DTGen Basics Demonstration

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

Table of Contents

Introduction:

The exercises in this demonstration are focused on basic DTGen functionality. 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.

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: Basic Generation

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" 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 DEMO1 Schema from DTGEN
create a DEMO1 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
```

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

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
```

Exercise #3 does not modify the database. This exercise can be repeated without problem.

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
```

Exercise #4 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.

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 demobld.sql script, DEPTNO was an implied foreign key from the EMP table to the DEPT table. (No foreign keys were actually created in that script.) 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 DEPT_ID and the foreign natural key 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
```

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

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 mgr_id_path, mgr_emp_id, id, ename,
  2        emp_dml.get_mgr_id_path(id) get_mgr_id_path
  3   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
```

Exercise #6 does not modify the database. This exercise can be repeated without problem.

Unlike the original demobld.sql, this demonstration 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
```

Exercise #7 does not modify the database. This exercise can be repeated without problem.

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 default setting for FOLD_STRINGS is TRUE.

Exercise #8: Full Procedural APIs

Command Line:

```
sqlplus /nolog @e8
```

Exercise #8 modifies the database. The "drop_demo_users.sql", "create_demo_users.sql", "e1.sql", and "e4.sql" scripts must be used to reset the database before re-running this exercise.

The EMP_DML package contains a set of APIs tailored to the EMP table. Each table generated by DTGen will have a matching "DML" package that contains APIs for use in the applications. In exercise #5, the "get_mgr_id_path", "get_mgr_nk_path", and "get_id" APIs were briefly introduced. SMITH will again be the target of this exercise.

```
SQL>
SQL> select ename, id, empno, mgr_id_path, mgr_nk_path
  2    from emp_act where empno = 7369;

ENAME              ID  EMPNO MGR_ID_PATH          MGR_NK_PATH
---------------- ---- ------ -------------------- ------------------------------
SMITH               6   7369 1:2:5                7839:7566:7902

1 row selected.

SQL>
SQL> select emp_dml.get_id(7369) id, emp_dml.get_nk(6) empno,
  2         emp_dml.get_mgr_id_path(6) mgr_id_path,
  3         emp_dml.get_mgr_nk_path(6) mgr_nk_path
  4    from dual;
```

```
  ID  EMPNO MGR_ID_PATH          MGR_NK_PATH
---- ------ -------------------- ------------------------------
   6   7369 1:2:5                7839:7566:7902

1 row selected.
```

As seen in this part of the exercise, the "get_mgr_id_path", "get_mgr_nk_path", and "get_id" functions are part of the EMP_DML package and return the same values as the active view. An additional function "get_nk" will return the natural key for an EMP.ID. Compound natural keys, or natural keys that are made of multiple columns, are returned as a single value. Each column is separated by the NK_SEP constant in the UTIL package.

Two additional function are available in the EMP_DML package

```
SQL>
SQL> select emp_dml.get_mgr_id_by_id_path('1:2:5') id
  2   from  dual;

  ID
----
   5

1 row selected.

SQL>
SQL> select emp_dml.get_mgr_id_by_nk_path('7839:7566:7902') id
  2   from  dual;

  ID
----
   5

1 row selected.
```

These functions (get_mgr_id_by_id_path and get_mgr_id_by_nk_path) will return the id for any full path in a heirarchy. The functions are named "mgr" because it is the manager to employee relationship that yeilds the hierarchy. Without the manager listed in the EMP table, there would be no hierarchy.

The EMP_DML package has insert, update, and delete procedures available

```
SQL>
SQL> declare
  2     id          emp_act.id%TYPE;
  3     empno       emp_act.empno%TYPE        := 21;
  4     ename       emp_act.ename%TYPE        := 'BOGUS';
  5     job         emp_act.job%TYPE          := 'CLERK';
  6     mgr_emp_id  emp_act.mgr_emp_id%TYPE;
  7     hiredate    emp_act.hiredate%TYPE     := sysdate;
  8     sal         emp_act.sal%TYPE          := 1;
  9     comm        emp_act.comm%TYPE;
 10     dept_id     emp_act.dept_id%TYPE      := 4;
 11  begin
 12     dbms_output.enable;
 13     dbms_output.put_line('id before emp_dml.ins is "' || id || '"');
 14     emp_dml.ins
 15        (n_id          => id
 16        ,n_empno       => empno
 17        ,n_ename       => ename
 18        ,n_job         => job
 19        ,n_mgr_emp_id  => mgr_emp_id
 20        ,n_hiredate    => hiredate
 21        ,n_sal         => sal
 22        ,n_comm        => comm
 23        ,n_dept_id     => dept_id
 24        );
 25     dbms_output.put_line('id after emp_dml.ins is "' || id || '"');
 26  end;
```

```
 27  /
id before emp_dml.ins is ""
id after emp_dml.ins is "17"

PL/SQL procedure successfully completed.

SQL>
SQL> select id, empno, ename, job, hiredate, sal, dept_id
  2    from emp_act where ename = 'BOGUS';

  ID  EMPNO ENAME            JOB       HIREDATE         SAL    DEPT_ID
---- ------ ---------------- --------- --------- ---------- ----------
  17     21 BOGUS            CLERK     13-APR-12          1          4

1 row selected.
```

