Database – SQL*Loader

Generating High-Quality Test Data with Oracle SQL*Loader

By Blake Couch

Editor's Note: *Ask any Oracle® database administrator or developer the number one task they dislike, and two answers come immediately to mind: documentation and creating test data. In this white paper, second time ORAtips contributor Blake Couch cannot help you with documentation, but he has developed a strategy to take the fear out of coming up with useful test data. And not just some test data, but multi-million rows of dummy information. Filled with best practices, sample scripts, tips, and tricks, this ORAtip is a winner!*

Introduction

Most of us have used the Oracle SQL*Loader utility at one time or another to populate tables with data from an outside source. SQL*Loader is a very powerful tool for loading large quantities of data from CSV or fixed-width ASCII files into an Oracle database in a hurry, but did you know that with SQL*Loader and a few common functions like MOD and DECODE, you can generate thousands or even millions of rows of good test data with just a few lines of code?

Be a Volume Dealer

Perhaps you are planning to test some procedure or application for high volume performance. You want a couple million rows to test against, and content is not important.

If your table is defined as:

```
CREATE TABLE FOO_TABLE
(
  FOO_ID              NUMBER(9)  NOT NULL,
  FOO_DATE            DATE,
  FOO_USER            VARCHAR2(100),
  FOO_MESSAGE         VARCHAR2(4000)
)
```

you could create a control file for SQL*Loader that looks something like

```
LOAD
TRUNCATE
INTO TABLE FOO_TABLE
(
  FOO_ID              CONSTANT 9999,
  FOO_DATE            SYSDATE,
  FOO_USER            CONSTANT 'fake user name',
  FOO_MESSAGE         CONSTANT 'fake message'
)
```

then save it as mycontrol.ctl and read it into SQL*Loader using the command line:

```
sqlldr myuser/mypassword@mydb control=mycontrol.ctl load=2000000
```

The load parameter tells SQL*Loader to create 2 million rows in mydb, while the control file tells it to put them in FOO_TABLE after first truncating (to load data into an empty table, change TRUNCATE to INSERT, or use APPEND to add to existing data), and to populate all 2 million

Copyright © 2006 by Klee Associates, Inc.

rows with the same 4 values. Note: SYSDATE is called for every insert, so that value will change as your load proceeds.

Creating that many rows could take a while, therefore a direct path load is preferred. Add

```
direct=true
```

to the command line. It will speed up the process considerably. Keep in mind that doing a direct path load causes Oracle to disable automatically all triggers and constraints on the table in question before loading the data, and then re-enable them when the load is done. If you load data that violates your constraints, you will need to rethink the load process and start over.

One constraint easily violated is the primary key constraint created for FOO_ID. You did declare a primary key for that table, didn't you? If FOO_ID is your primary key, you need SQL*Loader to insert unique values in that column. Therefore change the line in your control file for FOO_ID to

```
FOO_ID                  SEQUENCE(1000),
```

which will cause SQL*Loader to create its own sequence starting with 1000 and insert the values from that sequence into FOO_ID in your table.

Variety Is the Spice of Life

Not content with putting the same values in every row you create? Not a problem! SQL*Loader allows you to specify expressions as values instead of constants, and within those expressions, you can use a subset of Oracle's built-in functions.

You are probably thinking, it would be nice to use the DBMS_RANDOM package to generate randomized values for your testing. So sorry! There is no way to call the INITIALIZE procedure from SQL*Loader, and remember, you only have access to a *subset* of Oracle's functions in your expressions.

The fact is, random values may not be what you really want in the first place. Chances are you have a limited set of values for certain columns that you need to see in your tables. You will need to make sure your application works properly with known data. In other words, you have *test cases* you need to build and evaluate. SQL*Loader enables you to build those test cases in your data.

Let's say you have five known values for FOO_USER to use in your test data: BillG, LewisL, GeorgeB, FreddieM, and OprahW. If you are content having equal numbers of each in your table, you could replace your FOO_USER line in your control file with

```
FOO_USER     EXPRESSION “DECODE(MOD(:FOO_ID,5), 0, 'BillG', 1, 'LewisL', 2,
'GeorgeB', 3, 'FreddieM', 4, 'OprahW')”
```

:FOO_ID is a reference to your sequence column. The sequence value for the row currently inserting will be plugged in there. MOD(:FOO_ID,5) gives you the remainder when you do an integer divide on FOO_ID by 5. Your five possible values are 0, 1, 2, 3, and 4, and since your FOO_ID values are integers in a straight sequence, you will get equal or nearly equal numbers of the five different values over a large data set. The DECODE allows you to replace the five integers with five values of your choosing.

