Introduction and Features

Welcome to Golden!

Welcome to Golden, the professional SQL tool for Oracle.   This program allows you to create, edit and execute SQL statements and scripts.   This help file is designed to help you get the most out of your Benthic Software purchase.

Important Note:

Remember that you may be using a database which is being accessed by others.   Speak to your database administrator before using any ad-hoc sql tools.   You must understand how data locks, run-away queries, data contention, and other considerations will affect you and other users.

Features:

- Full 32bit for all versions of 32 bit Windows and Windows NT.
- SQL and PL/SQL editor with syntax highlighting.
- Point and Click SQLBuilder with Oracle keywords, all accessible Oracle Objects, column datatypes, primary key and index information.
- Running individual and multiple sql scripts.
- Timing of SQL statements.
- Explain Plan output.
- Spreadsheet results.
- Transposed Spreadsheet results.
- Editing of simple (single table) result sets.
- Editing of single table in a multi table query.
- Printing (with preview) of query results.
- Favorites menu with quick access to commonly used queries.
- Support for prompt variables (&,&&).
- Support for calling external script files (using the @ syntax.)
- Support for Bind variables including the REFCURSOR type.
- Support for connect keyword in scripts.
- Support for DESC and EXEC commands.
- Support for the SPOOL command.
- Direct data export to Excel, CSV files and XML.
- High speed external ascii delimited Import/Export.

License and disclaimer

Golden Copyright (C) 1996-2003 Benthic Software All Rights Reserved
http://www.benthicsoftware.com

1. License Agreement Golden (the "Software")

"Licensee" means the person or company identified below, who is being licensed
to use the "Software" identified in the title of this License Agreement.
"Licensor" means Benthic Software.

2. 30 Day Trial License
THIS IS NOT FREE SOFTWARE. Subject to the terms below, Licensee is hereby
licensed to use this software for evaluation purposes without charge
for a period of 30 days. If you use this software after the 30 day
evaluation period you are required to make payment for the software.

3. Limited Nonexclusive License

Licensee acknowledge that you are acquiring only a limited nonexclusive license
to use the Software (the "License"). Licensor shall remain the owner of all
right, title, and interest in the Software and in any copies of it.

The License permits Licensee to install the Software on more than one computer
system, e.g., a desktop computer and a laptop computer normally used by the
same person at different times, provided that there is no possibility that the
Software will be used on more than one computer system simultaneously; any such
simultaneous use requires a separate license for each computer system.

Licensee may make a reasonable number of backup copies of the Software solely
for backup purposes.

Licensee agrees not to make, nor to permit, the making of copies of the
Software (including its documentation) except as authorized by this License
Agreement or otherwise authorized in writing by Licensor.

Licensee agrees not to engage in, nor to permit, the de-compilation,
disassembly, or other reverse engineering of the Software.

4. Disclaimer of Warranty

THIS PROGRAM IS SUPPLIED AS IS. THE LICENSOR DISCLAIMS ALL WARRANTIES,
EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF
MERCHANTABILITY AND OF FITNESS FOR ANY PURPOSE. THE LICENSOR ASSUMES NO
LIABILITY FOR DAMAGES, DIRECT OR CONSEQUENTIAL, WHICH MAY RESULT FROM THE USE
OF THIS PROGRAM. THE LICENSEE MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM.
ANY LIABILITY OF THE LICENSOR WILL BE LIMITED EXCLUSIVELY TO PRODUCT
REPLACEMENT OR REFUND OF PURCHASE PRICE.

To the best of our knowledge Licensee's use of the Software in accordance with
its printed documentation will not, in and of itself, infringe any third
party's copyright, patent, or other intellectual property right.

No employee, agent, dealer, or distributor of ours is authorized to modify this
disclaimer of warranty, nor to make any additional warranties.

If implied warranties may not be disclaimed under applicable law, then ANY IMPLIED WARRANTIES ARE LIMITED IN DURATION TO 30 DAYS AFTER DELIVERY OF THIS COPY OF THE SOFTWARE TO LICENSEE.

Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to Licensee.

This warranty gives Licensee specific legal rights, and Licensee may also have other rights which vary from state to state.

5. No Incidental or Consequential Damages

Independent of the foregoing provisions, IN NO EVENT WILL LICENSOR BE LIABLE TO LICENSEE FOR ANY INCIDENTAL, SPECIAL, PUNITIVE, EXEMPLARY, OR CONSEQUENTIAL DAMAGES ARISING FROM OR CONNECTED WITH THIS AGREEMENT OR LICENSEE'S USE OF THE SOFTWARE, regardless whether Licensor know or have reason to know of the possibility of such damages.

Some states do not allow exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

6. Licensee's Indemnity to Licensor

Independent of the foregoing provisions, Licensee agrees to defend and indemnify Licensor against, and hold us harmless from, any and all claims, damages, losses, and expenses of any kind arising from or connected with the operation of Licensees business.

7. Termination

If Licensee materially breaches this License Agreement, Licensor may terminate Licensee's right to use the Software by notice to Licensee.

Licensee agrees that, upon termination of the License, Licensee will either return to Licensor or destroy all copies of the Software in Licensee's possession.

8. Entire Agreement, etc.

This written License Agreement is the exclusive agreement between Licensee and Licensor concerning the Software and supersedes any and all prior oral or written agreements, negotiations, or other dealings between the parties concerning the Software.

This License Agreement may be modified only by a writing signed by the parties.

In the event of litigation between Licensee and Licensor concerning the Software, the prevailing party in the litigation, and the prevailing party in any ancillary disputes (e.g., discovery disputes) will be entitled to recover attorneys' fees and expenses from the other party.

This License Agreement will be governed by the law of the State of Massachusetts applicable to contracts executed and performed entirely in, and by residents of, that state.

This License Agreement is effective upon the Licensee use of the Software. Licensee agrees that Licensor need not sign this License Agreement in order for it to take effect.

Login window

This window is where you login to Oracle. The login made is the 'public login' which is shared by all queries that are not on a 'private tab'. For more information on public vs private tabs, please see the How to work with tabs page. When you login, Benthic Software programs dynamically link to the Oracle libraries needed to connect to your database. If you have problems please see Connection/Login Problems in the Troubleshooting section of this help file.

You must supply the following information to login to Oracle:

Username:

This is the username supplied by your DBA. Example: SCOTT
Note: To login as SYSDBA or SYSOPER enter your username as "username as sysdba" or "username as sysoper"

Password:

This is the password supplied by your DBA. It will display as asterisks *. Example: TIGER

Database:

Ask your DBA for the connect string necessary for your database. Generally this will be a database alias setup using SQLNet EasyConfig or Net8 Assistant.