Several important aspects of the DML API need to be noted, starting with the definition of the insert API from the EMP_DML package specification.

```
   procedure ins
      (n_id  in out  NUMBER
      ,n_empno  in out  NUMBER
      ,n_ename  in out  VARCHAR2
      ,n_job  in out  VARCHAR2
      ,n_mgr_emp_id  in out  NUMBER
      ,n_mgr_id_path_in  in  VARCHAR2  default null
      ,n_mgr_nk_path_in  in  VARCHAR2  default null
      ,n_mgr_emp_nk1_in  in  NUMBER  default null
      ,n_hiredate  in out  DATE
      ,n_sal  in out  NUMBER
      ,n_comm  in out  NUMBER
      ,n_dept_id  in out  NUMBER
      ,n_dept_nk1_in  in  NUMBER  default null
      );
```

Each "in out" parameter must have a buffer (variables will work) to receive data returned from the API. This is particulalry useful for preventing additional round-trips to the database in order to check the data that was inserted. For instance, a sequence was used to generate a new surrogate key for the EMP table during the insert. That new surrogate key was returned from the call and can be seen in the DBMS_OUTPUT '*id after emp_dml.ins is "17"*'.

```
SQL>
SQL> begin
  2     dbms_output.enable;
  3     glob.fold_strings := TRUE;
  4     for buff in (
  5        select * from emp
  6         where ename = 'BOGUS' )
  7     loop
  8        buff.job   := 'SALESMAN';
  9        buff.ename := 'Bogus';
 10        dbms_output.put_line('buff.ename before emp_dml.up is "' ||
 11                             buff.ename || '"');
 12        emp_dml.upd
 13           (o_id_in       => buff.id
 14           ,n_empno       => buff.empno
 15           ,n_ename       => buff.ename
 16           ,n_job         => buff.job
 17           ,n_mgr_emp_id  => buff.mgr_emp_id
 18           ,n_hiredate    => buff.hiredate
 19           ,n_sal         => buff.sal
 20           ,n_comm        => buff.comm
 21           ,n_dept_id     => buff.dept_id
 22           );
 23        dbms_output.put_line('buff.ename after emp_dml.up is "' ||
 24                             buff.ename || '"');
 25     end loop;
 26  end;
 27  /
buff.ename before emp_dml.up is "Bogus"
buff.ename after emp_dml.up is "BOGUS"
```

```
PL/SQL procedure successfully completed.

SQL>
SQL> select id, empno, ename, job, hiredate, sal, dept_id
  2    from emp_act where ename = 'BOGUS';

  ID  EMPNO ENAME           JOB       HIREDATE         SAL    DEPT_ID
---- ------ ---------------- --------- --------- ---------- ----------
  17     21 BOGUS           SALESMAN  13-APR-12          1          4

1 row selected.
```

