An Oracle White Paper
October 2010

Implement Right-Time Data Loading with Oracle Warehouse Builder 11gR2

ORACLE

Introduction

This paper will introduce various aspects of the Data Integration offerings from Oracle that will help you leverage the newest features for right-time data loading. This will cover a spectrum of capabilities from file loading, to Oracle Data Pump to Oracle GoldenGate and Streams, and how customers who license Oracle Data Integrator, Enterprise Edition (ODI-EE) can exploit them from Oracle Warehouse Builder 11gR2.

You will learn how to leverage different code templates and some of the newest ETL features in Oracle Warehouse Builder 11gR2 to build change data capture and right-time mappings.

A version of this paper was presented at ODTUG 2010 Kaleidoscope.

Right-Time Data Warehousing: The Time is Now

The landscape of the data integration domain is forever changing, tools need to be flexible and support ETL, ELT, push up, push down or whatever. The bottom line is that users want to integrate more than ever, faster than ever and make better use of the assets they have to perform the data integration tasks. The open connectivity framework in Oracle's Data Integration tools empowers customers to do exactly that. The flexibility provided by Oracle Data Integrator's Knowledge Module framework is shared with the 11gR2 release of Oracle Warehouse Builder (OWB).

Oracle Warehouse Builder is built on top of a metadata repository that captures your design. Once you have created your design in the repository, Warehouse Builder generates the code, populating built-in code templates with metadata from your design. The objects are deployed to one or more databases or other hosts, where the code executes.

Prior to the 11gR2 release, for data integration routines, the tool generated a combination of SQL and PL/SQL (plus ABAP and SQL*Loader), allowing you to perform complex transformations on the data you are moving. It was primarily focused on Oracle, files and SAP sources with heterogeneous connectivity to all other systems using the Database Gateways. ETL workload was generally concentrated on the target database hosts where the code was deployed.

This approach worked well for a lot of cases for Oracle database customers who needed batch ETL, but:

- You could never reconfigure the physical execution of the mapping to make better use of the assets at hand.
- You could never change the built-in templates, to support features that were not supported out of the box.
- You had to jerry-rig stop-gap solutions to meet new requirements and new technologies that appeared between OWB releases.

And there's no escaping it-- your business needs and the technologies that can solve them are forever changing!

Warehouse Builder 11gR2 is all about giving you new ways to use technologies and resources to stay ahead of the demands your business places on your data warehouse. This paper focuses on how Warehouse Builder 11gR2 can keep your data warehouse updated in scenarios where traditional batch ETL is not enough.

A Spectrum of Requirements and Techniques

The terms "real-time data warehousing" and "right-time data warehousing" (which, conveniently, share an acronym RTDW) are thrown around a lot, and with pretty broad meanings. However, a

practical way of looking at RTDW is that data freshness is *"… as real time as it needs to be… for the operational requirements of the business."* There are two main categories with regards to data freshness:

Pure Real Time ("True real time?"): In this case the warehouse should be loaded as part of the source system transaction. The performance overhead of having the data warehouse synchronization part of the atomic source transaction is most often unacceptable. It is questionable whether such real-time is necessary for data warehousing.

*Near-Real Time***:** Any mechanism to meet the right-time demands of the warehouse that is not pure real time can be referred to as near-real time. Within this, the time interval between successive warehouse refreshes could be wide and varying: in some cases the requirement may be to update the warehouse every 5 minutes or every hour, while in other cases it may be required to update the warehouse continuously –as soon as possible—without slowing down the flow of transactions in source systems.

RTDW has implications for the following aspects of your data warehouse and ETL design:

- Loading Approaches – There may be cases where it is required to load data as soon as possible after it is published by the source system.
- Consistency – Since the data being loaded is immediately visible to the users of the warehouse it is important the warehouse be moved from one consistent state to another.
- Design – The RTDW should be designed so that it can be efficiently loaded while still meeting the query performance requirements of the business. It may not be acceptable for indexes and aggregates to be recomputed with each load.
- Resource Management – The warehouse resources are shared by both loading processes as well as by queries. Resource allocation for the loading processes and the queries may be required to prevent either workload from overrunning the other.

