Oracle database tips

Summary:	DB_BLOCK_BUFFERS size tuning.
Description:	Find Memory/Physical read Hit ratio. Select decode(name, 'physical reads', value)/ (decode(name, 'consistent gets',value) * decode(name,'db block gets',value)) as hit_ratio From v$sysstat where name IN ('physical reads', 'consistent gets', 'db block gets'); if ratio > 95 you may decrease the buffer size in case required for other purpose if hit_ratio between 70 and 94 then its OK if hit_ratio 60 to 69 then one has to add more buffers ================== suppose it is less than 70, then question arises as to how many buffers to add. In order to deterr only on SYS login and DB_BLOCK_LRU_EXTENDED_STATISTICS set to non-zero value Select 100*TRUNC(indx/100) + 1 \|\| '-' \|\| 100*(TRUNC(indx/100)+1 range, SUM(count) additional_hits FROM x$KCBRBH GROUP BY TRUNC(indx/100); suppose the result range additional_hits ========================= 1-100 78002 101-200 105000 201-300 1005 in this case if you notice increasing the buffer size by 200 is sufficient, further increases will n decrease, then perform above operation with X$KCBCDH view. For this, the init.ora prameter to true

Summary:	How to find
Description:	Ever spend s values you c can easily fi v$nls_valid_ "LANGUAG "CHARACT release of Or "SORT" ord
Summary:	NLS_LANG parameter
Description:	While running an export character set of WE8ISC

	generated a warning mes being used and possible I tried setting the NLS_L English_Canada.WE8IS properly. This is someth exports on any database default character set. Oth special characters like ac from your data! The env need something similar t NLS_LANG=English_C NLS_LANG

Summary:	"Query is executing..."?
Description:	"Query is executing..." isn't it? Who knows? This does, if your query uses rollback segments (updates, inserts, etc.) Using the statement below, you can find out if your UPDATE (or whatever) is proceeding or rolling back. Step 1: Obtain the SID for the session in question. There are a few ways to obtain the (proper) SID, but this may work as well as any: select sid, serial#, username, terminal from v$session; Look through the results to find the SID you need. In this example, the SID is 10. Step 2: `select taddr from v$session where sid = '10' ;` In this example, the result is 023ED71C Step 3: Using the value of taddr returned in step 2, do this:

	`select used_ublk from v$transaction where addr='023ED71C'`
	 `or, you can combine steps 2 and 3 as follows:`
	 `select used_ublk from v$transaction where addr= (select TADDR from v$session where sid='10');`
	
	 Step 4: Wait a few seconds, and run STEP 3 again. if the results show an increasing number, the transaction is proceeding. If the results show a smaller number, then a rollback is occurring.

Summary:	Migrating using ODMA
Description:	I'm migrating more than 242 oracle 7.3.4 databases to Oracle 8.1.6 Using ODMA (Oracle Data Migration Asistant) could find a big error in the Oracle Note

	76460.1 from Metalink (Checklist for Migrating from Oracle7 to Oracle8.1 on UNIX). Step 13, when you have to CONVERT the database, does not work as indicated, so, what you have to do is: 1) startup your db 2) backup your control file to trace 3) shutdown your db 4) startup nomount 5) recreate your control files using script generated in step 2). 6) alter database open 7) continue with Oracle

	Checklist. Believe me, it works fine !!! Hope this could be useful. Email me if any problem comes up.

Summary:	Migrating using ODMA
Description:	I'm migrating more than 242 oracle 7.3.4 databases to Oracle 8.1.6 Using ODMA (Oracle Data Migration Asistant) could find a big error in the Oracle Note 76460.1 from Metalink (Checklist for Migrating from Oracle7 to Oracle8.1 on UNIX). Step 13, when you have to CONVERT the database, does not work as indicated, so, what you have to do is: 1) startup your db 2) backup your control file to trace 3) shutdown your db 4) startup nomount 5) recreate your control files using script generated in step 2). 6) alter database open 7) continue with Oracle Checklist. Believe me, it works fine !!! Hope this could be useful. Email me if any problem comes up.

Summary:	Running SQLPLUS in NT scripts
Description:	Be aware, you can't setup an environment variable SQLPLUS in an NT script. It will fail. Oracle 8.1.6 has changed from plus80.exe to sqlplus.exe. If you have an environment variable set, i.e. SQLPLUS=d:\oracle\ora81\bin\sqlplus.exe, the batch job will fail. Change the environment variable to anything else other than SQLPLUS. i.e. set SPLUS=d:\oracle\ora81\bin\sqlplus.exe

Summary:	How to cleanup unused temporary segments
Description:	To cleanup or remove unused temporary

	segments you can use the following SQL statement on the temporary tablespace:
	
	`alter tablespace default`
	`storage (pctincrease )`
	
	use the pctincrease value you've already specified for this tablespace. This SQL "wakes up" the system monitor process, which removes all unused temp. segments

Summary:	How to load data from MS excel to an Oracle table?

Description:	If your Excel sheet has a simple table format then you can copy it as text file with tab delimiters. Then create a simple ctl-file from SQL*Loader just like that:
	
	`load data`
	`infile ""`
	`append`
	`into table`
	`fields terminated by ' ' (`
	`)`
	
	The key is that in apostrophes you should specify the tab character (ASCII code 9). Use one of the text editors that support quoting (usually with Ctrl+q combination keystroke). After that you can issue:
	
	`sqlldr80 userid= control= log=`
	`data=`
	
	That's all.
	Some clarification.
	Better use clause "...fields terminated by X'09'..." in order to specify the tab character explicitly.
	You can also try saving the file from Excel in text file CSV-format or TAB-delimited and load with the SQL*Loader (Oracle Utilities) or Borland DataPump (from Delphi or C++ distribut).

Summary:	How do I restrict a query by "ROWNUM" range?

Description:	If you try to use rownum to restrict a query by a range that does not start with 1, you will quickly find that it does not work. For example:
	
	`SQL> SELECT * from TABLE1`
	`WHERE rownum BETWEEN 5 AND 10`
	`no rows selected`
	
	The reason for this, is that rownum is a pseudo-column produced AFTER the query returns. Normally, it can only be used to restrict a query to return a rownumber range that starts with 1 (like rownum)
	
	<5). However, there is a way to achieve this using "in-line views".
	For this complete tip, visit:
	http://www.arrowsent.com/oratip/tip41.htm
	
	For more of Ken's Oracle tips, visit his main site at:
	http://www.arrowsent.com/oratip/frames.htm

Tip #15: SQL scripts that compare schemas in two different instances for differences. (Type: SQL)

So you have your application installed in three different instances (Development, Test, Production). Or maybe that is six (conversion, demo, training), or eight? Well, how do you keep all of that in sync? No matter how good your migration procedures, differences between the instances somehow seem to always creep in. (How is it that something gets into production, that has never been in development?)

This tip is a couple of SQL scripts that will use database links and the SQL MINUS operator to compare the objects and table definitions in the same schema in two different instances.

First, an example of using the two scripts will be shown, then the actual scripts themselves. They can also be downloaded below.

The first script lists the objects that are not in both of the selected instances:

```
SQL> select db_link from user_db_links;

DB_LINK
------------------------------------
TESTLINK.WORLD

SQL> @objdiff
Object Owner: SHARED
First instance DB Link (Include @):
Second instance DB Link (Include @):@TESTLINK

OBJDIFF                  OBJECT DIFFERENCE REPORT    Report
Date: 02/10/97

Page:         1
         OWNER:  SHARED
                          Objects in devl but not demo

Object Name                          Object Type     Status
----------------------------------- --------------- -------
---
TSU_SELECT_CAD_FN                    FUNCTION        VALID
TSU_SELECT_FN                        FUNCTION        VALID
FMU_PLS                              INDEX           VALID
FMU_PRECIP                           INDEX           VALID
SYSPIPE                              PACKAGE         VALID
SYSPIPE                              PACKAGE BODY    VALID
ABLE_FK_CONSTRAINTS                  PROCEDURE       INVALID
WTRSHD_SEQ                           SEQUENCE        VALID
ADMIN_CODE                           TABLE           VALID
FMPT_FORM_HELP                       TABLE           VALID
FMU_COUNTY_V                         VIEW            VALID
FMU_TRUST_V                          VIEW            VALID

12 rows selected.


OBJDIFF                  OBJECT DIFFERENCE REPORT    Report
Date: 02/10/97

Page:         1
         OWNER:  SHARED
                          Objects in demo but not devl

Object Name                          Object Type     Status
----------------------------------- --------------- -------
---
PPR_LABEL_NM                         FUNCTION        VALID
FMA_PEST_PEST_CD_I                   INDEX           VALID
GEO_ADMIN_UNIT_SEQ                   SEQUENCE        VALID
TR_FMA_STATUS                        TRIGGER         INVALID
TR_INSERT_FMA_AREA                   TRIGGER         INVALID
FMA_INS_FMA_AREA_TR                  TRIGGER         VALID
RX_SUM_RPT_VIEW                      VIEW            VALID

7 rows selected.
```

Notes: If no database link is entered, then the script uses the CURRENT instance. Also, the title uses the instance name in each instances v$parameter table (Objects in {instance_1_name} but not in {instance_2_name}.

The second script compares the actual table definitions in two instances:

```
SQL> @tabdiff
Table Owner: SHARED
First instance DB Link (Include @):
Second instance DB Link (Include @):@TESTLINK

TABDIFF                  SCHEMA DIFFERENCE REPORT    Report
Date: 02/10/97

Page:          1
         OWNER:  SHARED
                                    Differences between
devl and demo

Instance Table                     Column
DataType  Len   Pr Null?
-------- ------------------------- ------------------------
- -------- ---- ---- -----
demo     FMU                       FMU_RESTR_BEG_DT_BAD
DATE        7 =0   Y
demo     FMU                       FMU_RESTR_END_DT_BAD
DATE        7 =0   Y

2 rows selected.
```

And now listings of the actual scripts:

```
/*************************************************************
***************/
/* objdiff.sql - Lists the objects in a schema that are not
in both of   */
/*               two instances.   Uses database links and
the SQL MINUS  */
/*               operator to make the comparison.
*/
/*
*/
/*    Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*             http://www.arrowsent.com/oratip
*/
/*
*/
/*   Written:  5/11/95
*/
/*
*/
/* You need to have a database link setup for any instance
that you want */
/* to make a comparison for.
*/
/*
*/
```

```
/* Please feel free to use and modify this script as long
it is not sold */
/* or included in any software without the prior permission
of the author*/
/* If you do make some good improvements, please send them
to me, and I  */
/* can incorporate them in a future version and make them
available to   */
/* others (giving you credit of course!).
*/
/*
*/
/*************************************************************
***************/
set pagesize 60
set linesize 80
set verify off
set feedback off
set pause off;
--define obj_owner = '&1'
--define inst_1 = '&2'
--define inst_2 = '&3'
accept obj_owner prompt 'Object Owner: '
accept inst_1 prompt 'First instance DB Link (Include @):'
accept inst_2 prompt 'Second instance DB Link (Include @):'

clear breaks
ttitle off
set heading off

column datetime noprint new_value datetime
column inst_code1 noprint new_value inst_code1
column inst_code2 noprint new_value inst_code2

select to_char(sysdate,'MM/DD/YY') datetime
  from dual
/
select value inst_code1
  from v$parameter&inst_1
where name = 'db_name'
/
select value inst_code2
  from v$parameter&inst_2
where name = 'db_name'
/
set feedback on
set heading on
set newpage 0


ttitle  left 'OBJDIFF'-
  col 25 'OBJECT DIFFERENCE REPORT' -
        col 53 'Report Date: ' datetime -
 skip 1 col 60 'Page: ' sql.pno -
 skip 1 col 10 'OWNER:  ' obj_owner   -
 skip 1 center 'Objects in &inst_code1 but not &inst_code2'
-
```

```
 skip 2

set null=0

column object_type format a15 heading 'Object Type';
column object_name format a35 heading 'Object Name';
column status format a10 heading 'Status';
column inst_code format a10 heading 'Instance';
select object_name, object_type, status
from all_objects&inst_1
where owner = UPPER('&obj_owner')
--  and object_type != 'SYNONYM'
MINUS
select object_name, object_type, status
from all_objects&inst_2
where owner = UPPER('&obj_owner')
--  and object_type != 'SYNONYM'
order by 2,3
/
set heading off;
set feedback off;
select '
```

```
' from dual
/
set heading on;
set feedback on;
ttitle  left 'OBJDIFF'-
  col 25 'OBJECT DIFFERENCE REPORT' -
        col 53 'Report Date: ' datetime -
 skip 1 col 60 'Page: ' sql.pno -
 skip 1 col 10 'OWNER:  ' obj_owner   -
 skip 1 center 'Objects in &inst_code2 but not &inst_code1'
-
 skip 2

select object_name, object_type, status
from all_objects&inst_2
where owner = UPPER('&obj_owner')
  and object_type != 'SYNONYM'
MINUS
select object_name, object_type, status
from all_objects&inst_1
where owner = UPPER('&obj_owner')
  and object_type != 'SYNONYM'
order by 2,3
/
undefine datetime
undefine inst_code1
undefine inst_code2
undefine obj_owner
```

==
=

```
/************************************************************
***************/
/* tabdiff.sql - Lists the differences in table definitions
in the tables*/
/*                for a schema in two different instances.
Uses database*/
/*                links and the SQL MINUS operator to make
the comparison.*/
/*
*/
/*    Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*             http://www.arrowsent.com/oratip
*/
/*
*/
/*   Written:  5/11/95
*/
/*
*/
/* You need to have a database link setup for any instance
that you want */
/* to make a comparison for.
*/
```

```
/*
*/
/* Please feel free to use and modify this script as long
it is not sold */
/* or included in any software without the prior permission
of the author*/
/* If you do make some good improvements, please send them
to me, and I  */
/* can incorporate them in a future version and make them
available to   */
/* others (giving you credit of course!).
*/
/*
*/
/***********************************************************
***************/
set pagesize 60
set linesize 105
set verify off
set feedback off
set pause off;

--define obj_owner = '&1'
--define inst_1 = '&2'
--define inst_2 = '&3'
accept obj_owner prompt 'Table Owner: '
accept inst_1 prompt 'First instance DB Link (Include @):'
accept inst_2 prompt 'Second instance DB Link (Include @):'

clear breaks
ttitle off
set heading off

column datetime noprint new_value datetime
column inst_code1 noprint new_value inst_code1
column inst_code2 noprint new_value inst_code2

select to_char(sysdate,'MM/DD/YY') datetime
  from dual
/
select value inst_code1
  from v$parameter&inst_1
where name = 'db_name'
/
select value inst_code2
  from v$parameter&inst_2
where name = 'db_name'
/
set feedback on
set heading on
set newpage 0
ttitle  left 'TABDIFF'-
  col 25 'SCHEMA DIFFERENCE REPORT' -
        col 53 'Report Date: ' datetime -
 skip 1 col 60 'Page: ' sql.pno -
 skip 1 col 10 'OWNER:  ' obj_owner   -
 skip 1 center 'Differences between &inst_code1 and
&inst_code2' -
 skip 2


column table_name format a25 heading 'Table';
column column_name format a25 heading 'Column';
column data_type format a8 heading 'DataType';
column data_length format 999 heading 'Len';
column data_precision format 999 heading 'Pr';
column nullable format a5 heading 'Null?';
column inst_code format a8 heading 'Instance';
(
select '&inst_code1' inst_code, table_name, column_name,
data_type, data_length, data_precision, nullable
from all_tab_columns&inst_1
where owner = UPPER('&obj_owner')
  and table_name in (select table_name from
all_tables&inst_2
                     where owner = UPPER('&obj_owner'))
MINUS
select '&inst_code1' inst_code, table_name, column_name,
data_type, data_length, data_precision, nullable
from all_tab_columns&inst_2
where owner = UPPER('&obj_owner')
)
UNION
(
select '&inst_code2' inst_code, table_name, column_name,
data_type,
       data_length, data_precision, nullable
from all_tab_columns&inst_2
where owner = UPPER('&obj_owner')
  and table_name in (select table_name from
all_tables&inst_1
                     where owner = UPPER('&obj_owner'))
MINUS
select '&inst_code2' inst_code, table_name, column_name,
data_type,
       data_length, data_precision, nullable
from all_tab_columns&inst_1
where owner = UPPER('&obj_owner')
)
order by 2, 3
/
undefine datetime
undefine inst_code1
undefine inst_code2
undefine obj_owner
```

Tip #12: SQL Script to show 'hit ratio' of currently running processes. (Type: SQL)

Have you ever wondered why your server was running so slow? Who else is running queries and why are they bogging the system down?? So you go round up a DBA and ask them to monitor the database using one of those shnazzy DBA type tools. But DBAs are not always had for the asking, and you do not have access to the tools, so what do you do? This tip is a couple of simple SQL scripts which will show which Oracle processes are currently running in an instance, and what the buffer hit ratio is for those processes (low hit ratios are an indication of poorly tuned SQL, which can slow the WHOLE instance down).

