Warehouse Builder 11*g*

Development's Bag of Tricks for Warehouse Builder

May 2008

ORACLE

Development's Bag of Tricks for Warehouse Builder

INTRODUCTION

With Warehouse Builder 11g we have arrived at the 10th generation or major release of the product. Many features are built and ready for you to use. In this paper we look at some of the more useful, yet somehow often forgotten features in Warehouse Builder.

As in any tips and tricks paper we are looking at practical applications of the feature set. Focus is on enhanced productivity, better results and applications of Warehouse Builder to make your project more successful.

We will work our way through the core part (included in any database license – Standard Edition, Standard Edition One and Enterprise Edition) and the options. If we are looking at optional functionality (can only be licensed with database Enterprise Edition) we will make note of that fact.

DATA MATCHING AND MERGING

Incredible but true, this advanced functionality comes with the core edition of Warehouse Builder and is therefore included in the database license. Rather than writing lots of PL/SQL to do some string matching you should apply this advanced functionality.

This short example shows Oracle Warehouse Builder's match/merge capabilities in a fairly common scenario. We will de-duplicate customer data from three different systems using the match/merge operator.

The power of the match/merge operator is that it is rule based and makes use of powerful algorithms rather than of hand coded custom routines. As part of the core product, match/merge is one of the most powerful tools in your toolbox to resolve this common problem.

This small glossary explains some terms up-front and can act as a reference later on. It also has a definition of how the term is used in Warehouse Builder in this specific case.

Term	Definition
Match	The first step in match/merge. Match uses match rules leveraging algorithms to detect potential matches. Many match rules can be combined. Sets of attributes (or individual ones) can carry individual rules
Merge	Using rules to actually apply a merge on the potential matches. Each attribute can carry one or more individual merge rules.
Bin	The most important concept for performance of the matching. A bin groups or partitions (not the DB feature!) data and matching only occurs within the bin.
Match or Merge Rules	The individual algorithms applied to a set of attributes including the thresholds etc constitutes a match or merge rule. Rules can be composite via a custom rule.

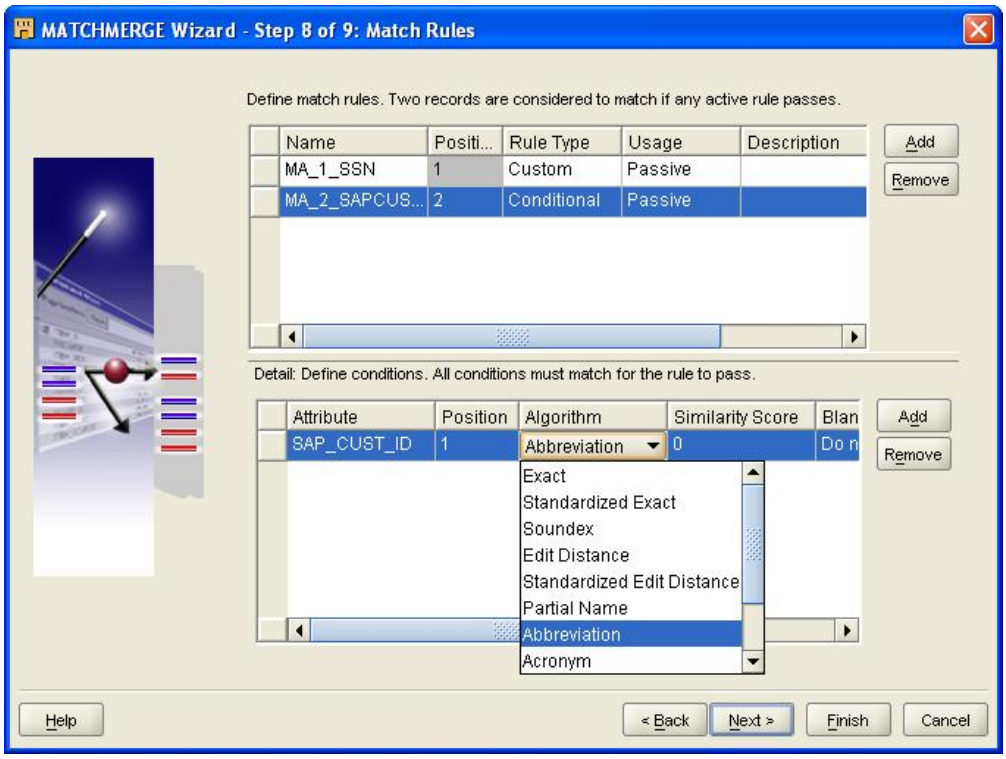
Match Rules

We will implement a complex rule scheme using the following rules:

Data Element	Rule Description
SSN	If the SSN is not null and not equal to 999-99-9999 then use fuzzy (edit distance) matching

Data Element	Rule Description
SAP_CUST_ID	If the SAP_CUST_ID is not null then use partial (abbreviation) matching
XYZ_CUST_ID	If the XYZ_CUST_ID is not null then use exact matching
ABC_CUST_ID	If the XYZ_CUST_ID is not null then use exact matching
NAME (composite)	If first name and the last name are not null then use fuzzy (soundex) matching

A completed set looks likes this:



Merge Rules

The merge rules (determining how we choose each attribute) are based on the following:

Data Element	Rule Description
NAME_M	The longest non-null middle name
SSN	The most common SSN

Data Element	Rule Description
NAME_F	The longest non-null first name from Table A
CUST_SEQ	From the same record as the merged SSN
SAP_CUST_ID	The most common SAP_CUST_ID with 7 characters

Rather than creating all this in the UI we will choose scripting in OWB to create these rules in the metadata. A short overview of the commands is as follows:

```
#Creating a merge rule set
#Rule 1:
#Note that the order of the last two settings has to be exactly like
#this! TYPE before Attribute.

OMBALTER MAPPING 'CUST_MATCH_MERGE' \
    ADD MERGE_RULES 'ME_1_NAMEM' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (TYPE) VALUES
('MM_MIN_MAX') \
    MODIFY MERGE_RULES 'ME_1_NAMEM' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (ATTRIBUTE_NAME) VALUES
('NAME_M') \
    MODIFY MERGE_RULES 'ME_1_NAMEM' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (MIN_MAX_TYPE) VALUES
('MM_LONGEST') \
    MODIFY MERGE_RULES 'ME_1_NAMEM' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (MIN_MAX_ATTRIBUTE)
VALUES ('NAME_M')
```

This first rule creates a pre-built type rule picking values for NAME_M (middle name) via the longest of the middle name fields in the matched records.

The following is an example of a more complex rule:

```
#Rule 2:
#Custom rule text is added to a variable and called from OMB.
#Note the escape characters in the PL/SQL text:
# Double quotes are escaped by a slash
# Single quotes are escaped by a single quote

set custom_rule2 "fName varchar2(2000) := null;
BEGIN
    -- return the longest first name from table a
    -- in table a, CUST_SEQ is not null
  FOR i IN M_MATCHES.FIRST .. M_MATCHES.LAST LOOP
      IF  M_MATCHES(i).\"NAME_F\" IS NOT NULL and
          M_MATCHES(i).\"CUST_SEQ\"  is not null THEN
        IF fName IS NULL OR LENGTH(RTRIM(M_MATCHES(i).\"NAME_F\")) >
LENGTH(RTRIM(fName)) THEN
          fName := M_MATCHES(i).\"NAME_F\";
        END IF;
      END IF;
  END LOOP;
  RETURN fName;
END;"

OMBALTER MAPPING 'CUST_MATCH_MERGE' \
    ADD MERGE_RULES 'ME_2_NAMEF' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (TYPE) VALUES
('MM_CUSTOM') \
```

```
    MODIFY MERGE_RULES 'ME_2_NAMEF' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (ATTRIBUTE_NAME) VALUES
('NAME_F') \
    MODIFY MERGE_RULES 'ME_2_NAMEF' \
        OF OPERATOR 'MATCHMERGE' SET PROPERTIES (CUSTOM_TEXT) VALUES
('$custom_rule2')
```

For more on the scripting on the merge rules have a look at the Warehouse Builder blog entry here.

Match Merge is one of the most powerful features in Oracle Warehouse Builder but for some reason rarely used. This session and paper aims to change this.

PIVOTTING DATA ELEMENTS

The pivot transformation operator enables you to transform a single row of attributes into multiple rows in an efficient manner. This example illustrates transforming a table that has a row for each year with the quarterly sales in a table with a row for each quarter. The OWB pivot operator makes this simple (there is also an unpivot).

