

SELECT[®] Online

The Journal of the International Oracle Users Group

Select Journal
Home

Regular Columns

Past Issues

General Information

IOUG Home

IOUG
Independent Oracle Users Group

Select Magazine - April 2001

Volume 7, No. 3

Changing Your Accounting Flexfield

By Vincent Giasolli

Why would you want to change a segment of your accounting flexfield? Where would you start looking for potential problems? What steps are needed to perform the upgrade? What would you test to ensure completion without complications? This article will address the approach to modifying the chart of accounts structure by taking advantage of the Oracle Financials database structure to preserve all accounting history.

There comes a time in almost every company's growth when the desire to modify the chart of accounts becomes a topic of conversation. Most of the time, the desired modification seems trivial, but the impact to an Oracle Applications implementation can be quite significant. Chart of accounts modification can range from expanding a single segment by one or two characters to redesigning the entire structure from the ground up.

Every module in the Oracle Financials suite of products is impacted by a change to the accounting flexfield. If there are any other external business-related applications, those are affected too. This article will describe a general approach including how a modification would work, mapping old values to new values, how any customizations would be impacted and the importance of testing. A technical approach will follow with information related to expanding a single segment by two characters in length. This example was selected because it is the easiest possible modification to the accounting flexfield. SQL scripts are included.

Oracle-Supported Approach

Oracle Corporation realizes that clients may wish to modify their chart of accounts. The supported upgrade procedure requires the creation of a new set of books with the new structure, closing the existing set of books and opening the new set of books with the new structure for future use. A lengthy well defined project would be required, and the resulting new set of books would not contain any historical data.

This approach does not consider any company-specific modification or custom programs that reference the chart of accounts. This additional concern may be a large part of the modification.

A Different Approach

There is a way to modify the chart of accounts without losing any history. Following the conversion, all information on your system will appear as if it had always used the new structure. Extreme care must be taken when performing this upgrade because of the impact a mistake or oversight could have on the company and any possible future audits. All old accounts must map to a new account following the modification to prevent any loss of historical data. This requires careful planning prior to the upgrade. Modification to the chart of accounts using this process is minor compared to other possible modifications to the chart of accounts.

It is relatively easy to modify all historical financial data because all stored data contains a reference or pointer (called a code combination ID, or CCID) that points to the actual chart of accounts combination. Each journal or purchase order record stored in the database contains only CCID information, which then points to another record in another table containing the actual account value. The focus of the conversion approach presented in

this paper is to leverage Oracle's architecture by changing the actual code combination record, leaving the pointer stored in all accounting records. Existing records in related systems (journal entries, purchase orders, invoices and inventory transactions) will not have to be modified since they all contain the CCID (pointer) and not the actual chart of accounts combination. This will give the users the appearance that the new account segments existed historically. Additional steps will be required to address other items, such as account defaults, the use of account ranges, AutoAccounting, Workflow, and all custom tables and programs.

Mapping Old Segment Values to New Segment Values There must be a corresponding new segment value for every old segment value being modified. This means that every value currently used in the GL account combination must have a corresponding substitute. This mapping must be complete prior to the beginning of the modification project. This mapping can be simple for many, if not all, values. Because each value must be mapped, consideration of why segments are being changed must be done in detail. If a segment is being expanded for the sake of allowing more values (you basically ran out of room), then mapping shouldn't be required. Simply add zeros to the beginning or end of a numeric field and mapping is not required.

Customizations

The modification to the chart of accounts will effect all external applications that generate or depend on a chart of accounts value. Each customization, external application or feeder system will have to be checked for this possibility. Each application that you determine is effected by this change will also have to be modified to work with the new structure.

Many companies have a web page that uses chart-of-account information. The web page is generally a complicated view of the account segment and it will require research.

