

An Oracle White Paper
September 2010

Data Transformations with Oracle Data Pump

Introduction

Database administrators (DBAs) sometimes need to modify the data being exported out of a database or imported into a database. For example, as part of an export a DBA may need to scrub sensitive data such as credit card numbers or social security numbers. Similarly, during an import, the DBA may want to convert internal identification numbers in a table to avoid collisions with existing data. Previously, doing these sorts of conversions required many additional steps to stage the data prior to modification. With Oracle Database 11g Release 2, the `REMAP_DATA` parameter for the Data Pump Export and Import commands helps to automate this process. The `REMAP_DATA` parameter allows a user-supplied PL/SQL function to modify user data as Data Pump transfers the data. This paper provides examples that demonstrate the power of the `REMAP_DATA` parameter.

Obscuring Sensitive Data

Suppose your database has a schema that consists of the following two tables:

```
CREATE TABLE HOLDER (
    NAME          VARCHAR2(100),
    CARDNO        NUMBER );
```

```
CREATE TABLE ACTIVITY (
    VENDOR        VARCHAR2(100),
    AMOUNT        NUMBER,
    CARDNO        NUMBER );
```

The table named **HOLDER** is used to describe credit card holders. The table named **ACTIVITY** is used to track credit card transactions. The **CARDNO** charges. Assume that the following sample data is inserted into these two tables:

```
-- HOLDER table:
```

NAME	CARDNO
John	1234567890123456
Dean	2345678901234561
Steve	3456789012345612

```
-- ACTIVITY table:
```

VENDOR	AMOUNT	CARDNO
Telco	\$26.43	1234567890123456
CatTV	\$87.54	1234567890123456

Grocer	\$36.28	3456789012345612
Telco	\$13.22	2345678901234561
Telco	\$37.17	3456789012345612

Suppose that a development group needs to gain access to this schema for testing the latest version of the application that works with this data. However, due to security policies, the DBA needs to scramble the data before providing a copy of it to the developers. To make the application work, the DBA needs to make sure that the scrambled numbers (1) pass validation tests that require the first and last digits to be unchanged and (2) the scrambling is consistent across both tables to maintain the referential relationships. To meet these challenges, the DBA would build the following package containing the conversion function:

```
CREATE OR REPLACE PACKAGE hidedata AUTHID CURRENT_USER AS
    FUNCTION newcc          (oldid IN  NUMBER) RETURN NUMBER;
END hidedata;
/
```

```
CREATE OR REPLACE PACKAGE BODY hidedata AS
    TYPE cc_list IS TABLE OF NUMBER
                INDEX BY VARCHAR2(16);
    cc_remaps cc_list;
    cc_seed NUMBER := 000000000000010;

    FUNCTION newcc (oldid IN  NUMBER) RETURN NUMBER IS
    BEGIN
        IF NOT cc_remaps.EXISTS(oldid) THEN
            cc_seed := cc_seed + 10;
            cc_remaps(oldid) :=
                ROUND (oldid, -15) + -- 1st digit
                cc_seed +          -- 2nd-15th digits
                MOD(oldid,10) ;    -- 16th digit
        END IF;
    END newcc;
END hidedata;
```

```

        END IF;

        RETURN cc_remaps(olddid);

    END;

END hidedata;

/

```

In the HIDEEDATA package, the CC_REMAPS structure is used to remember the previously assigned value for a credit card number that has already been seen. This allows tables across a job to be translated in a consistent fashion. The assignment of the remapped id is careful to preserve the first and last digits of the sixteen digit credit card number as is required by the application.

Now that the package for hiding the credit card numbers is in place, the DBA can perform the export and specify that the HIDEEDATA package should do the data conversions:

```

expdp myuser/pw \
tables=holder,activity \
    remap_data=holder.cardno:hidedata.newcc \
remap_data=activity.cardno:hidedata.newcc \
directory=dpump_dir dumpfile=hremp2.dmp

```

The REMAP_DATA parameter identifies the columns that need to be converted during the export. Each REMAP_DATA parameter contains two arguments that are separated by a colon. The first argument identifies a table and the column within that table that is to be remapped. It may also optionally include a schema name. The second argument specifies a PL/SQL package and the function within the package that is used to remap the column. It may also optionally include a schema name. The function must accept a single parameter whose datatype is the same as the column being converted. It must also return the same datatype. In this example, the datatype being converted is NUMBER.