The first script shows the active processes and their current hit ratio.

```
/**********************************************************
***************/
/* listproc.sql - Lists currently running processes and
their hit ratios */
/*
*/
/*    Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*                http://www.arrowsent.com/oratip
*/
/*
*/
/* You need select access to V$SESSION, V$PROCESS, and
V$SESS_IO          */
/*   to run this script.
*/
/*
*/
/*  The columns returned by this script are:
*/
/       Oracle ID (schemaname) = The oracle 'schema' or
'user' that is    */
/*                              running the SQL statement.
*/
/*      System ID (username)    = The system id that the
process is       */
/*                              running under.  Will be
the unix userid */
/*                              if Oracle running on unix.
*/
/*      Program        = The name of the program that is
running the SQL.*/
/*      Physical Reads = The number of physical block
reads.             */
/*      Hit Ratio      = The ratio of buffer to physical
block reads.    */
/*                        be an indication of the
efficiency of the query*/
/*                        running. Anything under 90% is
bad.  Very low   */
/*                        hit ratios (< 10-20%) in a
process can slow    */
/*                        down the whole system.
*/
```

```
/***********************************************************
***************/
column schemaname format a10 heading 'Oracle ID'
column username format a10 heading 'System ID'
column program format a32 heading 'Program'
column hit_ratio format 9.90 heading 'Hit Ratio'
column physical_reads format 9999999 heading 'Reads'
column sid format 99999
SELECT   s.schemaname
       , p.username
       , s.program
       ,io.physical_reads
       ,(io.block_gets+io.consistent_gets)/

(io.block_gets+io.consistent_gets+io.physical_reads)
hit_ratio
  FROM  V$Session s
       ,V$Process p
       ,V$Sess_io io
WHERE  s.paddr  = p.addr
  AND  s.sid    = io.sid
  -- Only look at active processes
  AND  s.status = 'ACTIVE'
  -- Need this predicate to prevent division by 0
  AND  (io.block_gets+io.consistent_gets+io.physical_reads)
> 0
/
```

An example of using the script:

```
SQL> @hitratio

Oracle ID  System ID  Program
Reads Hit Ratio
---------- ---------- -------------------------------- ----
---- ---------
SYS        oracle7
1.00
SYS        oracle7
10894       .83
SYS        oracle7
18       .95
BDES490    oracle7    C:\ORAWIN\BIN\PLUS31.EXE
1.00
BDES490    oracle7    sqlplus@larabee (TNS interface)
3478       .83
```

The next script is a simpler version that just shows all of processes and their status, sid and serial#. The sid and serial# are used in the ALTER SYSTEM KILL SESSION command to kill oracle processes that are 'stuck'.

```
/***********************************************************
***************/
/* listproc.sql - Lists currently processes, status, sid &
serial#       */
/*
*/
```

```
/*    Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*             http://www.arrowsent.com/oratip
*/
/*
*/
/* You need select access to V$SESSION, V$PROCESS to run
this script     */
/*
*/
/***********************************************************
***************/
column schemaname format a10 heading 'Oracle ID'
column username format a10 heading 'System ID'
column program format a30 heading 'Program'
column user_name format a15 heading 'User Name'
column sid format 99999
SELECT  s.schemaname
       ,p.username
       ,s.program
       ,s.sid
       ,s.serial#
       ,s.status
  FROM  V$Session s
       ,V$Process p
where s.paddr = p.addr
/
```

An example of running the script:

```
SQL> @listproc

Oracle ID  System ID  Program                           SID
SERIAL# STATUS
---------- ---------- ------------------------------ ------
--------- --------
KATK490               C:\WINDOWS\SYSTEM32\OLE2.DLL       21
447 KILLED
SYS        oracle7                                        1
1 ACTIVE
SYS        oracle7                                        2
1 ACTIVE
SYS        oracle7                                        3
1 ACTIVE
SYS        oracle7                                        4
1 ACTIVE
SYS        oracle7                                        5
1 ACTIVE
ORAPIPE    orapipe       ?  @gamera (TNS interface)       9
8021 INACTIVE
BDES490    oracle7    C:\ORAWIN\BIN\CKRON10L.DLL         12
105 INACTIVE
JOJJ490    oracle7    C:\ORAWIN\BIN\R25DES.EXE            7
32691 INACTIVE
BDES490    oracle7    C:\ORAWIN\BIN\PLUS31.EXE           16
275 ACTIVE
ARJJ490    oracle7    C:\ORAWIN\BIN\CKRON10L.DLL          6
2029 INACTIVE
```

```
BHAR490    oracle7    C:\ORAWIN\BIN\PLUS31.EXE          10
2545 INACTIVE
BDES490    oracle7    sqlplus@larabee (TNS interface)    17
619 ACTIVE
MAJJ490    oracle7    C:\ORAWIN\BIN\CKRON10L.DLL        13
35 INACTIVE
BHAR490    oracle7    C:\ORAWIN\BIN\R25DES.EXE          14
39 INACTIVE
ARJJ490    oracle7    C:\ORAWIN\BIN\R25DES.EXE           8
9173 INACTIVE
MAJJ490    oracle7    C:\ORAWIN\BIN\R25DES.EXE          11
2273 INACTIVE
SHARED     oracle7    C:\ORAWIN\BIN\PLUS31.EXE          15
67 INACTIVE
BDES490    oracle7    C:\ORAWIN\BIN\PLUS31.EXE          18
739 INACTIVE
```

Tip #11: Procedure to disable FK constraints TO a table. (Type: DBA)

So you have to reload the data in a table that is maintained in another system. But there are these pesky Foreign Keys defined TO this table from other tables in your database. Oh Well, Select the names of the FKs from the constraints table, enter the commands to disable them, now load the data. What? You missed one? Disable it, reload. Now enable all of the constraints again. Kind of tedious. This tip details a stored procedure that can automatically disable or enable all of the FK constraints *TO* a specified table.

The following procedure uses the following steps to enable or disable all of the FK constraints *TO* a specified table:

1. Finds the PK of the specified table.
2. Uses this PK to find all of the FKs that are linked to the PK.
3. Puts together an ALTER TABLE DISABLE CONSTRAINT command to disable each FK.
4. Uses dynamic SQL to execute the commands.

```
PROMPT
PROMPT Creating Procedure able_fk_constraints
CREATE OR REPLACE PROCEDURE able_fk_constraints(
  pTable IN VARCHAR2 ,
  pAble IN VARCHAR2 )
IS
vPKName VARCHAR2(80);

-- This cursor returns the list of FK constraints linked to
the specified
--   PK constraint.
CURSOR curFK(pcPKName IN VARCHAR2) IS
  SELECT constraint_name, table_name
    FROM user_constraints
   WHERE r_constraint_name = pcPKName;
  -- These two variables are used for the dynamic SQL
  nDDLCursor INTEGER;
  nDDLReturn INTEGER;
BEGIN
/**********************************************************
********************/
/* ABLE_FK_CONSTRAINTS - This procedure easily
enables/disables FK constraints */
/*                         pointing TO the specified table.
*/
/*
*/
/*       Parameters:  pTable - The name of the table to
dis/enable FK            */
/*                               constraints to.
*/
/*                    pAble   - One of: DISABLE or ENABLE
*/
/**********************************************************
********************/

  -- Get the name of the PK constraint for the specified
table.
  BEGIN
    SELECT constraint_name INTO vPKName
      FROM user_constraints
     WHERE table_name = pTable
       AND constraint_type = 'P';
  END;


  -- Now get the FK constraints linked to the PK constraint
of the specified table.
  FOR fk IN curFK(vPKName) LOOP

    -- Use dynamic SQL to execute the ALTER TABLE command
and dis/enable the constraint
    nDDLCursor  := dbms_sql.open_cursor;
    dbms_sql.parse(nDDLCursor,'ALTER TABLE
'||fk.table_name||' '
                 ||pAble||' CONSTRAINT
'||fk.constraint_name, 1);
    nDDLReturn  := dbms_sql.execute(nDDLCursor);
    dbms_sql.close_cursor(nDDLCursor);
  END LOOP;
END ABLE_FK_CONSTRAINTS;
/
```

An example of using the script:

```
execute able_fk_constraints('MYTABLE','DISABLE');
truncate table mytable;
@load_mytable
execute able_fk_constraints('MYTABLE','ENABLE');
```

Of course, the procedure has to be installed in a schema that has the ALTER TABLE system privelege, and

security to modify the specified table. Also, the data that is loaded into the table may cause an existing FK contraint to no longer be valid (like if an expected code is no longer there). In this case, the ENABLE will bomb, and the data will have to be fixed before the constraint can be re-enabled. Tip #52: Getting Rid of "Input truncated to # characters" (Type: SQL*Plus)

Are you getting the annoying message "Input truncated to # characters" whenever you run a SQL script in SQL*Plus? This can be very annoying, especially if you are running SQL scripts that produce reports or generate other SQL scripts. This tip will tell you how to get rid of this message!

An Example of the Problem

Consider the following SQL*Plus report:

```
set pagesize 30
set linesize 40
set feedback off
ttitle CENTER 'Test Employee Report' skip 2
break on dname skip 1
spool tstrep.lst
SELECT d.dname, e.empno, e.ename
  FROM Dept d, Emp e
 WHERE d.deptno = e.deptno
 ORDER BY d.dname, e.ename
/
spool off
```

If executed you might see:

```
Test Employee Report

DNAME                    EMPNO  ENAME
-------------- ---------- ----------
ACCOUNTING                7782  CLARK
                          7839  KING
                          7934  MILLER

RESEARCH                  7876  ADAMS
                          7902  FORD
```

```
                                7566 JONES
                                7788 SCOTT
                                7369 SMITH

SALES                           7499 ALLEN
                                7698 BLAKE
                                7900 JAMES
                                7654 MARTIN
                                7844 TURNER
                                7521 WARD

Input truncated to 9 characters
```

Problem Message!

As you can see, you have the unwanted message at the bottom of the report.

What Causes The Problem?

This problem is caused by having anything OTHER than a blank line at the bottom of your SQL*Plus script!   The last line of the script *must* be a blank line, that is a line with a carriage return and NOTHING ELSE. For example:

```
set pagesize 30
set linesize 40
set feedback off
ttitle CENTER 'Test Employee Report' skip 2
break on dname skip 1
spool tstrep.lst
SELECT d.dname, e.empno, e.ename
  FROM Dept d, Emp e
 WHERE d.deptno = e.deptno
 ORDER BY d.dname, e.ename
/
spool off

```

Blank Line!

Tip #44: Ordering by a Hierarchy (Type: SQL)

Have you ever tried to order a hierarchical query? The results are not encouraging. The ordering returned by Oracle is based on the hierarchy, and there is no easy way to order WITHIN the hierarchy levels. So how do we get around this problem? Well, there is no easy way to do it. However, with a little work, the solution presented in this tip will do it.

What Happens if I Order by

I will use the infamous EMP/DEPT tables to illustrate this technique. Using these tables, you might use the following SQL for a standard hierarchical query:

```
SQL>
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4*   START WITH emp.empno = 7839
SQL> /

    LEVEL ENAME                     EMPNO        MGR     DEPTNO
--------- -------------------- ---------- ---------- ----------
        1 KING                       7839                    10
        2   JONES                    7566       7839         20
        3     SCOTT                  7788       7566         20
        4       ADAMS                7876       7788         20
        3     FORD                   7902       7566         20
        4       SMITH                7369       7902         20
        2   BLAKE                    7698       7839         30
        3     ALLEN                  7499       7698         30
        3     WARD                   7521       7698         30
        3     MARTIN                 7654       7698         30
        3     TURNER                 7844       7698         30
        3     JAMES                  7900       7698         30
        2   CLARK                    7782       7839         10
        3     MILLER                 7934       7782         10
```

Now let's say you want to order alphabetically within each level (i.e. BLAKE, CLARK, JONES for level 2, and ALLEN, JAMES, MARTIN, TURNER, WARD for level 3 under BLAKE). Here are some standard attempts at this:

```
SQL> l
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4*   START WITH emp.empno = 7839

    LEVEL ENAME                     EMPNO        MGR     DEPTNO
--------- -------------------- ---------- ---------- ----------
        4       ADAMS                7876       7788         20
        4       SMITH                7369       7902         20
        3     ALLEN                  7499       7698         30
        3     FORD                   7902       7566         20
        3     JAMES                  7900       7698         30
        3     MARTIN                 7654       7698         30
        3     MILLER                 7934       7782         10
        3     SCOTT                  7788       7566         20
        3     TURNER                 7844       7698         30
        3     WARD                   7521       7698         30
        2   BLAKE                    7698       7839         30
        2   CLARK                    7782       7839         10
        2   JONES                    7566       7839         20
        1 KING                       7839                    10
```

```
14 rows selected.

SQL> l
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4    START WITH emp.empno = 7839
  5* order by emp.ename
SQL> /

    LEVEL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        4       ADAMS               7876      7788        20
        3     ALLEN                 7499      7698        30
        2   BLAKE                   7698      7839        30
        2   CLARK                   7782      7839        10
        3     FORD                  7902      7566        20
        3     JAMES                 7900      7698        30
        2   JONES                   7566      7839        20
        1 KING                      7839                  10
        3     MARTIN                7654      7698        30
        3     MILLER                7934      7782        10
        3     SCOTT                 7788      7566        20
        4       SMITH               7369      7902        20
        3     TURNER                7844      7698        30
        3     WARD                  7521      7698        30

14 rows selected.

SQL> l
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4    START WITH emp.empno = 7839
  5* order by level,emp.ename
SQL> /

    LEVEL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 KING                      7839                  10
        2   BLAKE                   7698      7839        30
        2   CLARK                   7782      7839        10
        2   JONES                   7566      7839        20
        3     ALLEN                 7499      7698        30
        3     FORD                  7902      7566        20
        3     JAMES                 7900      7698        30
        3     MARTIN                7654      7698        30
        3     MILLER                7934      7782        10
        3     SCOTT                 7788      7566        20
        3     TURNER                7844      7698        30
        3     WARD                  7521      7698        30
        4       ADAMS               7876      7788        20
        4       SMITH               7369      7902        20
```

None of these give us what we want.