Other Considerations

There will be times when a segment value used in the chart of accounts has the same set of values used in other key segments somewhere else in another Oracle Financials application. The Asset Key Flexfield and asset location flexfield are possibilities for using the same value set as the location segment of the accounting flexfield. If the segment being changed in the general ledger is the segment used in another application, the other segment will have to be modified as well. Custom programs and extensions that interact with the Oracle Financials applications may be affected by this change and must be considered as well.

Testing The modification to the chart of accounts will effect all applications. The testing of this modification must be thorough. Each Oracle Financials modules installed, as well as all external application, will be included in the test plan.

Investigation into the Impact of Custom Code

An analysis of the report must begin with a determination of the tables and programs, which must change. We will assume that all custom code has been registered correctly and exists only in the custom directory APPSX. Programs not registered within the application must be considered as part of this project, but will not be addressed directly in this paper.

Custom Tables

Custom tables should be owned by your custom applications user. For the purposes of this article, the custom applications user will be identified as the user APPSX. Suspect columns are determined by running the SQL statement -

```
[BEGIN CODE BOX]

select distinct table_name, column_name
from dba_tab_columns
where owner = 'APPSX'
and (column_name like 'SEGMENT%'
or column_name like '%CODE%COMB%')
```

```

or column_name like '%GL%'

or column_name like '%ACC%')

and column_name not like '%ATTRIBUTE%'

and column_name not like '%CODE_COMBINATION_ID'

```

The resulting tables must be examined for content. Exclude tables by examining the column names in the table. A general assumption is that columns named CODE_COMBINATION_ID are safe because these columns will contain the CCID pointer, which will not change. Any column containing the name ATTRIBUTE is also excluded because these columns are assumed to be description flexfields. Both these assumptions were added to the query to reduce the number of records.

Documentation of this process should be created summarizing the analysis. This documentation should include a table on showing the results of this query and the logic behind selecting the table with possible contention. In the example below, all tables resulting from this query are shown with the tables eliminated through column name and visual inspection of column contents displaying a strikethrough line.

TABLE_NAME	Suspect Column
APPSX_AP_IMPORT_TEMP	ACCOUNTING_DATE
APPSX_AP_IMPORT_TEMP	SEGMENT1
APPSX_AP_IMPORT_TEMP	SEGMENT2
APPSX_AP_IMPORT_TEMP	SEGMENT3
APPSX_AP_IMPORT_TEMP	SEGMENT4
APPSX_AP_IMPORT_TEMP	SEGMENT5
APPSX_AP_IMPORT_TEMP	SEGMENT6
APPSX_AR_LOCATIONS	GL_SITE_CODE
APPSX_BILLING_SUPPL	ACCOUNT_NO
APPSX_BILLING_SUPPL	ACCOUNT_STATUS
APPSX_BILLING_SUPPL	ROLLUP_ACCOUNT
APPSX_BILLING_SUPPL	USE_ROLLUP_ACCT

Custom Programs

Find the programs, which are registered as custom programs using the following SQL statement:

```

Column executable_name format a20

Column execution_file_name format a30

Column application_short_name format a15

select application_short_name,
executable_name,
execution_file_name,
decode(execution_method_code,
'L', 'SQL*Loader',
'I', 'Stored Proc',
'H', 'Host',
'P', 'Oracle Report',
'Q', 'SQL*Plus') method
from fnd_executables, fnd_application
where fnd_executables.application_id > 10000
and fnd_application.application_id =
fnd_executables.application_id
order by decode(execution_method_code,
'L', 'SQL*Loader',
'I', 'Stored Proc',
'H', 'Host',
'P', 'Oracle Report',
'Q', 'SQL*Plus')

```

1. Check the SQL*Plus programs for possible customization concerns. There is no quick way to identify which SQL*Plus routines will be effected by this modification since SQL is used in a variety of ways - internally in code, from the command line using a sequence of commands or registered in the application as a routine that can be run using the concurrent manager. Because of this complexity, each of the SQL routines found in the custom directory (\$APPSX_TOP/sql) should be reviewed visually.
2. Both SQL*Loader and Host routines are stored in the custom bin directory (\$APPSX_TOP/bin). Determine the type of files contained in this directory by running the UNIX command 'file *'. All programs in this directory identified as shell scripts, ASCII files, SQL*Loader routines or host shell scripts should be inspected visually.
3. Check the stored procedures by using SQL*Plus. Substitute the X in the statement below for each of

the stored procedures found in this step.