However, the widely differing data freshness characteristics imply that *different loading approaches* and different design techniques may be required depending on the exact nature of the environment being implemented.

- Common data loading approaches used in a batch environment also have a place in most RTDW scenarios. These include:
- Full data into empty tables - applicable if source table is very small or very volatile (i.e. large part of the table changes between successive refreshes)
- Incremental data refresh - applicable in most cases where the latency is more tolerable and where the source table is large or not very volatile. For example, refreshes of the data warehouse every 5 minutes or every hour.

In either case, batch data loading may be subject to time constraints, where the faster the data can be moved to the target and loaded, the better; and for the Oracle database, flat files are often

the best way to do this; but to minimize the ETL developer workload, developers should be able to switch among data movement mechanisms without redesigning their mappings.

In addition a continuous trickle feed loading approach is required where data freshness as close to real time is required. This approach will form the basis for integration with EAI products which is likely to emerge as a critical component for data acquisition in most RTDW deployments.

Oracle Data Integration and RTDW

Considering the breadth of the definition of a RTDW, it is clear that there is no single technique appropriate for an arbitrary RTDW requirement. Accordingly, Oracle Data Integration tools provide a platform that enables the design of solutions around data movement patterns and technologies that can meet the full spectrum of freshness requirements facing our customers.

In Warehouse Builder 11gR2, several major enhancements contribute to this support:

- Bulk data movement and heterogeneous database support come from the new, open mapping code generation framework, based on Oracle Data Integrator knowledge module support
- Changed data capture mappings and change capture processes are now easy to build and manage within OWB
- Trickle-feed mappings that continually process updates as they appear can address the near-real time requirements.

OWB 11g2: Under the Hood

To understand the functional value OWB 11gR2 adds beyond previous releases, it helps to look at the architecture and how it changed between 11gR1 and 11gR2. There are changes in the design-time and run-time architecture that bring to OWB flexible, hot-pluggable support for new platforms and data movement mechanisms, without forcing OWB developers to learn a new mapping development style. The existing paradigm has been extended, rather than replaced, so the skills you have today transfer to the new infrastructure.

Architectural Framework - Designer

The 11gR2 release supports both the existing code generation capabilities and an open extensible code template and connectivity framework based on the Oracle Data Integrator Knowledge Module framework. Figure 1 below illustrates the changes from 11gR1 to the 11gR2 release.

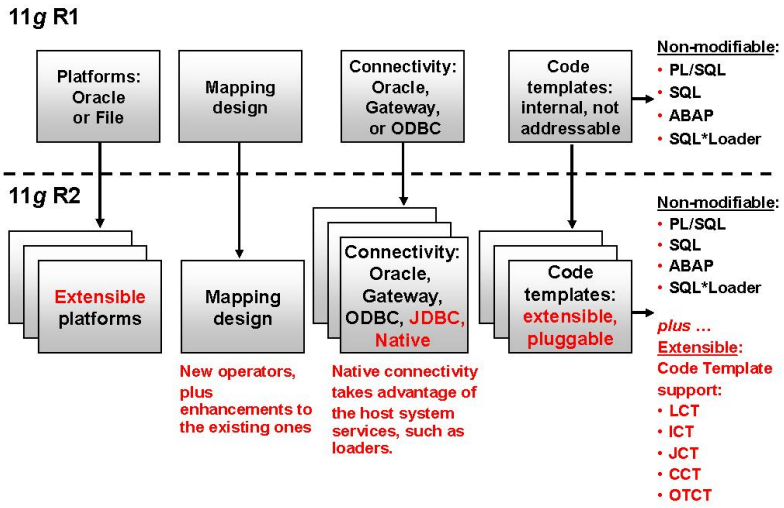

Figure 1. OWB 11gR2 Mapping Components.