Use a Hierarchy Order Key

The only way I have found to truly resolve this problem is to add a hierarchy ordering key column to the table with the hierarchy. This column needs to be populated programmatically in such a way that you get the desired ordering. This key has to be the concatenation of some sort of order key for EACH of the parent levels above the hierarchy node. This will allow the hierarchy to be ordered within each level while allowing the children to be placed directly underneath their parent. For example, consider the EMP_HIER_ORDER column that I added to the standard emp table below:

```
    EMPNO ENAME                EMP_HIER_ORDER
--------- -------------------- -----------------------------
     7369 SMITH                0008000700050012
     7499 ALLEN                000800030002
     7521 WARD                 000800030014
     7566 JONES                00080007
     7654 MARTIN               000800030009
     7698 BLAKE                00080003
     7782 CLARK                00080004
     7788 SCOTT                000800070011
     7839 KING                 0008
     7844 TURNER               000800030013
     7876 ADAMS                0008000700110001
     7900 JAMES                000800030006
     7902 FORD                 000800070005
     7934 MILLER               000800040010
```

Now if I order by EMP_HIER_ORDER I get:

```
SQL> l
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno,
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4    START WITH emp.empno = 7839
  5  order by emp_hier_order
SQL> /

    LEVEL ENAME                    EMPNO       MGR EMP_HIER_ORDER
--------- -------------------- --------- --------- --------------
---
        1 KING                      7839           0008
        2   BLAKE                   7698      7839 00080003
        3     ALLEN                 7499      7698 000800030002
        3     JAMES                 7900      7698 000800030006
        3     MARTIN                7654      7698 000800030009
        3     TURNER                7844      7698 000800030013
        3     WARD                  7521      7698 000800030014
        2   CLARK                   7782      7839 00080004
        3     MILLER                7934      7782 000800040010
        2   JONES                   7566      7839 00080007
        3     FORD                  7902      7566 000800070005
        4       SMITH               7369      7902
0008000700050012
        3     SCOTT                 7788      7566 000800070011
        4       ADAMS               7876      7788
0008000700110001
```

Which is exactly what I want. The first four characters of EMP_HIER_ORDER are used for ordering the top level of the hierarchy ("0008"), the second four are used for ordering the second level ("0003","0004","0007"), and the third four for the third level, etc.
NOTE: The above query used the hierarchical clauses (CONNECT BY, etc.) Using the hierarchy ordering column you could construct a query that does not need it. For instance:

```
SELECT length(emp_hier_order)/4 lvl, LPAD('
',(length(emp_hier_order)/2)-2)||emp.ename ename,
  FROM Emp
order by emp_hier_order
```

Populating The Hierarchy Ordering Key

The main problem with this technique is that it requires that extra code be written and executed to populate the hierarchy ordering key. I used the following stored procedure to populate the EMP_HIER_ORDER key in the above example:

```
CREATE OR REPLACE PROCEDURE Update_Emp_Hier IS

  -- Cursor to return the ordering key for emp
  CURSOR emp_order_cur IS
         SELECT empno
    FROM Emp
   ORDER BY ename;

  -- Hierarchy query
  CURSOR hier_cur IS
  SELECT LEVEL lvl, empno
    FROM Emp
  START WITH emp.empno = 7839
  CONNECT BY PRIOR emp.empno = emp.mgr;

  TYPE vc_tabtype IS TABLE OF VARCHAR2(4) INDEX BY
BINARY_INTEGER;
  t_ordkey vc_tabtype;
  t_key vc_tabtype;
  v_hier_key VARCHAR2(30);
  v_OrdCnt NUMBER := 0;
BEGIN
  --
  -- Load the ordering key into a PL/SQL table to save table
access
```

```
  FOR e IN emp_order_cur LOOP
    v_OrdCnt := v_OrdCnt + 1;
    t_ordkey(e.empno) := LPAD(TO_CHAR(v_OrdCnt),4,'0');
  END LOOP;

  -- Now open the hierarchy query
  FOR h IN hier_cur LOOP
    -- Store the order key for the current level in the hierarchy
    t_key(h.lvl) := t_ordkey(h.empno);

    -- Build the full ordering key for the current record.  This
will
    --  consist of the current record's ordering key preceded in
order
    --  by the ordering keys of every level above it in the
hierarchy.
    v_hier_key := '';
    FOR i IN 1..h.lvl LOOP
      v_hier_key := v_hier_key||t_key(i);
    END LOOP;

    UPDATE Emp
       SET emp_hier_order = v_hier_key
     WHERE empno = h.empno;

  END LOOP;
END;
/
```

This stored procedure can be called from the client that maintains the hierarchy, executed either manually (i.e. when the user says they are done editing the hierarchy), or automatically. But a better method would be to put a call to this procedure into a trigger for the table. The following trigger definition would work:

```
CREATE OR REPLACE TRIGGER emphierorder
 AFTER INSERT OR DELETE OR UPDATE OF mgr
    ON Emp
BEGIN
  update_emp_hier;
END;
```

This trigger would automatically maintain the hierarchy after any updates to the table that would affect the hierarchy (i.e. updates to the MGR column).

Drawbacks to This Technique

Of course there are a few drawbacks to this technique:

1. You have to create and maintain a "denormalized" column.
2. You have to write and maintain the code that populates the column.
3. Since ANY update to the table causes ALL of the rows to be updated, there may be some performance problems for large frequently updated hierarchies. (There

are ways to reduce this impact, but they are usually design specific, and out of the scope of this tip).

Tip #42: A Single Hierarchy View for Multiple Hierarchies (Type: SQL)

When you use hierarchical queries (queries using CONNECT BY and PRIOR), you always have to specify the top of a particular hierarchy using the "START WITH" syntax. This is often done by hard coding the PK of the top of the hierarchy in the "START WITH" clause. However, if you have many hierarchies in the same table, you might want to be able to have the same program use ANY of the hierarchies, and specify which hierarchy (and thus, which "START WITH" key) at runtime. Wouldn't it be nice if you could put the hierarchy query in a view, and simply specify the hierarchy to use at runtime? Well, you can! This tip will show one technique for doing this.

Starting With A Standard Hierarchy Query

I will use the infamous EMP table to illustrate this technique. However, since the standard emp table only has one hierarchy (starting with "KING"), I added a second hierarchy. I also updated the DEPTNO for all of the standard records to have the same DEPTNO (which I use to differentiate the two hierarchies). Using this tables, you might use the following SQL for a standard hierarchical query:

```
 SQL>
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4*   START WITH emp.empno = 7839
SQL> /

    LEVEL ENAME                     EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 KING                      7839                   10
        2   BLAKE                   7698      7839         10
        3     MARTIN                7654      7698         10
        3     ALLEN                 7499      7698         10
        3     TURNER                7844      7698         10
        3     JAMES                 7900      7698         10
        3     WARD                  7521      7698         10
        2   CLARK                   7782      7839         10
        3     MILLER                7934      7782         10
        2   JONES                   7566      7839         10
        3     FORD                  7902      7566         10
        4       SMITH               7369      7902         10
        5         Ken                999      7369         10
        3     SCOTT                 7788      7566         10
        4       ADAMS               7876      7788         10
```

```
I placed a second hierarchy in the same table, this one starting with
"SONG":
        SQL>
          1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
        emp.empno, emp.mgr, emp.deptno
```

```
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4*   START WITH emp.empno = 6000
SQL> /

     LEVEL ENAME                     EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 SONG                      6000                  40
        2   GOMEZ                   6001      6000        40
        3     WILLIAMS              6002      6001        40
        4       DIRKSEN             6003      6002        40
        5         ATKINS            6004      6003        40
        5         DESZELL           6005      6003        40
        5         DEVITT            6006      6003        40
        2   SMITH                   6007      6000        40
        3     GEORGE                6008      6007        40
        3     JONES                 6009      6007        40
        4       MILLER              6010      6009        40
        4       BAKER               6011      6009        40
```

Trying to Make The Query more Generic

Let's try leaving off the "START WITH" in a view in an attempt to make a generic hierarchy view:

```
SQL> CREATE OR REPLACE VIEW Emp_Hier AS
  2  SELECT level lvl, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  3    FROM Emp
  4  CONNECT BY PRIOR emp.empno = emp.mgr
  5  /
```

Now if we select from this view without any predicates, the query will still return, but it will return the results of a hierarchy starting with EVERY record in the table. For example:

```
SQL> SELECT lvl, ename, empno, mgr, deptno
  2    FROM Emp_Hier
  3  /

    LEVEL ENAME                     EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 KING                      7839                  10
        2   BLAKE                   7698      7839        10
        3     MARTIN                7654      7698        10
        3     ALLEN                 7499      7698        10
        3     TURNER                7844      7698        10
        3     JAMES                 7900      7698        10
        3     WARD                  7521      7698        10
        2   CLARK                   7782      7839        10
        3     MILLER                7934      7782        10
        2   JONES                   7566      7839        10
        3     FORD                  7902      7566        10
        4       SMITH               7369      7902        10
        5         Ken                999      7369        10
        3     SCOTT                 7788      7566        10
        4       ADAMS               7876      7788        10
        1 BLAKE                     7698      7839        10
        2   MARTIN                  7654      7698        10
        2   ALLEN                   7499      7698        10
```

```
      2   TURNER                  7844      7698        10
      2   JAMES                   7900      7698        10
      2   WARD                    7521      7698        10
      1 CLARK                     7782      7839        10
      2   MILLER                  7934      7782        10
      1 JONES                     7566      7839        10
      2   FORD                    7902      7566        10
      3     SMITH                 7369      7902        10
      4       Ken                  999      7369        10
      2   SCOTT                   7788      7566        10
      3     ADAMS                 7876      7788        10
      1 MARTIN                    7654      7698        10
      1 ALLEN                     7499      7698        10
      1 TURNER                    7844      7698        10
      1 JAMES                     7900      7698        10
      1 WARD                      7521      7698        10
      . .                         .         .           .
      . .                         .         .           .
. .                          .          .           .
1 SONG                       6000                   40
2   GOMEZ                    6001       6000        40
3     WILLIAMS               6002       6001        40
4       DIRKSEN              6003       6002        40
5         ATKINS             6004       6003        40
5         DESZELL            6005       6003        40
5         DEVITT             6006       6003        40
2   SMITH                    6007       6000        40
3     GEORGE                 6008       6007        40
3     JONES                  6009       6007        40
4       MILLER               6010       6009        40
4       BAKER                6011       6009        40
1 GOMEZ                      6001       6000        40
2   WILLIAMS                 6002       6001        40
3     DIRKSEN                6003       6002        40
. .                          .          .           .
. .                          .          .           .
. .                          .          .           .
```
(NOTE: I did not display the complete results of this query)

Notice that the results start with the standard hierarchy (beginning with "KING"), followed by another hierarchy starting with "BLAKE" (which is a child of "KING", and should not have it's own hierarchy), followed by "CLARK", "JONES", etc. There are even one level hierarchies for the records at the bottom of the tree (i.e. "MARTIN", "ALLEN", etc.). Now this query is generic, and it will also pick up the 2nd complete hierarchy (starting with "SONG"). You can use the DEPTNO column to select one or the other of the hierarchies:

```
SQL> SELECT lvl, ename, empno, mgr, deptno
  2    FROM Emp_Hier
  3   WHERE deptno = 40
  3  /

    LEVEL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------

1 SONG                       6000                   40
2   GOMEZ                    6001       6000        40
3     WILLIAMS               6002       6001        40
4       DIRKSEN              6003       6002        40
5         ATKINS             6004       6003        40
5         DESZELL            6005       6003        40
5         DEVITT             6006       6003        40
2   SMITH                    6007       6000        40
```

```
  3     GEORGE                  6008      6007        40
  3     JONES                   6009      6007        40
  4       MILLER                6010      6009        40
  4       BAKER                 6011      6009        40
  1 GOMEZ                       6001      6000        40
  2   WILLIAMS                  6002      6001        40
  3     DIRKSEN                 6003      6002        40
4       ATKINS                6004      6003        40
  4       DESZELL               6005      6003        40
  4       DEVITT                6006      6003        40
  1 WILLIAMS                    6002      6001        40
  2   DIRKSEN                   6003      6002        40
  3     ATKINS                  6004      6003        40
  3     DESZELL                 6005      6003        40
  3     DEVITT                  6006      6003        40
  1 DIRKSEN                     6003      6002        40
  2   ATKINS                    6004      6003        40
  2   DESZELL                   6005      6003        40
  2   DEVITT                    6006      6003        40
  1 ATKINS                      6004      6003        40
  1 DESZELL                     6005      6003        40
  1 DEVITT                      6006      6003        40
  1 SMITH                       6007      6000        40
  2   GEORGE                    6008      6007        40
  2   JONES                     6009      6007        40
  3     MILLER                  6010      6009        40
  3     BAKER                   6011      6009        40
  1 GEORGE                      6008      6007        40
  1 JONES                       6009      6007        40
  2   MILLER                    6010      6009        40
  2   BAKER                     6011      6009        40
  1 MILLER                      6010      6009        40
  1 BAKER                       6011      6009        40
```

This limits the query to one of the hierarchies, but it does not eliminate the spurious hierarchies. Therefore, leaving off the START WITH predicate is fairly useless

Using a Database Function to Dynamically Determine the Top of the Hierarchy

In order to add the "START WITH" back into the view, yet make the view dynamic, you can create a database function that returns the top parent of a hierarchy given a key that identifies the hierarchy (the deptno in this example). The following function does this for our example:

```
CREATE OR REPLACE FUNCTION Get_Emp_Top(p_DeptNo IN NUMBER) RETURN
NUMBER IS
  v_TopParent NUMBER;
BEGIN

  SELECT empno INTO v_TopParent
    FROM Emp
   WHERE Deptno = p_Deptno
     AND mgr IS NULL;

  RETURN(v_TopParent);

END;
/
```

Now, we update the view, adding a START WITH clause that uses the function:

```
SQL> CREATE OR REPLACE VIEW Emp_Hier AS
  2  SELECT level lvl, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  3    FROM Emp
  4  CONNECT BY PRIOR emp.empno = emp.mgr
  5  START WITH emp.empno = Get_Emp_Top(emp.deptno)
  6  /

SQL> SELECT lvl, ename, empno, mgr, deptno
  2    FROM Emp_Hier
  3  /
```

```
      LVL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 KING                      7839                  10
        2   BLAKE                   7698      7839        10
        3     MARTIN                7654      7698        10
        3     ALLEN                 7499      7698        10
        3     TURNER                7844      7698        10
        3     JAMES                 7900      7698        10
        3     WARD                  7521      7698        10
        2   CLARK                   7782      7839        10
        3     MILLER                7934      7782        10
        2   JONES                   7566      7839        10
        3     FORD                  7902      7566        10
        4       SMITH               7369      7902        10
        5         Ken                999      7369        10
        3     SCOTT                 7788      7566        10
        4       ADAMS               7876      7788        10
        1 SONG                      6000                  40
        2   GOMEZ                   6001      6000        40
        3     WILLIAMS              6002      6001        40
        4       DIRKSEN             6003      6002        40
        5         ATKINS            6004      6003        40
        5         DESZELL           6005      6003        40
        5         DEVITT            6006      6003        40
        2   SMITH                   6007      6000        40
        3     GEORGE                6008      6007        40
        3     JONES                 6009      6007        40
        4       MILLER              6010      6009        40
        4       BAKER               6011      6009        40
```

As you can see, we now have a view that will return ONLY the complete hierarchy of both hierarchies we have defined. We can simply add a predicate to the SELECT from the view to only display one of the hierarchies, thereby giving us the dynamic selection of the hierarchy from the view:

```
SQL> SELECT lvl, ename, empno, mgr, deptno
  2    FROM Emp_Hier
  3   WHERE DEPTNO = 40
  4  /
```

```
      LVL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 SONG                      6000                  40
        2   GOMEZ                   6001      6000        40
        3     WILLIAMS              6002      6001        40
        4       DIRKSEN             6003      6002        40
        5         ATKINS            6004      6003        40
        5         DESZELL           6005      6003        40
        5         DEVITT            6006      6003        40
        2   SMITH                   6007      6000        40
        3     GEORGE                6008      6007        40
        3     JONES                 6009      6007        40
        4       MILLER              6010      6009        40
        4       BAKER               6011      6009        40
```