Here are some examples:

benthic	{ Alias created with Net Manager, EasyConfig, etc. }
t:myserver:orcl	{ A SQLNet 1.x database connect string }
beq-local	{ The default local connection for Net8 }
2:	{ Personal Oracle or local Oracle7 server }

Save This Password:

Checking the Save checkbox will cause your password to be save as an encrypted string so that you don't have to enter it again. **SECURITY ALERT!! Although the Save password option this is very convenient, please discuss security issues with your DBA before you decide to use it.**

More >> (Less <<):

Clicking the LoginList button shows or hides the "Quick Loginlist." This is a list of all of the logins that have ever been made on this machine from any Benthic Software product. Click on an entry to copy the username, password (if you chose to save it in the past) and database into the appropriate fields or double-click an entry to quickly login to that account (if the password was saved.)

Options...:

Click the Options button to see the Login Options Window.

Hints:

Right click the Quick Loginlist to see options to remove entries or clear the list.

Login Options window

The Login Options Window is where options related to the login window and SQLNet/Net8 troubleshooting are set.

This window contains the following settings:

Set focus to username field on startup:
Sets the focus to the username field when the login window is opened.

Never save passwords:
When checked the option disables the Save This Password option and passwords will never be saved to the registry.

Clear all login information:
Check this option and click 'Ok' to clear all stored login information.

OCI Dll Name:
If login fails and indicates that the registry settings are incorrect or that the oci linking file can't be find, you can use this field to hardcode the correct oci dll. Look in your ORACLE_HOME\bin directory for a file named oraxxx.dll and set this field to that name (i.e. ora805.dll) Note that in most cases this setting should be blank (not filled in.) The correct file for Oracle 8i and 9i is "OCI.DLL" which is the default setting when this field is blank.

Main window

The Main Window consists of the following parts:

The menu and toolbar:

The menu items and toolbar buttons all have quickhelp hints. As you pass over the button or menu item a short description will appear in the message area (at the bottom of the main window.)

Tabs:

The main work area can contain multiple 'tabs'. One tab is open for each file or script that you are working on. Please see the topic **'How to work with tabs.'**

The SQL Editing area:

Type your SQL statement(s) here. You may use tabs and returns to format your statement. You may quickly execute your statement(s) by typing <Shift-Enter>.

Script and delimiter examples.
Enter and run SQL statements.

The splitter bar:

This horizontal bar can be dragged up or down to change the relative sizes of the editing and results areas.

The spreadsheet results area:

When you execute an sql query, the results will show up here. You may resize the columns by sliding the header dividers left and right. You may copy results to the clipboard with the Edit menu. Note that results are put in the clipboard in a standard spreadsheet format that may be easily pasted into Excel or other spreadsheets.

The status bar and timing information:

The status bar at the bottom of the main window has three "panes" The pane on the left shows informative messages and status messages while running statements and scripts. The middle pane shows how many records were affected by the last statement. The pane on the right shows the elapsed time of the last statement/script execution.

Options window

The Options Window is where most program options are set. Changed options can be just for the current session or can be made the default. The Options Window contains a tab control that allows access to different sections of options. The sections are:

General:

These options are either global in nature, or didn't fit anywhere else!

Editor:

These options affect the formatting of the sql statements. Options for SQL Builder are also available here.

Spreadsheet:

Options that affect the spreadsheet result area.

Menus:

These options affect the Favorites menu and the Tools menu. Note that these menus may not be visible depending upon these settings.

Printing:

These options affect printing and the print preview window.

SQLBuilder window

The SQL Builder allows you to build an SQL statement by pointing and clicking. Clicking on a word inserts it into the cursor position of the current sql buffer. If the Smart checkbox is checked then the statement will be formatted nicely. The dictionary checkbox toggles the table list between the current schema and the data dictionary.

See "How to use the SQL Builder to build SQL statements" for an example!

The SQL Builder window consists of the following parts:

The SQL Builder toolbar:
The toolbar buttons all have quickhelp hints, just pause the mouse cursor over a button for a second, and an informative message will pop up. The small buttons insert a comma, a space, and an apostrophe respectively. The "Refresh" button will update the object tree using the user account of the currently displayed tab.

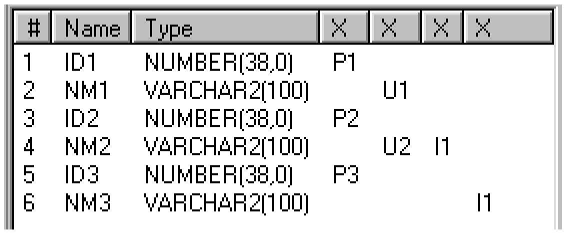
The SQL keyword list:
This list has commonly used SQL keywords. Click on a word to insert it into the active sql text buffer at the current cursor position (or replace the current selection.) Note that this list comes from the sqlbuild.txt text file. You may edit this file with any text editor to create a customized keyword list.

The splitter bars:
The horizontal bars can be dragged up or down to change the relative sizes of the lists.

The object tree:
This is a list of the tables, etc. available to you. Click a table name to fill in the listbox below with the column names of the table. Double click a name to insert it into the current sql text buffer. Note that the view can be changed to show User Objects, All Accessible Objects, or Dictionary Objects. Right click this field for more options.

The object tree search field:
Enter some text in the field and press Enter (or use the up and down arrow buttons) to search for the text in the current object tree. Note that only loaded information will be searched. You may have to expand some of the tree items in order to load the object information. There is also a right click menu called 'Fully expand this item' to quickly fill in an owner's objects.

The object detail list (Column List):
Click on the object tree above to fill in this list. Click on an item to insert it into the current tab text. For tables, this list also shows the datatype of each column, the primary key and any unique or nonunique indexes. Click on the column headers to sort by that column (click again to reverse sort.) Right click this field for more options.



#	Name	Type	X	X	X	X
1	ID1	NUMBER(38,0)	P1			
2	NM1	VARCHAR2(100)		U1		
3	ID2	NUMBER(38,0)	P2			
4	NM2	VARCHAR2(100)		U2	I1	
5	ID3	NUMBER(38,0)	P3			
6	NM3	VARCHAR2(100)				I1

This column list shows a table with 6 columns and:

- A primary key consisting of (ID1, ID2, ID3)
- A unique index consisting of (NM1, NM2)

- A nonunique index consisting of (NM2)
- A nonunique index consisting of (NM3)

Print Preview window

The Print Preview Window shows a representation of what a printed SQL script or result set would look like.   It allows you to change margins, display of grid lines, page size, orientation, and other printer related settings.