Set arraysize 1

select text from dba_source where name = 'X';

4. Through visual analysis, determine if there are any concerns related to a stored procedure. Oracle Reports are a concern. Currently, I know of no way to analyze them except visually, and that may take a while.

Oracle Financials Standard Tables

There are tables within the application that may contain accounting information directly. These must be found and modified as well. Run the following queries to find all tables that may contain the field 'SEGMENT' or 'CONCATENATED_SEGMENT,' suggesting that they may contain actual account information and not just a reference. For the purposes of this paper, we will use SEGMENTx as the segment value being modified.

```

Set headings off

Set pagesize 0

Spool getcount.sql

Select 'set echo on' from dual;

Select 'spool segcount.lst' from dual;

Select distinct 'select count(*) from ' ||
DBA_TABLES.TABLE_NAME || ';'
from DBA_TAB_COLUMNS, DBA_TABLES
Where DBA_TAB_COLUMNS.TABLE_NAME =
DBA_TABLES.TABLE_NAME
and COLUMN_NAME like '%SEGMENT%';

Select 'Spool off' from dual;

Spool off

@getcount

```

Spooling the output and running the file as shown will give a list of tables with the column name like SEGMENT and the number of records the table contains. Of the resulting list, the following tables were determined to contain direct account information. Your results may vary depending on the applications installed at your site. The results listed below are taken from one client by looking at the tables with at least one record.

Set

```

TABLE      COLUMN NAME
GL_CODE_COMBINATIONS      SEGMENTx
GL_POSTING_INTERIM        SEGMENTx
GL_INTERFACE              SEGMENTx
GL_ALLOC_FORMULA_LINES    SEGMENTx
GL_BUDGET_INTERFACE       SEGMENTx
GL_BUDGET_INTERIM         SEGMENTx
GL_BUDGET_ASSIGNMENT_RANGES      SEGMENTx_HIGH, SEGMENTx_LOW
GL_JE_HEADERS      BALANCING_SEGMENT_VALUE
GL_OASIS_FIN_ASSIGNMENTS      ACCOUNT_SEGMENT_VALUE
FA_LOCATIONS      SEGMENTx
FA_ASSET_KEYWORDS      SEGMENTx
RA_ACCOUNT_DEFAULT_SEGMENTS      CONSTANT
PO_CONTROL_RULES      SEGMENTx_HIGH, SEGMENTx_
LOW RG_REPORT_CONTENT_OVERRIDES SEGMENTx_HIGH, SEGMENTx_LOW AP_
FLEX_SEGMENT_MAPPINGS
RANGE_HIGH,
RANGE_LOW

```

A second query will find all tables with concatenated segment values. These will need to be changed as well. The segment being modified should be extracted and replaced. Use the following query to find the tables that might be candidates for modification. Examine the column names for the possibility of data contained in each to determine if they are truly candidates for modification.

```

select  ALL_TAB_COLUMNS.TABLE_NAME, ALL_TAB_COLUMNS.COLUMN_NAME
from    ALL_TAB_COLUMNS, DBA_TABLES
where   ALL_TAB_COLUMNS.COLUMN_NAME
        like 'CONCAT%'
and     ALL_TAB_COLUMNS.TABLE_NAME = DBA_TABLES.TABLE_NAME