Putting a couple of Tips Together

In a previous tip (Tip #40), I detailed a method to allow joins to hierarchical queries. We can combine that technique with the one from this tip to give us a very powerful view:

```
CREATE OR REPLACE VIEW emp_hier AS
SELECT emphier.emplevel, emphier.ename ind_ename, emphier.ename
,emphier.empno
      ,dept.deptno, dept.dname, dept.loc
      ,emp.ename mgr_ename
FROM Dept, Emp
     ,(select level emplevel, LPAD(' ',2*level-2)||ename ename,
empno, mgr, deptno
        from Emp
      connect by prior empno = mgr
      start with empno = Get_Emp_Top(emp.deptno)
     ) emphier
WHERE emphier.deptno = dept.deptno
  AND emphier.mgr     = emp.empno (+)
```

Here is an example of using the view:

```
SQL> select ind_ename, mgr_ename, dname, loc
  2  from emp_hier
  3  where deptno = 40
  4  /

IND_ENAME            MGR_ENAME  DNAME          LOC
-------------------- ---------- -------------- -------------
SONG                            OPERATIONS     BOSTON
  GOMEZ              SONG       OPERATIONS     BOSTON
    WILLIAMS         GOMEZ      OPERATIONS     BOSTON
      DIRKSEN        WILLIAMS   OPERATIONS     BOSTON
        ATKINS       DIRKSEN    OPERATIONS     BOSTON
        DESZELL      DIRKSEN    OPERATIONS     BOSTON
        DEVITT       DIRKSEN    OPERATIONS     BOSTON
  SMITH              SONG       OPERATIONS     BOSTON
    GEORGE           SMITH      OPERATIONS     BOSTON
    JONES            SMITH      OPERATIONS     BOSTON
      MILLER         JONES      OPERATIONS     BOSTON
      BAKER          JONES      OPERATIONS     BOSTON
```

Click [here] for a SQL script that creates and populates the EMP table used in this example, then runs the example queries. Tip #40: Using "Inline Views" to Join to Hierarchical Queries (Type: SQL)

Have you ever tried to join to a hierarchical query (a query using CONNECT BY and PRIOR) only to get this message:

```
ORA-01437: cannot have join with CONNECT BY
```

One of the limitations of hierarchical queries is that you cannot join to them. However, there are often times you would like to join to them anyway. For instance, if the hierarchy table only has surrogate keys, and you would like to display the real value. This tip shows how you can use "Inline Views" (which are SELECTs in the FROM clause) to join tables to a hierarchical query.

Starting With A Standard Hierarchy Query

I will use the infamous EMP/DEPT tables to illustrate this technique. Using these tables, you might use the following SQL for a standard hierarchical query:

```
SQL>
  1  SELECT level, LPAD(' ',2*level-2)||emp.ename ename,
emp.empno, emp.mgr, emp.deptno
  2    FROM Emp
  3  CONNECT BY PRIOR emp.empno = emp.mgr
  4*   START WITH emp.empno = 7839
SQL> /

    LEVEL ENAME                    EMPNO       MGR    DEPTNO
--------- -------------------- --------- --------- ---------
        1 KING                      7839                  10
        2   BLAKE                   7698      7839        30
        3     MARTIN                7654      7698        30
        3     ALLEN                 7499      7698        30
        3     TURNER                7844      7698        30
        3     JAMES                 7900      7698        30
        3     WARD                  7521      7698        30
        2   CLARK                   7782      7839        30
        3     MILLER                7934      7782        10
        2   JONES                   7566      7839        20
        3     FORD                  7902      7566        20
        4       SMITH               7369      7902        20
        5         Ken                999      7369        20
        3     SCOTT                 7788      7566        20
        4       ADAMS               7876      7788        20
```

Try to Join This Query To the DEPT Table

If you try to join this query to the DEPT table, it won't work:

```
SQL> l
  1  select level, LPAD(' ',2*level-2)||ename ename, empno, mgr,
dept.deptno, dept.dname
  2  from emp, dept
  3  where emp.deptno = dept.deptno
  4  connect by prior empno = mgr
  5* start with empno = 7839
SQL> /
from emp, dept
     *
ERROR at line 2:
ORA-01437: cannot have join with CONNECT BY
```

Place the Hierarchical Query in an "Inline View"

Since Oracle 7.3, we could actually use a complete SELECT statement as one of the "tables" in a query. Using this technique, we can turn the hierarchical query into a "table" and join it do the DEPT table:

```
SQL> l
  1  SELECT emphier.emplevel, emphier.ename, emphier.empno,
dept.deptno, dept.dname
  2  FROM Dept
  3       ,(select level emplevel, LPAD(' ',2*level-2)||ename
ename, empno, mgr, deptno
```

```
  4           from Emp
  5         connect by prior empno = mgr
  6         start with empno = 7839
  7       ) emphier
  8* WHERE emphier.deptno = dept.deptno
SQL> /
```

```
 EMPLEVEL ENAME                    EMPNO    DEPTNO DNAME
--------- -------------------- --------- --------- --------------
        1 KING                      7839        10 ACCOUNTING
        2   BLAKE                   7698        30 SALES
        3     MARTIN                7654        30 SALES
        3     ALLEN                 7499        30 SALES
        3     TURNER                7844        30 SALES
        3     JAMES                 7900        30 SALES
        3     WARD                  7521        30 SALES
        2   CLARK                   7782        30 SALES
        3     MILLER                7934        10 ACCOUNTING
        2   JONES                   7566        20 RESEARCH
        3     FORD                  7902        20 RESEARCH
        4       SMITH               7369        20 RESEARCH
        5         Ken                999        20 RESEARCH
        3     SCOTT                 7788        20 RESEARCH
        4       ADAMS               7876        20 RESEARCH
```

The SELECT statement inside the parentheses is treated just as if it were a view that you are joining to. It is given an alias, "emphier", which is used to refer to it in the SELECT clause (i.e. "emphier.ename"), and in the WHERE clause (i.e. "emphier.deptno"). Since it is treated like a view, we can join it to the Dept table with the following predicate:

```
WHERE emphier.deptno = dept.deptno
```

This will allow you to display the department name ("DNAME") in your hierarchical query.

Putting the Query into a View

Quite often, these hierarchical queries can be useful in many programs and reports. It is often helpful to create a view that lists the hierarchy and joins to useful tables. Here is an example of a view using the EMP/DEPT tables. This view allows you to list the department name and location and the manager name in the query:

```
CREATE OR REPLACE VIEW emp_hier AS
SELECT emphier.emplevel, emphier.ename ind_ename, emphier.ename
,emphier.empno
      ,dept.deptno, dept.dname, dept.loc
      ,emp.ename mgr_ename
FROM Dept, Emp
     ,(select level emplevel, LPAD(' ',2*level-2)||ename ename,
empno, mgr, deptno
        from Emp
      connect by prior empno = mgr
      start with empno = 7839
     ) emphier
WHERE emphier.deptno = dept.deptno
  AND emphier.mgr    = emp.empno (+)
```

Here is an example of using the view:

```
SQL> select ind_ename, mgr_ename, dname, loc
  2  from emp_hier
  3
SQL> /

IND_ENAME            MGR_ENAME  DNAME          LOC
-------------------- ---------- -------------- -------------
KING                            ACCOUNTING     NEW YORK
  BLAKE              KING       SALES          CHICAGO
    MARTIN           BLAKE      SALES          CHICAGO
    ALLEN            BLAKE      SALES          CHICAGO
    TURNER           BLAKE      SALES          CHICAGO
    JAMES            BLAKE      SALES          CHICAGO
    WARD             BLAKE      SALES          CHICAGO
  CLARK              KING       SALES          CHICAGO
    MILLER           CLARK      ACCOUNTING     NEW YORK
  JONES              KING       RESEARCH       DALLAS
    FORD             JONES      RESEARCH       DALLAS
      SMITH          FORD       RESEARCH       DALLAS
        Ken          SMITH      RESEARCH       DALLAS
    SCOTT            JONES      RESEARCH       DALLAS
      ADAMS          SCOTT      RESEARCH       DALLAS
```

Tip #38: Listing Records with the Highest Values using SQL Only. (Type: SQL)

There are times where you want to simply return the rows with a certain number of the highest (or lowest) values for a certain column. This type of functionality is easy to implement in PL/SQL (just order by the column and grab the first *n* rows from the query), but more difficult to do using SQL only. This tip shows you a method to do this in SQL.

Data Used for The Examples in this Tip

The following data (from the infamous EMP table) will be used for all of the examples in this Tip:

```
SQL> desc emp
 Name                            Null?    Type
 ------------------------------- -------- ----
 EMPNO                           NOT NULL NUMBER(4)
 ENAME                                    CHAR(10)
 JOB                                      CHAR(9)
 MGR                                      NUMBER(4)
 HIREDATE                                 DATE
 SAL                                      NUMBER(7,2)
 COMM                                     NUMBER(7,2)
 DEPTNO                          NOT NULL NUMBER(2)

SQL> SELECT empno, sal FROM EMP;

    EMPNO       SAL
--------- ---------
       41      4200
       46      6800
       99      9000
       23      2000
       11      4000
       10      3500
       51      4500
       52      4500
       53       8000
       54       2900

10 rows selected.
```

ROWNUM does not work!

Many SQL begginers are tempted to try to use ROWNUM along with an ORDER BY to limit the rows returned to the highest values. However, this does not work, becuase Oracle sets the ROWNUM value *before* the query results are ordered! Consider the following query:

```
SQL> SELECT empno, sal, rownum
  2    FROM Emp
  3  ORDER BY sal DESC
  4
SQL> /

    EMPNO        SAL     ROWNUM
--------- ---------- ---------
       99       9000          3
       53       8000          9
       46       6800          2
       51       4500          7
       52       4500          8
       41       4200          1
       11       4000          5
       10       3500          6
       54       2900         10
       23       2000          4
```

Notice that the records are ordered by the SAL column, but not the ROWNUM column. If you added a where clause to limit the query to the first three ROWNUMs, you would get:

```
SQL> l
  1  SELECT empno, sal, rownum
  2    FROM Emp
  3  WHERE ROWNUM < 4
  4* ORDER BY sal DESC
SQL> /

    EMPNO        SAL     ROWNUM
--------- ---------- ---------
       99       9000          3
       46       6800          2
       41       4200          1
```

Which does NOT return the three highest SALs!

Solution: Correlated SubQuery to Same Table

One solution for this problem is to use a correlated subquery to the same table. The following select will return the correct rows:

```
SQL> l
  1  SELECT empno, sal
  2    FROM Emp e1
  3   WHERE 3 > (SELECT COUNT(*) FROM Emp e2
  4                 WHERE e1.sal < e2.sal)
  5* ORDER BY SAL desc
SQL> /

    EMPNO       SAL
--------- ---------
       99      9000
       53      8000
       46      6800
```

For every row processed by the main query, the correlated subquery returns a count (*COUNT(*)*) of the number of rows with higher salaries (*WHERE e1.sal < e2.sal)*. Then the main query only returns rows that have fewer than three salaries that are higher (*WHERE 3 > ...)*. For example, for EMPNO=46, the salary is "6800". There is only 1 row with a higher salary (EMPNO=99), so the subquery returns "1", which is less than 3, causing the "WHERE 3 > ..." to evaluate to TRUE, thereby returning the row.

A Problem With This Technique

However, there is a problem with this method. What if there are more than one row with the same salary? Consider the following query, where we change it to return the first 4 rows:

```
SQL> l
  1  SELECT empno, sal
  2    FROM Emp e1
  3   WHERE 4 > (SELECT COUNT(*) FROM Emp e2
  4                 WHERE e1.sal < e2.sal)
  5* ORDER BY SAL desc
SQL> /

    EMPNO       SAL
--------- ---------
       99      9000
       53      8000
       46      6800
       51      4500
       52      4500
```

Instead of returning 4 rows, it returned 5! This is because this technique returns ALL of the rows with the highest 4 salaries, not the first 4 rows. This is a problem with this technique, so you need to make sure that it is acceptible in your design before you use it.

An Alternative Technique which Lists Rank

If you want to use a join instead of a correlated subquery, you could use the following select:

```
SQL> l
  1  SELECT e1.deptno, e1.empno, e1.sal, COUNT(distinct e2.empno)
  2    FROM Emp e1, Emp e2
  3   WHERE e1.sal <= e2.sal
  4  GROUP BY e1.deptno, e1.empno, e1.sal
  5* ORDER BY COUNT(distinct e2.empno)
SQL> /

   DEPTNO      EMPNO        SAL COUNT(DISTINCTE2.EMPNO)
--------- ---------- ---------- -----------------------
       30         99       9000                       1
       50         53       8000                       2
       20         46       6800                       3
       40         51       4500                       5
       50         52       4500                       5
       20         41       4200                       6
       40         11       4000                       7
       40         10       3500                       8
       50         54       2900                       9
       30         23       2000                      10
```

This select turns the correlated subquery into a self-join with a GROUP BY. This allows us to change the count into a sort of RANK. However, the problem with equal salaries remains (notice the two records with a "rank" of 5).

This rank can then be used to select the first three rows:

```
SQL> l
  1  SELECT e1.deptno, e1.empno, e1.sal, COUNT(distinct e2.empno)
  2    FROM Emp e1, Emp e2
  3   WHERE e1.sal <= e2.sal
  4  GROUP BY e1.deptno, e1.empno, e1.sal
  5  HAVING COUNT(distinct e2.empno) < 4
  6* ORDER BY COUNT(distinct e2.empno)
SQL> /

   DEPTNO      EMPNO        SAL COUNT(DISTINCTE2.EMPNO)
--------- ---------- ---------- -----------------------
       30         99       9000                       1
       50         53       8000                       2
       20         46       6800                       3
```

One advantage of this method is that it can be easily used to return the record for just one ranking. For example:

```
SQL> l
  1  SELECT e1.deptno, e1.empno, e1.sal, COUNT(distinct e2.empno)
  2    FROM Emp e1, Emp e2
  3   WHERE e1.sal <= e2.sal
  4  GROUP BY e1.deptno, e1.empno, e1.sal
  5  HAVING COUNT(distinct e2.empno) = 7
  6* ORDER BY COUNT(distinct e2.empno)
SQL> /

   DEPTNO      EMPNO        SAL COUNT(DISTINCTE2.EMPNO)
--------- ---------- --------- -----------------------
       40         11       4000                       7
```

Tip #32: Script to List Trigger Errors & Line Numbers (Type: PL/SQL)

When a trigger is created for a table, sometimes there are compilation errors (hey, were not all perfect programmers). When you have errors, you can list the trigger's syntax errors by using the SHOW ERRORS TRIGGER <Trigger_Name> command. However, many times these messages are cryptic, and it can be difficult to match the error to the specific line of code in the trigger. This tip shows a method that can be used to list the errors, along with the trigger source, and indicating which line of source has each error.

My thanks to Tim Onions, a Principal Technical Consultant at AT&T in the UK for asking if it was possible to do this, and spurring me on to attempting it. He also reviewed the script and made improvement suggestions. Also, thanks to Jurij Modic of the Republic of Slovenia   Ministry of Finance for pointing out a major flaw in the original tip.