The Print Preview Window consists of the following parts:

The Toolbar:

The toolbar contains controls for viewing the pages and changing the attributes of the page and printer.

The Page View:

The page view shows a graphic of what the current report will print.

DBMS Output window

The DBMS Output window is for use with PL/SQL to output data.

Showing the window initializes the DBMS_Output package, you do not have to do it yourself. Then just call the dbms.put and dbms.put_line statements in your code. This is often used for pl/sql debugging. There is an example in the How to use the DBMS Output window topic.

The DBMS Output contents can be printed, cut to the clipboard, or cleared via the toolbar buttons.

How to Enter and run SQL statements

First login to the database when prompted after you start Golden.   This is the account that will be used for all queries unless you create a Private Tab (a tab that has it's own separate connection to a database.)   Once Golden finishes the login, you will see at least one 'tab' window.   A tab window is a window inside the Golden main window which has an sql editor on top and a spreadsheet on the bottom.   You type sql statements or scripts into the editor portion and see the results of queries in the spreadsheet.   After typing a statement (or opening a script file) you may run it by using the 'Run Script' menu option (on the Script menu), pressing F5, or by clicking the Run Script button on the button bar.   If your statement was a query, the results will be displayed in the spreadsheet.   The elapsed time to run the statement(s) will be displayed on the right hand side of the status bar.   The number of records affected by your statement will be displayed in the middle of the status bar.

You have the ability to run an entire script, run a script starting at the current cursor position, or run a single statement and stop.   Please see the rest of the help pages for more information.   Make sure to check out the keyboard shortcuts page!

There are many other ways to run scripts.   The options can be seen by looking at the Script menu.

How to Use the SQLBuilder to build SQL statements

Here is an example of using the SQL Builder to create the sql statement "select empno, ename, job from emp where mgr is not null"   This example assumes that the user logged on as SCOTT/TIGER.

1. Using an empty sql edit window, click on the SQL Builder toolbar button.
2. Click on 'Select' in the keyword pane of the SQL Builder, this will copy the word 'Select' into the sql edit window.
3. Now click on the 'emp' table in the table pane of SQL Builder.   Note that the columns list fill with the column names and datatypes of the emp table.   Notice that the 'emp' word does NOT get pasted into the sql edit window.
4. Now click on 'empno' and then 'ename, and then 'job' in the column pane of sql builder.   Note that the fields and commas are pasted into the sql edit window.
5. Now click on 'from' in the keyword pane.
6. Now DOUBLE CLICK 'emp' in the table pane.   You must double click a table name to paste it into the edit window because single clicking it just displays its fields.
7. Now click on 'where' in the keywords pane, 'mgr' in the columns pane, and 'is not null' in the keyword pane.
8. You're done!   Click the Run button to run the query!

How to Edit data with Edit Mode

Overview:

Besides writing insert, update and delete SQL statements you can edit data in Golden's spreadsheet after running a query in Edit Mode. Golden will allow you to edit simple queries (queries of a single table) or the first table of multi-table queries. Golden works best when the first column in the select statement is the ROWID of the table you wish to edit (an example would be 'select e.rowid, e.* from emp e') If you do not include the ROWID as the first column then the table must have a primary or unique key to be edited. Note that queries with multiple tables MUST include the ROWID of the first table in the from list (which is the editable table) as the first column. A simple example of this would be

```
select e.rowid, e.empno, e.ename, d.deptname from emp e, dept d where e.deptno = d.deptno
```

This query would let you edit the empno and ename fields of the EMP table. The editable table name is shown on the toolbar above the spreadsheet. Columns that can't be edited are shown in RED.

Entering Edit Mode:

Type in a query and click the data editing button (the one that looks like the run button but with an 'E' on it.) You may then directly edit cells in the spreadsheet. Records get saved to the database when you move the current focus to a different record (by clicking or using the arrow keys.) You can also use the 'Accept Edit' button on the editing button bar. You may press escape to abort editing the current cell and then the current row (so you may have to press escape twice to restore a row to it's pre-edited state.

Transactions:

The following only applies if you do not have AutoCommit on in the program options.
No other users of the database will see your changes until you click the commit button or logout. You will see the changed data if you requery the database. This can be very useful to check your work. Then if you are satisfied with the results you can commit, or rollback if you don't want to accept the changes and return the database to the state of your last commit. It is a good idea to commit your data after you have made sure that it is correct. Oracle will commit your data changes when you cleanly logout from the database. Oracle will rollback your data changes if you lose your connection to the database.

Selecting and editing a cell:

Note that a spreadsheet cell can be the 'current' cell but not be in edit mode. You can tell if a cell is in edit mode if the text cursor is shown inside the cell. You can enter edit mode by clicking on data after you have selected the cell (but not double clicking, click once to select and again after a moment to move the cursor into the text.) You can also press the F2 key to begin editing (this is a Windows standard.) Once you are finished editing, you can press Enter or Tab to accept the edit.

Inserting Data:

You may insert records by typing them into the bottom empty record displayed in the spreadsheet. You may also paste from the clipboard into the spreadsheet. The data should be tab delimited text data. You will be prompted if you are trying to paste more than one record.

Updating Records:

You may update records by typing values right over the existing ones. You may update by pasting data from the clipboard. You will be prompted if you are trying to paste more than one record. The paste will occur starting at the currently selected cell.

Deleting Records:

You may delete records by moving to the record and then clicking the 'Delete records' button. You can select multiple records to delete by clicking and dragging a selection on the left most column of the spreadsheet.   Ctrl-Clicking allows you to select multiple individual records.   You will be asked to confirm your deletion.

Note: Be careful that data fields contain all the date information that you want.   You may want to use the date format mm/dd/yyyy hh24:mi:ss in order to include all date information.

Hint:

You can still edit a table that doesn't have a primary key if you use the rowid as the first field in the query
(i.e. select rowid, emp.* from emp)
If you are getting a message saying "Another user has modified the record" and you know that this hasn't happened, try adding 'ROWID' as the first column and re-run the query in edit mode.

How to Use the DBMS Output window to get pl/sql output

To display output from the DBMS_OUTPUT package you must first show Golden's DBMS Output window (from the View menu.) If the window is displayed, any DBMS_OUTPUT statements (put, put_line) will display data in the window.

Here is an example of a pl/sql block that output data to the DBMS Output window:

Make sure to show the DBMS Output window before running it. Note that you can cut and paste from this window.

```
declare
    CurrTime varchar(20);
begin
    select to_char( sysdate, 'HH24:MI:SS' ) into CurrTime from dual;
    dbms_output.put_line( 'Current Server Time:' );
    dbms_output.put( '     ' );
    dbms_output.put_line( CurrTime );
end;
/
```

Produces the output:

```
Current Server Time:
   09:39:09
```

How to Use the DESCRIBE command

The DESC[RIBE] command is used to display the structure of tables and the arguments of stored procedures.

