

# **AUTOMATIC TEST DATA GENERATION FOR AN 11G DATABASE - A CASE STUDY & DEMO**

*Chandu Patel, Deloitte Consulting LLP*

## **INTRODUCTION**

One of the challenges developers often face during the development and testing phase is to get the test data timely populated in the development and test or UAT databases. In the absence of data conversion programs, it becomes tedious and manual effort to come up with test data so that various components of an application can be tested. Often the test data provided or created are not good enough to test end conditions. It is highly desirable to have large quantity data with end conditions and with proper referential integrity.

This paper discusses a case study where a technique is used data in about 132 relational tables across three schemas were automatically generated. The technique or utility to generate data takes configurable parameters such as number of rows to be generated and populated into each table. It discusses what all dictionary views are used to generate data. It also describes how to generate data for the columns accepting only valid list of values.

The utility is capable of populating generate millions of rows making it usable to populate OLTP as well as OLAP and Data Warehouse databases. Its modular designs make it easy to extend and customize.

## **ENVIRONMENT**

The environment for the case study was Oracle Database 11g Enterprise Edition Release 11.1.0.6.0. Data was generated in 132 tables in and referencing tables across three schemas in the same database. The script generator is primarily SQL and PL/SQL, and the generated script is in PL/SQL.

## **ASSUMPTIONS**

- The script generator has been tested on Oracle Database 11gR1 as well as 10gR2
- Tables have single column surrogate PKs and values generated from dedicated sequence objects strictly following a naming convention
- Handles table hierarchy up to 10 levels of depth
- XML column data types not supported

## **PREREQUISITES/PRIVILEGES**

- A user with DBA access
- SELECT privilege should be directly granted to this DBA user by SYS on
  - SYS.DBA\_CONSTRAINTS
  - SYS.DBA\_CONS\_COLUMNS
- SELECT privileges by table owners directly on
  - All the tables for which data needs to be generated
  - All the tables that are referenced by the these tables

## **SETUP STEPS**

- Create a user with DBA access
- Install supporting objects
  - 2 tables
  - 9 functions
- Populate Reference Code Mapping

## **SUPPORT FUNCTIONS**

All the support functions are organized into a package. Each of the functions carries out an atomic task. These functions use dictionary views DBA\_CONSTRAINTS and DBA\_CONS\_COLUMNS as the main views to query data from.

- has\_common\_pkfk\_col (p\_owner, p\_tab): Finds out if a table has any column that is part of both Primary Key (PK) as well as a Foreign Key (FK).
- is\_pk\_col (p\_owner , p\_tab , p\_col): Finds out if a column is part of the PK;
- is\_fk\_col (p\_owner , p\_tab , p\_col): Finds out if a column is part of a FK.
- has\_child (p\_owner , p\_tab ): Finds out if a table has any child table(s);
- get\_pk\_count (p\_owner , p\_tab ): Finds out count of constituent PK columns.
- get\_pk\_val\_from\_parent (p\_child\_owner , p\_child\_tab , p\_child\_col ): Obtains a value from the parent table for the given child column.
- get\_random\_date (p\_begin\_julian\_day, p\_end\_julian\_day): Generates a random date between the specified begin and end dates.
- is\_ref\_code\_col (p\_owner , p\_tab , p\_col ): Finds out if the given column should have a valid reference code value.
- get\_ref\_code\_val (p\_owner , p\_tab , p\_col ): Obtains one of the valid reference codes at random.

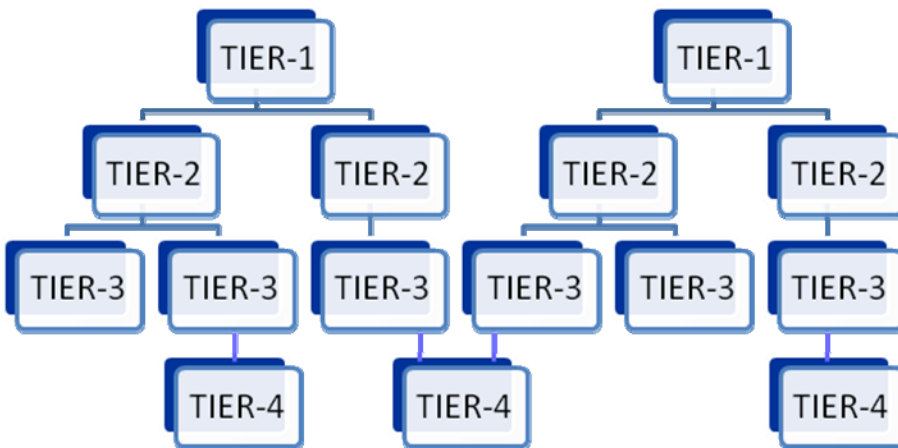
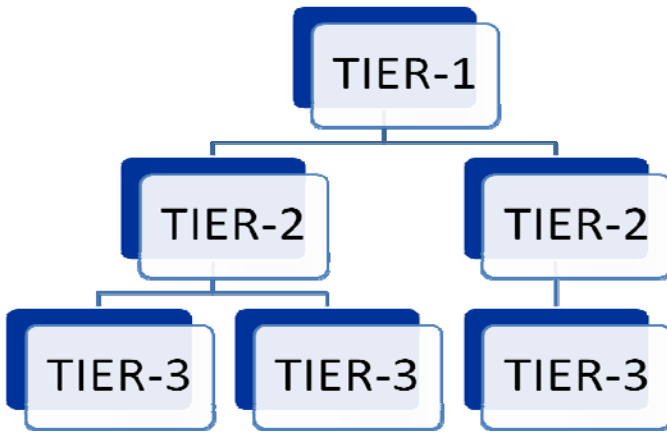
## **CORE LOGIC**