```

Technical Process

The technical process discussed here is based on a simple example of expanding the fourth segment of the accounting flexfield from six digits to eight characters. Segment4 is the location segment. Because the segment is the location field, we will have to modify the Fixed Assets key flexfield as part of the technical process described below. Segment1 of the key asset flexfield will contain the exact same value set as those in segment4 of the accounting flexfield. Because segment4 will be changed from a digit to a character, a complete mapping will be required. The table appsx.mapping_table will be used for the purpose of mapping all segment4 entries. There should be one entry for each value in segment4 of the accounting flexfield.

```

insert into fnd_user (
    USER_ID,
    USER_NAME,
    LAST_UPDATE_DATE,
    LAST_UPDATED_BY,
    CREATION_DATE,
    CREATED_BY,
    ENCRYPTED_FOUNDATION_PASSWORD,
    ENCRYPTED_USER_PASSWORD,
    SESSION_NUMBER,
    START_DATE,
    DESCRIPTION)
select -7, 'ACCOUNTING CODE MODIFICATION',
sysdate, -7, sysdate, -7,
ENCRYPTED_FOUNDATION_PASSWORD,

```

```

        ENCRYPTED_USER_PASSWORD ,
        0, sysdate,
        'SQL scripts for the GL structure upgrade'
from FND_USER where user_name = 'ANONYMOUS'
/

```

Steps to Increase the Segment Size

1. Create an Oracle Financials user to identify the records being update here. The WHO columns of each update will contain a record of this user. We will use the user_id -7 as our unique number.

```

                                Create table mapping_table
(mapped_column  varchar2(8),
original_value  varchar2(6));

```

2. Create a mapping table. The table will basically have two columns. One column, named original_value, will contain values currently in segment4 of the gl_code_combinations table. The second column, named mapped_column, will contain the value being mapped to when the conversion is complete. Other columns may be added to include other information related to the mapping, but these basic two columns are required.

```

Select distinct segment4 from gl_code_combinations
minus
select original_value from appsx.mapping_table;

```

3. All values used in segment4 of the chart of accounts must be contained in the mapping table prior to beginning. The mapping table must be complete and contain no duplicates. Use the following scripts to guarantee that the table is clean and ready for the conversion.

A. Validate that everything is mapped by running the following SQL. Do not continue until all values are mapped.

```

col flex_value format a9
col description format a50

select distinct ffv.flex_value, ffv.description
from fnd_flex_value_sets ffvs, fnd_flex_values_vl ffv,
gl_code_combinations gcc
where ffvs.flex_value_Set_name = 'APPSX_LOCATION'
and ffvs.flex_value_set_id = ffv.flex_value_set_id
and ffv.flex_value = gcc.segment4
and gcc.segment4 in (
select distinct segment4 from gl_code_combinations where length(segment4) = 6
minus (select original_value from appsx.mapping_table));

```

This query should return no values found. If any records are found, use the following script to identify the description associated with the segment values.

```

select original_value from appsx.mapping_table
minus
select distinct segment4 from gl_code_combinations;

```

The script is based on the assumption that there is a value set named APPSX_LOCATION, which contains valid values for this segment. The reverse of the above query must be guaranteed as well. Run the following query to determine if there are any values in the mapping_table which are not in gl_code_combinations.

Last, we must be certain that there are no double entries in the mapping table. Use the following query to find any duplicate entries.

```

select a.original_value, a.mapped_column
from appsx.mapping_table a
where a.original_value = (select b.original_value
from appsx.mapping_table b
where a.original_value =
b.original_value
and a.rowed <>
b.rowed )

```

Any rows returned indicate a duplicate entry in the mapping table. Delete one of the entries.

