DTGen Demonstration #1

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

Table of Contents

Introduction:

The set of exercises in this demonstration is focused on basic DTGen functionality. All functionality in this demonstration is available through both command line and graphical user interface (GUI) forms. For simplicity in understanding the under-lying workings of DTGen, this demonstration is conducted entirely by command-line. (No GUIs will be injured during the execution of this demonstration.)

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

Exercise #1: Basic Generation

Command Line:

```
sqlplus /nolog @e1
```

Based on Oracle's 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" script
5. Runs the "install_db.sql" script
6. Loads 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 DEMO Schema from DTGEN
create a DEMO Schema in DTGEN
Generate Demo1 Application
Capture install_db.sql Script
```

Step 4 is captured in the "install_db.sql" file. This file is about 78 kbytes and has over 3,000 lines. Due to its size, it is not listed here. It contains all the code generated by DTGen for this application.

Steps 5 and 6 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_mods
```

The above listing represents a successful installation of the application generated by DTGen. This application is small in that it only has 2 tables, 1 tier (the database tier), and no user schema.

The DEPT table is silently loaded with data. A query of column comments on the DEPT table from

the data dictionary help identify what each column's data represents. Following the column comments is a report of all the data in the DEPT table (active view) for the selected columns.

```
COLUMN_NAME         COMMENTS
------------------- ------------------------------------------------------------
DEPTNO              Department Number
DNAME               Name of the Department
LOC                 Location for the Department

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

The EMP table is also silently loaded with data. The same queries of column comments and data on the EMP table (active view) are shown.

```
COLUMN_NAME         COMMENTS
------------------- ------------------------------------------------------------
EMPNO               Employee Number
ENAME               Employee Name
JOB                 Job Title
MGR_EMP_NK1         EMP Natural Key Value 1: Employee Number
HIREDATE            Date the Employee was hired
SAL                 Employee's Salary
DEPT_NK1            DEPT Natural Key Value 1: Department Number


     EMPNO ENAME            JOB       MGR_EMP_NK1 HIREDATE     SAL   DEPT_NK1
---------- ---------------- --------- ----------- --------- ------ ----------
      7782 CLARK            MANAGER          7839 09-JUN-81   2450         10
      7698 BLAKE            MANAGER          7839 01-MAY-81   2850         30
      7566 JONES            MANAGER          7839 02-APR-81   2975         20
      7902 FORD             ANALYST          7566 03-DEC-81   3000         20
      7788 SCOTT            ANALYST          7566 09-DEC-82   3000         20
      7876 ADAMS            CLERK            7788 12-JAN-83   1100         20
      7369 SMITH            CLERK            7902 17-DEC-80    800         20
      7900 JAMES            CLERK            7698 03-DEC-81    950         30
      7844 TURNER           SALESMAN         7698 08-SEP-81   1500         30
      7654 MARTIN           SALESMAN         7698 28-SEP-81   1250         30
      7521 WARD             SALESMAN         7698 22-FEB-81   1250         30
      7499 ALLEN            SALESMAN         7698 20-FEB-81   1600         30
      7934 MILLER           CLERK            7782 23-JAN-82   1300         10
      7839 KING             PRESIDENT              17-NOV-81   5000         10
```

With the completion of exercise 1, a new application was defined in DTGen, generated, and loaded into the database.

Exercise #2: Sequences and Surrogate Primary Keys

Command Line:

```
sqlplus /nolog @e2
```

In the exercise #1, a basic generation was completed. The results of that generation were loaded into a new schema. This exercise, and the following exercises, will examine more closely what was generated. In this exercise, the use of sequences and surrogate keys are reviewed.

Exercise #2 has 4 queries. The first query shows the sequences that were generated by DTGen for each of the tables DEPT and EMP.

```
SEQUENCE_NAME
------------------------------
DEPT_SEQ
```

```
EMP_SEQ

TABLE_NAME           CONSTRAINT_NAME      COLUMN_NAME            POSITION
-------------------- -------------------- -------------------- ----------
DEPT                 DEPT_PK              ID                            1
EMP                  EMP_PK               ID                            1
```

Every table that is defined in DTGen gets a sequence. That sequence is used to generate a surrogate key for each record in the table. The surrogate key is the primary key for the record. The surrogate keys for the DEPT and EMP tables can be seen in the results of the second 2 queries:

```
        ID     DEPTNO DNAME          LOC
---------- ---------- -------------- -------------
         1         10 ACCOUNTING     NEW YORK
         2         20 RESEARCH       DALLAS
         3         30 SALES          CHICAGO
         4         40 OPERATIONS     BOSTON