Here is an example of its use:

```
  DESC EMP
```

This will display the columns and datatypes that make up the EMP table.

```
  DESC UPDATE_ACCOUNTS
```

Will display information about the arguments and return value (if it is a function) of a stored procedure. You can also describe functions and procedures in packages by describing the package.

How to Use the EXECUTE command

The EXECUTE command (can also use EXEC) is used to run stored procedures on the server.

Bind variables can be used with the EXEC command to pass arguments and return values including refcursors.   Please see the 'How to use Bind Variables' help topic for more information and examples.

Output of OUT arguements and function results are displayed using 'BIND variables.'   Bind variables are names starting with a colon (i.e. :RETVAL)   The contents of bind variables are displayed when the exec call is complete.

Here is an example of its use:

```
REM My sample function:
create or replace function marktest( arg1 number, arg2 out varchar ) return
number is
begin
   arg2 := arg1 + 10;
   return arg1-5;
end;
/
REM Here is the execute call:
EXEC :RETVAL := marktest( 11, :OUTARG );
PRINT
```

Running the above script will display:

```
RETVAL = 6
OUTARG = 21
```

Use the SPOOL command

Golden now supports the 'SPOOL' command. This command allows you to store the results (and optionally the SQL commands) of a script to a text file. Here are the commands related to this feature:

SPOOL FILENAME
Will copy results and timings to a text file. If the filename contains spaces enclose it with double quotes (i.e. SPOOL "C:\MY FILES\MY SPOOLFILE.TXT"

SPOOL OFF
Turns off spooling and closes the spool file.

SET ECHO ON/OFF
If ECHO is on then commands are placed in the spool file before the results.

Example script using the SPOOL commands:

```
REM Start a spool file:
SPOOL "C:\MYSPOOLFILE.TXT"
SET ECHO ON
SELECT * FROM DEPT;
INSERT INTO DEPT ( DEPTNO, DNAME, LOC ) VALUES ( 60, 'HR', 'ORLANDO' );
SPOOL OFF
```

```
will generate the file:

06/25/2001 09:53:10 am Golden Spool File

SQL> SELECT * FROM DEPT

DEPTNO DNAME      LOC
------ ---------- --------
10     ACCOUNTING NEW YORK
20     RESEARCH     DALLAS
30     SALES       CHICAGO
40     OPERATIONS   BOSTON

4 rows selected in 0.002 Secs

SQL> INSERT INTO DEPT ( DEPTNO, DNAME, LOC ) VALUES ( 60, 'HR', 'ORLANDO' )

1 rows created in 0.002 Secs
```

Use Prompt Variables

Prompt variables allow your scripts to be more interactive.   Prompt variables are variables that can be defined and given a value which will be substituted in place of the variable before statements are passed to the database for processing.   Prompt variables are specified using the & and && characters.   The use of these can be a bit confusing at first, but they work in Golden exactly as they work in SQLPlus.

Each Tab in Golden (editor 'page') manages its own list of prompt variables.   You can use the 'Variables' submenu on the 'Script' menu to edit and clear the current tab's prompt variables.

In a script, a prompt variable that starts with a single & character is a temporary prompt variable.   You will be prompted for its value each time it is seen by the parser.   Using the && character before a prompt variable makes it permanent.   You will no longer be prompted for its value even if it appears later with a single &.

Here's an example of a statment with a temporary prompt variable:

```
  select * from emp where ename like '&Lastname';
  select * from emp where ename like '&Lastname';
```

When you run this statement in Golden, you will be prompted to enter a value for &Lastname twice (once for each statement)

Here's an example of a statment with a single prompt variable:

```
  select * from emp where ename like '&&Lastname';
  select * from emp where ename like '&Lastname';
```

Here you will only be prompted for Lastname once and that value will be used for both statements.

You can define a prompt variable in script using the DEFINE command:

```
  define lastname = "FORD";
  select * from emp where ename like '&lastname';
```

Note that the define command acts like a && prompt variable (it is permanent and you won't get prompted for that variable again.

You can undefine a prompt variable using the UNDEFINE command:

```
  undefine lastname;
```

Use Bind Variables

Bind variables allow your scripts to be more interactive and can be very useful in calling pl/sql stored procedures.   Bind variables are variables that can be defined and given a value which will be passed to the server and processed by the database.   Bind variables are created using the VAR command and used in statements by placing a colon : in front of the Bind Variable name.   Note that each Script tab in Golden manages its own Bind variables (they aren't shared among tabs.)

The var command syntax to create a bind variable is:
  var varname vartype
    The choices for vartype are:
      INTEGER: In Oracle this is a NUMBER(38,0) type.
      NUMBER: This is a floating point value.
      STRING: A string of 2000 (4000 for Oracle 8+) max characters.
      PLSQLSTRING: A string of 32K characters (varchar size in pl/sql)
      REFCURSOR: A referenced cursor (see example below.)

    Note that if you use string or plsqlstring, Oracle will convert your variable to the appropriate type (except REFCURSOR) automatically.
    Another Note: Golden will autodefine any bind variables it sees in your statements as string type and give them an initial value of "" if they don't already exist.

The var command syntax to assign a value to a bind variable is:
  var varname = "Data" (remove the quotes if it isn't a string.)

The var command for clearing all bind variables (you can also use the 'Clear all bind variables in this tab' menu choice Script | Variables) is:
  var clearall

To see all the defined bind variables use:
  var
    or
  print

To see a single bind variables use:
  var varname

Note that Golden displays one REFCURSOR at a time in the spreadsheet results area.   You can switch between REFCURSORs by using the 'Results | Bind Variable Cursors' menu.   The REFCURSOR will be displayed in the spreadsheet window.   All other bind variables are displayed in the text view window.