The Stanard SHOW_ERRORS command

Here is an example of using the SHOW ERRORS command for triggers:

```
  KATKINS> CREATE OR REPLACE TRIGGER Test_Trigger
  2  before insert or update of price on items
  3  for each row
  4  BEGIN
  5    -- If extended amout is greater than $10,000 set the
status
  6    --   to pending approval, else approve the item.
  7    IF (new.price*new.amount) > 10000 THEN
  8      :new.status = 'P';
  9    ELSE
 10      :new.status := 'A';
 11    END IF;
 12  END;
 /

Warning: Trigger created with compilation errors.

KATKINS> show errors trigger Test_Trigger
LINE/COL
ERROR
```

```
         --------- -------------------------------------------------
         -----------
         5/17      PLS-00103: Encountered the symbol "=" when
         expecting one of the following:
                   := . ( @ % ; indicator
                   The symbol ":= was inserted before "=" to
         continue.
```

The errors go into USER_ERRORS, just like stored procedure errors:

```
         DES2OWNER> l
           1  SELECT line, text FROM User_Errors
           2  WHERE name = 'PVNT_DUP_ENT_ER_TR'
           3* order by sequence
         DES2OWNER> /
         >

               LINE
         TEXT

         ---------- -------------------------------------------------
         ------------
                  5 PLS-00103: Encountered the symbol "=" when
         expecting one of the following:
                    := . ( @ % ; indicator
                    The symbol ":= was inserted before "=" to
         continue.

         DES2OWNER>
```

Merging the errors with the source.

As you can see, it is fairly easy to select from this table to see the errors. But I wanted to do something more. I want to list the code of the trigger along with the arrows, and have the listing point to the line in the code with the error, just like I did for packages, procedures, & Functions in Tip #3.

The PL/SQL source for packages, procedures, & functions is stored in USER_SOURCE. However, the source of the trigger is NOT stored in USER_SOURCE. Instead, it is stored in the TRIGGER_BODY column of USER_TRIGGER. Since TRIGGER_BODY is a LONG column, it is difficult to work with directly (i.e. you cannot use SUBSTR and INSTR on it). Because of this, I wrote a stored procedure that parses it into lines, gives the lines line numbers, then matches those lines to the value in the LINE column of USER_ERRORS. Then, using DBMS_OUTPUT to display the results, I can get the output of this program to look very similar to the stored programs error output I got in Tip #3.

Here is the procedure:

```
/***********************************************************************
**/
/* LIST_TRIG_ERR - A procedure that uses DBMS_OUTPUT to list the
*/
```

```
/*                  compilation errors of a trigger, along with the
*/
/*                  trigger's source.  Also indicates the source line
*/
/*                  with the error.
*/
/*
*/
/*       Inputs: p_Trigger = The trigger name.
*/
/*
*/
/*  Author:  Ken Atkins
(Ken@arrowsent.com)                                *
/*            Principal Consultant - ARIS Corporation
*/
/*
*/
/*  Please feel free to use and modify this script as long as it is not
*/
/*  sold or included in any software without the prior permission of
*/
/*  the author.  If you do make good improvements, please send them to
*/
/*  me and I will incorporate them in a future version of the script
*/
/*  (giving you credit of course!).
*/
/*
*/
/* Modifications:
*/
/*----------------------------------------------------------------------
-*/
/* Ver Date      By             Change
*/
/* --- --------- -------------- ---------------------------------------
-*/
/* 1.0 05-MAR-98 Ken Atkins     Written.
*/
/***********************************************************************
**/
CREATE OR REPLACE PROCEDURE LIST_TRIG_ERR(p_Trigger IN VARCHAR2) AS
  v_Trig LONG;
  b_Continue BOOLEAN := True;
  v_NumLines NUMBER := 0;
  v_Line VARCHAR2(240);
  v_NxtChr NUMBER := 0;
  v_LstChr NUMBER := 0;
  TYPE LineTabTyp IS TABLE OF VARCHAR2(240)
       INDEX by BINARY_INTEGER;
  t_Lines LineTabTyp;
  CURSOR err_cur IS SELECT line, text
                      FROM User_Errors
                     WHERE name = p_Trigger
                       AND text not like '%Statement ignored%';
  i NUMBER;
  v_Prefix CHAR(10);
  v_DDLCursor NUMBER;
  v_DDLReturn NUMBER;
BEGIN
  --
  -- Fetch the trigger code into a variable that will be used to parse
it.
  --
  BEGIN
    SELECT trigger_body INTO v_Trig
      FROM User_Triggers
     WHERE Trigger_Name = p_Trigger;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      Raise_Application_Error(-20001,'Trigger does not exist:
'||p_Trigger);
  END;

  --
  -- Use DBMS_SQL to execute the command that places the errors into
USER_ERRORS
  --
  v_DDLCursor := dbms_sql.open_cursor;
  DBMS_SQL.Parse(v_DDLCursor,'ALTER TRIGGER '||p_Trigger||' COMPILE
DEBUG',1);
  v_DDLReturn := dbms_sql.execute(v_DDLCursor);
  DBMS_SQL.Close_Cursor(v_DDLCursor);

  --
  -- Now loop through the lines in the trigger code and parse it into
separate
  --   record in a PL/SQL table.
  --
  WHILE b_Continue LOOP
    v_NumLines := v_NumLines + 1;
    v_NxtChr := INSTR(v_Trig, CHR(10),1,v_NumLines);
    v_Line := SUBSTR(v_Trig, v_LstChr+1, (v_NxtChr-v_LstChr));
    t_Lines(v_NumLines) := v_Line;
    IF v_NxtChr = 0 THEN
      b_Continue := False;
    ELSE
      v_LstChr := v_NxtChr;
    END IF;
  END LOOP;

  --
  -- Loop through all of the errors in USER_ERRORS for this trigger,
displaying
  --    each error, followed by the triggers code, with an ===>
pointing to the
  --    error line.
  --
  DBMS_OUTPUT.Put_Line('.');
  FOR e IN err_cur LOOP
    DBMS_OUTPUT.Put_Line(e.text);
    DBMS_OUTPUT.Put_Line('.');
    FOR i IN 1..v_NumLines LOOP
```

```
      IF e.line = i THEN

        v_Prefix := '========> ';
      ELSE
        -- Note:  The dots (...) are used because DBMS_OUTPUT normally
strips leading spaces
        v_Prefix := '......... ';
      END IF;
      DBMS_OUTPUT.Put_Line(v_Prefix||to_char(i)||': '||t_Lines(i));
    END LOOP;
    DBMS_OUTPUT.Put_Line('.');
  END LOOP;

END;
/
```

Here is an example of using the procedure. First, I created a simple sql script to call the procedure:

```
      set serveroutput on size 100000
      execute list_trig_err(UPPER('&1'));
```

Then, after creating the trigger, I can call this script to detail the errors.

An Example of Using the Scripts

Here is an example of listing the errors for a trigger::

```
      KATKINS> CREATE OR REPLACE TRIGGER Test_Trigger
        2  before insert or update of price on items
        3  for each row
        4  BEGIN
        5    -- If extended amout is greater than $10,000 set the
      status
        6    --   to pending approval, else approve the item.
        7    IF (new.price*new.amount) > 10000 THEN
        8      :new.status = 'P';
        9    ELSE
       10      :new.status := 'A';
       11    END IF;
       12  END;
       /

      Warning: Trigger created with compilation errors.

      KATKINS> @trigerr TEST_TRIGGER
      PLS-00103: Encountered the symbol "=" when expecting one of
      the following:
         := . ( @ % ; indicator
      The symbol ":= was inserted before "=" to continue.

      .

      ......... 1: BEGIN


      ......... 2:   -- If extended amout is greater than $10,000
      set the status
```

```
......... 3:   --   to pending approval, else approve the
item.

......... 4:   IF (new.price*new.amount) > 10000 THEN

========> 5:     :new.status = 'P';


......... 6:   ELSE


......... 7:     :new.status := 'A';


......... 8:   END IF;


......... 9: END;


.........
10:

.
```

Tip #30: An easy way to EXPLAIN and get some statistics on your SQL. (Type: SQL*Plus)

Haven't you ever thought there should be an easier way to do the EXPLAIN PLAN and TKPROF statistics than to edit your queries to add the commands (like EXPLAIN PLAN SET...), or to have to find or write a script that automates this? It should be an automatic part of SQL*Plus. Well, as of SQL*Plus 3.3 it is!! The command is called 'SET AUTOTRACE ON'!

My thanks go out to Jack Applewhite for pointing out this command in a post to the ODTUG email lists.

The SET AUTOTRACE Command

In SQL*Plus 3.3 there is a little known command (at least I didn't know about it until recently) called SET AUTOTRACE. It is documented in the newest SQL*Plus document set, but who really reads the whole document set for changes? Well I did not. It is very simple to use. Just type the command:

```
SET AUTOTRACE ON
```

And then run your select statement. Example:

```
SQL> SET AUTOTRACE ON

SQL> SELECT d.deptno, d.dname, e.empno, e.ename

  2    FROM dept d, emp e

  3   WHERE d.deptno = e.deptno

  4  /


    DEPTNO DNAME               EMPNO ENAME

---------- -------------- ---------- ----------

        10 ACCOUNTING           7839 KING

.

.

        30 SALES                7900 JAMES

        30 SALES                7521 WARD


14 rows selected.


Execution Plan

----------------------------------------------------------

   0      SELECT STATEMENT Optimizer=CHOOSE

   1    0   MERGE JOIN

   2    1     SORT (JOIN)

   3    2       TABLE ACCESS (FULL) OF 'EMP'

   4    1     SORT (JOIN)

   5    4       TABLE ACCESS (FULL) OF 'DEPT'


Statistics
```

```
----------------------------------------------------------

          0  recursive calls
          4  db block gets
          2  consistent gets
          0  physical reads
          0  redo size
        670  bytes sent via SQL*Net to client
        376  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
         14  rows processed
```

There are also some other options, for example there is a TRACEONLY option which supresses the SQL output. See the SQL*Plus 3.3 manual for a full description.

Some setup issues:

If you go off and try this on your instance, you may run into some problems. There are a few setup steps that need to be taken to make this work:

1. Make sure you have access to PLAN_TABLE. If you don't, create it using *utlxplan.sql* (It should be in a directory like $ORACLE_HOME/rdbms73/admin/) and make sure you have access to it from the user you are using to tune the SQL.
2. You also need to create the PLUSTRACE role, and grant it to the desired users. The script to create this role is in:

   $ORACLE_HOME/plus33/Plustrce.sql

   It has to be run from SYS in order to have the correct security access. Then grant the role to the desired users or ROLEs.

Tip #28: Setting the SQL*Plus prompt to the current directory. (Type: SQL*Plus)

Have you ever had a SQL*Plus window open, and did not remember what it's current directory was? This tip will show a method to set your SQL*Plus prompt to include the current directory.

The SET SQLPROMPT command can be used to set the SQL*Plus prompt to any text string you want. The trick is to get access to the current directory from SQL*Plus, so you can use it in SET SQLPROMPT.

We have access to the current directory from the OS (via 'cd' in DOS/NT and 'pwd' in Unix). We can also call an OS script by using the SQL*Plus HOST command. Using these two capabilities, I wrote two scripts that together performed the function I wanted:

1. A .bat script that writes a SQL script to do the actual SET SQLPROMPT, using the 'cd' command to insert the current directory.
2. A sql script that executes the .bat script and runs the SQL script that it has written.

Here are the two scripts:

```
setprmpt.bat
-----------------
echo set define $ > tmp.sql
echo column curdir noprint new_value curdir >> tmp.sql
echo SELECT REPLACE(' >> tmp.sql
cd >> tmp.sql
echo ',CHR(10),'') curdir FROM DUAL; >> tmp.sql
echo set sqlprompt "($curdir) SQL> "  >> tmp.sql

setprmpt.sql
-----------------
host setprmpt.bat
@tmp
set define &
```

Here is an example of using the scripts to set the prompt:

```
SQL> @setprmpt



(C:\d2k\working) SQL>
```

How these scripts work

The 'host' command in the .sql scriptexecutes the .bat script. The .bat script then writes the following commands to a temporary sql script (called tmp.sql):

```
set define $

column curdir noprint new_value curdir

SELECT REPLACE('

C:\d2k\working

',CHR(10),'') curdir FROM DUAL;

set sqlprompt "($curdir) SQL> "
```

The .sql script then executes this temporary script file. The TMP.SQL script had to be so complicated because I was only using the DOS output redirection capabilities ('>' and '>>') and I could only get the current directory into the file on it's own line. The REPLACE(..,CHR(10),'') command removes the carriage return before and after the directory line.

The 'column .. new_value' command is a SQL*PLUS command that allows the value of a selected column to be placed into a SQL*Plus variable (in this case 'curdir'). Therefore, when the following SELECT.. is run, the text string of the current directory is placed in the 'curdir' SQL*Plus variable. Then the SET SQLPROMPT uses this variable to set the prompt.

Another feature of this technique, is that you now have the SQL*Plus variable with the current directory available for other uses in this SQL*Plus session. For example, it can be used in a select like:

```
(C:\fmpt\sql) SQL> select '&curdir' from dual;

old   1: select '&curdir' from dual

new   1: select 'C:\fmpt\sql' from dual



'C:\FMPT\SQ

-----------

C:\fmpt\sql
```

The above .bat file is NOT pretty. Using Perl or some other scripting language, I could write a simple script that would just write the SET SQLPROMPT command directly without the use of the 'column' command or the 'SELECT'.

Tip #24: Ordering numerically in a VARCHAR2 column. (Type: SQL)

Have you ever tried to order by a VARCHAR2 column that has numeric information in it? Your query is sorted ALPHABETICALLY instead of numerically. That is, your order is 1,10,2,20,200,3,4... instead of 1,2,3,4,10,20,200. If you try to use TO_NUMBER in the order_by your query blows up if there are any alpha characters in the column. This tip details a method that can be used to have the order_by return the columns numerically even if there are some alpha characters in the column.

Consider the following table:

```
SQL> desc NUMBER_SORT


Name                 Null?    Type
-------------------- -------- ------------
sortby               NOT NULL VARCHAR2(20)

SQL> SELECT * from NUMBER_SORT

SORTBY

--------------------


100

A

1

10

1AB

2

20

BBBB

1000

11

30

3

200

21



14 rows selected.
```

If you do a simple order_by your result will be:

```
SQL> SELECT sortby
  2    FROM Number_Sort
  3   ORDER BY sortby;

SORTBY

--------------------


1

10

100

1000

11

1AB

2

20

200

21

3

30

A

BBBB


14 rows selected.
```

Which is not what you want! However, the following select WILL return the column ordered numerically:

```
SQL> SELECT sortby
  2    FROM Number_Sort
  3   ORDER BY
DECODE(TO_CHAR(NVL(LENGTH(TRANSLATE(sortby,'A1234567890','A')),0)),'0',LPAD(sor
tby,8),sortby)
  4  /

SORTBY

--------------------


1

2
```

```
3

10

11

20

21


30


100

200

1000

1AB

A

BBBB


14 rows selected.
```

Now let's take that construct apart to see how it works:

1. **TRANSLATE(*sortby*,'A1234567890','A')** - This usage of TRANSLATE strips all of the numeric characters (1..9) out of the value of *sortby*. For any row where *sortby* ONLY contains numeric characters, it returns a null string ('').
2. **NVL(LENGTH(....),0))** - This part determines the length of the TRANSLATEd string, and NVLs it to 0 if the string is null. The SQL construct up to this point will return 0 if *sortby* has only numeric characters, and will return a positive integer if there are any non-numeric characters. .
3. **TO_CHAR(....)** - Converts the number returned by the LENGTH into a varchar. This needs to be done so that the DECODE will work correctly.
4. **DECODE(....,'0',LPAD(sortby,8),sortby)** - Now we come to the guts of this technique. This DECODE checks to see if the value is numeric only (a LENGTH of '0'), and if so, returns the value of *sortby* LPADed to 8 characters. If the value has any alpha characters (LENGTH > 0) it just returns *sortby* without any modification.