4. Values used in segment1 of the key asset flexfield which map will be converted. This client has values used as asset keys that are not directly related to a location, and thus will not map. Values not mapped will remain as they were. Use the SQL below to review asset key values that don't map.

```

select distinct segment1 from fa.fa_asset_keywords
minus
select original_value from appsx.mapping_table;

```

Use the following script to identify the description associated with the segment values.

```

This script looks at records not in
appsx.mapping_table but have gl code
combinations values.
These values are all defined in the list of values for
the location segment of the chart of accounts, but
they are not defined in the appsx.mapping_table.
col flex_value format a9
col description format a50

```

```

select distinct ffv.flex_value, ffv.description
from fnd_flex_value_sets ffvs,
fnd_flex_values_vl ffv,
fa_asset_keywords fak
where ffvs.flex_value_set_name = 'APPSX_LOCATION'
and ffvs.flex_value_set_id = ffv.flex_value_set_id
and ffv.flex_value = fak.segment1
and fak.segment1 in (
select distinct segment1
from fa_asset_keywords
where length(segment1) = 6
minus

```

```
(select mapped_column
from appsx.mapping_table));
```

Run the following query to determine if there are any values in the mapping_table which are not used as an asset key.

```
select original_value from appsx.mapping_table
minus
select distinct segment1 from fa_asset_keywords;
```

Any record retrieved by the above query must have one record enabled and the other with enabled_flag. If this is not the case, stop here and fix the problem before continuing.

5. Create synonyms and grants to allow the APPS user to access the necessary custom table owned by APPSX.

Log onto SQL*Plus as the user APPSX and run the following command:

```
grant select on MAPPING_TABLE to APPS;
```

Then log on as APPS and run the following commands:

```
create synonym MAPPING_TABLE
```

```
for APPSX. MAPPING_TABLE;
```

6. Create a new value set for the accounting flexfield segment. The validation for this value set may be configured a number of different ways, including the use of MAPPING_TABLE as the table for validation (ongoing maintenance of the mapping table is definitely a consideration). Using the System Administrator responsibility: Nav: Application > Validation > Set. Enter a new record with the following information.

7. Create a new value set for locations in Fixed Assets. Arrow down to a new record with the following information.

```
Value Set Name: APPSX_GL_NEW_LOCATION
Description:    Post conversion location
Format Type:   Char
Maximum Size:  8
Validation Type: To be determined....
```

Change the value set on the chart of accounts segment to the new one just created.

Nav: Flexfield > Key > Segments. Run a query to obtain the Accounting Flexfield record.

Unfreeze the record and proceed to the Segments screen.

Click on the Value Set field of the fourth segment (GL_LOCATION) and type:

Value Set: APPSX_GL_NEW_LOCATION

9. Change the value set for any custom programs. You can use an SQL shortcut for this update because of the complexity involved in updating it within the application. You cannot change the value set directly because the set may be used by many different parameters in many different programs. You would have to create a new value set, then change each of the parameters to use the new value set. Using SQL, we can accomplish the same thing by changing the value set behind the scene directly. You will need to know what value set is being used by the program, then look in `fnf_flex_validation_tables` and modify the column names `id_column_name`, `id_column_size`, and `id_column_type`.

10. Asset key values will be maintained separately from the value set for the accounting flexfield. We must update the affected values for the asset keys. First, determine the `flex_value_set_id` for the value set used for the asset keys. We will assume the `flex_value_set_id = 1002625`. Use the following SQL to update the value set values.

```

update fnf_flex_values ffv
set
  ffv.flex_value = (select a1.mapped_column
                    from appsx.mapping_table a1
                    where ffv.flex_value = a1.original_value),
  ffv.last_updated_by = -7,
  ffv.last_update_date = sysdate
where ffv.flex_value in
  ( select c.original_value from appsx.mapping_table c)
  and ffv.flex_value_set_id = 1002625
/
update fnf_flex_values ffv
set ffv.enabled_flag = 'N',
  ffv.last_updated_by = -7,
  ffv.last_update_date = sysdate
where ffv.flex_value in
  ( select c.original_value from appsx.mapping_table c)
  and ffv.last_updated_by <> -7
  and ffv.flex_value_set_id = 1002625

