# DTGen Tiers Demonstration

*Developed by DMSTEX (http://dmstex.com)*



## Table of Contents



## Introduction:

The exercises in this demonstration are focused on DTGen functionality that enables and enhances tiered deployment. All functionality in these exercises is available through both command line and graphical user interface (GUI) mode. For simplicity in understanding the under-lying workings of DTGen, these exercises are conducted entirely in command-line mode.

Tiered deployment as described here includes 2 forms of deployment:

- database/mid-tier server tiers
- database user/schema tiers

Multi-tiered hardware deployment is a common aspect of many systems. The database and mid-tier servers use a software deployment stragey that increases capacity and improves security. Capacity can be increased by adding servers to existing systems instead of replacing centralized servers with larger servers. Common deployment architectures used by Oracle for the Transaction Performance Processing Council's TCP-C Benchmark (http://www.tpc.org/tpcc/default.asp) were used as a model for DTGen's multi-tiered deployment in that all possible functionality is moved from the database tier to the mid-tier.

Security can be improved through the user of layered access to applications and data. Layered security can also be found in many systems. Oracle e-Business Suite layers application schema behind a common database application login. The layered security generated by DTGen was created to assist in compliance with "The Defense Information Services (DISA) Oracle Database Security Readiness Review (SRS)" ("http://iase.disa.mil/stigs/downloads/zip/unclassified_oracle10_v8r1.8_checklist_20100827.zip") Specific references in the "U_DB_oracle10_v8r1.8_Checklist_20100827.pdf" file include:

- V0005683 - Application object owner accounts should be disabled when not performing

installation or maintenance actions.
- V0015613 - Each database user, application or process should have an individually assigned account.
- V0015629 - Application users privileges should be restricted to assignment using application user roles.
- V0003847 - Database application user accounts should be denied storage usage for object creation within the database.

The exercises in this demonstration do not entirely conform to the DISA Oracle Database SRS because they use only 1 database and there is only 1 configuration for the application. Full compliance is obtained with multiple databases in that the same shema name is be duplicated between database and mid-tier servers.

The "basic" and "asof" demonstrations should be reviwed before running these exercises. Serveral concepts introduced in those exercises are not explained here. Exercise #1 in this demonstration is similar to Exercise #1 in the asof demonstration. Additionally, the "DML & API Calls" and "Other Object Location" diagrams in the "DTGen_Notes.pdf" document in the "docs" directory provide a graphical layout of the multi-tier deployments created by DTGen.

This demonstration requires the creation of a database link, which uses Oracle's loopback to the same database. Configuration of this loopback is a normal part of the installation of an Oracle database. If this loopback is not working or has been disabled, these exercises will return errors.

The exercises in this directory are numbered and must be executed in sequential order. The demonstration users must be created with the "create_demo_users.sql" script in the parent directory before the first exercise is run. The demonstration users must be dropped with the "drop_demo_users.sql" script before the "create_demo_users.sql" script can be re-run. These exercises also assume that the default username/password (dtgen/dtgen) is still in use for the generator. Names and passwords are set in the "vars.sql" script and can be modified, if necessary. Also, the DTGen database objects must be installed in the database and the DTGen must be ready to generate code.

# Exercise #1: Simple Mid-Tier

## Command Line:

```
sqlplus /nolog @e1
```

*Exercise #1 modifies the database. The "drop_demo_users.sql" and "create_demo_users.sql" scripts must be used to reset the database before re-running this exercise.*

Based on the demobld.sql script, this exercise implements the EMP and DEPT tables using DTGen. The script for this exercise performs the following functions:

1. Removes any old DEMO1 Items from DTGEN
2. Creates new DEMO1 Items in DTGEN
3. Generates the DEMO1 Application in DTGEN
4. Creates the "install_db.sql" and "install_mt.sql" scripts
5. Runs the "install_db.sql" script

6. Loads Data and Shows EXPLAIN PLAN
7. Runs the "install_mt.sql" script
8. Shows EXPLAIN PLAN and Reports Data

Steps 1-3 are captured in the "e1.LST" file. Following is a example of e1.LST.

```
Login to dtgen
Connected.
Remove old DEMO3 Schema from DTGEN
create a DEMO3 Schema in DTGEN
Generate Demo3 Application
Capture SQL Scripts
```

Step 4 is captured in the "install_db.sql" and "install_mt.sql" files. These files have a combined size of about 216 kbytes and over 8,000 lines. Due to their size, they are not listed here. They contain all the code generated by DTGen for this exercise.