You are probably starting to see the power that sequence column gives you – but what if you do

not have a column like that in your table? Perhaps you are thinking “hmmm, wonder if I could use ROWNUM?” Sorry again! SQL*Loader does have a value called RECNUM that can be used by itself as a column value, but it cannot be used in an expression. I recommend you simply add a numeric column to your table, temporarily, if you do not already have one you can use in this fashion, and then drop the column when you are done with your load process. You can use RECNUM in place of SEQUENCE, but you will still need a column in your table to hold that value. SEQUENCE also gives you some additional options. Specify an increment value, with the default being one, and direct SQL*Loader to start the sequence with either the current maximum value in the column or with the row count of the table, plus the increment value.

Now back to our example: if you want 7 different values in the table, do a MOD 7. If you have 23, do a MOD 23. It’s that simple. But what if you do not want an equal number of each value in your data? Add 2 to the MOD value and repeat a couple of your desired values, as shown here:

```
FOO_USER      EXPRESSION “DECODE(MOD(:FOO_ID,7), 0, 'BillG', 1, 'LewisL', 2,
'GeorgeB', 3, 'FreddieM', 4, 'OprahW', 5, 'LewisL', 6, 'FreddieM')”
```

and you will get an extra LewisL and FreddieM for every cycle through the values.

For some added variability, consider that you can nest function calls, making one or more of your DECODE values the result of an embedded call to DECODE or some other function.

To get some variation in date values, you can add or subtract from SYSDATE, remembering that the default unit when doing arithmetic on SYSDATE is “day.”

```
FOO_DATE      EXPRESSION "SYSDATE + MOD(:FOO_ID,18) – 9"
```

This will generate a range of date values from nine days before to nine days after today. You can add or subtract fractions of days by dividing the operand by a number at least as large as your MOD value. By tweaking this expression, you can generate a range of date values that are only minutes or even seconds apart from each other, and, of course, you can create a range that is measured in years, if you so desire.

Putting It All Together

Applying these techniques to our control file, it now becomes:

```
LOAD
TRUNCATE
INTO TABLE FOO_TABLE
(
  FOO_ID              SEQUENCE(1000),
  FOO_DATE            EXPRESSION "SYSDATE + MOD(:FOO_ID,18) – 9",
  FOO_USER            EXPRESSION “DECODE(MOD(:FOO_ID,7), 0, 'BillG', 1,
                      'LewisL', 2, 'GeorgeB', 3, 'FreddieM', 4, 'OprahW', 5,
                      'LewisL', 6, 'FreddieM')”,
  FOO_MESSAGE         CONSTANT 'fake message'
)
```

If you have loaded a few thousand rows using this control file, and later decide you want to generate a few thousand more using a different set of values, change the expressions. Set a new starting value for the sequence, and then change the TRUNCATE directive to APPEND. This will leave the existing rows intact and insert new ones.

Copyright © 2006 by Klee Associates, Inc.

Another feature of SQL*Loader that is useful is its ability to load data into more than one table at a time. You can have as many "INTO TABLE xxx (…)" sections in your control file as you have tables to populate. Simply list them one after the other in the file, as follows:

```
LOAD
TRUNCATE
INTO TABLE FOO_TABLE_1
(
  <column specifications for first table>
)
INTO TABLE FOO_TABLE_2
(
  <column specifications for second table>
)
INTO TABLE FOO_TABLE_3
(
  <column specifications for third table>
)
```

Each table loaded in this fashion can have its own independent sequence values, and, therefore, its own set of values generated by functions like the ones I described previously.

To learn more about using SQL*Loader to generate test data, review:

http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96652/ch06.htm#1008235

Conclusion

There are any number of third-party tools you can use to accomplish the often-tedious task of populating Oracle tables with test data. They all come with their own price tag and learning curve. You have an excellent tool for this task already at your disposal in SQL*Loader, which comes standard with your Oracle database investment. With a little imagination and this article as a guide, you should be able to generate tons of useful test data with ease.

About the Author

Blake Couch – Blake is a 25-year veteran of the computing wars who can be taught a new trick or two, but hasn't forgotten paper tape, punch cards, or Teletype machines. Blake resides in Colorado with his wife, daughter, and dog, Stanley Pup, who came along right after the Avalanche last captured Lord Stanley's trophy. Blake may be contacted at Blake.Couch@ERPtips.com.

Copyright © 2006 by Klee Associates, Inc.

The information on our website and in our publications is the copyrighted work of Klee Associates, Inc. and is owned by Klee Associates, Inc.

NO WARRANTY: This documentation is delivered as is, and Klee Associates, Inc. makes no warranty as to its accuracy or use. Any use of this documentation is at the risk of the user. Although we make every good faith effort to ensure accuracy, this document may include technical or other inaccuracies or typographical errors. Klee Associates, Inc. reserves the right to make changes without prior notice.

NO AFFILIATION: Klee Associates, Inc. and this publication are not affiliated with or endorsed by Oracle Corporation. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Klee Associates, Inc. is a member of the Oracle Partner Network.

All other company and product names used herein may be trademarks or registered trademarks of their respective owners.

This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips.

For training, consulting, and articles on JD Edwards or SAP, please visit our websites: www.JDEtips.com & www.SAPtips.com.