```

The best response to get from the last statement would be 'No records found', indicating that all mappings were successful. Update the existing asset items key flexfield values with the new segment value using the following SQL statement. Most of the asset keys (but not all) were equivalent to the location segment of the chart of accounts. The request was to map all asset keys that matched and leave the rest as is.

```

Update FA_ASSET_KEYWORDS FAK
set FAK.SEGMENT1 = (select mapped_column
from appsx.mapping_table
where fak.segment1 =
appsx.mapping_table.original_value),
last_updated_by = -7,
last_update_date = sysdate
where FAK.SEGMENT1 in
(select original_value from appsx.mapping_table

```

11. Update the existing chart of accounts values with the new segment value using the following SQL statement. The WHERE clause on the update statement prevents the update of records which do not match (even though there should not be any).

```

Update GL_CODE_COMBINATIONS GCC
set GCC.SEGMENT4 =
(select mapped_column
from appsx.mapping_table
where GCC.SEGMENT4 =
appsx.mapping_table.original_value),
last_updated_by = -7,
last_update_date = sysdate
where GCC.SEGMENT4 in
(select original_value from appsx.mapping_table )
/

Select GCC.SEGMENT4
from GL_CODE_COMBINATIONS GCC
where length(GCC.SEGMENT4) = 6
/

```

The best response to get from the last statement would be 'No records found', indicating that all mappings were successful.

Update the existing asset items key flexfield values with the new segment value using the following SQL statement. Most of the asset keys (but not all) were equivalent to the location segment of the chart of accounts. The request was to map all asset keys that matched and leave the rest as is.

```

Update FA_ASSET_KEYWORDS FAK
set FAK.SEGMENT1 = (select mapped_column
from appsx.mapping_table
where fak.segment1 =
appsx.mapping_table.original_value),
last_updated_by = -7,
last_update_date = sysdate
where FAK.SEGMENT1 in
(select original_value from appsx.mapping_table)
/

```

12. Update the existing records waiting to be posted with the new segment value using the following SQL statement.

```

Update RG_REPORT_AXIS_CONTENTS
set SEGMENT4_LOW = '00000000'
where SEGMENT4_LOW = '000000'
/
Update RG_REPORT_AXIS_CONTENTS
set SEGMENT4_LOW = 'ZZZZZZZZ'
where SEGMENT4_LOW = '999999'
/
Update RG_REPORT_AXIS_CONTENTS
set SEGMENT4_HIGH = '00000000'
where SEGMENT4_HIGH = '000000'
/
Update RG_REPORT_AXIS_CONTENTS
set SEGMENT4_HIGH = 'ZZZZZZZZ'
where SEGMENT4_HIGH = '999999'

```

17. In the Purchasing module, there is one table that stores account combinations by segments instead of the CCID. There is a range value that needs to be enlarged to accommodate the larger segment size. Run the following SQL:

```

Update PO_CONTROL_RULES
set SEGMENT4_HIGH = '00000000'
where SEGMENT4_HIGH = '000000'
/
Update PO_CONTROL_RULES
set SEGMENT4_HIGH = 'ZZZZZZZZ'
where SEGMENT4_HIGH = 'ZZZZZZ'
/

```

18. In the Accounts Receivables module, there is one form that stores account combinations by segments instead of the Code Combination ID (CCID). The Define Automatic Accounting form uses the RA_ACCOUNT_DEFAULT_SEGMENTS table to store AutoAccounting Rules. Each segment in the chart of accounts is stored as a separate row in this table. There is a constant value that needs to be enlarged to accommodate the larger segment size. Run the following SQL:

```

update RA_ACCOUNT_DEFAULT_SEGMENTS
set constant = constant || '00'
where segment_num = 4
/