Steps 5 though 8 are captured in the "install.LST" file. Step 5 is the execution of the install_db.sql script.

```
Login to dtgen_db_demo
Connected.

FILE_NAME
-----------------
 -) create_glob

FILE_NAME
----------------
 -) create_ods

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
------------------
 -) create_integ

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
-----------------
 -) create_oltp

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
---------------
 -) create_aa

TABLE_NAME
--------------
```

```
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
-----------------
 -) create_mods
```

The above listing represents a successful installation of the simulated database tier in the "dtgen_db_demo" schema. The application is small in that it only has 2 tables. The data is loaded silently (Step 6). An explain plan for a simple query is run to demonstrate the query runs locally.

```
SQL>
SQL> explain plan set statement_id = 'D3_E1_Q1'
  2    into plan_table for select * from emp where empno = 7900;
SQL>
SQL> select plan_table_output from table (
  2    dbms_xplan.display('PLAN_TABLE', 'D3_E1_Q1') );

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------
Plan hash value: 750405628
-----------------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |   146 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |   146 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | EMP_NK |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO"=7900)
```

The above explain plan shows that the query of EMP has no REMOTE operations. This query was run locally. Step 7 is the execution of the install_mt.sql script below.

```
Login to dtgen_mt_demo
Connected.

FILE_NAME
-----------------
 -) create_gdst

FILE_NAME
-----------------
 -) create_dist

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
-----------------
 -) create_oltp

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***

FILE_NAME
```

```
-----------------
 -) create_mods

FILE_NAME
--------------------
 -) create_gdst_sec

FILE_NAME
--------------------
 -) create_dist_sec

FILE_NAME
--------------------
 -) create_oltp_sec

FILE_NAME
--------------------
 -) create_mods_sec
```

The above listing represents a successful installation of the simulated mid-tier in the "dtgen_mt_demo". A database link is used to access the data in the tables at the simulated database tier ("dtgen_db_demo" schema). An explain plan for the same simple query below (step 8). It demonstrates the query is actually run remotely.

```
SQL> explain plan set statement_id = 'D3_E1_Q2'
  2    into plan_table for select * from emp where empno = 7900;
SQL>
SQL> select plan_table_output from table (
  2    dbms_xplan.display('PLAN_TABLE', 'D3_E1_Q2') );

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
Plan hash value: 750405628
-----------------------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)| Time     |Inst   |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT REMOTE     |        |     1 |   146 |     2   (0)| 00:00:01 |       |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |   146 |     2   (0)| 00:00:01 |   XE  |
|*  2 |   INDEX UNIQUE SCAN         | EMP_NK |     1 |       |     1   (0)| 00:00:01 |   XE  |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("A1"."EMPNO"=7900)
Note
-----
   - fully remote statement
```