This is a huge change in the product and opens the door to make it an extensible platform for integrating data. The framework can be extended with new platforms to represent new systems and has an open, extensible code template support that lets users:

- Have much greater architectural flexibility; you can apply technology at the right place and at the right time (push code to execute wherever it can be optimally executed).
- Use alternative data movement strategies (bulk extract, ftp, scp, piped etc.)
- Integrate with additional systems through the Open Connectivity framework (platforms and code templates)
- Provide implementations for standard design patterns for these systems (CDC, control etc.)
- Integrate database features that were not incorporated in the core product (for example, data pump external tables, pre-process files for external table) out of product release cycles
- Construct templates in a modular manner to share across common patterns.

The mapping design is mostly as before, a single logical design with a major shift that you can now configure the physical design and assign customizable code templates or assign the existing Oracle Target template(which allows some level of customization also, for example you could natively compile the package after deployment. The mapping can include a mixture of *both*, so you can leverage the flexibility offered for staging data in Oracle via the open connectivity framework then apply the complex transformation and data warehouse operators (for example) available in classic Warehouse Builder mappings. For example, you can leverage the CDC code templates for capturing changes and the data warehouse operators to build the warehouse.

The pluggable nature of the code templates is a great capability. Mapping designs can be designed once and tuned to drastically different physical realizations simply by assigning different code templates to the execution units. This lets you configure *how* data is captured, moved, checked and integrated.

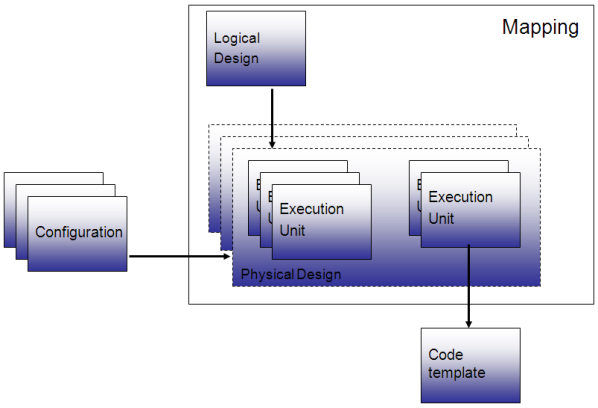

Figure 2. Logical Mapping – physical configuration to code template.

The figure above illustrates how a single logical mapping design can have many configurations, each configuration can have many execution units and each assigned a code template. The pluggable code templates also apply to modules which we will see later – code templates for CDC are assigned to modules.

Architectural Framework - Runtime

The code template based mappings are facilitated through the Control Center Agent, a program that can be run on a machine to orchestrate the tasks in the mappings. The agent itself could be placed on a source or transformation system for example if you wanted to unload data using a native un-loader. The code templates are based on the Oracle Data Integrator 10g Substitution Reference interface, this interface (its only an interface, you can't deploy OWB maps to an ODI agent) has been implemented based on the metadata OWB deploys for locations, connectors, mappings, web services and OWB Control Center Agent components.

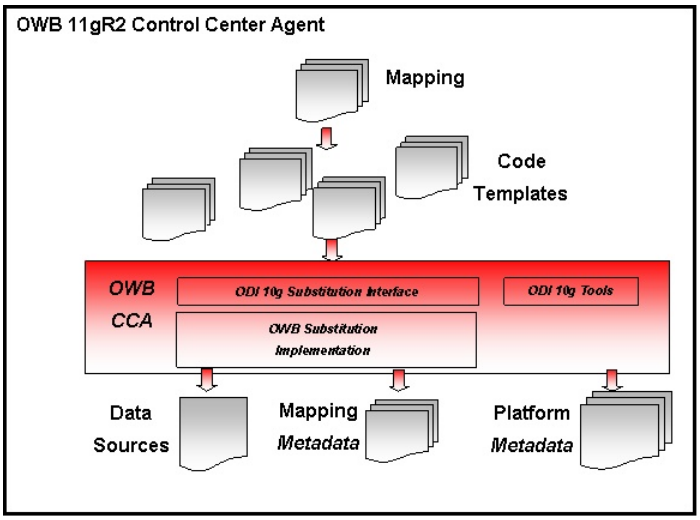

Figure 3. Control Center Agent.