Now why do we want to LPAD the numeric values only? Because space (' ') sorts BEFORE the numbers! This allows for a decimal place by decimal place comparison of the two numbers. The '1' will return with 7 leading blanks, the '10' with 6, ect. When the alphabetical sort is done, the values are compared, character by character, and blank sorts before the numbers. If you selected the SQL construct that you are sorting by, you would see something like:

```
SQL> SELECT
DECODE(TO_CHAR(NVL(LENGTH(TRANSLATE(sortby,'A1234567890','A')),0)),'0',LPAD(sortby,8),s
ortby)
  2    FROM Number_Sort
  3   ORDER BY
DECODE(TO_CHAR(NVL(LENGTH(TRANSLATE(sortby,'A1234567890','A')),0)),'0',LPAD(sortby,8),s
ortby)
  4  /

SORTBY

--------------------



1


2


3


10


11


20


21



30




100


200


1000

1AB

A

BBBB
```

```
14 rows selected.
```

The '8' I used in the LPAD is just arbitrary. If I had 15 digit numbers in the SORTBY column, I would use a value greater than 15 so that any number would sort correctlyTip #23: Using a Database Function to Query by a LONG Column. (Type: SQL)

Have you ever tried to use LONG columns in the WHERE clause of your SQL statement? Something
like 'WHERE long_column like '%SEARCH%'? If so, you know this does not work! (You get 'ORA-00932:
inconsistent datatypes') This tip shows how you can use database functions to avoid this limitation, and
query by LONG columns anyway.

Consider the following table:

```
LONG_WHERE
==========
Name                 Null?    Type
-------------------- -------- ----
LONG_ID              NOT NULL NUMBER
LONG_DESC                     LONG
```

If you tried to select by the long column, you might use something like:

```
SQL> SELECT * FROM Long_Where
  2  WHERE long_desc like '%SEARCH%';
WHERE long_desc like '%SEARCH%'
      *
ERROR at line 2:
ORA-00932: inconsistent datatypes
```

Which does not work! One way around this limitation is to write a database function 'wrapper' for the
long column. This function would accept the PK of the table as an input parameter, and return the
LONG column's value, converted to a VARCHAR2. Here is an example of such a function:

```
CREATE OR REPLACE FUNCTION vc_desc(p_ID IN NUMBER) RETURN
VARCHAR2 IS
v_desc VARCHAR2(2000);
v_Long LONG;
BEGIN
  SELECT long_desc INTO v_Long
    FROM Long_where
   WHERE long_id = p_ID;
  v_Desc := SUBSTR(v_Long,1,2000);
  RETURN(v_Desc);
END;
```

Now you can use this function in the where clause instead of using the LONG column directly. For example:

```
SQL> SELECT long_id, long_desc
  2    FROM Long_Where
  3   WHERE vc_desc(long_id) like '%SEARCH%'
  4  /

  LONG_ID
LONG_DESC

--------- -------------------------------------------------------

        2 Another bunch of text to
          place into a long value.  Search for SEARCH2 somewhere

        4 Search for SEARCH3 in here
```

You are limited to searching the first 2000 characters of the long column, but this is often good enough.

Tip #19: Selecting ONLY the group with the maximum Sum in a group query. (Type: SQL)

Let's say you have a select that is summing by a key value, and you want to only return the key that has the maximum sum (not ALL of the rows like a group by will). This tip will show a SQL statement that does this.

Consider the following tables and data:

```
SUM_PARENT
----------
 Name                    Null?    Type
 ------------------- --------- ----
 SUM_ID                         NUMBER
 SUM_NAME                       VARCHAR2(10)

SELECT * FROM Sum_Parent;

   SUM_ID SUM_NAME
--------- ----------
        1 ONE
        2 TWO
        3 THREE
        4 FOUR
        5 FIVE


SUM_CHILD
---------
 Name                    Null?    Type
 ------------------- --------- ----
 SUM_ID                         NUMBER
 QTY                          NUMBER

SELECT * FROM Sum_Child;

   SUM_ID        QTY
--------- ---------
        1        10
        1        20
        1         5
        2        10
        2         5
        3         3
        3         2
        4        30
        4         2
        5        10
```

The following simple GROUP BY select will return the sum of the QTY for each key (SUM_NAME):

```
SELECT p.sum_name, sum(c.qty)
  FROM Sum_Parent p, Sum_Child c
 WHERE p.sum_id = c.sum_id
GROUP BY p.sum_name
/

SUM_NAME   SUM(C.QTY)
---------- ----------
FIVE               10
FOUR               32
ONE                35
THREE               5
TWO                15
```

Let's say you only want to return the row with the MAXIMUM quantity (SUM_NAME=ONE). To do this, you can add a HAVING predicate related to a sub-query. The HAVING is needed to be able to use the SUM(c.qty) in a predicate. An example of this follows:

```
SELECT p.sum_name, sum(c.qty)
  FROM Sum_Parent p, Sum_Child c
 WHERE p.sum_id = c.sum_id
GROUP BY p.sum_name
HAVING SUM(c.qty) =
   (SELECT MAX(SUM(c2.qty))
      FROM Sum_Child c2
    GROUP BY c2.sum_id
   )
/

SUM_NAME   SUM(C.QTY)
---------- ----------
ONE                35
```

Which is the desired result. One caution however, if there are more than one key with the maximum sum, the query will return ALL of them. For instance, if the following data is added to the SUM_CHILD table:

```
   SUM_ID        QTY
--------- ---------
```

```
                3         30
```

The above query will return the following result.

```
SUM_NAME   SUM(C.QTY)
---------- ----------
ONE                35
THREE              35
```

This query is not very efficient however. If you have PL/SQL available to you, you can get the same result by creating a cursor with the first query, adding an order by clause, and only fetching the first row. This will avoid the second sum, which will give it better performance. Here is an example of this:

```
set serveroutput on
DECLARE
  CURSOR sum_cur IS
  SELECT p.sum_name, sum(c.qty)
    FROM Sum_Parent p, Sum_Child c
   WHERE p.sum_id = c.sum_id
  GROUP BY p.sum_name
  ORDER BY sum(c.qty) desc;
 SumName VARCHAR2(10);
 SumQty  NUMBER;
BEGIN
  OPEN sum_cur;
  FETCH sum_cur INTO SumName, SumQty;
  CLOSE sum_cur;
  DBMS_OUTPUT.PUT_LINE('SUM_NAME: '||SumName||' Sum(Qty):
'||to_char(SumQty));
END;
/
SUM_NAME: ONE Sum(Qty): 35

PL/SQL procedure successfully completed.
```

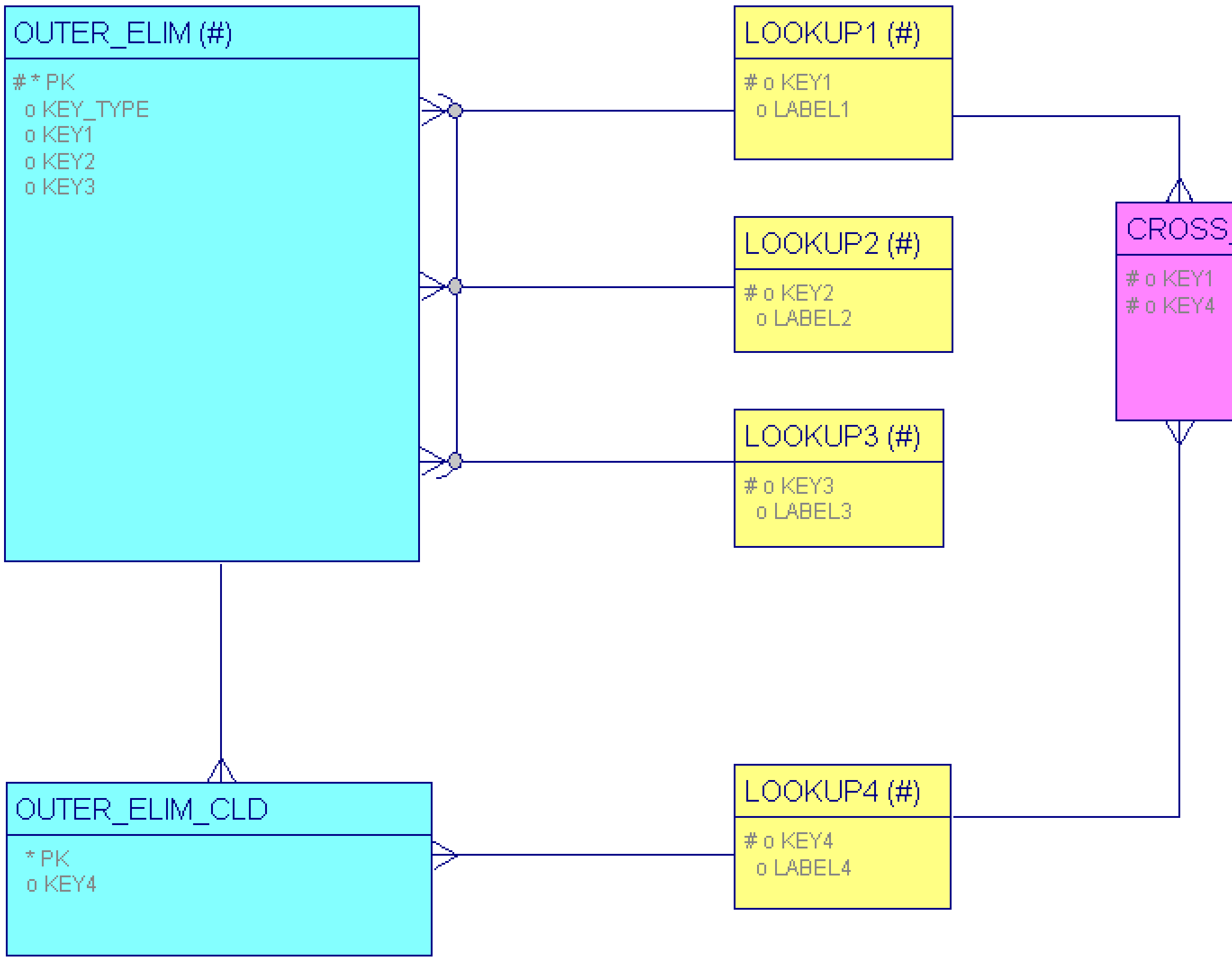
If there are more than one row with the same maximum sum, this PL/SQL program will only return the first one it encounters. Therefore, the program should be expanded to handle this by either returning multiple values, or ordering by the key also (which will ensure consistancy if nothing else).

Tip #18: Using Database Functions to Eliminate Outer Joins. (Type: PL/SQL & SQL)

Outer joins are very useful in SQL to return data from queries where some of the relationships are optional. However, there are times when the outer joins can cause some problems. Sometimes they will make the query run very slowly. There is also a restriction that you can only outer a table to ONE other table. In these cases, database functions can be used to eliminate the need for the outer joins. This tip will detail how to do this for the following two examples:

- Elimination of Multiple Outer Joins in ARC usage.
- Allowing 'Outer Join' to Multiple Tables.

Consider the following data model:



Assuming the data looks like:

```
OUTER_ELIM
==========
       PK KTYPE      KEY1      KEY2      KEY3
--------- ----- --------- --------- ---------
        1 1             1
        2 1             2
        3 1             3
        5 2                       1
        6 2                       2
        7 2                       3
        9
       10 3                                 1
       11 3                                 2


LOOKUP1                  LOOKUP2                  LOOKUP3
=======                  =======                  =======
     KEY1 LABEL1              KEY2 LABEL2              KEY3
LABEL3
--------- ----------     --------- ----------     ---------
----------
        1 1-ONE                  1 1-TWO                  1
1-THREE
        2 2-ONE                  2 2-TWO                  2
2-THREE
        3 3-ONE                  3 3-TWO                  3
3-THREE


OUTER_ELIM_CLD               CROSS_LABEL
==============               ===========
       PK      KEY4               KEY1      KEY4
CROSS_LABEL
--------- ---------          --------- ---------   --------
-------
        1         1                  1         2   L1-1 X
L4-2
        1         2                  3         1   L1-3 X
L4-1
        1         3
        2         1
        2         2
        2         3
        3         1
        3         2
        3         3
```

This data model will be used to illustrate both examples of using a database function to eliminate outer joins:

Elimination of Multiple Outer Joins in ARC usage.

Let's say you need a SQL select statement to implement the above arc (from OUTER_ELIM to LOOKUP1,2,3). For example, you want to display only ONE of the labels (either LABEL1, LABEL2, or LABEL3) depending on the value of the KEY_TYPE column. The following SQL statement could be used to do this:

```
/* tip18q1.sql */
SELECT oe.pk,
DECODE(ktype,'1',l1.label1,'2',l2.label2,'3',l3.label3)
Label
  FROM outer_elim oe, lookup1 l1, lookup2 l2, lookup3 l3
 WHERE oe.key1 = l1.key1 (+)
   AND oe.key2 = l2.key2 (+)
   AND oe.key3 = l3.key3 (+)
```

```
ORDER BY
DECODE(ktype,'1',l1.label1,'2',l2.label2,'3',l3.label3)
/
```

This SQL statement will return:

```
PK        LABEL
--------- ----------
        1 1-ONE
       10 1-THREE
        5 1-TWO
        2 2-ONE
       11 2-THREE
        6 2-TWO
        3 3-ONE
        7 3-TWO
        9
```

However, sometimes having multiple outer joins can cause performance problems. Especially when there are many tables joined together.
A database function can be used to get the same query results without using an outer join.
To do this, first create the following database function:

```
/* tip18fun.sql */
CREATE OR REPLACE
FUNCTION comb_label(pType IN VARCHAR2
         ,pKey1 IN NUMBER,pKey2 IN NUMBER,pKey3 IN NUMBER)
RETURN VARCHAR2 IS
  vReturn VARCHAR2(10);
BEGIN
  IF pType = '1' THEN
    BEGIN
      SELECT label1 INTO vReturn
  FROM lookup1
       WHERE key1 = pKey1;
    END;
  ELSIF pType = '2' THEN
    BEGIN
      SELECT label2 INTO vReturn
  FROM lookup2
       WHERE key2 = pKey2;
    END;
  ELSIF pType = '3' THEN
    BEGIN
      SELECT label3 INTO vReturn
  FROM lookup3
       WHERE key3 = pKey3;
    END;
  END IF;
RETURN(vReturn);
END;
/
```

This function can be used in the following SQL to return the same results as the first query, but without any outer joins:

```
/* tip18q2.sql */
column label format a12;
SELECT oe.pk, comb_label(ktype,key1,key2,key3) Label
  FROM outer_elim oe
ORDER BY comb_label(ktype,key1,key2,key3)
/
```

Using a Database Function to allow Outer Join to Multiple Tables.

There are times when you really want to outer join one table to two different tables. When this happens, the restriction can be frustrating. In the above data model, you might want to do an outer join from the CROSS_LOOKUP table to both the OUTER_ELIM and OUTER_ELIM_CLD tables. To do this, you might try to use a SQL statement like:

```
/* tip18q3.sql */
SELECT oe.pk, l1.label1, l4.label4, cl.cross_label
  FROM outer_elim oe,  outer_elim_cld cld, lookup1 l1,
lookup4 l4
  ,cross_lookup cl
 WHERE oe.pk = cld.pk
   AND oe.key1 = l1.key1
   AND cld.key4 = l4.key4
   AND cl.key1 (+) = oe.key1
   AND cl.key4 (+) = cld.key4
/
```

Which will cause the following error:

```
   AND cl.key1 (+) = oe.key1
                   *
ERROR at line 7:
ORA-01417: a table may be outer joined to at most one other
table
```

This error occurs because Oracle will not let you have an outer join from ONE table to two DIFFERENT tables.