The explain plan above is identical to the previous explain plan, except for the note showing that it is a "fully remote statement". This confirms that the simulated tiers are working. The queries below also confirm that the data is accessible from the simulated mid-tier.

```
SQL>
SQL> select deptno, id, dname, loc, aud_beg_usr, aud_beg_dtm
  2    from dept_act order by deptno, id;

DEPTNO  ID DNAME      LOC      AUD_BEG_ AUD_BEG_D
------ --- ---------- -------- -------- ---------
    10   1 ACCOUNTING NEW YORK Dataload 01-NOV-80
    20   2 RESEARCH   DALLAS   Dataload 01-NOV-80
    30   3 SALES      CHICAGO  THOMPSON 17-AUG-82
    40   4 OPERATIONS BOSTON   JAMES    12-FEB-82

SQL>
SQL> select deptno, dept_id, dname, loc, aud_beg_usr,
  2         aud_beg_dtm, aud_end_usr, aud_end_dtm
  3    from dept_aud
  4   order by deptno, dept_id, aud_beg_dtm;

DEPTNO DEPT_ID DNAME      LOC      AUD_BEG_ AUD_BEG_D AUD_END_ AUD_END_D
------ ------- ---------- -------- -------- --------- -------- ---------
```

```
    20       3 SALES      ST LOUIS Dataload 01-NOV-80 THOMPSON 17-AUG-82
    40       4 OPERATIONS BUFFALO  Dataload 01-NOV-80 JAMES    12-FEB-82

SQL>
SQL> select empno, id, ename, job, mgr_emp_nk1, hiredate,
  2         sal, dept_nk1, aud_beg_usr, aud_beg_dtm
  3   from  emp_act
  4   order by empno, id;

EMPNO  ID ENAME    JOB        MGR_ HIREDATE    SAL  DEPT_ AUD_BEG_ AUD_BEG_D
----- --- -------- --------- ----- --------- ----- ------ -------- ---------
 7369   3 SMITH    CLERK      7902 17-DEC-80   800     20 SMITH    26-FEB-83
 7499   4 ALLEN    SALESMAN   7698 20-FEB-81  1600     30 THOMPSON 12-MAY-81
 7521   5 WARD     SALESMAN   7698 22-FEB-81  1250     30 THOMPSON 14-MAY-81
 7566   6 JONES    MANAGER    7839 02-APR-81  2975     20 SMITH    29-NOV-81
 7654  14 MARTIN   SALESMAN   7698 28-SEP-81  1250     30 SMITH    26-SEP-81
 7698   8 BLAKE    MANAGER    7839 01-MAY-81  2850     30 SMITH    30-NOV-81
 7782   9 CLARK    MANAGER    7839 09-JUN-81  2450     10 SMITH    26-NOV-81
 7788  20 SCOTT    ANALYST    7566 09-DEC-82  3000     20 JAMES    11-DEC-82
 7839  15 KING     PRESIDENT       17-NOV-81  5000     10 SMITH    18-NOV-81
 7844  13 TURNER   SALESMAN   7698 08-SEP-81  1500     30 SMITH    06-SEP-81
 7876  21 ADAMS    CLERK      7788 12-JAN-83  1100     20 SMITH    12-JAN-83
 7900  17 JAMES    CLERK      7698 03-DEC-81   950     30 SMITH    05-DEC-81
 7902  18 FORD     ANALYST    7566 03-DEC-81  3000     20 SMITH    01-DEC-81
 7934  19 MILLER   CLERK      7782 23-JAN-82  1300     10 JAMES    21-JAN-82

SQL>
SQL> select empno, emp_id, ename, job, mgr_emp_id,
  2         aud_beg_usr, aud_beg_dtm, aud_end_usr, aud_end_dtm
  3   from  emp_hist
  4   order by empno, emp_id, aud_beg_dtm;

EMPNO EMP_ID ENAME    JOB        MGR_ AUD_BEG_ AUD_BEG_D AUD_END_ AUD_END_D
----- ------ -------- --------- ----- -------- --------- -------- ---------
 7301      1 ELLISON  PRESIDENT       DAVIS    04-NOV-80 THOMPSON 28-JUN-81
 7344      2 DAVIS    CLERK         1 DAVIS    14-NOV-80 THOMPSON 25-JUN-81
 7344      2 DAVIS    CLERK        11 THOMPSON 25-JUN-81 THOMPSON 20-AUG-81
 7344      2 DAVIS    CLERK        12 THOMPSON 20-AUG-81 SMITH    29-NOV-81
 7344      2 DAVIS    CLERK        15 SMITH    29-NOV-81 SMITH    06-DEC-81
 7369      3 THOMPSON CLERK         1 DAVIS    15-DEC-80 THOMPSON 25-JUN-81
 7369      3 THOMPSON CLERK        11 THOMPSON 25-JUN-81 SMITH    21-AUG-81
 7369      3 SMITH    CLERK        12 SMITH    21-AUG-81 SMITH    01-DEC-81
 7369      3 SMITH    CLERK        15 SMITH    01-DEC-81 SMITH    26-FEB-83
 7499      4 ALLEN    SALESMAN      1 THOMPSON 17-FEB-81 THOMPSON 12-MAY-81
 7521      5 WARD     SALESMAN      1 THOMPSON 24-FEB-81 THOMPSON 14-MAY-81
 7566      6 JONES    MANAGER       1 THOMPSON 03-APR-81 THOMPSON 24-JUN-81
 7566      6 JONES    MANAGER      11 THOMPSON 24-JUN-81 SMITH    22-AUG-81
 7566      6 JONES    MANAGER      12 SMITH    22-AUG-81 SMITH    29-NOV-81
 7654      7 MARTIN   SALESMAN      1 THOMPSON 16-APR-81 THOMPSON 13-MAY-81
 7698      8 BLAKE    MANAGER       1 THOMPSON 02-MAY-81 THOMPSON 24-JUN-81
 7698      8 BLAKE    MANAGER      11 THOMPSON 24-JUN-81 THOMPSON 19-AUG-81
 7698      8 BLAKE    MANAGER      12 THOMPSON 19-AUG-81 SMITH    30-NOV-81
 7782      9 CLARK    MANAGER       1 THOMPSON 07-JUN-81 THOMPSON 23-JUN-81
 7782      9 CLARK    MANAGER      11 THOMPSON 23-JUN-81 SMITH    23-AUG-81
 7782      9 CLARK    MANAGER      12 SMITH    23-AUG-81 SMITH    26-NOV-81
 7788     10 SCOTT    ANALYST       6 THOMPSON 10-JUN-81 JAMES    09-MAR-82
 7839     11 KING     PRESIDENT       THOMPSON 18-JUN-81 SMITH    30-AUG-81
 7840     12 LANE     PRESIDENT       THOMPSON 12-AUG-81 SMITH    29-NOV-81
 7876     16 ADAMS    CLERK         6 SMITH    20-NOV-81 JAMES    13-JUN-82
```