The mappings are deployed to this OWB agent in a completely headless manner. Figure 3 illustrates the mechanics and dependencies of code template based mappings within the Control Center Agent. They are deployed as J2EE applications but are not heavyweight java applications - the application is basically a very small driver script and some metadata about the map. The connectors (from agent to source and target system) are deployed as J2EE data sources. The design paradigm is the same as before its just now you can get much more control of what capabilities of a system you would like to leverage.

CDC Mappings and Journalizing Knowledge Modules

The Oracle Data Integrator journalizing knowledge modules are supported in OWB as CDC code templates. These components help you orchestrate the process to capture changes in real time; the changes are then consumed in mini-batches. The heterogeneous framework includes certified templates for Oracle, DB2 UDB and Microsoft SQLServer:

- Oracle Consistent LogMiner (asynchronous w' HotLog)
- Oracle Simple and Consistent (trigger based)
- DB2 UDB Simple and Consistent (trigger based)
- SQLServer Simple and Consistent (trigger based)

The ODI-EE KM for GoldenGate is Oracle's newest addition, Oracle GoldenGate is a best-of-breed, easy-to-deploy product used to replicate and integrate transactional data with sub second speed among a variety of enterprise systems. Oracle GoldenGate provides the flexibility to move data between like-to-like and heterogeneous systems. The JKM integrates the GoldenGate extract and capture into the Oracle Data Integration journalizing framework.

Based on the template you choose, the data changes are captured from the sources and incorporated into the Oracle Data Integration CDC/journalize framework. ODI or OWB mappings are designed irrespective of load or increment and process the changed data in mini-batches, scheduled with the desired frequency.

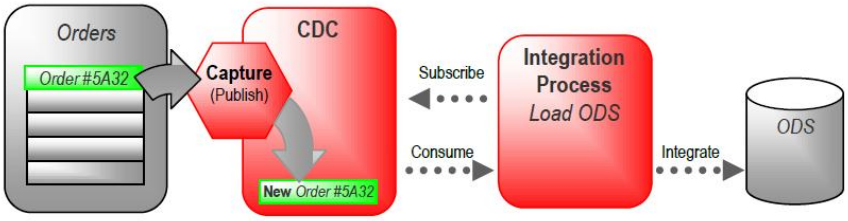

Figure 4. Change Data Capture framework.

OWB Advanced Queues for Streams Support in Mappings

The 11gR2 release introduced real time capabilities based on the Oracle Streams feature in the database server, wrapped up in the new Advanced Queue operator. Mappings that use advanced queues as sources can operate in a batch mode, where messages build up in the queue and are processed periodically, or in a continuous trickle feed mode (see Figure 5).

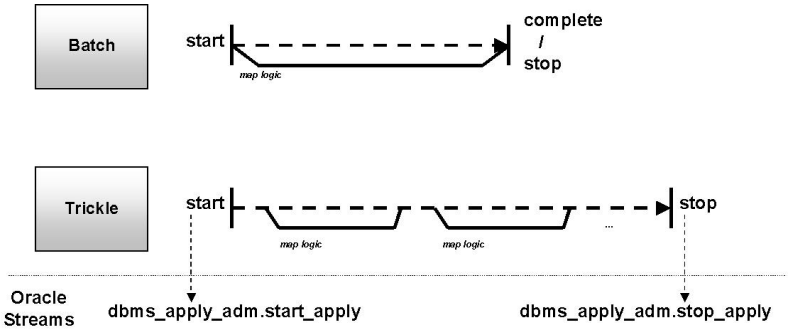

Figure 5. The different mapping execution modes.

Messages can arrive in a queue from a Streams environment capturing changes from a data source, in which case the message encapsulates a DML change record. Alternatively, a message can arrive in a queue due to an application publishing the message. In either case, the arrival of a message on the driving queue (where data trickles in) kicks off the rest of the mapping, (treated as a Streams apply handler). The result is continuous processing of updates in near-real time. The rest of the mapping can transform/load each message/event that arrives from the AQ source, and even write new messages to other queues to feed other systems.