        ID      EMPNO ENAME            MGR_EMP_ID    DEPT_ID
---------- ---------- ---------------- ---------- ----------
         1       7839 KING                                 1
         2       7566 JONES                     1          2
         3       7788 SCOTT                     2          2
         4       7876 ADAMS                     3          2
         5       7902 FORD                      2          2
         6       7369 SMITH                     5          2
         7       7698 BLAKE                     1          3
         8       7499 ALLEN                     7          3
         9       7521 WARD                      7          3
        10       7654 MARTIN                    7          3
        11       7844 TURNER                    7          3
        12       7900 JAMES                     7          3
        13       7782 CLARK                     1          1
        14       7934 MILLER                   13          1
```

Notice that "DEPT_ID" is a foreign key to "ID" in the DEPT table. Also, "MGR_EMP_ID" is a foriegn key to "ID" in the EMP table. These surrogate keys are used to maintain referential integrity across foreign keys.

Exercise #3: Indexed Foreign Keys and Natural Keys

Command Line:

```
sqlplus /nolog @e3
```

In this exercise, indexes on foreign keys and natural keys are explored. Following is a query of the DTGen setup used to generate this application

```
Login to dtgen
Connected.

COLUMN_NAME          COMMENTS
-------------------- ------------------------------------------------------------
TABLES_NK2           TABLES Natural Key Value 2: Abbreviation for this table
NAME                 Name of this column
SEQ                  Sequence number for this column
NK                   Natural key sequence number for this column.  Implies this
                     column requires data (not null).
FK_PREFIX            Foreign key prefix for multiple foreign keys to the same
                     table
FK_TABLES_NK2        TABLES Natural Key Value 2: Abbreviation for this table
TYPE                 Type for this column
LEN                  The total number of significant decimal digits in a number,
                     or the length of a string, or the number of digits for
                     fractional seconds in a timestamp

TABLES_NK2           NAME                   SEQ   NK TYPE                    LEN
-------------------- -------------- ---------- ---- --------------- ----------
DEPT                 deptno                 10    1 NUMBER                   2
```

```
EMP                empno                    10    1 NUMBER                    4

TABLES_NK2         NAME                  SEQ FK_PREFIX  FK_TABLES_NK2
------------------ --------------- ---------- ---------- ------------------
EMP                dept_id                80            DEPT
EMP                mgr_emp_id             40 mgr_       EMP
```

Foreign keys and natural keys are defined in the DTGen TAB_COLS_ACT view. The output shown above gives a description of the TAB_COLS_ACT columns and reports the selected data that creates the foreign and natural keys in this application.

The exercise 3 script then logs into the application to query the data dictionary.

```
Login to dtgen_db_demo
Connected.

CONSTRAINT_NAME TABLE_NAME      COLUMN_NAME       POSITION INDEX_NAME
--------------- --------------- --------------- ---------- ---------------
DEPT_NK         DEPT            DEPTNO                   1 DEPT_NK
EMP_FK1         EMP             MGR_EMP_ID               1 EMP_FX1
EMP_FK2         EMP             DEPT_ID                  1 EMP_FX2
EMP_NK          EMP             EMPNO                    1 EMP_NK
```

There is a natural key on each of the 2 tables, which is confirmed by constraints "DEPT_NK" and "EMP_NK". Also, the EMP table has 2 foreign keys, which are confirmed by constraints "EMP_FK1" and "EMP_FK2". Note that all natural keys and foreign keys have indexes. These indexes are automatically generated by DTGen.

Exercise #4: Natural Key Updatable Views

Command Line:

```
sqlplus /nolog @e4
```

Each table defined in DTGen is generated with a corresponding "active view". The DEPT and EMP tables have an active view called "DEPT_ACT" and "EMP_ACT", respectively. In most cases, these views should be used for all DML (Data Manipulation Language - insert, update, and delete) instead of the tables. The active views include a feature that allows foreign key data to be referenced using the natural key of the foriegn key table. (In reality, all foreign keys reference the surrogate/primary key from the foreign key table. The active view automatically translates the natural key.)

In the original scott/tiger dmobld.sql script, DEPTNO was used as a foreign key from the EMP table to the DEPT table. In exercise #1, DEPTNO was identified as the natural key for the DEPT table. DTGen then produced the EMP_ACT active view with the foreign surrogate key D_DEPT_ID and the foreign natural key D_DEPT_NK1.

This exercise performs inserts and updates on the EMP_ACT active view using both foreign surrogate keys and foreign natural keys for the department. 2 queries will confirm that the OPERATIONS department has no employees.

```
SQL> select deptno, dname, loc from dept_act
  2   where dname = 'OPERATIONS';

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON

1 row selected.
```

```
SQL>
SQL> select empno, ename, job, mgr_emp_nk1, hiredate,
  2   sal, dept_nk1 from emp_act
  3   where dept_nk1 = 40;

no rows selected
```

2 insert statements will add 2 new employees to the OPERATIONS department. The first insert uses a foreign surrogate key for the department. The second insert uses a foreign natural key for the department.