So taking a simple example as follows:

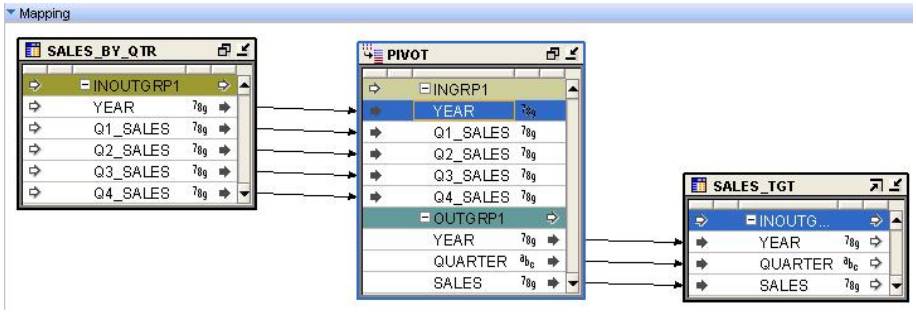
YEAR	Q1_sales	Q2_sales	Q3_sales	Q4_sales
2005	10000	15000	14000	25000
2006	12000	16000	15000	35000
2007	16000	19000	15000	34000

we wish to transform the data set to the following with a row for each quarter:

YEAR	QTR	SALES
2005	Q1	10000
2006	Q1	12000
2007	Q1	16000
2005	Q2	15000
2006	Q2	16000
2007	Q2	19000
2005	Q3	14000

etc..

We can design this in the OWB mapping as



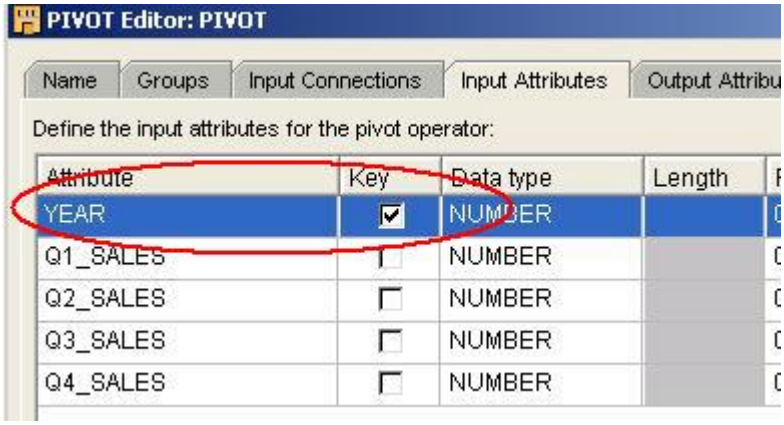
Looking at the internals of the operator we see how this is described. The pivot operator allows you to define the input columns, the output columns and how the data is pivoted. This is achieved by defining a few pieces of information;

the key columns (the columns from the source that will appear in the output of the pivoted data)

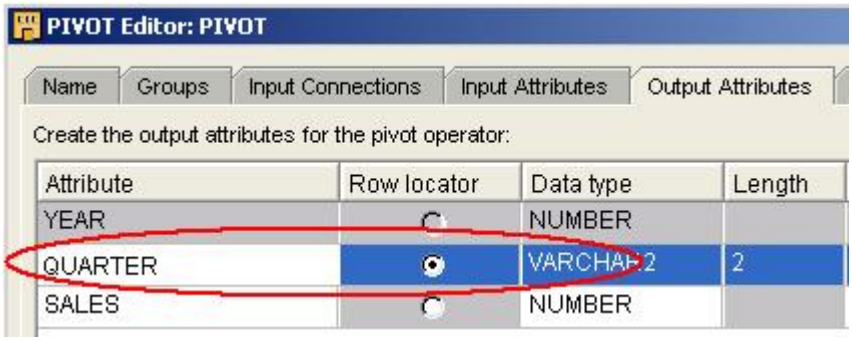
the row locator (this is the pivot column)

the pivot transformation (which values to project for the pivoted columns

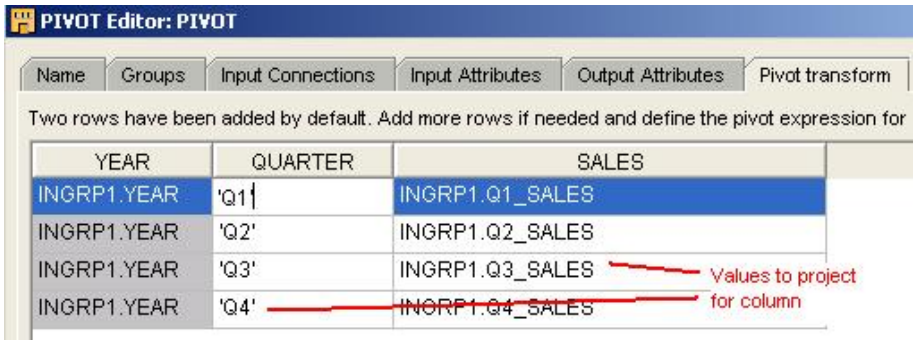
Firstly in our example we define the key column to be YEAR, this will be the same for each pivoted row.



Then we define the QUARTER column as the pivot column, this is the row locator (in OWB terms).



Finally we define how the row is transformed from a row with columns to a number of rows, we do this by entering a row in the table for each case we desire (so we have a row for Q1, a row for Q2, a row ... etc.).



This makes the map design so much simpler as you can see, since the operator

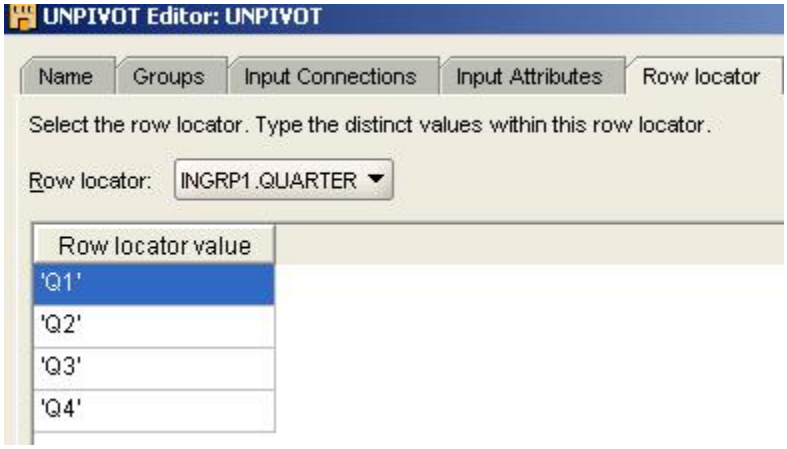
encapsulates the transformation in a simple manner. The example has been scripted (get the script here) so that you can create it and have a look around at how this is done....

Here is the sample data I used also:

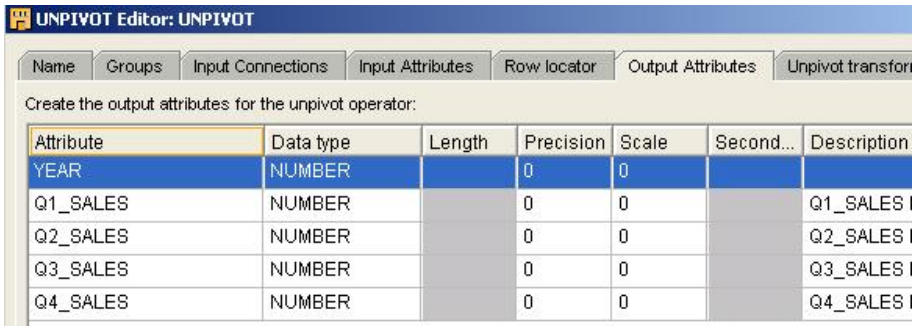
```
--
-- Execute the following where you deployed SALES_BY_QTR
-- I manually added some rows in SALES_BY_QTR for the example:
--
insert into SALES_BY_QTR values (2005, 10000, 15000, 14000, 25000);
insert into SALES_BY_QTR values (2006, 12000, 16000, 15000, 35000);
insert into SALES_BY_QTR values (2007, 16000, 19000, 15000, 34000);
commit;
```

The reverse of this scenario is the unpivot, the script for the unpivot example can be found here.