The overall logic involves 3 steps:

- Generate list of tables by level in parent-child hierarchy
- Generate Declaration section of PL/SQL script
- Generate INSERT statements
  - Parent tables first, followed by child tables

## **GENERATE PARENT-CHILD HIERARCHY**

In a relational database, tables in a schema are related via foreign keys (referential integrity constraints). Tables can be laid out in parent-child hierarchies, as illustrated in the following diagram. Tables at the top of the hierarchies, that is having no parents, are said to be at tier 1, T1 tables. The tables which are direct child of T1 tables are tier 2, T2 tables. The tables that are direct child of T2 are tier 3, T3 tables, and henceforth. A given table may be part of more than one hierarchy and may be at different tier for each of the hierarchies it is part of.



However, there are two approaches the parent-child table hierarchies can be generated.

**APPROACH 1: ROOT TABLE APPROACH**

While generating the list of parent-child hierarchy, this approach allows to generate list only for the tables that are child or descendent of a given parent table. However, it should be ensured that (a) if any outlier tables referenced by any of these descendent tables are already populated (should have at least one record) or (b) all the references should be within this hierarchy. In essence, there will be only one hierarchy, as at the top there would be only one table.

## APPROACH 2: TIERED TABLES APPROACH

In this approach all the tables in a schema are considered for the parent-child hierarchy generation. In essence, there will be multiple hierarchies. At the top of each hierarchy, will be a T1 table, that is a table referencing none. It is only referenced by one or more child or descendent tables.

## GENERATE DECLARATION SECTION

A typical PL/SQL program has a declaration section, the main body and an optional exception handling section. The program generates the declaration section, which basically declares one local variable per table to hold the value of auto-generated PK column value to be used subsequently as foreign key in a child table.

```
DECLARE
    variable_name DATA TYPE := NULL;
    ...
    ...
BEGIN
    SQL Statements
EXCEPTION
    WHEN exception_name
        Handling statements;
END;
/
```

## GENERATE INSERT STATEMENTS

- Generate INSERT statements
  - Parent tables first, followed by child tables
  - T1, T2, T3, ..., T10 in that order
  - PK value from parent is used in child table maintaining referential integrity
- What about the tables with multiple FKs?
  - All are considered
- What about the tables referencing lower tables in the hierarchy
  - Such tables are skipped during first iteration, and considered during subsequent iteration after data for all child tables are generated

The following table depicts what values are generated for each data type.

Data Type	Value
DATE	Within the specified range
NUMBER	nnnnn...nnn (full column length)

VARCHAR2	n-testdata-testdata-testdata...
CLOB	N-clobdata-clobdata-clobdata

Besides this, if for a specifically named columns, some special values are needed to be generated, that can be done as well.

Column	Value
DTE_CHANGE_LAST	SYSTIMESTAMP (current date and time)
DTE_CRTD	SYSTIMESTAMP (current date and time)
IDN_USER_CHANGE_LAST	AUTO_DATAGEN (literal)
CDE_%	REFERENCE CODE (value to be picked from a valid list of values)
NAM_%	N-testName-testName...

## GENERATING REFERENCE CODES

Often values for code columns are supposed to be from a list of valid values only. For example, a column for gender type can have values M or F. Values for a column for State Code can have one of 50 valid list of state codes (NJ, NY, PA, MA, MD, ..., etc.). This utility needs to know 2 additional information to support automatic generation of a random valid value.

Firstly, the mapping between a code column and corresponding code type. Secondly, the list of valid values for a given code type.

The Ref Code Column mapping table - T\_REF\_CODE\_MAP layout should be as follows:

<i>OWNER</i>	<i>TABLE_NAME</i>	<i>COLUMN_NAME</i>	<i>CODE_TYPE</i>
CP	T_EXP	CDE_STATE	STATE_CODE
CP	T_EMP	CDE_GENDER	GENDER_CODE
SAM	T_INCOME	IS_PROCESSED	YES_NO_CODE
SAM	T_PAYMENT	CDE_EXPENSE	EXPENSE_CODE
SAM	T_PAYMENT	CDE_METHOD	METHOD_CODE

The Ref Code table - T\_CODES is typically as follows.

<i>CODE_TYPE</i>	<i>CODE</i>
GENDER_CODE	M
GENDER_CODE	F
YES_NO_CODE	Y
YES_NO_CODE	N
STATE_CODE	AL
STATE_CODE	AZ
STATE_CODE	MA
STATE_CODE	NJ
EXPENSE_CODE	01

EXPENSE_CODE	02
EXPENSE_CODE	03
EXPENSE_CODE	04
METHOD_CODE	CC
METHOD_CODE	CHECK

### **ENHANCEMENT OPTIONS**

This utility can be extended and/or enhanced to meet specific requirement and inter-relationships of tables within a schema.

- If the depth of table is more than 10, the code can be enhanced to handle more levels.
- If a database has its own way to store reference codes, then rather than creating a mapping table, a view can be created on top of these tables. This view would essentially have 4 columns – OWNER, TABLE\_NAME, COLUMN\_NAME and CODE\_TYPE.
- If the CLOB columns need to generate more than 50 characters, just change the CLOB\_LENGTH constant in the Declaration section of the program.

### **SUMMARY/TAKEAWAYS**

- There are ways to reduce developer's pain
- It is possible to generate data automatically
- A Development DBA can help!