In this example, the update API is used to change the job for the new employee. Notice that the GLOB.FOLD_STRINGS is set to TRUE (See exercise #7) and the correct ename was returned from the update API. These APIs will work with the natural keys as well.

```
SQL>
SQL> declare
  2     rec  emp%ROWTYPE;
  3  begin
  4     dbms_output.enable;
  5     rec.id            := null;
  6     rec.empno         := 22;
  7     rec.ename         := 'BOGUS';
  8     rec.job           := 'CLERK';
  9     rec.mgr_emp_id    := null;
 10     rec.hiredate      := sysdate;
 11     rec.sal           := 1;
 12     rec.comm          := null;
 13     rec.dept_id       := null;
 14     dbms_output.put_line('rec.mgr_emp_id before emp_dml.ins is "' ||
 15                          rec.mgr_emp_id || '"');
 16     dbms_output.put_line('rec.dept_id before emp_dml.ins is "' ||
 17                          rec.dept_id || '"');
 18     emp_dml.ins
 19        (n_id             => rec.id
 20        ,n_empno          => rec.empno
 21        ,n_ename          => rec.ename
 22        ,n_job            => rec.job
 23        ,n_mgr_emp_id     => rec.mgr_emp_id
 24        ,n_mgr_nk_path_in  => '7839:7566:7902'
 25        ,n_hiredate       => rec.hiredate
 26        ,n_sal            => rec.sal
 27        ,n_comm           => rec.comm
 28        ,n_dept_id        => rec.dept_id
 29        ,n_dept_nk1_in    => 40
 30        );
 31     dbms_output.put_line('rec.mgr_emp_id after emp_dml.ins is "' ||
 32                          rec.mgr_emp_id || '"');
 33     dbms_output.put_line('rec.dept_id after emp_dml.ins is "' ||
 34                          rec.dept_id || '"');
 35  end;
 36  /
rec.mgr_emp_id before emp_dml.ins is ""
rec.dept_id before emp_dml.ins is ""
rec.mgr_emp_id after emp_dml.ins is "5"
rec.dept_id after emp_dml.ins is "4"

PL/SQL procedure successfully completed.

SQL>
SQL> select id, empno, ename, job, sal, mgr_emp_nk1, dept_nk1
  2    from emp_act where ename = 'BOGUS';

  ID  EMPNO ENAME           JOB              SAL MGR_EMP_NK1   DEPT_NK1
---- ------ ---------------- --------- ---------- ----------- ----------
  17     21 BOGUS           SALESMAN           1                     40
  18     22 BOGUS           CLERK              1        7902         40

2 rows selected.
```

In the above example, another BOGUS employee is created in the EMP table. In this case, the

DEPTNO natural key of 40 was used to identify the OPERATIONS department for this employee. Notice that the DEPT_ID surrogate foreign key was returned by the insert API. "Full path" values can also be used to identify a foreign key record.

```
SQL>
SQL> declare
  2     type empcurtype is ref cursor return emp%ROWTYPE;
  3     c1   empcurtype;
  4     buff emp%rowtype;
  5  begin
  6     dbms_output.enable;
  7     glob.fold_strings := TRUE;
  8     open c1 for
  9        select * from emp
 10         where empno = 21;
 11     fetch c1 into buff;
 12     close c1;
 13     dbms_output.put_line('buff.mgr_emp_id after emp_dml.ins is "' ||
 14                          buff.mgr_emp_id || '"');
 15     emp_dml.upd
 16        (o_id_in             => buff.id
 17        ,n_empno             => buff.empno
 18        ,n_ename             => buff.ename
 19        ,n_job               => buff.job
 20        ,n_mgr_emp_id        => buff.mgr_emp_id
 21        ,n_mgr_nk_path_in    => '7839:7566:7902'
 22        ,n_hiredate          => buff.hiredate
 23        ,n_sal               => buff.sal
 24        ,n_comm              => buff.comm
 25        ,n_dept_id           => buff.dept_id
 26        ,nkdata_provided_in  => 'T'
 27        );
 28     dbms_output.put_line('buff.mgr_emp_id after emp_dml.ins is "' ||
 29                          buff.mgr_emp_id || '"');
 30  end;
 31  /
buff.mgr_emp_id after emp_dml.ins is ""
buff.mgr_emp_id after emp_dml.ins is "5"

PL/SQL procedure successfully completed.

SQL>
SQL> select id, empno, ename, job, sal, mgr_emp_nk1, dept_nk1
  2    from emp_act where ename = 'BOGUS';

  ID  EMPNO ENAME            JOB              SAL MGR_EMP_NK1   DEPT_NK1
---- ------ ---------------- --------- ---------- ----------- ----------
  17     21 BOGUS            SALESMAN           1        7902         40
  18     22 BOGUS            CLERK              1        7902         40

2 rows selected.
```

In the above example, the natural key full path value was used to identify the MANAGER. The surrogate key for MGR_EMP_ID was returned by the update API. Notice that the "NKDATA_PROVIDED_IN" parameter was used to signal the update API that natural keys were being used. In this example, when NKDATA_PROVIDED_IN is set to T (any string starting with a t or y), MGR_EMP_IS is determined using value changes in the following order:

1. MGR_EMP_ID
2. MGR_EMP_ID_PATH
3. MGR_EMP_NK_PATH
4. MGR_EMP_NK1

In the example above, there was no change to MGR_EMP_ID, or MGR_EMP_ID_PATH (both were null and unchanged). When the change was found in NK_EMP_NK_PATH, the new value for MGR_EMP_ID was determined from it. Also, if NKDATA_PROVIDED_IN is not set to T (any

string starting with a t or y), any values in the natrual key fields N_MGR_EMP_ID_PATH_IN, N_MGR_EMP_NK_PATH_IN, N_MGR_EMP_NK1_IN, and N_DEPT_NK1_IN field would have been ignored.