Here is an example of a very simple script using bind variables:

```
REM First we define some bind variables.
var MyVar1 string
var MyVar2 string
var MyVar3 number
var MyVar4 integer
REM This next line allows you to set a value for a bind variable.
var MyVar2="Software";
REM Now we run a pl/sql block with the bind variables in it.
begin
  :MyVar1 := 'Benthic ' || :MyVar2;
  :MyVar3 := 3*1.4;
  :MyVar4 := 5+6.2;
end;
```

```
  /
  print
```

And the result of running this script is:

```
  -- 01/04/2000 08:24:30 pm
  MYVAR1 STRING = Benthic Software
  MYVAR2 STRING = Software
  MYVAR3 NUMBER = 4.2
  MYVAR4 INTEGER = 11
```

Here is an example of a very simple EXEC call using bind variables:

```
var arg1 string
var arg2 string
var arg3 string
var arg4 string
var ret1 string
var arg1 = "Gort, ";
var arg2 = "Klaatu ";
var arg3 = "barada ";
var arg4 = "nikto";
exec :ret1 := MyConcatFunction( :arg1, :arg2, :arg3, :arg4 );
print ret1
```

And the result of running this script is:

```
  -- 01/04/2000 08:45:07 pm
  RET1 STRING = Gort, Klaatu barada nikto
```

Using and Displaying REFCURSOR bind variables

Refcursor bind variables work the same way as other bind variables except that you can not give them an initial value and they are displayed in the spreadsheet window. Each REFCURSOR containing data is shown on the 'Results' menu on the 'Bind Variable Cursors' submenu. Just select a REFCURSOR to see the data in the spreadsheet.

Note that if you call your REFCURSOR bind variable "cursor", you do not need to declare it with a VAR statement.

The example uses the following package and function:

```
create or replace package types
as
    type cursorType is ref cursor;
end;
/
create or replace function GetMyTablesCursor return types.cursortype
as
    l_cursor      types.cursorType;
begin
    open l_cursor for select * from tab;
    return l_cursor;
end;
/
```

Here is the example:

```
var MyCursor refcursor
exec :MyCursor := GetMyTablesCursor;
```

Or you could just use:

```
exec :cursor := GetMyTablesCursor;
```

How to Create and use the favorites menu

On the "Menus" page of the Options Window, is a field called "Favorites Menu Root" If this field is set to a disk directory that contains *.sql files, a Favorites menu will appear on the main window. This menu will contain items for each of your .sql files and also will contain folders for any subdirectories that have .sql files.

For items in your Favorites menu, you can:

Click item: Will load the file into the active sql editor window and run it. Be careful when autocommit is set! I suggest only using queries on your favorites menu!

Shift-Click item: Will load the file into the active sql editor window, but will not run it.

Control-Click item: Will prompt you to delete the .sql file for this item.

For example:

I have the following directory structure:

```
C:\Benthic\              <-- Contains Golden32.exe
C:\Benthic\SQL\
    File:  Show Tabs.sql
    File:  Show Indexes.sql
C:\Benthic\SQL\Patient Queries\
    File:  Show Patients with Hypertension.sql
```

If my "Favorites Menu Root" field is set as "C:\Benthic\SQL" (without the quotes), My favorites menu will contains the Items "Show Tabs" and "Show Indexes". It contains a subfolder called "Patient Queries" with one item, "Show Patients with Hypertension"

If the favorites menu doesn't seem to work:

A minimum test would be to create a .sql file called test.sql that contains the text "select * from tab". Place it into the same directory as Golden32.exe and set the Favorites Menu Root to "." (without the quotes.) Your favorites menu should then contain "Refresh Favorites" and "test"

How to Work with public and private tabs

The main work area consists of a tabbed 'notebook' where multiple files can be opened at once. Tab text is underlined in red when the tab is 'dirty' and needs to be saved.

There are two types of tabs, public and private.

Public Tabs:

Public tabs share the main program login displayed in the caption of the main window. For example, if you logged into scott/tiger at startup, all new public tabs will share that single connection to the database. This conserves resources. Since they share a connection to the database, some operations can cause blocking. Basically, long operation that don't involve retrieving records (updates, deletes, etc.) can block other public tabs from operating. Note that as soon as the blocking operation is finished, work will continue as normal.

Private Tabs:

Private tabs are tabs that have their own private database session. When creating a private tab (from the 'File' menu) you will be prompted for the user/password/server information. Private tabs display their username and server above the script window. Besides allowing you to connect to multiple accounts/servers at once, private tabs will never cause blocking as discribed above. The file menu has choices for creating public and private tabs.

Using Workspaces

Workspaces allow you to save the current tab information to a file.   The workspace can then be opened, restoring your tabs and scripts.

What is saved:

The 'Save Workspace files as links' options setting (on the General tab) specifies how tabs that contain files are saved.   If the option is checked than tabs that have filenames associated with them only save the filename to the workspace.   When the workspace is loaded, linked filenames are found and their text is loaded.

Setting tab captions:

Tab captions default to 'QueryN' or the current filename loaded in the tab.   To change a tab's caption, make the tab the current tab by clicking on it.   Then right click the tab area to brink up a window where the tab can be changed.   Tab names are only saved if you then save the workspace.

FAQ (Frequently Asked Questions)

Why can't I connect?

Assuming that the database and network is in working order, the most common cause of connection problems is a damaged SQLNet or Net8 installation. In many cases, re-installing SQLNET and checking the database aliases in Oracle EasyConfig will solve the problem. Benthic products try to load the most current version of SQLNET/Net8 that is installed on your system. It does this by searching for the OCI (Oracle Call Interface) .dll file on your machine. It is possible that there is an 'orphaned' OCI file that is causing the problem. Re-installing can solve the problem as Oracle scans for OCI dll files during the install and can remove them. If you are still having a problem, you can enter an oci dll filename on the Login Options Window to fix the problem. Of course, if you are still having problems, please contact Benthic Software and we will be glad to help you!

How can I select a range of cells with the mouse?

If you move the mouse to the left side of a spreadsheet cell, you should see the cursor change slightly to signify that you are in 'range select mode.' Just left click and drag as normal to select a range of cells. Note that besides selecting ranges with the mouse, you can also select rows and columns by clicking and dragging. You can select the entire spreadsheet by clicking the upper left cell.

Why am I getting date related errors?

In many Oracle tools, the default date format is 'DD-MON-YY' The problem is that data can hidden by this particular format (not even mentioning the year 2000 consideration!)

For example, say you insert the SYSDATE value into a date field in the database and then display it in SQLPlus. You will see something like '15-MAR-67'. Now select it using to_char(mydate, 'MM/DD/YYYY HH24:MI:SS') and you will see something like '03/15/1967 08:32:53' The first call is hiding the 4 digit year and the time.

You can change Golden's default date format to 'DD-MON-YY' to be compatible with Oracle's tools, but it is not really recommended. It is much better to use the TO_DATE(and TO_CHAR(commands to explicitly perform the conversion. This way the sql will be compatible no matter what the default settings are.

Note that Golden will not let you update a date field that is currently hiding information in Edit mode.

Quick Tips

Use sql to make sql:

It can be useful to use an sql query to generate other sql statements. This is often useful in maintainance.