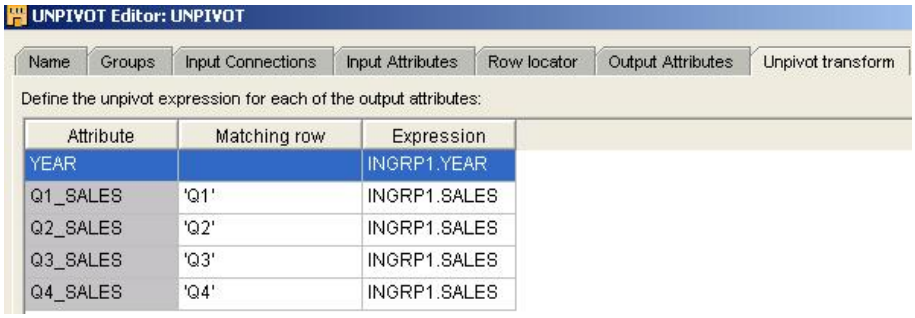
You select the key just like the pivot above, then define the row locator (or unpivot column), defining the values for each match row:



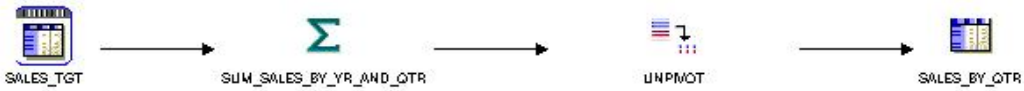
Then define the output attributes for the unpivot:



Finally define the unpivot transformations (how the column data is taken from the matching row):



If your data has many rows with sales values for a quarter (for a single year) you will need to aggregate the data before unpivoting, for example the map below first aggregates and sums sales before unpivoting. The data is grouped by YEAR (key) and QUARTER (row locator) and the output expression has SUM(SALES), the map then unpivots that data. (you cannot tweak the agg function just now in the unpivot)



As with the Match/Merge operator, the pivot and unpivot capabilities are part of the core edition of Warehouse Builder.

ADVANCED SQL CAPABILITIES

One of the amazing things in Warehouse Builder is the SQL you can produce in the product. You can do very simple and straightforward mappings to load data, but in many cases (like loading fact tables) you will need to start looking at more complex SQL queries. Here we are covering a few examples that are either new or interesting, and all are just part of the core product.

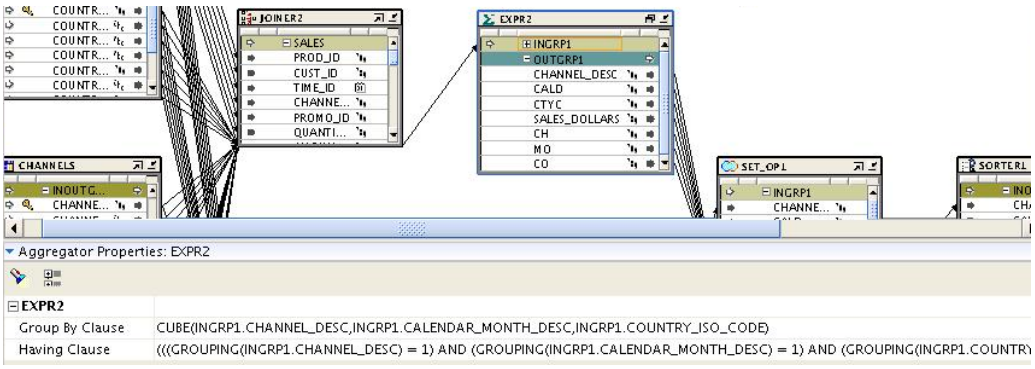
Aggregation using advanced SQL

The OWB 10.2.0.3 and 11g releases of OWB extended the aggregation capabilities to support Oracle's advanced aggregation capabilities. For example the CUBE and ROLLUP clauses can now be used in the aggregation operator.

To illustrate the usage, we have use the example from the Oracle Data Warehousing guide (Example 20-8 GROUPING combined with HAVING):