As the Data Pump processes each row of the HOLDER table, the value of the CARDNO column is replaced by the result of calling the HIDEEDATA.NEWCC function using the old

CARDNO value as an input. Due to the nature of the HIDE DATA package, these translations are cached in the CC_LIST table so that the translation is preserved for identical credit card numbers. Because the same structure is used for caching values from the ACTIVITY table, the translations are consistent across the two tables. Therefore, joins between the tables on the CARDNO columns will yield the same results before and after the remapping.

After the hremap2.dmp file is imported, you can see the new data that has been substituted for the CARDNO columns in the two tables:

```
-- HOLDER table:
```

NAME	CARDNO
John	10000000000000026
Dean	20000000000000041
Steve	30000000000000032

```
-- ACTIVITY table:
```

VENDOR	AMOUNT	CARDNO
Telco	\$26.43	10000000000000026
CatTV	\$87.54	10000000000000026
Grocer	\$36.28	30000000000000032
Telco	\$13.22	20000000000000041
Telco	\$37.17	30000000000000032

In each table, the sensitive data in the CARDNO column has been modified to protect it. However, the reassignment of the values is consistent so that joins based upon the CARDNO column will produce the same results as would be seen in the original database.

Special Considerations When Using REMAP_DATA

As coded, the HIDE DATA package has several limitations. Because the CC_LIST PL/SQL table is stored in memory, there is a limit to the number of values that may be cached. If the job is stopped between the processing of the HOLDER and ACTIVITY tables, the consistency between the translations will be lost after restarting the Data Pump job because the contents of the CC_LIST is not persistent across sessions. For a similar reason, the results would be inconsistent if PARALLEL=2 were used for the job because each thread of execution would use a different copy of the CC_LIST table. The DBA could avoid all of these issues by replacing the CC_LIST PL/SQL table with a persistent database table that was updated via an autonomous transaction every time a remapping was required.

There are other restrictions for the REMAP_DATA parameter that cannot be worked around. Because this parameter alters data, it cannot be used in a Data Pump job that uses the transportable method for moving data. It also cannot be used for modifying tables that contain columns with the LONG or LONG RAW datatypes.

The function that performs the remapping is executed under the username used to run the Data Pump job. Therefore, the DBA should only use remap packages that are supplied by reliable sources; otherwise these functions could use the elevated privileges of a DBA's account to compromise security within the database.

Reassigning Unique Identifiers on Table Merges

The REMAP_DATA parameter is available during import as well. A typical usage of REMAP_DATA at import time is to reset unique identifiers when peer databases are merged into a central database. For example, suppose there is a database in New York and a database in Los Angeles. The goal is to merge the Los Angeles data into the New York instance. The personnel information for these databases is stored in the following tables:

```
CREATE TABLE EMP (
    NAME          VARCHAR2 (10) ,
    EMPNO         NUMBER );
```

```
CREATE TABLE TIMECARD (
```

```
HOURS      NUMBER,  
WEEK       DATE,  
EMPNO      NUMBER );
```

The EMP table is used to describe the employees of a company. The TIMECARD table is used to track their hours worked. The EMPNO column is used as a common key between the tables to connect employees with their timecards. The New York copy of these tables looks as follows:

```
-- Table EMP for New York
```

```
NAME      EMPNO  
-----  -  
Helen      1  
Lee        2  
Rod        3  
Stu        4
```

```
-- Table TIMECARD for New York
```

```
HOURS WEEK  EMPNO  
-----  -  
42 02-APR  1  
43 09-APR  1  
23 02-APR  2  
15 09-APR  2  
45 02-APR  3  
47 09-APR  3  
41 02-APR  4  
49 09-APR  4
```