Enter an sql query like

```
select
    'alter ' || object_type || ' ' || object_name || ' compile;'
from
    user_objects
where
    object_type = 'PROCEDURE' and status = 'INVALID';
```

Execute it and cut the result set from the spreadsheet. Paste it into a query window and execute it.

Script delimiters:

Separate SQL statements with a semicolon and Enter (;<crlf>).

Example:
```
delete * from emp where ename like 'F%';
select * from emp;
```

If you have pl/sql in your scripts, separate statements with <crlf>/<crlf>.

Example:
```
delete * from emp where ename like 'F%';
begin
    -- Kind of a useless example, but hopefully you get the idea!

    insert into emp select * from emp2;
end;
/
select * from emp;
```

Comments:

In scripts, you can use the REM comment. Inside sql statements or pl/sql use -- for single line comments or /* */ for block comments.

Keyboard shortcuts

Application Help

Display Help Contents	F1

Tab and File Controls

New Tab	Ctrl-N
Clear Current Tab (New current Tab?)	Shift-Ctrl-N
New Tab with Private Login	Shift-Ctrl-Alt-N
Open File Into New Tab	Ctrl-O
Open File Into Current Tab	Shift-Ctrl-O
Open File Into New Tab	Shift-Ctrl-Alt-O
Save File	Ctrl-S
Close Tab	Ctrl-F4
Close Program	Alt-F4
Goto Next Tab	Ctrl-Tab
Goto Prior Tab	Shift-Ctrl-Tab
Goto Tab	Alt-# (# = keys 1 thru 0 for tabs 1 thru 10)

Editing Keys

Standard Editing:

Undo	Ctrl-Z
Redo	Shift-Ctrl-Z
Cut	Ctrl-X
Copy	Ctrl-C
Paste	Ctrl-V
Select All	Ctrl-A

Find & Replace:

Find	Ctrl-F
Find Next	F3
Replace	Ctrl-H

Bookmarks and Navigation:

Top of script	Ctrl-Home
Bottom of script	Ctrl-End
Set a bookmark within a script	Shift-Ctrl-# (0-9)
Goto a bookmark	Ctrl-# (0-9)
Toggle Between Beginning Of Line And First Non-Whitespace Character	Home

Text Block Commands:

Block Indent	Ctrl-I or Tab when selection
Block Unindent	Ctrl-U or Shift-Tab when selection
Comment Out Selected Lines	Ctrl-- (Ctrl - Dash)
Uncomment out selected Lines	Shift-Ctrl-- (Shift-Ctrl-Dash)

Find Matching brackets:

Highlight inside brackets or quotes	Ctrl-B
Find previous matching ({[<	Ctrl-[(Place caret on bracket first)
Find next matching ({[<	Ctrl-] (Place caret on bracket first)

Miscellaneous:

Toggle case of selection	Ctrl-T

Oracle Session

Login	Ctrl-L or Ctrl-J

Script Execution

Run Script	F5 or Shift-Enter while typing.
Run Script From Cursor	F6
Run One Statement At Cursor	F7 or Ctrl-Enter while typing or Ctrl-RightClick.
Run Script And Go To Edit Mode	Ctrl-E

Commit	Ctrl-F5
Rollback	Ctrl-F6
Show Execution Plan	Ctrl-P

Clipboard Extensions

Copy SQL To Clipboard With Language Formatting	Shft-Ctrl-C
Paste SQL From Clipboard Stripping Language Formatting	Shft-Ctrl-V

Window Controls

Toggle Between Edit And Results Windows	Ctrl-R or F8
Toggle SQL Builder	F9
Toggle DBMS Output Window	F10
Toggle Scratch Results Window	F11
Toggle Log View	F12

Thanks to Paul Jakins for compiling this list!

Connection/Login Problems

Assuming that the database and network is in working order, the most common cause of connection problems is a damaged SQLNet or Net8 installation. In many cases, re-installing SQLNET and checking the database aliases in Oracle EasyConfig will solve the problem. Benthic products try to load the most current version of SQLNET/Net8 that is installed on your system. It does this by searching for the OCI (Oracle Call Interface) .dll file on your machine. It is possible that there is an 'orphaned' OCI file that is causing the problem. Re-installing can solve the problem as Oracle scans for OCI dll files during the install and can remove them. If you are still having a problem, you can enter an oci dll filename on the Login Options Window to fix the problem. Of course, if you are still having problems, please contact Benthic Software and we will be glad to help you!

Overview

Note: This section of the help file was written by Keith Wood. Keith Wood is an analyst/programmer/author who has written many fine articles for Delphi Informant Magazine. Keith has no relationship to Benthic Software so please don't contact him for help with Golden!

Before Oracle can execute a SQL query, it must decide on how to efficiently access the data.

To do this Oracle invokes an optimizer to determine the best method to use. It has three modes of operation available:

RULE	The rule-based optimizer evaluates alternative execution paths based on a series of rules.
COST	The cost-based optimizer uses database statistics to weigh the effectiveness of various access methods to select the best one to use.
CHOOSE	This mode calls the cost-based optimizer if the tables have been analyzed and the rule-based optimizer if not.

After selecting the optimizer to use, Oracle has many operations that it can apply to the underlying tables to effectively retrieve the data. These are applied in combination to complete the query. An example shows how this works in practice. There are several things to keep in mind while optimizing a query. In addition, hints can be supplied to guide Oracle in its determinations.
Some operations require the entire set of records to be available before they can be processed and returned to the client. These are marked with the following symbol 🕒. Operations that return the first record quickly to the client while continuing to process the query are marked with the following symbol ①.

The operations are divided up by function as follows:

Filter and View
Index Access
Join Operations
Multiple Index Scans
RowNum Operations
Set Operations
Sorting Operations
Table Access
Update Locks
Connect By
Sequences

Or they can be accessed individually:

And-Equal
Concatenation
Connect By
Count Stopkey
Count
Filter
For Update
Hash Join
Index Range Scan
Index Unique Scan
Intersection
Merge Join
Minus
Nested Loops Outer
Nested Loops
Sequence
Sort Aggregate
Sort Group By
Sort Join
Sort Order By
Sort Unique
Table Access By RowId
Table Access Cluster
Table Access Hash
Table Access Full
Union-All
View

Table Access
These operations directly access the tables.

TABLE ACCESS FULL 🕒

The entire table is scanned sequentially. This method is applied when no indexes are available for use or when there is no where clause in the query. For small tables this operation can be very useful but as the size of the table grows it becomes less efficient.

For example:
```
select LAST_NAME
from EMPLOYEE
```

TABLE ACCESS BY ROWID ①

The table is accessed via its ROWID pseudo-column. This column encodes the physical location of the record in the table and so is a very effective access path. Access by ROWID is used when ROWID appears in the **where** clause or following retrieval via an index.