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
       TO_CHAR( SUM(amount_sold), '9,999,999,999') SALES$,
       GROUPING(channel_desc) CH,
       GROUPING (calendar_month_desc) MO,
       GROUPING(country_iso_code) CO
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND
sales.cust_id=customers.cust_id AND customers.country_id =
countries.country_id AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet') AND
times.calendar_month_desc IN ('2000-09', '2000-10') AND
country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc,
country_iso_code)
HAVING (GROUPING(channel_desc)=1 AND
GROUPING(calendar_month_desc)= 1 AND
GROUPING(country_iso_code)=1) OR (GROUPING(channel_desc)=1 AND
GROUPING (calendar_month_desc)= 1) OR
(GROUPING(country_iso_code)=1 AND GROUPING(calendar_month_desc)=
1);
```

In OWB use the AGGREGATOR operator and construct the group by and having clause like this:



As you can see the above statement is now in its entirety modeled within a mapping. Here we are inserting this into a target table, but you can replace the

target table with a view or materialized view and generate that object with the above query. So you can turn Warehouse Builder into a very smart MV generator.

DML Error Logging

Error logging enables the processing of DML statements to continue despite errors being encountered during the statement execution. The details of the error such as the error code and the associated error message are stored in an error table. After the DML operation completes, you can use the error table to correct rows with errors and subsequently process. DML error logging is supported for SQL statements such as INSERT, UPDATE, MERGE, and multi-table insert.

It is useful in long-running, bulk DML statements - for example processing 1 million records and 10 fail, with DML error logging all good records can be committed and the 10 error rows recorded in an error table. Until OWB 10.2.0.3 this was only possible with row based mapping code, now it is possible in set based mode also.

Warehouse Builder provides error logging for the tables, views, and materialized views used in set-based PL/SQL mappings. DML error logging is supported only for target schemas created in Oracle DB 10g R2 or later.

The Error Table

Error tables store error details. You can define error tables for tables, views, and materialized views only. Error tables are used for the following purposes:

- DML error logging (including physical errors).
- Capturing logical errors when data rules are applied to tables, views, or materialized views.

An error table is generated and deployed along with the base table, view, or materialized view if the shadow table name is set. The error table will have the following columns for DML errors.

Column Name	Description
ORA_ERR_NUMBER$	Oracle Error number causing the error
ORA_ERR_MESG$	Oracle Error message text describing the error
ORA_ERR_ROWID$	Rowid of the row causing the error (this is for updates and deletes)
ORA_ERR_OPTYPE$	Type of operation causing the error: I = Insert, U = Update, D = Delete
ORA_ERR_TAG$	Step or detail audit ID from the Warehouse Builder runtime audit data. This is populated with the STEP_ID in

	the ALL_RT_AUDIT_STEP_RUNS runtime public view

If you do not want OWB to generate the error table, you can always build your own error table and supply the name to the mapping, the database provides a function DBMS_ERRLOG.CREATE_ERROR_LOG for generating an error table that can also be used.

Enabling DML Error Logging in ETL

DML error logging is generated for set-based PL/SQL mappings if the following conditions are satisfied:

- The Error table name property is set for the operator (table/view/mv)
- The PL/SQL Generated Mode of the module that contains the mapping is set to 10gR2 and above or Default.

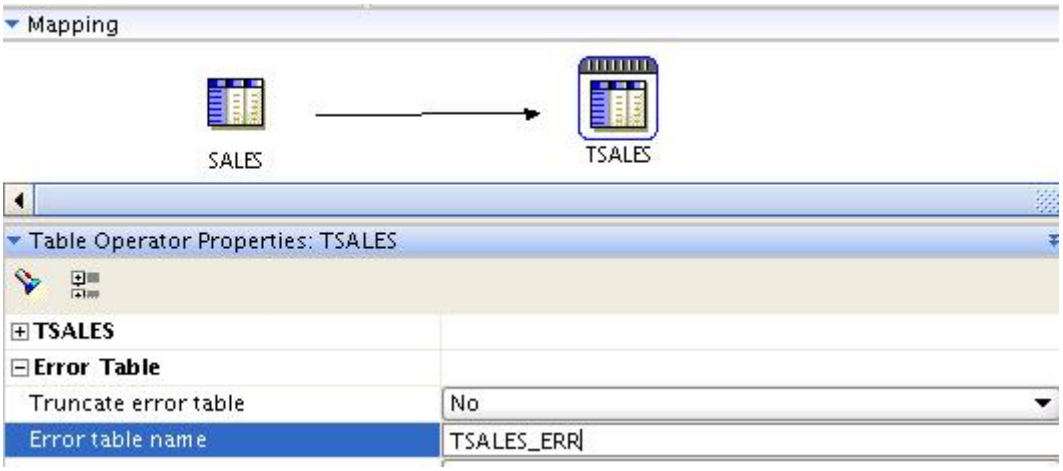
If the value is set to Default, ensure that location associated with the module has the Version set to 10.2 or above.

When you use a data object in a mapping, the Error Table Name property for this data object is derived from the shadow table name property of the data object . If you modify the error table name of a data object (using the shadow table name property), you must synchronize all the operators bound to this data object.

The execution of mappings that contain data objects for which DML error logging is enabled fails if any of the following conditions occur:

- The number of errors generated exceeds the specified maximum number of errors for the mapping. The default set for this value is 50. You can modify this value by setting the Maximum number of errors configuration property of the mapping. In the Project Explorer, right-click the mapping and select Configure . In the Maximum number of errors property, specify the maximum number of errors that can generated before the mapping execution is terminated.
- Errors occur due to functionality that is not supported. (see SQL Reference manual for details of DML Error logging feature restrictions). Depending on your error logging needs you can configure the table operator in a mapping to use the APPEND or NOAPPEND hint. For example, direct-path insert does not support error logging for unique key violations. To log unique key violations, use the NOAPPEND hint (be aware of performance implications of this, there is an interesting article here on some performance findings of using direct path and conventional path modes with the DML error logging feature).

So in my map I specify the error table name as follows:



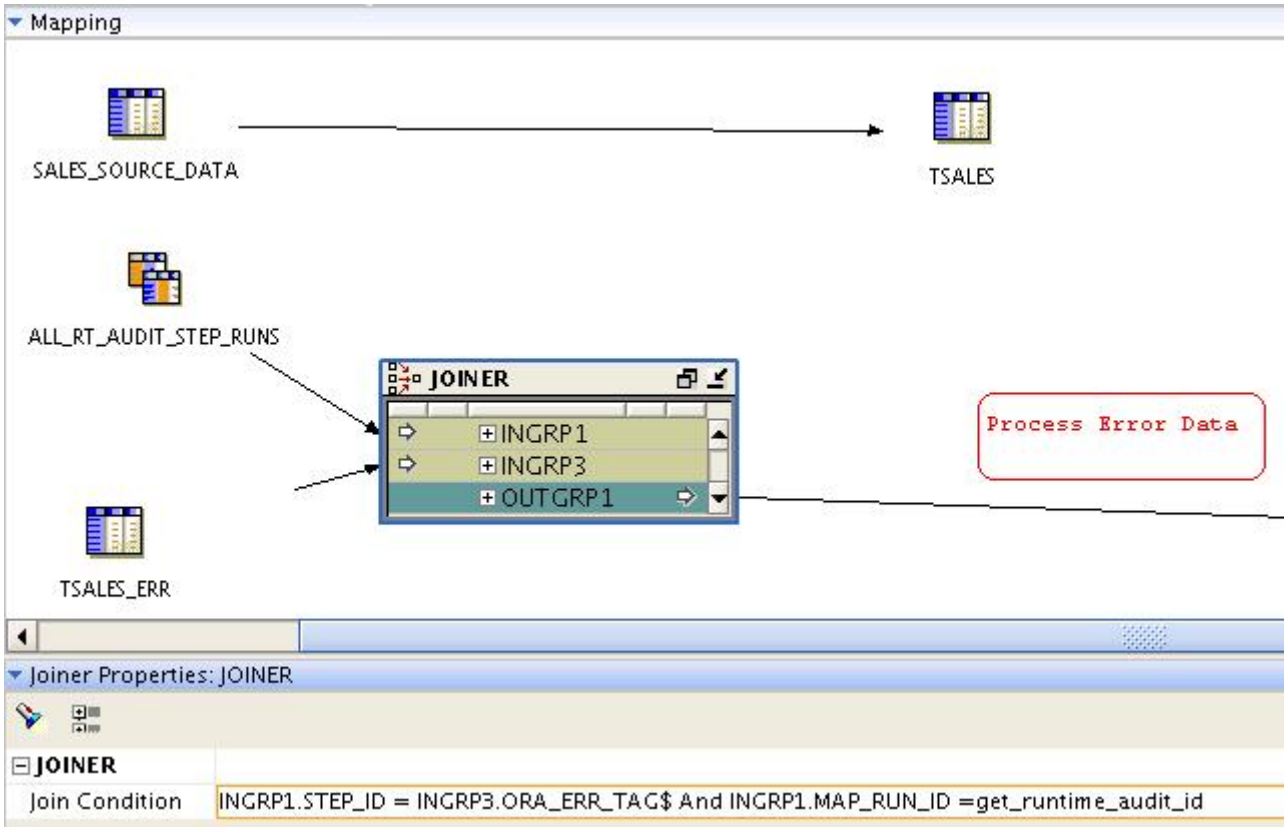
The maximum number of errors was set to 50 (the default in mapping configuration).

Now I get the following SQL generated now with the 'LOG ERRORS INTO ' clause generated;

```
INSERT INTO "TSALES"
  ("PROD_ID",
....
  "SALES"."AMOUNT_SOLD" "AMOUNT_SOLD"
FROM  "SALES"  "SALES"
  )
LOG ERRORS INTO TSALES_ERR (get_audit_detail_id) REJECT LIMIT 50;
```

There is also a truncate error table property on the mapping that can be used for housekeeping.

If you want to process the errors from within the mapping you can retrieve the current execution errors (that is if your error table has errors for more than one execution, it may not since your may choose to truncate at the start of the map execution) by joining the error table with the ALL_RT_AUDIT_STEP_RUNS views and use the get_runtime_audit_id variable in the join condition



In the above map you see the first step load the TSALES table, the second step will join the error table with the ALL_RT_AUDIT_STEP_RUNS using the audit id recorded in the error table in the column ORA_ERR_TAG$ and the STEP_ID column. You could choose a different strategy here either always truncating the error table for the map or using the max step id for example.

HANDLING MULTIPLE ENVIRONMENTS WITH MULTI-CONFIGURATION

Ok, you have to admit it, you were expecting this at some point in time. This feature (multi-config for short) is not free. It is part of the enterprise ETL option for Warehouse Builder, but it is well worth the money!

Problem Definition

Almost every project has the challenge to release new updates into a production system on a regular basis. The problem consists out of many particular issues to be addressed like:

- Version or change management – controlling the various versions of an object through the life cycle of a project
- Release management – creating and testing the appropriate releases before applying them to production

Here we will look primarily at the latter challenge, but of course we should at least briefly discuss the version management point.

Version Management using MDL and Collections

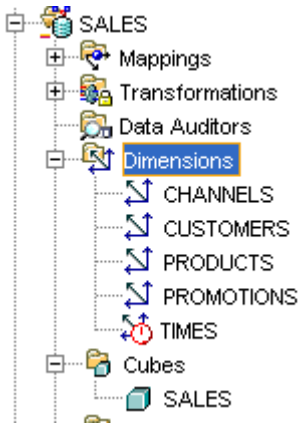
All of these are part of the core product (only multi-configuration – see next topic is part of Enterprise ETL). What it allows you to do is the following:

- Group objects to release in a collection
- Export the collection
- Check the collection's metadata into any versioning system
- Provide the entire collection for deployment to the operations staff

This works all based on some procedures you need to put in place. For example, you should do naming conventions for the collections etc.

A Small Example

In this example we have a simple data mart system with a few dimensions, a fact table (called a cube in Warehouse Builder) and the mappings to feed this cube.

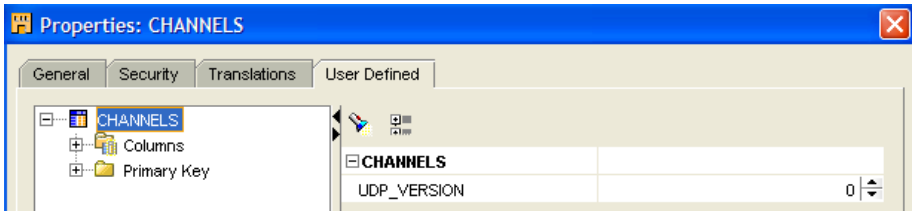


Here we are going to do our initial release of the system.

Step 1: Add a user defined property with the release number – if this property / number is filled out it goes into the collection. In this example we mark tables, dimensions, cubes and mappings with this property:

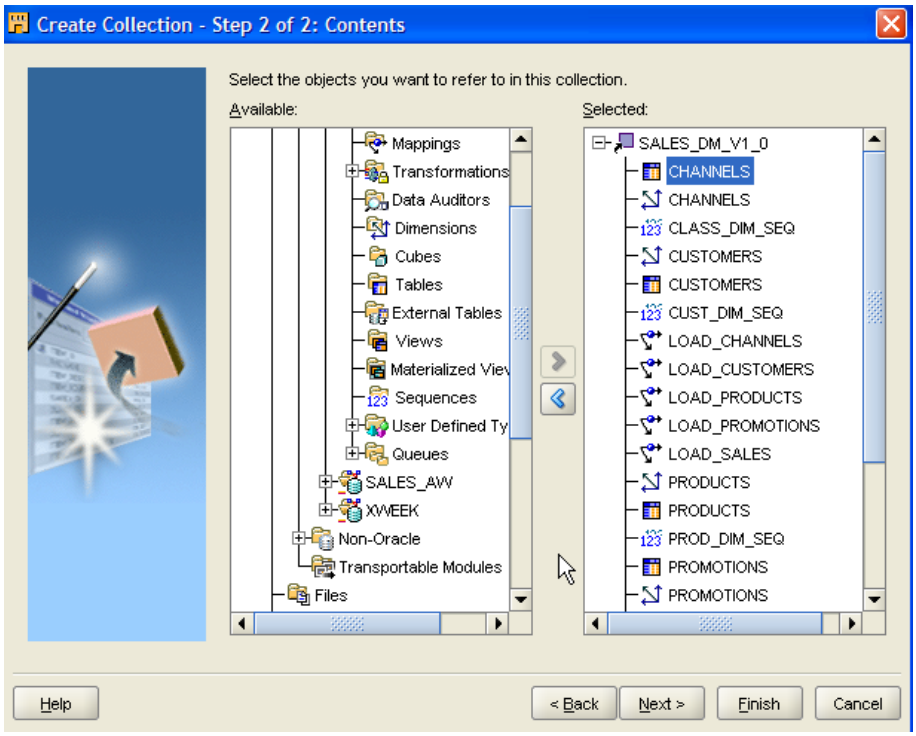
```
OMBREDEFINE CLASS_DEFINITION 'TABLE' \