The Los Angeles copy of these tables looks as follows:

```
-- Table EMP for Los Angeles
```

NAME	EMPNO
-----	-----
Jim	1
Mike	2
Cindy	3

```
-- Table TIMECARD for Los Angeles
```

HOURS	WEEK	EMPNO
-----	-----	-----
16	02-APR	1
22	09-APR	1
38	02-APR	2
46	09-APR	2
44	02-APR	3
48	09-APR	3

For these tables, new employee ids are allocated through the use of the LASTEMP sequence. Because employees have been assigned the same employee numbers in both databases, the numbers used by Los Angeles employees need to be remapped when the data is imported. As in the Export example, the remapping needs to be consistent across the tables being imported so that an employee's timecard data will follow him or her. The following package, named FIXUSERS defines the Import remap function to support this:

```

CREATE OR REPLACE PACKAGE fixusers AUTHID CURRENT_USER AS
    FUNCTION newempid      (oldempid IN  NUMBER) RETURN NUMBER;
END fixusers;
/

```

```

CREATE OR REPLACE PACKAGE BODY fixusers AS
    TYPE id_list IS TABLE OF NUMBER INDEX BY VARCHAR2(16);
    id_remaps id_list;
    FUNCTION newempid (oldempid IN  NUMBER)
RETURN NUMBER IS
    newid NUMBER;
    BEGIN
        IF NOT id_remaps.EXISTS(oldempid) THEN
            -- Allocate a value from sequence LASTEMP
            SELECT lastemp.NEXTVAL INTO newid FROM DUAL;
            id_remaps(oldempid) := newid;
        END IF;
        RETURN id_remaps(oldempid);
    END;
END fixusers;
/

```

This FIXUSERS package defines the function that allocates a new employee id from the LASTEMP sequence and uses it as a remap value for the imported employees. As in the export example, the values are cached into a local table so the all occurrences of a number are remapped to the same target value.

We are now ready to import the dump file containing the Los Angeles employee information (called laemps.dmp). The following command will import this dumpfile while remapping all of the imported users to new employee ids:

```
impdp user2/pw directory=dpump_dir \  
dumpfile=laemps.dmp \  
  remap_data=emp.empno:fixusers.newempid \  
  remap_data=timecard.empno:fixusers.newempid \  
  table_exists_action=append
```

After this import, the merged tables will look as follows:

```
-- Combined EMP table
```

NAME	EMPNO
Helen	1
Lee	2
Rod	3
Stu	4
Jim	5
Mike	6
Cindy	7

```
--Combined TIMECARD table
```

HOURS	WEEK	EMPNO
42	02-APR	1
43	09-APR	1
23	02-APR	2
15	09-APR	2
45	02-APR	3
47	09-APR	3
41	02-APR	4

49	09-APR	4
16	02-APR	5
22	09-APR	5
38	02-APR	6
46	09-APR	6
44	02-APR	7
48	09-APR	7

Note that the EMPNO for Jim has changed from 1 to 5 in both tables. Many of the same restrictions apply to the FIXUSERS package that applied to the HIDE DATA package in the Export example. In particular, the in-memory ID_REMAPS table may limit the size of databases that could be handled. Also, restarting the import job or using a high degree of parallelism could have unpredictable results. As mentioned for the export example, these restrictions could be relaxed if the package were rewritten to use a persistent store for caching the remap information.

Conclusion

The new REMAP_DATA parameter within the Data Pump offers a powerful tool for manipulating data values during export and import jobs. The export example in this paper showed how columns can be modified at export time to protect sensitive client information. The import example showed a way to modify data to match the requirements of a target database. These examples are only the beginning of how this feature can help the DBA in managing data that must be moved between Oracle databases. Although the REMAP_DATA parameter is a powerful tool, DBAs must be aware of the security and performance impacts when using this feature.



Data Transformations with Oracle Data Pump
September 2010
Author: Bill Fisher

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2010, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.