For example:
```
select LAST_NAME
from EMPLOYEE
where ROWID = '00000123.0001.0001'
```
(note, format of ROWID depends upon your database version)

TABLE ACCESS CLUSTER 🕒

This operation may be used to read the table if it is stored in a cluster. Data manipulation on clustered tables is generally less efficient than the same tasks on non-clustered tables.

TABLE ACCESS HASH

This operation may be used to read the table if it is stored in a hash cluster. The physical location of a record is determined by the values in the row in a hash cluster.

Index Access

Oracle supports two major types of indexes: *unique* and *non-unique*.
Indexes are used to resolve a query when you

- look for an indexed field with an exact value
- look for a range of values on an indexed field (including **between** and **like** 'xyz%')
- don't use a function on the column in the **where** clause
- don't use **is null** or **is not null** (indexes don't store null values)
- use equivalence operators (i.e. don't use **!=** or **not in**)
- set the leading column of a multi-column index equal to a value
- use the **max** or **min** functions on an indexed field
- use a selective index (i.e. there are few records for each index value) and the cost-based optimizer

INDEX UNIQUE SCAN ①

This operation denotes direct access to a single record via a unique index. As such it is very efficient. To be used the query must have a **where** clause that equates the indexed field with another value. If the columns requested are part of the index then they are returned from there. Otherwise the index provides the ROWID which is used to retrieve the record with a **TABLE ACCESS BY ROWID** operation.

For example:

```
select *
from EMPLOYEE
where EMP_NO = 123
```

INDEX RANGE SCAN ①

When searching for a range of values in an indexed field or via a non-unique index, this operation is used. As above the requested columns are returned directly from the index if available, or are retrieved from the table using the ROWID.

For example:

```
select *
from EMPLOYEE
where LAST_NAME like 'SMI%'
```

Multiple Index Scans

If limiting conditions are applied to more than one indexed field or more than once to the same field, then multiple index scans may be used to retrieve the records. These are then combined using the operations below.

AND-EQUAL

When two different indexes are used the resulting lists of ROWIDs may be combined with this operation. The combined list then feeds into a **TABLE ACCESS BY ROWID** operation to retrieve the records.

For example:

```
select *
from EMPLOYEE
where LAST_NAME > 'SMI'
  and EMP_NO > 123
```

CONCATENATION

When querying multiple values on the same indexed field this operation combines the resulting ROWIDs. The combined list then feeds into a **TABLE ACCESS BY ROWID** operation to retrieve the records.

For example:

```
select *
from EMPLOYEE
where LAST_NAME in ('SMITH', 'SMYTHE')
```

Sorting Operations

These operations work on sets of data retrieved by previous operations, such as **INDEX RANGE SCAN** or **TABLE ACCESS FULL**. Due to their nature these operations cannot return any records until the entire function has completed.

SORT ORDER BY ⏲

Invoked when an **order by** clause appears in the query, sorting by a field that is not indexed.

For example:

```
select *
from EMPLOYEE
order by FIRST_NAME
```

SORT UNIQUE ⏲

Invoked when the **distinct** keyword appears in the query, this operation eliminates duplicates in the selected fields. Unique sorts are also performed when **union**, **minus** or **intersect** are used in a query.

For example:

```
select distinct FIRST_NAME
from EMPLOYEE
```

SORT JOIN

This operation is only used as preparation for a **MERGE JOIN**.

SORT AGGREGATE ⏲

Used when grouping functions (such as **max**, **min**, and **count**) appear in the query, but there is no **group by** clause. Note that **min** and **max** on an indexed field will use an index scan only.

For example:

```
select max(SALARY)
from EMPLOYEE
```

SORT GROUP BY ⏲

Used when grouping functions (such as **max**, **min**, and **count**) appear in the query, and there is a **group by** clause.

For example:

```
select DEPT_NO, max(SALARY)
from EMPLOYEE
group by DEPT_NO
```

RowNum Operations
The ROWNUM pseudo-column is a sequential number generated for each record returned by a query. It operates on the output of a previous operation that retrieves the records.

COUNT

This operation adds the ROWNUM pseudo-column to each record.

For example:
```
      select LAST_NAME, ROWNUM
      from EMPLOYEE
```

COUNT STOPKEY

This operation adds the ROWNUM pseudo-column to each record and is used when a condition is placed on the count.

For example:
```
      select LAST_NAME, ROWNUM
      from EMPLOYEE
      where ROWNUM < 20
```

Set Operations

These operations work on sets of data.

UNION-ALL

This operation combines two sets of records without checking for duplicates, i.e. when a **union all** appears in the query. If only a **union** is used in the query then this operation is followed by a **SORT UNIQUE** operation.

For example:

```
select LAST_NAME, FIRST_NAME
from EMPLOYEE
union
select LAST_NAME, FIRST_NAME
from CUSTOMER
```

MINUS

This operation removes records selected in a second query. To achieve this the input to this process must be the output from two **SORT UNIQUE** operations on the underlying queries.

For example:

```
select LAST_NAME, FIRST_NAME
from EMPLOYEE
minus
select LAST_NAME, FIRST_NAME
from CUSTOMER
```

INTERSECTION

This operation retains only those records appearing in both sub-queries. Like the previous operation, the input comes from two **SORT UNIQUE** operations on the underlying queries.

For example:

```
select LAST_NAME, FIRST_NAME
from EMPLOYEE
intersect
select LAST_NAME, FIRST_NAME
from CUSTOMER
```

Update Locks
FOR UPDATE

Another set operation, this one places locks on the retrieved records to prevent others from accessing them.

For example:
```
      select *
      from EMPLOYEE
      for update of DEPT_NO
```

Filter and View

FILTER

Conditions placed on grouping functions are applied by this operation.

For example:

```
      select DEPT_NO, count(DEPT_NO)
      from EMPLOYEE
      group by DEPT_NO
      where count(DEPT_NO) > 10
```

VIEW

The results of a previous operation are returned to the client, typically from a query that is defined as a view.

Join Operations

Join operations are used to link two or more tables together. If more than two tables are linked then Oracle treats them as a series of two-table joins. The first is joined with the second, then the result is joined with the third, etc. Because of this the smallest tables should be the first to be joined, reducing the processing load in subsequent steps.

MERGE JOIN 🕘

The simplest of the join operations, this one is used to combine two separate inputs into one. The inputs must already exist and so this operation is slow to return to the client. Each input is processed by a **SORT JOIN** operation before being merged. It is usually invoked when there are no indexes available for use.