With the completion of exercise 1, a new application was defined in DTGen, generated, and loaded into a single database using 2 different schema names, which correcly simulates a simple mid-tier.

# Exercise #2: Materialized Views

## Command Line:

```
sqlplus /nolog @e2
```

*Exercise #2 does not modify the database. This exercise can be repeated without problem.*

In the exercise #2, a basic generation and installation of a simple mid-tier was completed. Part of that installation was a materialized view for the DEPT table on the simulated mid-tier.

```
Login to dtgen
Connected.

SQL> select seq, name, type, mv_refresh_hr from tables_act
  2   where applications_nk1 = 'DEMO3' order by seq;

       SEQ NAME            TYP MV_REFRESH_HR
---------- --------------- --- -------------
        10 dept            LOG            24
        20 emp             EFF
```

The results above show the DTGen configuration to setup a materialized view. The value in the MV_REFRESH_HR field signals the generator to make this table a materialized view and identifies the refresh period in hours. The materialized view on the simulated mid-tier will be refreshed daily.

```
Login to dtgen_mt_demo
Connected.

SQL>
SQL> select mview_name, last_refresh_date from user_mviews;

MVIEW_NAME                     LAST_REFR
------------------------------ ---------
DEPT                           23-APR-12
```

The results above confirm that a materialized view exists in the simulated mid-tier schema. Below is an explain plan on a query of that view.

```
SQL>
SQL> explain plan set statement_id = 'D3_E2_Q1'
  2    into plan_table for select * from dept where deptno = 40;

SQL>
SQL> select plan_table_output from table (
  2    dbms_xplan.display('PLAN_TABLE', 'D3_E2_Q1') );

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
Plan hash value: 1799509361
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     1 |    73 |     1   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS BY INDEX ROWID| DEPT    |     1 |    73 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN           | DEPT_NK |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=40)
```

Step 2 in the PLAN_TABLE_OUTPUT above show a MAT_VIEW, or mataerialized view, access operation was performed. The data returned for this query is local to the simulated mid-tier, contrasted with the EMP query in exercise #1 ther returned data from the remote simulated database tier. Also note that DTGen automatically generated the index DEPT_NK on the materialized view.