```
SQL> -- Add a new manager to the Operations Department
SQL> --   using the surrogate key for the department
SQL> --   in the active view
SQL> insert into emp_act (empno, ename, job,
  2     mgr_emp_nk1, hiredate, sal, dept_id)
  3   values (8156, 'MCMURRY', 'MANAGER',
  4     7839, sysdate, 2975, 4);

1 row created.

SQL>
SQL> -- Add a new analyst to the Operations Department
SQL> --   using the natural key for the department
SQL> --   in the active view
SQL> insert into emp_act (empno, ename, job,
  2     mgr_emp_nk1, hiredate, sal, dept_nk1)
  3   values (8157, 'WALKER', 'ANALYST',
  4     8156, sysdate, 3000, 40);

1 row created.
```

2 update statements will add transfer 2 existing employees to the OPERATIONS department. The first update uses a foreign surrogate key for the department. The second update uses a foreign natural key for the department.

```
SQL> -- Transfer an analyst to the Operations Department
SQL> -- using the surrogate key for the department
SQL> -- in the active view
SQL> update emp_act
  2    set  dept_id    = 4
  3        ,mgr_emp_nk1 = 8156
  4   where empno = 7788;

1 row updated.

SQL>
SQL> -- Transfer a clerk to the Operations Department
SQL> --   using the natural key for the department
SQL> --   in the active view
SQL> update emp_act
  2    set  dept_nk1   = 40
  3        ,mgr_emp_nk1 = 8156
  4   where empno = 7900;

1 row updated.
```

Finally, a query of the employees table shows the 4 employees in the OPERATIONS department.

```
SQL> select empno, ename, job, mgr_emp_nk1, hiredate,
  2   sal, dept_nk1 from emp_act
  3   where dept_nk1 = 40;

     EMPNO ENAME            JOB       MGR_EMP_NK1 HIREDATE     SAL   DEPT_NK1
---------- ---------------- --------- ----------- --------- ------ ----------
      8156 MCMURRY          MANAGER          7839 12-APR-12   2975         40
      8157 WALKER           ANALYST          8156 12-APR-12   3000         40
      7900 JAMES            CLERK            8156 03-DEC-81    950         40
      7788 SCOTT            ANALYST          8156 09-DEC-82   3000         40

4 rows selected.
```

Exercise #5: Full Path Hierarchy Data

Command Line:

```
sqlplus /nolog @e5
```

The EMP table has a self-referencing foreign key. It is the relationship between employees and managers. Since managers are also employees, they have managers as well, with the exception of the PRESIDENT. This self-referencing foreign key produces as hierarchy of relationships. In the case of the EMP table, that hierarchy basically shows who works for who. Every employee in the EMP table is in the management hierarchy that starts with the PRESIDENT.

When a self-referencing foreign key is setup in DTGen, hierarchial path functions are created to work with the hierarchy implied by the foreign key. Those functions are also included in the active view. One set of hierarchial path functions are based on surrogate keys.

```
COLUMN_NAME         COMMENTS
------------------- ------------------------------------------------------------
ID                  Surrogate Primary Key for this table
ENAME               Employee Name
MGR_EMP_ID          Surrogate Key of Employee's Manager
MGR_ID_PATH         Path of ancestor IDs hierarchy for this record

4 rows selected.

SQL>
SQL> select substr(mgr_id_path,1,20) mgr_id_path,
  2         mgr_emp_id, id, ename,
  3         substr(emp_dml.get_mgr_id_path(id),1,20) get_mgr_id_path
  4   from  emp_act where ename = 'SMITH';

MGR_ID_PATH          MGR_EMP_ID         ID ENAME            GET_MGR_ID_PATH
-------------------- ---------- ---------- ---------------- --------------------
1:2:5                         5          6 SMITH            1:2:5

1 row selected.
```

In this example, SMITH is ID 6. SMITH works for ID 5, which is the surrogate key for SMITH's manager. ID 5 works for ID 2, and ID 2 works for ID 1. The GET_M_ID_PATH function that is used by the active view to produce the M_ID_PATH is shown in the last column and is part of the EMP_DML package.

Another set of hierarchical path functions are based on natural keys.

```
COLUMN_NAME         COMMENTS
------------------- ------------------------------------------------------------
EMPNO               Employee Number
ENAME               Employee Name
MGR_NK_PATH         Path of ancestor Natural Key Sets hierarchy for this record
MGR_EMP_NK1         EMP Natural Key Value 1: Employee Number

4 rows selected.