For example:

```
select D.DEPT_NAME, E.LAST_NAME
from DEPARTMENT D, EMPLOYEE E
where D.DEPT_NO = E.DEPT_NO
```

NESTED LOOPS ①

Nested loops combine two tables by stepping through one and accessing the other for each record in turn. This allows the first records to be returned almost immediately. For this to work there must be an index available on one of the joining fields.

For example:

```
select D.DEPT_NAME, E.LAST_NAME
from DEPARTMENT D, EMPLOYEE E
where D.DEPT_NO = E.DEPT_NO
```

HASH JOIN 🕘

This operation requires the two input sets to be available before it can start. It reads one set and applies a hashing function to its key as it is saved to memory. The second set is then read, the same hashing function applied and any corresponding record located. Use of this operation depends on the optimizer.

For example:

```
select D.DEPT_NAME, E.LAST_NAME
from DEPARTMENT D, EMPLOYEE E
where D.DEPT_NO = E.DEPT_NO
```

NESTED LOOPS OUTER ①

This variant of the **NESTED LOOPS** operation is used during an outer join. The driving table is selected to be the *outer* table.

For example:

```
select D.DEPT_NAME, E.LAST_NAME
from DEPARTMENT D, EMPLOYEE E
where D.DEPT_NO = E.DEPT_NO(+)
```

Connect By

CONNECT BY

This operation is invoked by the **connect by** clause in a query. It requires three inputs, a table scan to find the root node, and a pair of table access operations to traverse the tree. Appropriate indexes can bypass the need for **TABLE ACCESS FULL** operations.

For example:

```
select EMP_NO, LAST_NAME
from EMPLOYEE
start with LAST_NAME = 'SMITH'
connect by EMP_NO = prior MANAGER_NO
```

Sequences

SEQUENCE

The next value from a sequence is selected by this operation.

For example:

```
      select SEQ.NEXTVAL
      from DUAL
```

Hints

Oracle allows hints to be supplied in the SQL query to assist in determining the execution path to be used. These are supplied in modified comments (delimited by /*+ and */) following the initial command keyword. The hints may not be used at all.

Hint	**Purpose**
ALL_ROWS	Use the plan that returns all of the rows as quickly as possible.
FIRST_ROWS	Use the plan that returns the first row as quickly as possible.
FULL(table_name)	Use the **TABLE ACCESS FULL** operation on the specified table.
INDEX(table_name)	Use an index on the specified table.
INDEX(table_name index_name)	Use the specified index on the nominated table.
USE_NL	Use the **NESTED LOOPS** operation.
USE_NL(table_name)	Use the **NESTED LOOPS** operation with the specified table as the inner table.
ORDERED	Join the tables in the order specified in the **from** clause.
USE_MERGE(table_name, table_name)	Use the **MERGE JOIN** operation on the specified table.
USE_HASH	Use the **HASH JOIN** operation.
RULE	Use the rule-based optimizer.

For example:

```
select /*+ FIRST_ROWS */ D.DEPT_NAME, E.LAST_NAME
from DEPARTMENT D, EMPLOYEE E
where D.DEPT_NO = E.DEPT_NO
```

A Complete Example

To show how these operations combine to provide the results of the query consider the following SQL:

```
select XXX.YYY, CCC.AAA, HHH.JJJ
from XXX, CCC, HHH
where XXX.ZZZ = CCC.AAA
  and CCC.BBB = HHH.KKK
```

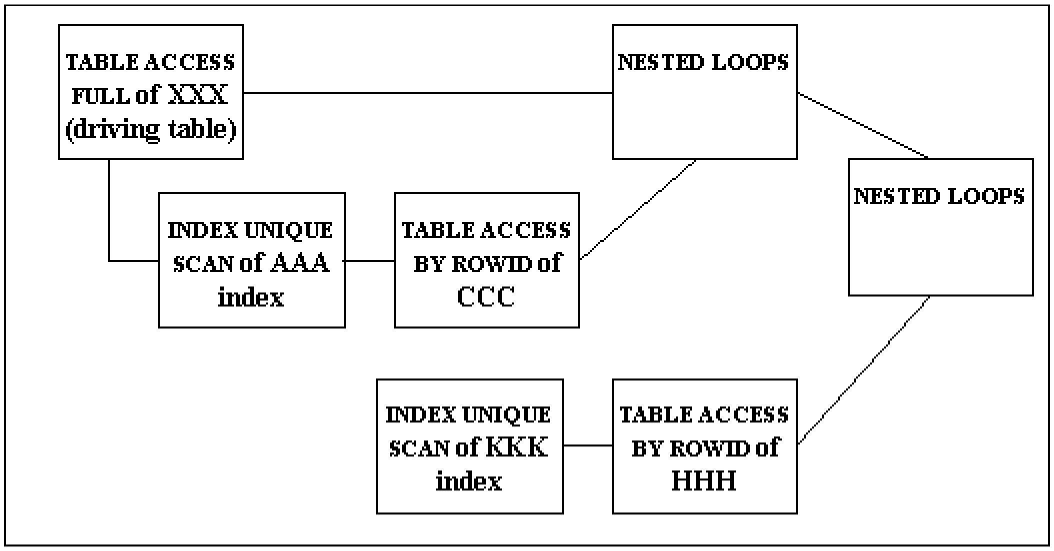


TABLE ACCESS FULL of XXX (driving table)
NESTED LOOPS
NESTED LOOPS
INDEX UNIQUE SCAN of AAA index
TABLE ACCESS BY ROWID of CCC
INDEX UNIQUE SCAN of KKK index
TABLE ACCESS BY ROWID of HHH

This involves two joins, and with appropriate indexes can use the execution plan shown below. Table XXX is accessed in full. For each record therein, the appropriate record from table CCC is retrieved. The resulting set is then used as input to another nested loop to add the values from the third table.

Suggestions

When optimizing a query several things need to be considered:

- Do you want the first rows back quickly (typically during online processing) or is the total time for the query (typically a batch process) more important?
- Are the tables properly indexed to take advantage of the various operations available?
- How large are the tables? Joining smaller tables first is usually more efficient.
- How selective are the indexes? Indexes on fields that have only a few values don't really help.
- How is sorting done? Are sorting and grouping operations necessary?

This window allows you to rename or reorder the tabs. You may drag and drop the tabs in the list or select one and use the Up or Down arrow buttons to move it. Clicking the 'E' button (for edit) or pressing F2 will allow you to edit the name of the currently selected tab in the list. Note that the tab names are only 'persistant' if you save a workgroup. Otherwise the tab name will default to the filename when loaded.

Legend

① Operation returns first row quickly.

🕒 Operation requires the entire set of records to be available before they can be processed and returned to the client.