Exercise #9: Custom Check Constraints

Command Line:

```
sqlplus /nolog @e9
```

Exercise #9 does not modify the database. This exercise can be repeated without problem.

During the setup of DTGen for this application, a custom check constraint was configured on the EMP table in DTGen. Below is a login and query of the DTGen configuration for that check constraint.

```
Login to dtgen
Connected.

COLUMN_NAME          COMMENTS
-------------------- ------------------------------------------------------------
TABLES_NK2           TABLES Natural Key Value 2: Abbreviation for this table
SEQ                  Sequence number of this check constraint
TEXT                 Execution (PL/SQL) text for this check constraint
DESCRIPTION          Description of this check constraint

TABLES_NK2 SEQ TEXT                              DESCRIPTION
---------- --- ------------------------------- --------------------------------
EMP         10 (comm is null) or (comm is not   Only SALESMAN can be on
               null and job = 'SALESMAN')       commission
```

From the description above, an error should be generated, if a COMM is given to SMITH, who is a CLERK.

```
Login to dtgen_db_demo
Connected.

glob.db_constraints := TRUE;

SQL>
SQL> select empno, ename, job, mgr_emp_nk1, sal, comm, dept_nk1
  2    from emp_act where ename = 'SMITH';

     EMPNO ENAME            JOB       MGR_EMP_NK1    SAL   COMM   DEPT_NK1
---------- ---------------- --------- ----------- ------ ------ ----------
      7369 SMITH            CLERK            7902    800                20

1 row selected.

SQL>
SQL> update emp_act
  2    set  comm  = 1000
  3   where empno = 7369;
update emp_act
*
ERROR at line 1:
ORA-20006: emp_tab.check_rec(): Only SALESMAN can be
on commission
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 82
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 165
ORA-06512: at "DTGEN_DB_DEMO.EMP_BU", line 10
ORA-04088: error during execution of trigger 'DTGEN_DB_DEMO.EMP_BU'
ORA-06512: at "DTGEN_DB_DEMO.EMP_VIEW", line 210
ORA-06512: at "DTGEN_DB_DEMO.EMP_IOU", line 24
ORA-04088: error during execution of trigger 'DTGEN_DB_DEMO.EMP_IOU'
```

In the above example, the connection was changed from DTGen the application. DB_CONSTRAINTS was changed to TRUE from the default of FALSE. The query confirms that SMTIH is a CLERK and currently has no COMM. The update statement fails with an ORA-20006 and a large stack trace. While this is the expected behavior, it should be noted that the ORA-20006 error text is in fairly plain language. Also, note the stack trace:

1. The EMP_ACT before update view trigger called
2. the EMP_VIEW pachage, which ran
3. an EMP table update, which ran
4. the EMP before update table trigger, which called
5. the EMP_TAB package.

Contrast that stack trace to the next example.

```
glob.db_constraints := FALSE;

SQL>
SQL> update emp_act
  2    set  comm  = 1000
  3   where empno = 7369;
update emp_act
*
ERROR at line 1:
ORA-20006: emp_tab.check_rec(): Only SALESMAN can be
on commission
ORA-06512: at "DTGEN_DB_DEMO.EMP_TAB", line 82
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 above example produced the same error, but with a much smaller stack trace.

1. The EMP_ACT before update view trigger called
2. the EMP_VIEW pachage, which ran
3. the EMP_TAB package.

The default value of FALSE for DB_CONSTRAINTS executes less code, especially if the table triggers and table constraints have been disabled. A possible downside is that this integrity error would have been missed if the same update statement was run against the EMP table instead of the EMP_ACT view.

```
SQL>
SQL> update emp
  2    set  comm  = 1000
  3   where empno = 7369;
update emp
*
ERROR at line 1:
ORA-02290: check constraint (DTGEN_DB_DEMO.EMP_CK10) violated
```

By using the EMP_TAB package to run the check constraints, an easier to understand error message can be produced. Also, care must be taken to ensure table inserts do not occur if DB_CONSTRAINTS is set to the defaul value of FALSE, and table triggers/constraints have been removed for efficiency.