SQL>
SQL> select mgr_nk_path, mgr_emp_nk1, empno, ename,
  2         emp_dml.get_mgr_nk_path(emp_dml.get_id(empno)) get_mgr_nk_path
  3   from  emp_act where ename = 'SMITH';

MGR_NK_PATH          MGR_EMP_NK1   EMPNO ENAME            GET_MGR_NK_PATH
-------------------- ----------- ------- ---------------- --------------------
7839:7566:7902              7902    7369 SMITH            7839:7566:7902

1 row selected.
```

In this example, SMITH is EMPNO 7369. SMITH works for EMPNO 7902, which is the natural key for SMITH's manager. EMPNO 7902 works for EMPNO 7566, and EMPNO 7566 works for

EMPNO 7839. The GET_M_NK_PATH function that is used by the active view to produce the M_ID_PATH is shown in the last column and is part of the EML_DML package.

The path delimiter can also be modified as required, The constant PATH_SEP is defined in the UTIL package specification. This change can be permenantly done in the UTIL package for the entire application. A complete restart of the application will be necessary after making this change.

Since the hierarchy functions are used in the view, searching the view on these functions can be quite slow if there are a large number of rows in the table. Other filters should be used as much as possible to help limit searching through the heiararchical paths.

Exercise #6: Enforced Descrete Domains

Command Line:

```
sqlplus /nolog @e6
```

Unlike the original scott/tiger dmobld.sql from Oracle, this demo includes built in domain checking on the JOB column in the EMP table. The configuration of DTGen included a domain specification for all possible company jobs. Unlike a foreign key table, a domain is embedded into the error checking of the application and is very difficult to change. It should only be used for value sets that are not likely to change, or in applications that can easliy be re-generated with new domain values.

```
SQL>
SQL> -- Attempt to alter SMITH's job incorrectly
SQL> update emp_act
  2    set  job = 'FIREMAN'
  3   where ename = 'SMITH';
update emp_act
*
ERROR at line 1:
ORA-20005: emp_tab.check_rec(): job must be one of (
"PRESIDENT", "MANAGER", "ANALYST", "SALESMAN", "CLERK").
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 70
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 159
ORA-06512: at "DTGEN_DB_DEMO.EMP_VIEW", line 190
ORA-06512: at "DTGEN_DB_DEMO.EMP_IOU", line 24
ORA-04088: error during execution of trigger 'DTGEN_DB_DEMO.EMP_IOU'
```

Since FIREMAN is not a correct job name, the application produced an error. This error was generated by DTGen. It identifies the list of correct job names as part of the error. One reason small value sets make better domain candidates is because all correct values for the domain will be returned in this error message.

This error message also gives a good view of the call stack for integrity processing. The EMP_IOU (instead of update) trigger on the EMP_ACT active view called the EMP_VIEW package, which called the EMP_TAB package, which used the CHECK_REC function to enforce the domain integrity. The EMP_VIEW package is also known as a view package. The EMP_TAB package is also know as a table package. DTGen geneates a view package and a table package for each table. Most of the integrity checking on table data occurs in the CHECK_REC function in the table packages.

Exercise #7: Enforced Case Folding

Command Line:

```
sqlplus /nolog @e7
```

Enforced case folding has 2 options. The option is selected based on the PL/SQL boolean variable FOLD_STRINGS in the GLOB package. This exercise will work with SMITH.

```
SQL>
SQL> select empno, ename
  2   from  emp
  3   where empno = 7369;

     EMPNO ENAME
---------- ----------------
      7369 SMITH

1 row selected.
```

When FOLD_STRINGS set to TRUE, any case problems are repaired.

```
glob.fold_strings := TRUE;

SQL>
SQL> -- Change SMITH's name to mixed-case
SQL> update emp_act
  2    set  ename = 'Smith'
  3   where empno = 7369;

1 row updated.

SQL>
SQL> select empno, ename
  2   from  emp
  3   where empno = 7369;

     EMPNO ENAME
---------- ----------------
      7369 SMITH

1 row selected.
```

When FOLD_STRINGS set to FALSE, any case problems result in an exception being raised.

```
glob.fold_strings := FALSE;

SQL>
SQL> -- Change SMITH's name to mixed-case
SQL> update emp_act
  2    set  ename = 'Smith'
  3   where empno = 7369;
update emp_act
       *
ERROR at line 1:
ORA-20003: emp_tab.check_rec(): ename must be upp
er case.
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 33
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 165
ORA-06512: at "DTGEN_DB_DEMO.EMP_VIEW", line 190
ORA-06512: at "DTGEN_DB_DEMO.EMP_IOU", line 24
ORA-04088: error during execution of trigger 'DTGEN_DB_DEMO.EMP_IOU'
```

The current default setting for FOLD_STRINGS is TRUE.

Exercise #8: Full Procedural APIs

Command Line:

```
sqlplus /nolog @e8
```

Enforced case folding has 2 options. The option is selected based on the PL/SQL boolean variable

Exercise #9: Custom Check Constraints