```

19. Update any custom tables. Remember to consider any column containing account code information. One example I found was a table with an ACCOUNT column actually containing the equivalent of the concatenated accounting structure (SEGMENT1||'-'||SEGMENT2||'-'||SEGMENT3 ||'-'||SEGMENT4||'-'||SEGMENT5||'-'||SEGMENT6) from gl_code_combinations. In this case, I updated the table using the following SQL:

```

Update APPSX.APPSX_AP_IMPORT_TEMP A set ACCOUNT = (select
    substr(b.ACCOUNT,1,14) ||
    mapped_column ||
    substr(b.ACCOUNT,21,9)
from APPSX.MAPPING_TABLE MT,
APPSX.APPSX_AP_IMPORT_TEMP B
    where substr(b.ACCOUNT,15,6) = original_value
and A.ACCOUNT = B.ACCOUNT)
where substr(A.ACCOUNT,15,6) in (select original_value
    from APPSX.MAPPING_TABLE)
/

```

Testing

1. Testing done prior to the final production conversion can be accomplished if all existing original_value columns are mapped to mapped_column values. Before complete mapping can be accomplished, the process described in this paper should be tested. A number of scripts were written to fill in the missing values prior to completing the mapping portion of the project. A basic description of the scripts is included here for reference.

A. Null_map.sql

Fill in missing mapped values by mapping numbers to letters. These are substitute values used for intermediate testing before the actual mappings are finished.

```

update appsx.mapping_table
set mapped_value =
decode(substr(original_value,1,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
decode(substr(original_value,2,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
decode(substr(original_value,3,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
decode(substr(original_value,4,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
decode(substr(original_value,5,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
decode(substr(original_value,6,1),
'1','A','2','B','3','C','4','D','5','E',
'6','F','7','G','8','H','9','I','0','J','X') ||
'XX',
last_updated_by = -7777777,
last_update_date = sysdate
where clli is null
/

```

B. This script looks at records not in either the mapping_table but that have gl_code combinations values. These values are all defined in the list of values for the location segment of the chart of accounts, but they are not defined in the mapping table.

```

col flex_value format a9
col description format a50

select distinct ffv.flex_value, ffv.description
from apps.fnd_flex_value_sets ffvs,
apps.fnd_flex_values_vl ffv,
apps.gl_code_combinations gcc
where ffvs.flex_value_set_name =
      'APPSX_LOCATION'
and ffvs.flex_value_set_id = ffv.flex_value_set_id
and ffv.flex_value = gcc.segment4
and gcc.segment4 in (
select distinct segment4
      from apps.gl_code_combinations
      where length(segment4) = 6
minus
(select gl_site_code from appsx.mapping_table ))
/

```

2. Each external application will be tested to ensure that the modifications do not affect them. If the modifications affect an external application, that application will be modified to incorporate the change. A complete list of external application will be necessary to start this effort.

3. All feeder systems passing data to the General Ledger, including Purchasing, Accounts Payable, Accounts Receivables, Inventory and Fixed Assets, will be tested by processing a specific set of transactions in each feeder system and then processing a monthly close.

4. A selection of reports encompassing the General Ledger and all feeder systems will be printed prior to the expansion of the chart of accounts segment. They will then be compared to the same reports printed after the expansion of the segment.

Other Considerations

One thing to consider for this project is developing a simple form for the mapping table. This will allow the people involved in the mapping process access to the accounts and their mapped equivalence. The form will have the advantage of :

- letting the accounting department help in the mapping process;
- provide a visual means of recalling what the mapped value was prior to the modification; and
- allowing on-line searches by users trying to map old printed documents to their new account

equivalent without having to ask a technical person for the information.

About the Author

Vincent Giasolli is the Senior Applications Architect with SofTec Solutions. He has over 20 years of experience in information technology, 14 years experience with the Oracle database, and 12 years experience with Oracle Financials. Throughout his career he has been involved with database administration, training, software design, development, testing, maintenance, documentation and technical support. He has been an instructor having trained other trainers, developers and end-users as well as classes at the University of Texas.

[Download Acrobat Reader](#)

Copyright 2003 by the International Oracle Users Group