However, you CAN get the desired effect using a database function. If the following function is created:

```
/* tip18fn2.sql */
CREATE OR REPLACE
FUNCTION cross_label(pKey1 IN NUMBER,pKey4 IN NUMBER)
RETURN VARCHAR2 IS
  vReturn VARCHAR2(15);
BEGIN
  SELECT cross_label INTO vReturn
    FROM cross_lookup
   WHERE key1 = pKey1
     AND key4 = pKey4;
  RETURN(vReturn);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN('');
END;
/
```

then the following SQL can be used to return the 'outer join' values from the CROSS_LOOKUP:

```
/* tip18q4.sql */
column cross_label format a15
SELECT oe.pk, l1.label1, l4.label4, cross_label(oe.key1,
cld.key4) cross_label
  FROM outer_elim oe,  outer_elim_cld cld, lookup1 l1,
lookup4 l4
 WHERE oe.pk = cld.pk
   AND oe.key1 = l1.key1
   AND cld.key4 = l4.key4
/
```

This select will return:

```
       PK LABEL1     LABEL4     CROSS_LABEL
--------- ---------- ---------- ---------------
        1 1-ONE      1-FOUR
        1 1-ONE      2-FOUR     L1-1 X L4-2
        1 1-ONE      3-FOUR
        2 2-ONE      1-FOUR
        2 2-ONE      2-FOUR
        2 2-ONE      3-FOUR
        3 3-ONE      1-FOUR     L1-3 X L4-1
        3 3-ONE      2-FOUR
        3 3-ONE      3-FOUR
```

Stored procedures and functions are very powerful. These are just some simple examples to get people thinking about some possible ways of using them.

Tip #17: Calculating a running total with SQL only. (Type: SQL)

There are applications where the running total for a series of numbers needs to be calculated and displayed. While this might be normally considered something that would be done with 3GL programming techniques, it IS possible to calculate and query a running total with SQL only.

First, we need to have a 'key' column to order the query by

Let's assume we have a table with a numeric column that we want. We also need to have a column to order the display by, or a running total does not make sense! Many times this will be a date column. This could also be the PK of the table, or a single or multiple column UK. I will use a date column in the following example. Consider the table:

```
SQL> desc Run_Total

Name                            Null?    Type
------------------------------- -------- ----
RUN_DATE                                 DATE
RUN_VALUE                                NUMBER

SQL> select * from Run_Total;

RUN_DATE  RUN_VALUE
```

```
        --------- ---------
        02-APR-97        10
        03-APR-97         5
        04-APR-97        20
        05-APR-97        15
        06-APR-97        45
        07-APR-97        12
        08-APR-97        37
        09-APR-97         9
        10-APR-97        23
        11-APR-97        19
        12-APR-97        10
        11 rows selected.
```

A self-join is needed to perform the correct sum

In order to produce the running total, a self-join is needed to sum all of the values of the table less than or equal to each row. The query is grouped by the records in the 'driving' table in the join, and ordered by the run_date:

```
        SELECT r1.run_date, r1.run_value, sum(r2.run_value)
Running_Total
          FROM Run_Total r1, Run_Total r2
         WHERE r2.run_date <= r1.run_date
         GROUP BY r1.run_date, r1.run_value
         ORDER BY r1.run_date
        /
```

This query will produce the following output:

```
        RUN_DATE  RUN_VALUE RUNNING_TOTAL
        --------- --------- -------------
        02-APR-97        10            10
        03-APR-97         5            15
        04-APR-97        20            35
        05-APR-97        15            50
        06-APR-97        45            95
        07-APR-97        12           107
        08-APR-97        37           144
        09-APR-97         9           153
        10-APR-97        23           176
        11-APR-97        19           195
        12-APR-97        10           205

        11 rows selected.
```

Tip #6: Using SQL only to Determine the Business Days Between Two Dates. (Type: SQL)

There are many times in reports or in calculations for forms where the number of business days between two dates needs to be determined. Here is a method for caluculating this (excluding holidays) using SQL only.

The following SQL script shows an algorythm that uses the standard Oracle date functions to calculate the number of business days between to dates. This method cannot exclude holidays (obviously), however there are many times that just the standard business days is useful. I am sure there are other algorythms that could be used, however this one has worked for me. The algorythm is described below in the comments of the script.

```
/*********************************************************
*******/
/* An example of business days calculation in SQL
*/
/*
*/
/*  The algorythm is:
*/
/*
*/
/*   1) Take the absolute difference between the dates
*/
/*        to_date('&todate') - to_date('&frdate')
*/
/*   2) Subtract the weekends (number of weeks in the range
*/
/*        TRUNC(to_date('&todate'),'D') = 1st day of week
that   */
/*                                        end of period is
in    */
/*        TRUNC(to_date('&frdate'),'D') = Last day of week
that  */
/*                                        start of period
is in  */
/*        So subtracting these two gives the number of days
*/
/*        between the two dates but including all of the
days in */
/*        the weeks that the dates start and end in.  When
this  */
/*        number is divided by 7 it gives the number of
weeks.   */
/*        Multiplying by 2 gives the number of weekend
days.     */
/*   3) Subtract 1 day if the ending date is on a saturday
*/
/*        DECODE(to_char(to_date('&todate'),'D'),7,-1,0)
*/
/*        --> If the day of the week is saturday (7),
returns -1 */
/*   4) Subtract 1 day if the start date is on a sunday
*/
```

```
/*        DECODE(to_char(to_date('&frdate'),'D'),1,-1)
*/
/*        --> If the day of the week is sunday (1), returns
1    */
/*   5) Add one day to make the range inclusive (The '1 + '
)    */
/*-----------------------------------------------------------
------*/
/* Author:  Kenneth Atkins  (Ken@arrowsent.com)
*/
/*          http://www.olywa.net/katkins/oratip
*/
/***********************************************************
*******/
define frdate = '&1'
define todate = '&2'
set verify off
select
   '&frdate' From_Date
  ,'&todate' To_Date,
  1 + to_date('&todate') - to_date('&frdate') -
  ((TRUNC(to_date('&todate'),'D') -
TRUNC(to_date('&frdate'),'D'))/7)*2
  + DECODE(to_char(to_date('&todate'),'D'),7,-1,0)
        + DECODE(to_char(to_date('&frdate'),'D'),1,-1,0)
Business_Days
 from dual
/
```

Here is an example of running the script:

```
SQL> @busdays 01-AUG-96 15-AUG-96

FROM_DATE TO_DATE   BUSINESS_DAYS
--------- --------- -------------
01-AUG-96 15-AUG-96            11

1 row selected.
```

This same algorythm can also be put into a stored function:

```
CREATE OR REPLACE FUNCTION business_days(p_from_date IN
DATE, p_to_date IN DATE)
   RETURN NUMBER IS
busdays NUMBER;
BEGIN
/***********************************************************
*******/
/* BUSINESS_DAYS  -  Database Function to Calculate number
of    */
/*                   business days between two dates
*/
/*-----------------------------------------------------------
------*/
```

```
/* Author:  Kenneth Atkins  (Ken@arrowsent.com)
*/
/*          http://www.olywa.net/katkins/oratip
*/
/***********************************************************
*******/

  -- Get the absolute date range
  busdays := TRUNC(p_to_date) - TRUNC(p_from_date)
        -- Now subtract the weekends
         --  this statement rounds the range to whole weeks
(using
         --  TRUNC and determines the number of days in the
range.
         --  then it divides by 7 to get the number of
weeks, and
         --  multiplies by 2 to get the number of weekend
days.
   - ((TRUNC(p_to_date,'D')-
TRUNC(to_date(p_from_date),'D'))/7)*2
         -- Add one to make the range inclusive
   + 1;

  /* Adjust for ending date on a saturday */
  IF TO_CHAR(p_to_date,'D') = '7' THEN
    busdays := busdays - 1;
  END IF;

  /* Adjust for starting date on a sunday */
  IF TO_CHAR(p_from_date,'D') = '1' THEN
    busdays := busdays - 1;
  END IF;
  RETURN(busdays);
END;
/
show errors;
```

Here is an example of calling this database function:

```
SQL> select business_days('01-AUG-96','15-AUG-96') from
dual;

BUSINESS_DAYS('01-AUG-96','15-AUG-96')
--------------------------------------
                                    11

1 row selected.
```

Of course in a stored function, you could add code to substract holidays also. Perhaps something like:

```
SELECT COUNT(*) INTO nHolidays
  FROM Holiday_Table
 WHERE holiday_date BETWEEN p_from_date AND p_to_date;
```

Then substract nHolidays from your business_days variable before returning.

Tip #3: Script to List PL/SQL Errors & Line Numbers (Type: PL/SQL)

When PL/SQL stored packages, procedures, or functions are loaded into the database and compiled (using CREATE OR REPLACE) any syntax errors in the code can be listed using the 'SHOW ERROR' command. However, many times these messages are cryptic, and the line numbers specified do not correspond to the line number in the SQL file used to load the code (because blank lines and comments before the 'CREATE' statement are not loaded. For large procedures, this can be very frustrating. This hint will show how a SQL script can be used to list the errors along with the actual source lines, indicating the line with the error (with a '-->').

The Oracle views USER_SOURCE, and USER_ERRORS (or ALL_SOURCE and ALL_ERRORS) can be used to list the source of the program the way the compiler sees it (without blank lines and leading comments). USER_ERRORS is the view used by SHOW ERRORS to display the errors for a PL/SQL program. The following select statement will join these two views to list all of the compile errors, and point out specifically which line the errors are on (using '-->').

```
/**********************************************************
******************/
/* listerr.sql   -  Lists errors and source for errors for
PL/SQL programs.  */
/*
*/
/*     Parameter:  &1 = The name of the PROCEDURE, PACKAGE,
or FUNCTION      */
/*
*/
/*        Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*                 http://www.arrowsent.com/oratip
*/
/*
*/
/*    This script uses the 'USER' views.  It could easily
be modified to use */
/*    the 'ALL' views by changing the 'user_' to 'all_' in
the view names    */
/*    and by adding an owner as another parameter
*/
/**********************************************************
******************/
set verify off
define obj_name = '&1';
column outline format a105 heading 'Error Listing';
break on err_text skip 2;
set linesize 105;
set pagesize 0;
set pause off;
spool listerr
SELECT
decode(to_char(us.line), to_char(ue.line-7),ue.text,
                         to_char(ue.line-6),'',
                         to_char(ue.line+6),'',
                         to_char(ue.line)  ,'   -->
'||to_char(us.line,'99990')
```

```
                                                                           ||'
      '||us.text
                                                                 ,'
      '||to_char(us.line,'99990')
                                                                           ||'
      '||us.text) outline
      from user_source us, user_errors ue
       where us.name = '&obj_name'
      and us.line between (ue.line-7) and (ue.line+6)
      and us.name = ue.name
      and us.type = ue.type
      -- This predicate is put here to elminate this useless
      fallout error
      and ue.text != 'PL/SQL: Statement ignored'
      /
      spool off
      set pause on;
      set pagesize 22;
```

Here is an example of using the script. A SQL script called 'hint3pck' has the definition of a package with a few errors. The above SQL script is called 'listerr.sql':

```
      CASE:KENNEA> @hint3pck

      Package created.

      No errors.

      Warning: Package Body created with compilation errors.

      Errors for PACKAGE BODY PACK_WITH_ERROR:
      >
      LINE/COL ERROR
      -------- --------------------------------------------------
      ---------------
      7/1      PL/SQL: Statement ignored
      9/1      PLS-00201: identifier 'VVAR' must be declared
      9/1      PL/SQL: Statement ignored
      19/11    PLS-00201: identifier 'NO_DTA_FOUND' must be
      declared

      CASE:KENNEA> @listerr PACK_WITH_ERROR
      PLS-00201: identifier 'VVAR' must be declared

                   4 vUSER VARCHAR2(30);
                   5 vTest VARCHAR2(30);
                   6 nTest NUMBER;
                   7 BEGIN
                   8 /* Comment line */
         -->       9 vVar := to_num('12');
                  10 /* Comment line */
                  11 /* The next line has an error (missing
      semicolon) */
                  12   nVar := 1;
                  13 /* more comments */
                  14  nVar := 3;

      PLS-00201: identifier 'NO_DTA_FOUND' must be declared
```

```
             14  nVar := 3;
             15   BEGIN
             16     SELECT user INTO vUSER
             17       FROM dual;
             18   EXCEPTION
   -->       19      WHEN NO_DTA_FOUND THEN
             20    NULL;
             21   END;
             22   BEGIN
             23     SELECT user INTO vUSER
             24       FROM dual;

28 rows selected.
```

The following select can also be used to just list the source for a stored procedure, package or function, putting in the line numbers the compiler uses:

```
/***********************************************************
*******************/
/* listsource.sql   -  Lists source for PL/SQL programs.
*/
/*
*/
/*     Parameter:  &1 = The name of the PROCEDURE, PACKAGE,
or FUNCTION      */
/*
*/
/*        Author:  Ken Atkins (Ken@arrowsent.com)
*/
/*                 http://www.arrowsent.com/oratip
*/
/*
*/
/***********************************************************
*******************/
define obj_name = '&1';
column text format a74 heading 'Source Listing';
column line format 9999 heading 'Line';
set verify off
set linesize 80;
set pagesize 0;
set pause off;
spool &obj_name
SELECT
us.line, us.text
from user_source us
 where us.name = '&obj_name'
order by type, line
/
spool off
set pause on;
set pagesize 22;
```

Tip #2: Determining Instance Name from SQL. (Type: SQL*Plus)

Sometimes it is useful to be able to get the name of the current instance from within SQL*Plus or another development tool (such as Oracle*Forms or Oracle*Reports). Here is a simple SQL statement that can be used to get the instance name. Also presented is an example of using the SQL statement to set the SQL prompt to the instance name.

There is an internal oracle view called 'V$PARAMETER' which holds the values of many parameters that the database uses. One of these parameters is called 'DB_NAME'. This parameter holds the name of the database (kind of makes sense, doesn't it?). The V$PARAMETER view looks like:

```
desc v$parameter

 Name                            Null?    Type
 ------------------------------- -------- ----
 NUM                                      NUMBER
 NAME                                     VARCHAR2(64)
 TYPE                                     NUMBER
 VALUE                                    VARCHAR2(512)
 ISDEFAULT                                VARCHAR2(9)
```

The following SQL statement will return the database name from this view:

```
SELECT UPPER(value)
  FROM V$Parameter
 WHERE UPPER(name) = 'DB_NAME';
```

A SQL statement like this can be used to replace the standard sql prompt ('SQL>') with the instance name. This can be useful if you are accessing many different instances in SQL*Plus. It has saved me from messing up data in the wrong instance many times (like deleting data from the production instance instead of test). Add the following SQL to your login.sql file:

```
rem Create _DB_Name variable for general use.
set termout off
column upper(VALUE) new_value _DB_NAME;
select upper(value), from v$parameter
  where upper(name) = 'DB_NAME';
rem
rem Put SGA Name in sql prompt
set SQLPROMPT '&_DB_NAME.> '
rem
rem If pause is on, say something when needed.
set pause '> '
rem
clear breaks
set termout on
set pause on
set feedback on
```

If the instance name is something like 'TESTINST' then the above SQL in the login.sql file will change the prompt to:

TESTINST>

Ken Atkins' Oracle Database Tip of the week.