ADD PROPERTY_DEFINITION 'UDP_VERSION' \

SET PROPERTIES (TYPE, DEFAULT_VALUE) VALUES ('INTEGER', 00)


OMBREDEFINE CLASS_DEFINITION 'CUBE' \

ADD PROPERTY_DEFINITION 'UDP_VERSION' \

SET PROPERTIES (TYPE, DEFAULT_VALUE) VALUES ('INTEGER', 00)


OMBREDEFINE CLASS_DEFINITION 'DIMENSION' \

ADD PROPERTY_DEFINITION 'UDP_VERSION' \

SET PROPERTIES (TYPE, DEFAULT_VALUE) VALUES ('INTEGER', 00)


OMBREDEFINE CLASS_DEFINITION 'MAPPING' \

ADD PROPERTY_DEFINITION 'UDP_VERSION' \

SET PROPERTIES (TYPE, DEFAULT_VALUE) VALUES ('INTEGER', 00)
```

The result of this set of statements is the following (use right mouse click on an object → Properties):



Step 2: Create a collection to capture all the objects

In a collection you collect all required objects.

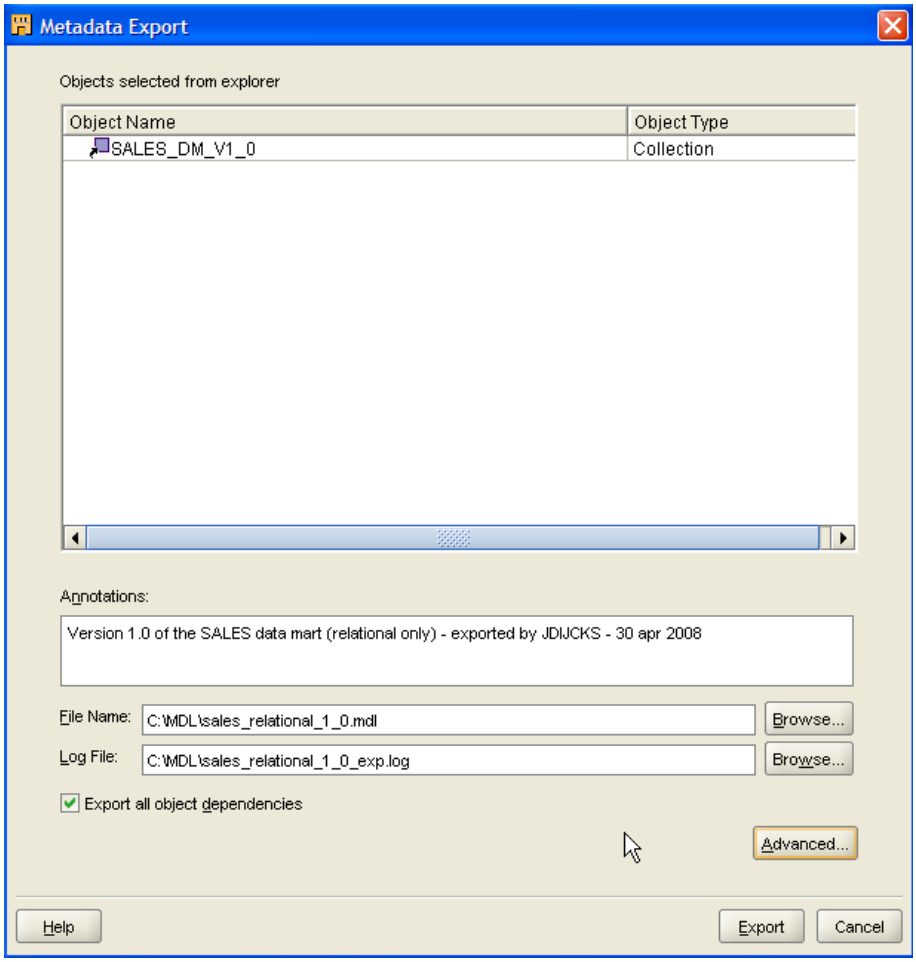


Here we moved them over by hand, but you can use scripting to do the following:

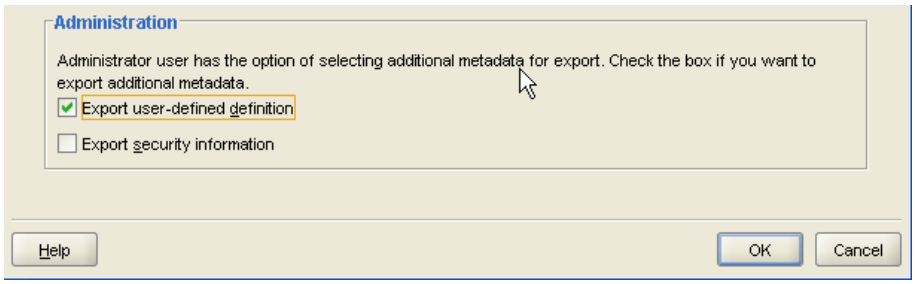
- Determine the version number in the UDP we just created
- Pick those objects for inclusion in the collection

Step 3: Export the collection

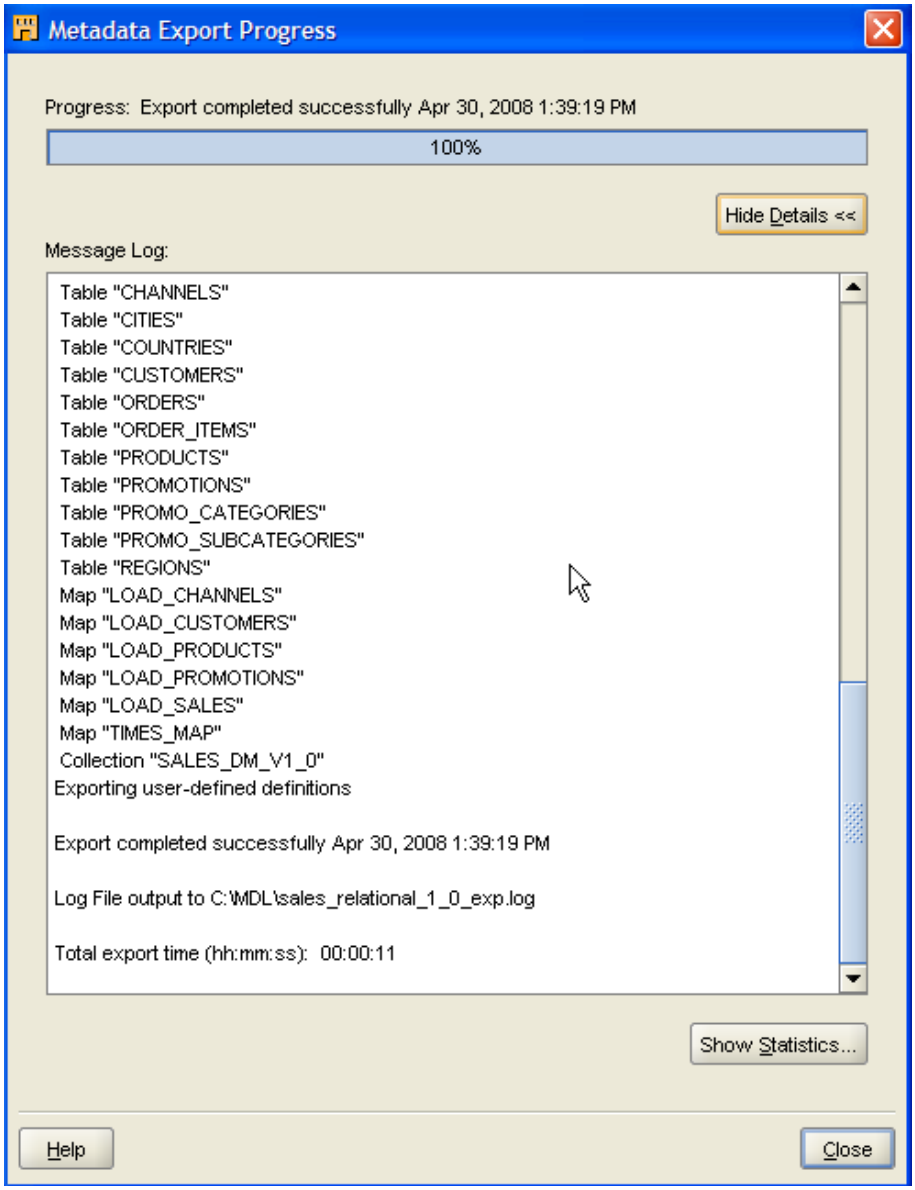
This is again done either from the UI from the scripting interface which makes it automatic. One thing you should do is enter a comment in the export to describe what is in the collection and other version information. This can be done in the Annotations field. When importing you can read these annotations and determine whether this is the correct version / collection / mdl file.



Also make sure to go to the advanced section to export the user defined objects (properties in this case).



Next is the actual import, since we picked the collection all objects in it are exported:



Step 4: Check it all into your versioning system

This step is simply taking the MDL file and checking it into the versioning system.

Step 5: Operations checks it out

Here you get it out of the versioning system after testing and then moving it to production.

Step 6: Operations runs the install of the version (this imports the MDL, deploys the entire thing, runs the mappings)

The last step is to import it all again. If your operations staff is well versed in Warehouse Builder they can use the UI to import. In general however you want to script this up into an install application.

The first script you will need is the import:

```
OMBIMPORT MDL_FILE 'c:\\mdl\\sales_relational_1_0.mdl' \
USE UPDATE_MODE \
OUTPUT LOG 'c:\\mdl\\sr_1_0_import_prod.log'
```

After this you can go much further and automate validation, deployment and starting of the mappings.

More commands can be found on OTN:
http://www.oracle.com/technology/products/warehouse/sdk/scripting%20pages/Scripting_3_services.htm

Alternatively:

If you are going to check in the generated code, you might want to work with the generation comments on each of the objects. You would do a preliminary step setting the comments so you track the change in the actual generated script. But in this case if you run the scripts, you will not have any Warehouse Builder audit trail.

Even more alternatively:

Take a look at snapshots. If you work with collections, you can snapshot them and get comparisons between the versions. This is quite handy and all available from within Warehouse Builder. You can even see property changes…

But you need more…

What is missing here is the answer to dealing with different environments between the test and the production systems in areas such as parallelism, tablespace names, compression, error handling in mappings and other physical changes to the system like the locations used.

Multi-configuration to the Rescue

The above works fine, until we start looking at deployments and locations / control centers and the above mentioned physical characteristics. Multi-config addresses that issue as part of Enterprise ETL.

Basics on Configuration

Configuration properties have been in the product for the longest time. In Warehouse Builder each object has configuration settings. These are not considered part of the logical design and can change per environment without affecting the logical design.

For example, a table has columns, indexes, partitions and is stored in a tablespace with parallel degree 8. The first three are logical design elements, the latter two are

physical characteristics. Since your production system is typically much larger than the development system you want to change the parallel degree. That is where the configurations are used.

Where is the Problem?

The problem comes when you have 1 configuration and 2 systems (simple case with just development and production). In order to get parallel degree 8 on your table you must change the object. This is a step you typically do not want to do when you are going live with the system since it introduces risk.

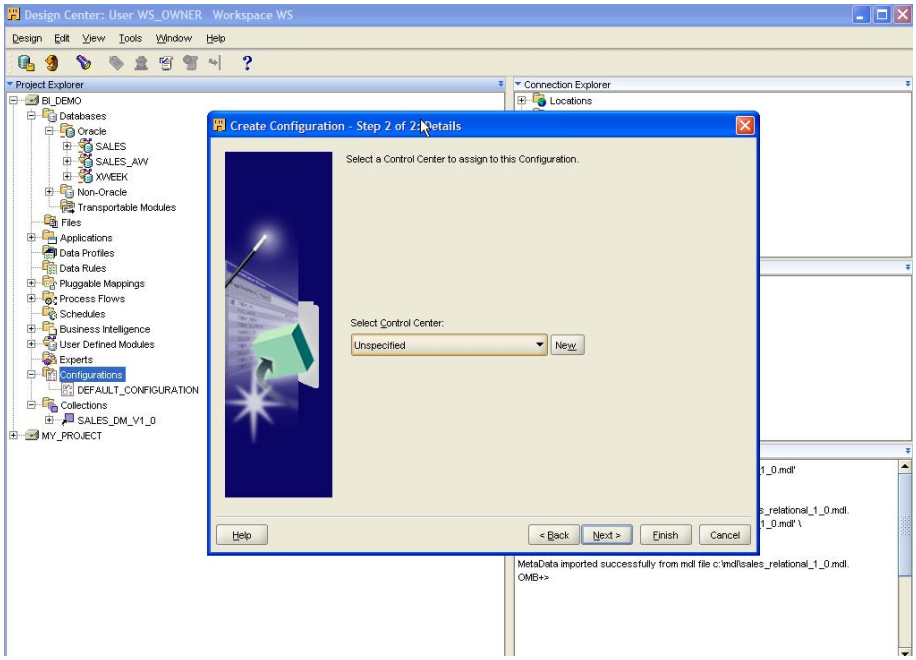
The solution is multiple configurations attached to the same object in each of the environments. This is of course Multi-configuration in Warehouse Builder.

A Small Example

In our scenario we have the two environments, development and production for the little data mart we discussed earlier. Here we are going to work with two configurations and will discuss setting this up.

Step 1: Create a new configuration and an associated control center

Create the new configuration in the Configurations node.

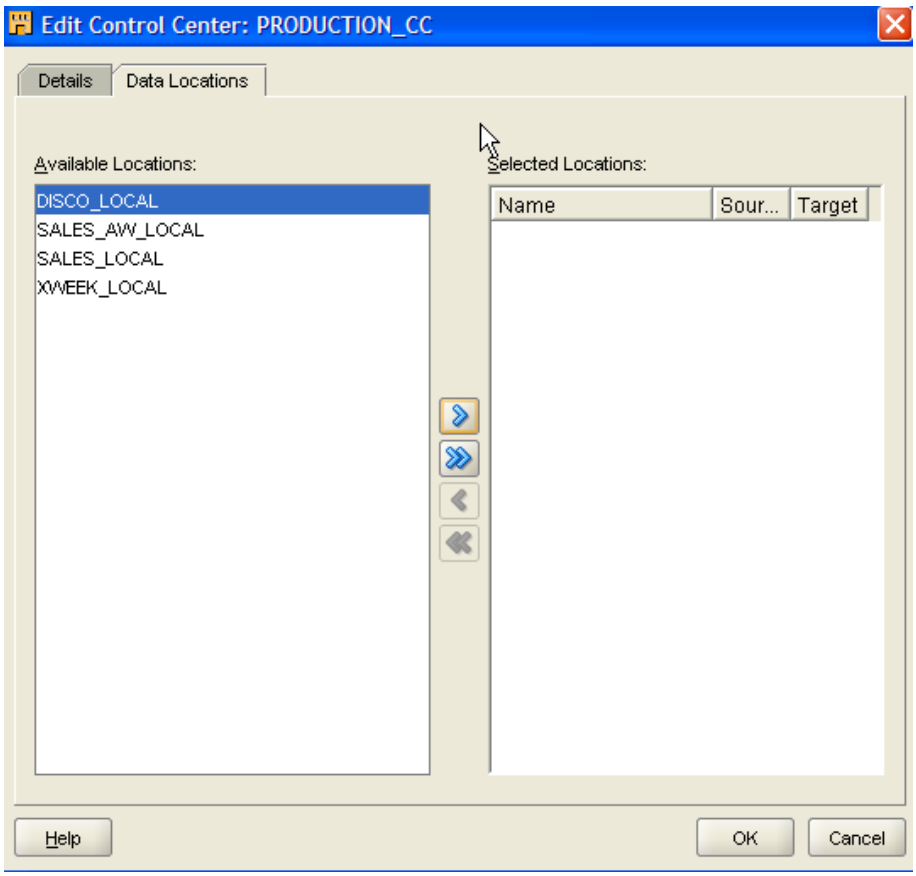


Here you should associate a control center, or create a new one. This links the set of locations to the configuration. It should point to the production system, even though you will not directly deploy there.

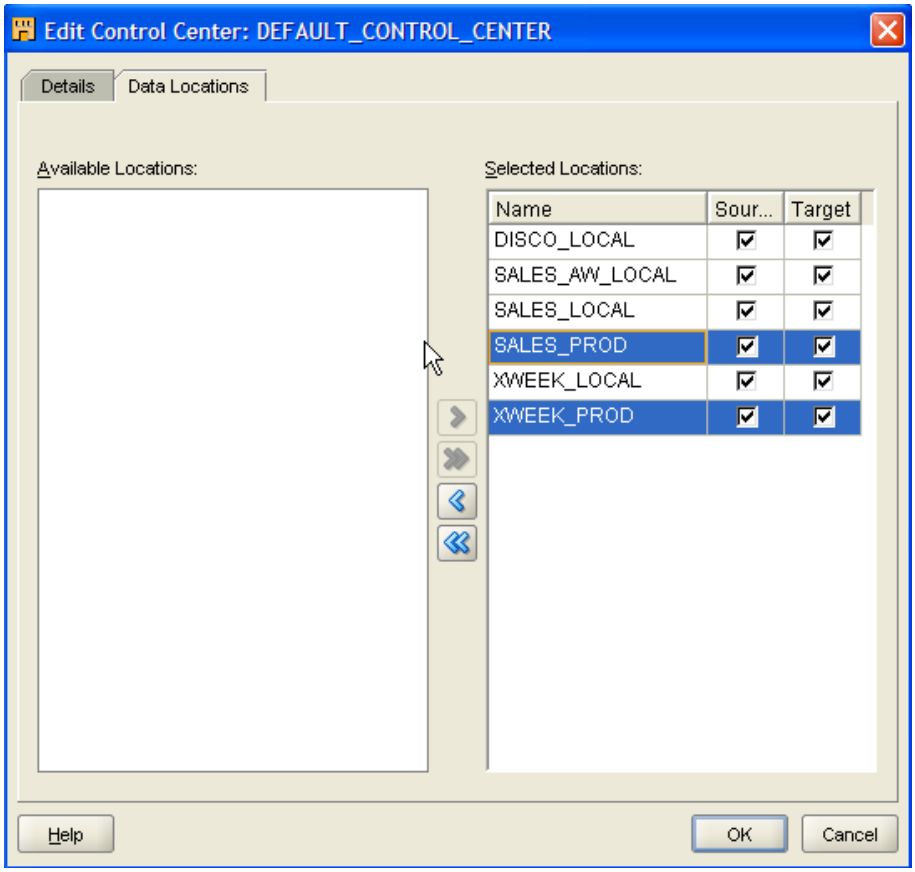
The reason for this is that when we first export this metadata we will include this control center and its locations. Then the administrator sets the passwords once and the system is up and running.

Step 2: Create the production locations and associate them with a control center

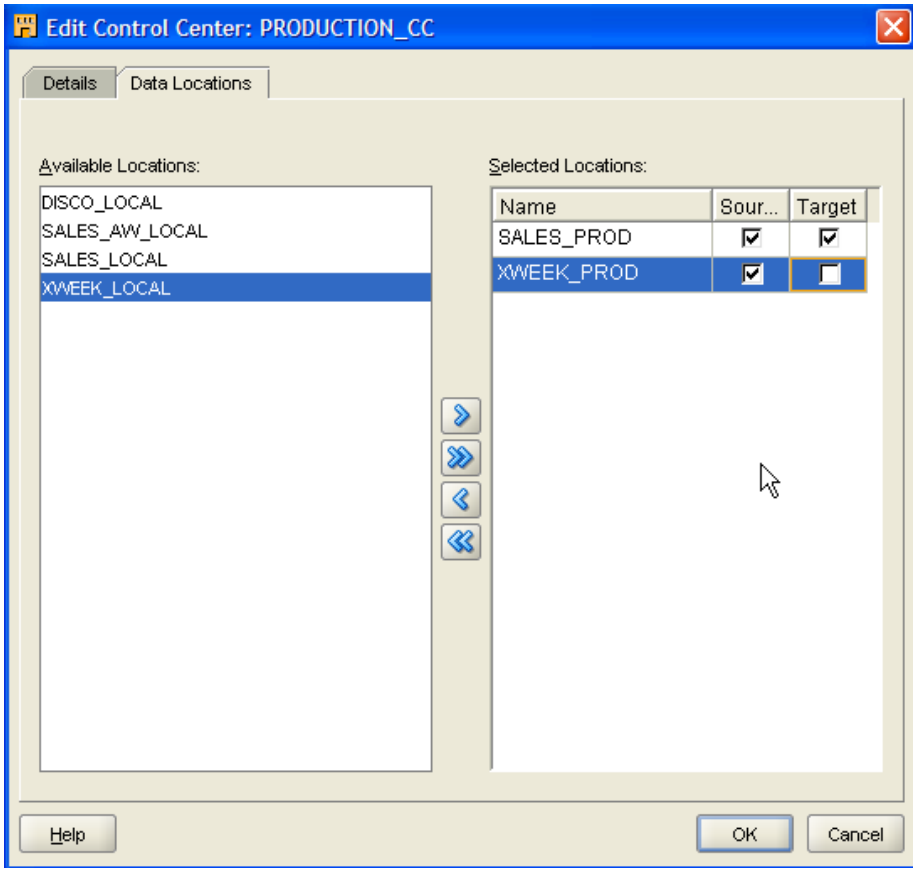
If you open the new control center, PRODUCTION_CC you will see no locations.



Now create a few locations in the Locations node in the Connection Explorer. Because the DEFAULT_CONFIGURATION is active the new locations are automatically added to the associated control center (DEFAULT_CONTROL_CENTER).