# Exercise #3: User Security

## Command Line:

```
sqlplus /nolog @e3
```

*Exercise #3 modifies the database. The "drop_demo_users.sql", "create_demo_users.sql", and "e1.sql" scripts must be used to reset the database before re-running this exercise.*

In this exercise, the user security layer is demonstrated. The security layer is implmented by seperating the user login from the schema that contains the datbase objects. Synonyms are used to easily access the database objects in the other schema. Below is a query showing the DTGen configuration that identifies the name of the schema with that contains the database objects

```
Login to dtgen
Connected.

SQL> select name, db_schema from applications where abbr = 'DEMO3';


NAME                           DB_SCHEMA
------------------------------ ------------------------------
Dtgen Tiers Demonstration      DTGEN_MT_DEMO
```

From the configuration listed above an "install_usr.sql" script was created in Exercise #1 that defines the required synonyms. This exercise continues with the login and installation of the user synonyms in the install_usr.sql script.

```
Login to dtgen_usr_demo
Connected.

FILE_NAME
-----------------
 -) create_usyn

TABLE_NAME
--------------
***  dept  ***

TABLE_NAME
-------------
***  emp  ***
```

Ths listing above is a successful installation of the user synonyms. An explain plan for the same simple query run in Exercise #1 demonstrates the query is actually run remotely.

```
SQL> explain plan set statement_id = 'D3_E3_Q1'
  2     into plan_table for select * from emp where empno = 7900;

Explained.

SQL> select plan_table_output from table (
  2    dbms_xplan.display('PLAN_TABLE', 'D3_E3_Q1') );

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 750405628
-------------------------------------------------------------------------------
|Id |Operation                    |Name  |Rows |Bytes|Cost(%CPU)|Time    |Inst|
-------------------------------------------------------------------------------
|  0|SELECT STATEMENT REMOTE      |      |    1|  146|    2  (0)|00:00:01|    |
|  1| TABLE ACCESS BY INDEX ROWID|EMP   |    1|  146|    2  (0)|00:00:01| XE |
|* 2|  INDEX UNIQUE SCAN          |EMP_NK|    1|     |    1  (0)|00:00:01| XE |
-------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("A1"."EMPNO"=7900)

Note
-----
   - fully remote statement
```

The explain plan above is identical to the previous explain plan, except for the note showing that it is a "fully remote statement". This confirms that the simulated tiers are working. A query of the ASOF data is run to demonstrate user access to the database objects in the other schema.

```
SQL>
SQL> execute glob.set_asof_dtm(to_timestamp('1981-06-01', 'YYYY-MM-DD'))
SQL>
SQL> select empno, ename, job, mgr_emp_nk1, hiredate, sal, deptno, dname, loc
  2   from  emp_asof e, dept_asof d where e.dept_id = d.id
  3   order by empno;

EMPNO ENAME    JOB       MGR_EMP_NK1 HIREDATE    SAL DEPTNO DNAME      LOC
----- -------- --------- ----------- --------- ----- ------ ---------- --------
 7301 ELLISON  PRESIDENT             02-NOV-80  4000     10 ACCOUNTING NEW YORK
 7344 DAVIS    CLERK            7301 16-NOV-80  1400     10 ACCOUNTING NEW YORK
 7369 THOMPSON CLERK            7301 17-DEC-80   800     10 ACCOUNTING NEW YORK
 7499 ALLEN    SALESMAN         7698 20-FEB-81  1600     20 SALES      ST LOUIS
 7521 WARD     SALESMAN         7698 22-FEB-81  1250     20 SALES      ST LOUIS
 7566 JONES    MANAGER          7301 02-APR-81  2975     20 RESEARCH   DALLAS
 7698 BLAKE    MANAGER          7301 01-MAY-81  2850     20 SALES      ST LOUIS
```

With the execution of the above query, data was successfully retrieved from a remote database link (database/mid-tier server tiers) via a synonym pointed to another schema (database user/schema tiers).