Tip #1: Conditional loading of PL/SQL code. (Type: PL/SQL)

Because of varying requirements in distributed databases or multi-organizational companies, there is sometimes a need to impliment a PL/SQL package or procedure with slight differences in the code when installed on separate instances or schemas. Quite often this was done by maintaining a version of the code for each site. However, the following technique can be used to keep all of the source in one file, and have the differences implemented when the code is installed.
A SQL*Plus variable (& (ampersand) variable) can be dynamically loaded using the 'new_value' clause of the SQL*Plus column command. This variable can be dynamically based upon the schema, or instance, or data in an existing table. These variables are scanned and replaced in the SQL code before the code is installed or compiled. Therefore the '&' variables can be used to change the code that is installed. Below is an example of using this technique to 'comment out' a call to a procedure for one schema only:

```
set pause off
column comvar new_value comment_var
SELECT DECODE(user,'CWVND','--','') comvar
  FROM DUAL
/
CREATE OR REPLACE PACKAGE test_package AS
       PROCEDURE main;
       PROCEDURE conditional_proc(parm1 IN VARCHAR2);
END test_package;
/
CREATE OR REPLACE PACKAGE BODY test_package AS
       PROCEDURE main IS
       BEGIN
         DBMS_OUTPUT.PUT_LINE('Beginning of main
procedure.');
         &comment_var conditional_proc('Test');
         DBMS_OUTPUT.PUT_LINE('End of main procedure.');
       END main;
       PROCEDURE conditional_proc(parm1 IN VARCHAR2) IS
       BEGIN
         DBMS_OUTPUT.PUT_LINE('Conditional_proc called with
parm1='||parm1);
       END conditional_proc;
END test_package;
/
```

If the above package is installed in the 'USER1' schema, then the 'conditional_proc' procedure will be commented out, and therefore will not run. For all other schemas, the procedure will NOT be commented out, and will run. Some other ways to use this technique:

- Make the parameter to a procedure conditional. For example, the call to the 'conditional_proc' procedure above could be changed to:
    - `column userval new_value user_val;`
    - `SELECT user userval FROM DUAL;`

  - .
  - .
  - `conditional_proc('&user_val');`
- Make the name of a procedure conditional. The following lines could be used to call a procedure that is called _setup, where is the name of the oracle instance.
  - `column procname new_value proc_name;`
  - `SELECT value||'_setup' procname`
  - `  FROM v$parameter`
  - ` WHERE upper(name) = 'DB_NAME';`
  - .
  - .
  - `    &proc_name;`

Tip #57: Procedure to show all of an Oracle Portal session variable's attributes (Type: Oracle on the Web)

Oracle When you are using Oracle Portal session variables, it is useful to be able to see the values of it's attributes when you are testing and debugging an application that uses them. This tip is a PL/SQL package that can be run from the browser to list all of a session variable's attributes to the browser.

A PL/SQL Procedure to do the job

Here is a simple PL/SQL procedure that accepts the domain and sub-domain of the Oracle Portal session and prints the names and values of all of the sessions attributes to the browser.

```
CREATE OR REPLACE PROCEDURE show_session(p_domain IN
VARCHAR2, p_subdomain IN VARCHAR2) IS
/**********************************************************
******************************/
/* SHOW_SESSION - Shows the names and values of all of the
attributes of an Oracle         */
/*                  Portal session variable.
/*
/*                  Parameters:  p_domain    = The name of
the session domain.             */
/*                               p_subdomain = The name of
the session subdomain.          */
/*
/* Written by ken atkins (ken@arrowsent.com).  Copywrite
2001, all rights reserved.      */
/* You may use this script for any purpose as long as you
do not include it in any        */
/* commercial software for sale. If you make changes to
improve the script, please      */
/* send them to me so I can make them available for other
users.                   */
/*
```

```
/* Check out my oracle tip site at:
http://www.arrowset.com/oratips                 */
/*
/* Vers  Date       By              Change History
/* ----- ---------  --------------- -----------------------
-------------------------         */
/* 1.0   14-FEB-01  Ken Atkins      Written.
/************************************************************
******************************/
  v_session portal30.wwsto_api_session;
  v_elements portal30.wwsto_session_elements;
  v_num_attr INTEGER;
  v_element portal30.wwsto_session_element;
BEGIN
  v_session := wwsto_api_session.load_session(p_domain,
p_subdomain);
  v_elements := v_session."_element_data";

  v_num_attr := v_elements.count;
  htp.p('Session attributes for:<BR><BR><TABLE
BORDER=0><TR><TD>DOMAIN:</TD><TD><B>'||p_domain||'</B></TD>
</TR>');
  htp.p('<TR><TD>SUB-
DOMAIN:</TD><TD><B>'||p_subdomain||'</B></TD></TR></TABLE>'
);
  htp.br;
  htp.p('Number of Attributes: '||to_char(v_num_attr));
  htp.br;
  htp.br;
  htp.p('<TABLE BORDER="0">');
  FOR i IN 1..v_num_attr LOOP

htp.p('<TR><TD>'||v_elements(i).name||'</TD><TD>=</TD><TD>'
||v_elements(i).varchar2_data||'</TD></TR>');
  END LOOP;
  htp.p('</TABLE>');

END;
/
show errors
```

You can download the script by clicking here.

Installation

To install the procedure, do the following:

1. Install the procedure either in the PORTAL30 schema, or a schema that has access to the PORTAL30 programs (for instance a provider schema).
2. Grant execute on the procedure to the portal30 public schema (usually PORTAL30_PUBLIC). For instance:

```
   3.    GRANT EXECUTE ON show_session TO portal30_public;
```

Using the procedure

To use the procedure, simply open a separate navigator window (after you have run the application which has set some session attributes), and type a call to the procedure into the URL field. For example, the following URL will call the procedure on a local install of portal30 with the standard dad:

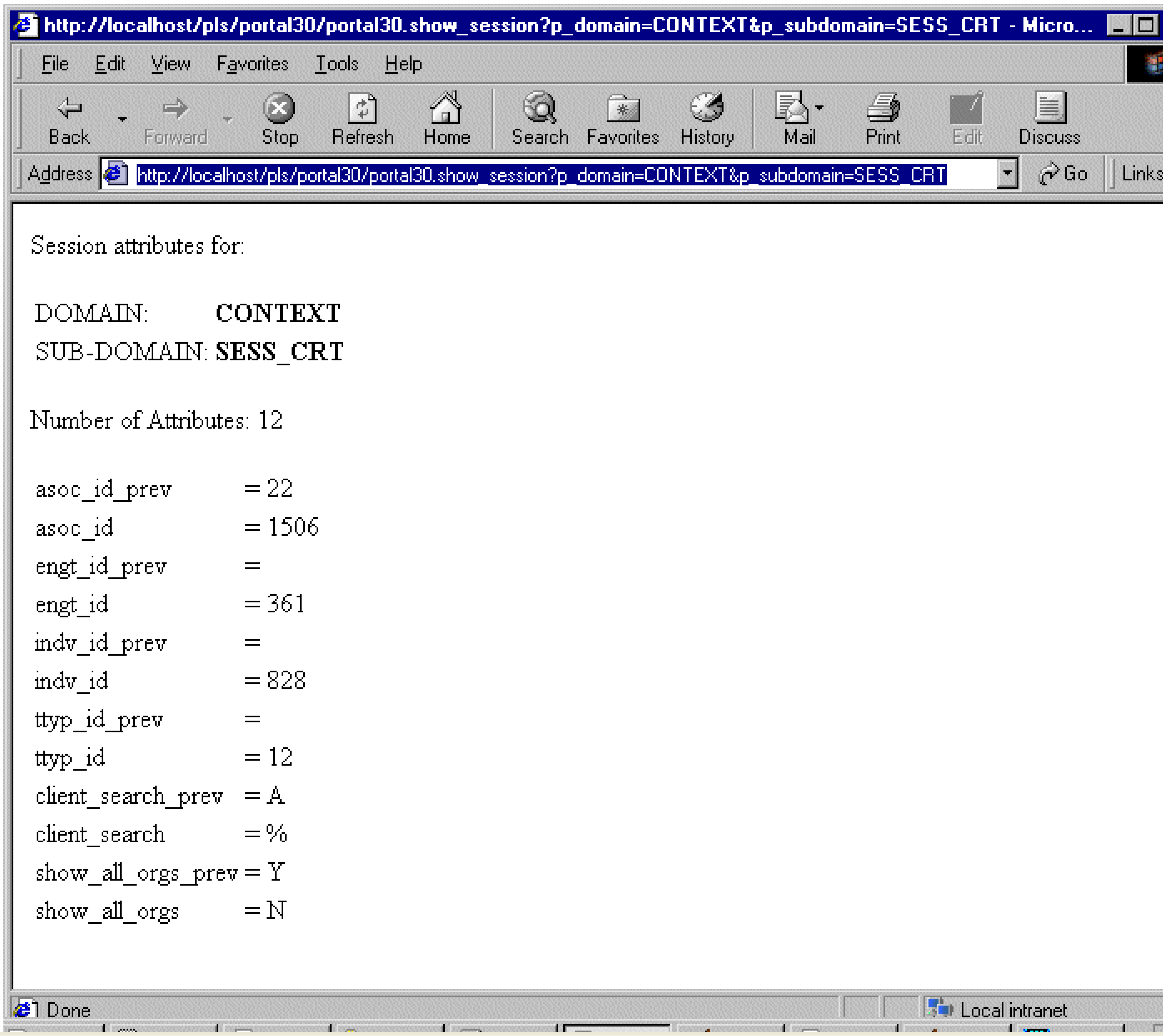
```
http://localhost/pls/portal30/portal30.show_session?p_domain=CONTEX
T&p_subdomain=SESS_CRT
```

This call assumes the procedure is installed in the PORTAL30 schema. If it is installed in another schema, replace the "portal30" immediately before "show_session" with the name of the schema in which it is installed.

Replace "localhost/pls/portal30" with the appropriate dad-path for your installation.

The value for *p_domain* and *p_subdomain* are set to the domain and subdomain of the session variable you want to see.

Here is an example of the output:


http://localhost/pls/portal30/portal30.show_session?p_domain=CONTEXT&p_subdomain=SESS_CRT - Micro...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://localhost/pls/portal30/portal30.show_session?p_domain=CONTEXT&p_subdomain=SESS_CRT Go Links
Done
Local intranet

Session attributes for:

DOMAIN: **CONTEXT**
SUB-DOMAIN: **SESS_CRT**

Number of Attributes: 12

```
asoc_id_prev       = 22
asoc_id            = 1506
engt_id_prev       =
engt_id            = 361
indv_id_prev       =
indv_id            = 828
ttyp_id_prev       =
ttyp_id            = 12
client_search_prev = A
client_search      = %
show_all_orgs_prev = Y
show_all_orgs      = N
```

Summary:	How do I implement a ru
Description:	Many form implementati be annoying to the user. This tip details how you c For the complete tip, visi For more of Ken's Oracle

Summary:	

Description:
Summary:
Description:

Summary:	
Description:	
Summary:	
Description:	
Summary:	
Description:	
Summary:	
Description:	
Summary:	
Description:	

Summary:
Description:
Summary:
Description:

Summary:	
Description:	

Summary:
Description:

Summary:
Description:
Summary:
Description:

Summary:
Description:
Summary:
Description:

Summary:
Description:
Summary:
Description:

Summary:
Description:

Summary:	
Description:	

Summary:	
Description:	

Summary:	How to determine the time and date
Description:	There is information available in the this informatoin is not in a convenie started. I have found it to be handier `SELECT TO_DATE(VALUE, 'J`

	` FROM V$INSTANCE`
	`WHERE key = 'STARTUP TIM`
	
	`SELECT TRUNC(VALUE/(60*6`
	`startup time"`
	` FROM V$INSTANCE`
	`WHERE KEY = 'STARTUP TIM`
	
	This information applies to version
	
	This will not work for Oracle 8, so u
	
	`select to_char(startup_t`
	` from v$instance`

Summary:	How do I load a large (more than 4K)
Description:	To load a large HTML file into a varc program to split the data into multiple table. Then you would also need a program this on its own. The other option is to than the VARCHAR2 datatype. This length, but there are some restrictions

Summary:	How do I import data from an excel spre Oracle table?
Description:	Two ways I can think of to import data f spreadsheet into an Oracle table: 1. Use an ODBC connection from Excel spreadsheet into table. 2. Export data from Excel spreadsheet to sqlloader to load the table from the flat f Either way is available, however, ODBC set up - sql*net has to be set up on the p quirks also, but it is usually faster to loa empty table rather than insert row by row

Summary:	Use NLS_UPPER fur speed of case-insensit
Description:	The NLS_UPPER fur you want to create an where your search qu

	insensitive, and the da
	either upper or lower
	
	To use:
	
	`create index x`
	`(NLS_UPPER(col_`
	
	`select * from t`
	`NLS_UPPER(col_1`
	
	This is much faster (s
	
	`create index x`
	`table_y(col_1),`
	
	`select * from t`
	`upper(col_1)='A`
	
	..because in the first c
	already converted eve
	case, and in the secon
	the conversion as it qu

Summary:	8.1.5 Export and it's parameters behav differently.
Description:	I happened on a problem regarding ex parameter files in 8.1.5. I have the foll parameter file: userid=scott/tiger rows=n indexes=n grants=n file=temp_inside.dmp tables=(user1.Table_x) In 7.3.3 if I run the following statemen exp parfile=temp.par file=temp_outsic the file written to is temp_outside.dmp In 8.1.5 if I run the same statement the

	written to is temp_inside.dmp. According to Oracle's 8.1.5 document 'You can use a combination of the firs second options. That is, you can list p both in the parameters file and on the line. In fact, you can specify the same in both places. The position of the PA parameter and other parameters on the line determines what parameters overr For example, assume the parameters f params.dat contains the parameter IN and Export is invoked with the follow exp system/manager PARFILE=paran INDEXES=N In this case, because INDEXES=N oc PARFILE=params.dat, INDEXES=N the value of the INDEXES parameter PARFILE.' Clearly this is not true. Be problem.

Summary:	How much "true" data is in a table, in blocks?
Description:	Need to know how much data is in a table, I don't mean reserved space from dba_extents, I mean real amount in something useful like blocks: For v7.x DBs: select count(distinct substr(rowid,1,8)||substr(rowid,15,4)) from ; For v8.x DBs: select count(distinct substr(rowid,10,6)||substr(rowid,7,3)) from ; Replace with your table name. This will produce a total of

	blocks that have been used by the data in the table, simply multiply the figure by your block size to get the actual bytes figure.

Summary:	Adjustment of scripts following de-support of svrmgrl.

Description:	Oracle has announced future desupport of svrmgrl. You may ask yourself, self, how am I going to adjust my scripts to SQLPLUS and include the shutdown/startup stuff.
	Use the following command:
	
	`sqlplus /nolog`
	`SQL>connect internal;`
	`connected`
	`SQL>shutdown immediate;`
	
	Cool huh!

Summary:	How can I write a message to the ORACLE Alert log from a PL/SQL program?
Description:	Use the following pl/sql: dbms_system.ksdwrt(2,'A

	line of text'); Will write the text to the alert log. Use 1 instead of 2 to write to the trace file Use 3 to write to both. Thanks to: Jonathan Lewis Yet another Oracle-related web site: http://www.jlcomp.demon.co.uk

Summary:

Description:

Summary:
Description:
Summary:
Description:
Summary:
Description:

Summary:	
Description:	
Summary:	Oracle8 installation creates a new account, tracesvr/trace, with SELECT
Description:	The Oracle8 install has provided a very dangerous back door. It creates a Trace. The new account is TRACESVR, and the default password is wel account is granted SELECT ANY TABLE. This feature was discovered on a V8.0.6 install.
Summary:	SQL Loader parameters when fields contain line feeds or carriage returns
Description:	SQL Loader control files use line feed or CR to signify the end of the rec entry of these characters (most commonly as part of a description or text

	encounters these line feeds, it assumes the end of the record and generate To get around this: 1. when you generate the ascii file, put a field delimiter after each colum character as the delimiter. 2. in the SQL Loader control file, add the line: continueif last !='|' This will cause SQL Loader to correctly interpret data where line feeds o CRs are a legitimate part of the data.
Summary:	SQL Loader parameters when fields contain line feeds or carriage return
Description:	SQL Loader control files use line feed or CR to signify the end of the rec entry of these characters (most commonly as part of a description or text encounters these line feeds, it assumes the end of the record and generate To get around this: 1. when you generate the ascii file, put a field delimiter after each colum '|' character as the delimiter. 2. in the SQL Loader control file, add the line: continueif last !='|' This will cause SQL Loader to correctly interpret data where line feeds o CRs are a legitimate part of the data.
Summary:	Stored procedure to recompile schema
Description:	EXEC DBMS_UTILITY.COMPILE_SCHEMA('schema-name');