This is NOT what we wanted, so lets remove the first. Now go to the PRODUCTION_CC and add these locations to this control center.



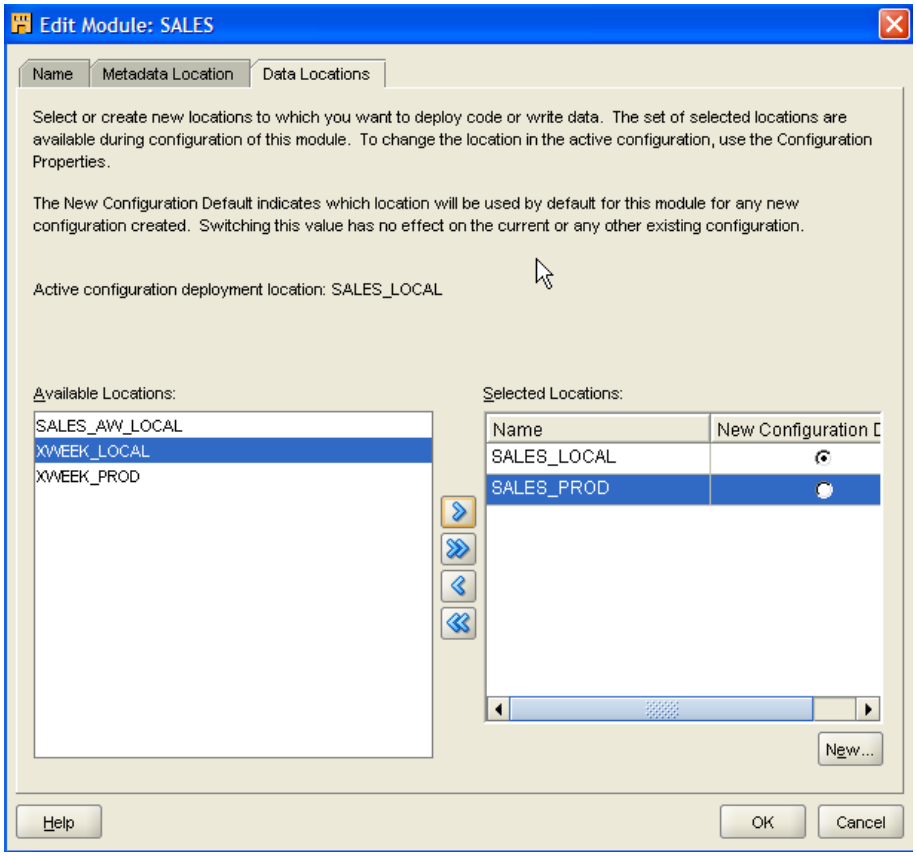
Now notice that we have unchecked the Target check box for Xweek, so now we can make sure we do not magically deploy to this target and overwrite the source. So in the control center manager you will see no objects to deploy… kind of cool right?

Now that this is set up, every time you flip the configuration to the production side, the control center is this PRODUCTION_CC and the locations are set to these production locations. But how does this reflect in the modules and objects? Lets look at that in the next step…

Step 3: Configure the objects

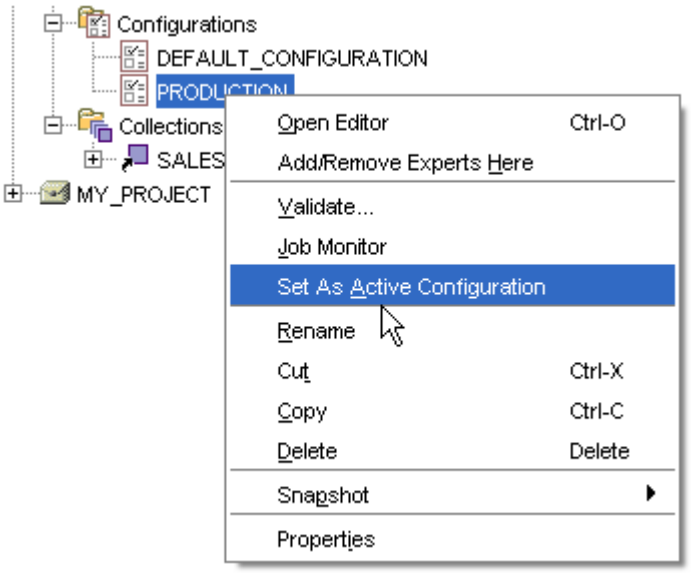
First things first, how do I get these locations to be reflected in the modules so that if I switch configuration, I switch locations for the right modules?

Edit the desired module to create a list of locations for the module and add the SALES_PROD location (we are editing the SALES module).

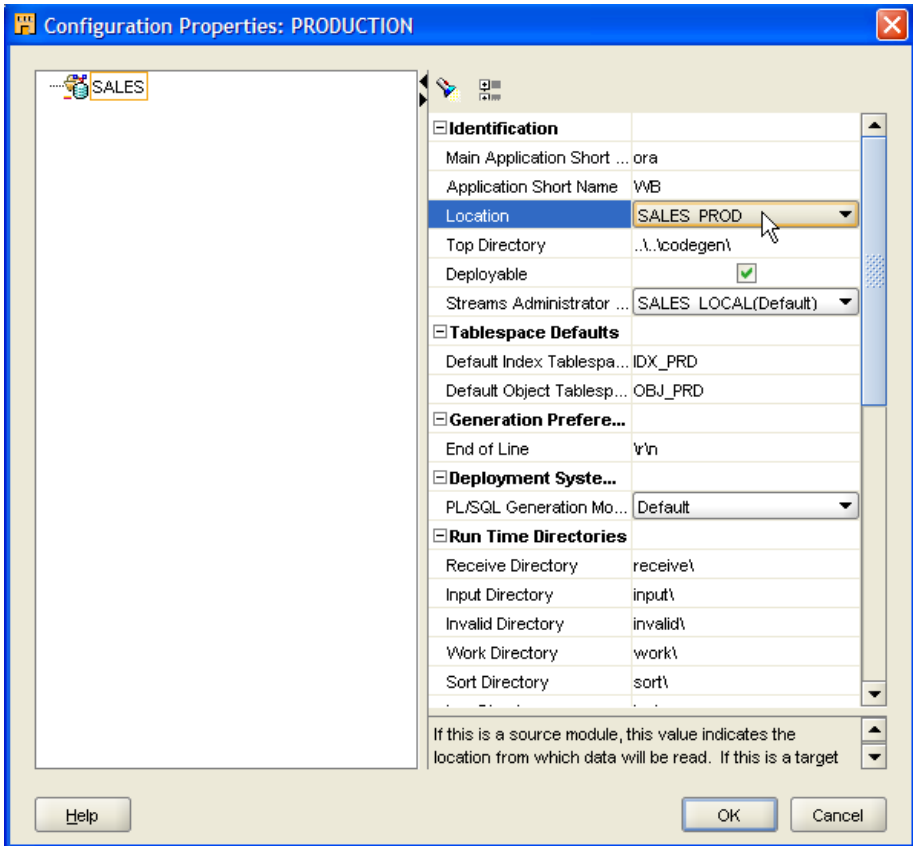


Same action should be applied for the Xweek location of course. Now that this is done we have the candidate list for the locations. Note that the little radio button is there to indicate which location gets defaulted to a new configuration. So if we were to create a TEST configuration, the SALES_LOCAL location would be picked up for that configuration by default.

To set the configuration for the module in the PRODUCTION configuration, we must first make that configuration active. Do this by right mouse clicking on the PRODUCTION configuration and choose Set as Active.



You will see this change reflected in the status bar of the product. Now, right mouse click on the module (SALES) and choose Configure. Set the location to SALES_PROD.



If you want to you can also set for example tablespace for all objects in the module at this level. Here we set these to IDX_PRD and OBJ_PRD. If you switch back to the DEFAULT_CONFIGURATION and configure the module you of course will not see these changes.

As you configured the module, you will also configure the objects. Rather than doing this one by one, consider downloading and installing the multi-configuration expert from the OTN exchange:

http://www.oracle.com/technology/products/warehouse/htdocs/Experts/multi_config.zip

This expert allows you to configure object types per configuration making this much easier to achieve than going by individual objects.

Solving the Puzzle

If you combine the multi-configuration with the steps around checking in metadata you are almost at the solution. One more tip to consider is that you need to set the default configuration for the repository to the correct configuration. If you import the metadata into the production repository, then setting that repository with a PRODUCTION configuration as default makes sense. This way you get the configuration set and you can directly deploy without manually or scripted setting of the configuration. One less thing you can forget…

SUMMARY

As you can see in this paper, Warehouse Builder has many interesting features for you to look into. Here we only showed off a few of the ones that are not so obvious in the product or that are relatively new.

You can do very advanced matching and merging in Warehouse Builder for a very low price point. You can also use the most advanced SQL available in Oracle Databases. On top of that you get to easily pivot and unpivot data to ensure better reporting structures for your end users (and avoid them using Excel everywhere).

For managing complex environments you can work with multi-configuration to make your life and your operations staff's life better and simpler. You can also avoid lengthy test cycles due to system changes.

Development's Bag of Tricks for Warehouse Builder
May 2008
Author: Jean-Pierre Dijcks

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