# Exercise #4: Global Locks

## Command Line:

```
sqlplus /nolog @e4
```

*Exercise #5 does not modify the database. This exercise can be repeated without problem.*

In an application that spans multiple nodes, it is sometimes necessary to single-thread a process or ensure sole access to an global resource. This functionality is performed by the GLOB.package.

```
Login to dtgen_usr_demo
Connected.

SQL>
SQL> describe glob

FUNCTION GET_ASOF_DTM RETURNS TIMESTAMP WITH TIME ZONE

FUNCTION GET_DB_CONSTRAINTS RETURNS BOOLEAN

FUNCTION GET_DTM RETURNS TIMESTAMP WITH LOCAL TIME ZONE

FUNCTION GET_FOLD_STRINGS RETURNS BOOLEAN

FUNCTION RELEASE_LOCK RETURNS VARCHAR2

FUNCTION REQUEST_LOCK RETURNS VARCHAR2
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 LOCKNAME_IN                    VARCHAR2                IN
```

```
PROCEDURE SET_ASOF_DTM
 Argument Name                  Type                      In/Out Default?
 ------------------------------ ------------------------ ------ --------
 ASOF_DTM_IN                    TIMESTAMP WITH TIME ZONE IN

PROCEDURE SET_DB_CONSTRAINTS
 Argument Name                  Type                      In/Out Default?
 ------------------------------ ------------------------ ------ --------
 BOOL_IN                        BOOLEAN                  IN

PROCEDURE SET_FOLD_STRINGS
 Argument Name                  Type                      In/Out Default?
 ------------------------------ ------------------------ ------ --------
 BOOL_IN                        BOOLEAN                  IN

PROCEDURE UPD_EARLY_EFF
 Argument Name                  Type                      In/Out Default?
 ------------------------------ ------------------------ ------ --------
 TABLE_NAME                     VARCHAR2                 IN
 EFF_DTM_IN                     TIMESTAMP                IN
```

The ouptut above from e4.LST has a describe command that shows the names and parameters of all the available functions and procedures in the GLOB package. The next portion of the e4.sql script will request a lock.

```
SQL>
SQL> execute util.set_usr('DEMO3');
SQL> execute dbms_output.put_line(glob.request_lock('DEMO3'));
SUCCESS
SQL>
SQL> set echo off

Open another window to run "e4a.sql"
Ex: sqlplus /nolog @e4a

Press Enter when prompted from other window
```

In the output above, the GLOB.REQUEST_LOCK function returned the status of SUCCESS. The session at the database tier (simulated) is now holding the lock for the session at the mid-tier (simulated). A pause in the e4.sql script allows another mid-tier (and database tier) session to be started. The same lock is requested from the other session using the e4a.sql script.

```
Login to dtgen_usr_demo
Connected.

SQL> execute util.set_usr('DEMO3');
SQL> execute dbms_output.put_line(glob.request_lock('DEMO3', 2));
TIMEOUT

SQL> -- Press Enter on the other window to continue
SQL> execute dbms_output.put_line(glob.request_lock('DEMO3'));
```

In the output above from e4a.LST, the first lock request failed with a timeout of 2 seconds. The second request halts the execution of the script, waiting to aquire the lock without a timeout value set. Moving back to the initial window, ENTER is pressed to continue the execution of the e4.sql script.

```
SQL> execute dbms_output.put_line(glob.request_lock('DEMO2'));
RELEASE ONLY

SQL> execute dbms_output.put_line(glob.request_lock('DEMO3'));
SUCCESS

SQL> execute dbms_output.put_line(glob.release_lock);
SUCCESS

SQL> execute dbms_output.put_line(glob.release_lock);
SUCCESS
```

A request for a lock, other than the one already held, is executed and fails. The RELEASE ONLY return confirms that the first lock must be released before another lock can be aquired. A request for the the lock that is already held succeeds. Both lock release calls succeed, the first because the lock is actually released, and the second because not lock is currently held by the session. As soon as the lock was release, execution continued on e4a.sql in the other window.

```
SUCCESS

SQL> execute dbms_output.put_line(glob.release_lock);
SUCCESS
```

The lock that was held by the initial window is successfully requested and released. This locking mechanism works amoung all processes running on any database tier node (RAC) or any mid-tier node.