OWB RTDW: Some Scenarios

With all that as background, you probably want to know what you can really do, better than you could do it before. Some scenarios will make this clearer:

More Data, More Efficient – File Staging Made Easy

The operator library in OWB is designed to *simplify* the complex task of integrating and transforming information. The code template support helps greatly here too, since we can now load information from more places than ever before and much easier than before. The logical mapping design we will look at couldn't even be designed in prior releases (users had to design the external tables), using code templates also provides fantastic opportunities such as loading sets of dynamically named files into the warehouse in a flexible and powerful manner. This mapping is an example of the hybrid style allowing both open connectivity execution units (limited set of operators) and classic OWB code generation execution units. One logical map design is realized into physical execution units each assigned their own code templates. The example is loading a sales cube. The cube operator performs surrogate key lookup and error handling automatically. This alleviates the user from doing all the repetitive lookups etc which would be needed if it were to be done using the base operators such as the table operator.

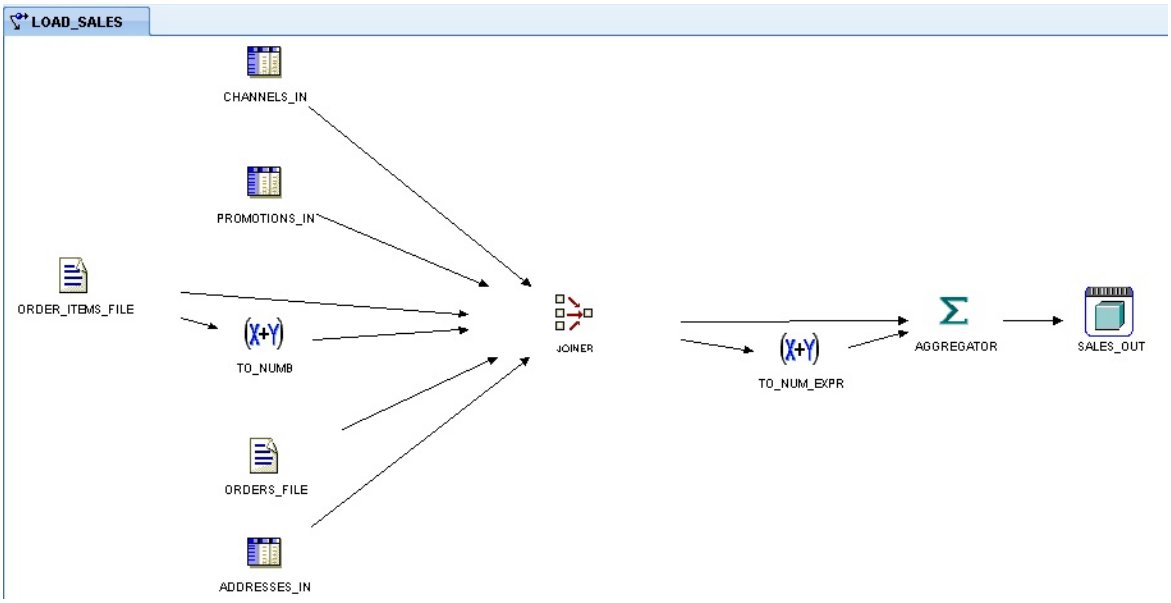

Figure 6. Sales cube loaded from a number of tables and files.

The mapping initially can use the file to external table load code template to stage the file in Oracle. This template is shipped with OWB and loads a single file via an external table. It is very easy to extend this to provide a number of valuable pieces of functionality;

- Integrate preprocessor into external table as an option in the template.
- Stage a number of files based on a regular expression, names may not be known at design time.

Let's illustrate the changes just mentioned. Based on the existing single file code template, we can enrich with these changes, parameters can be added to the template to support a regular expression for the file names and a preprocessor directive. These parameters are assigned values when the template is assigned in a mapping.

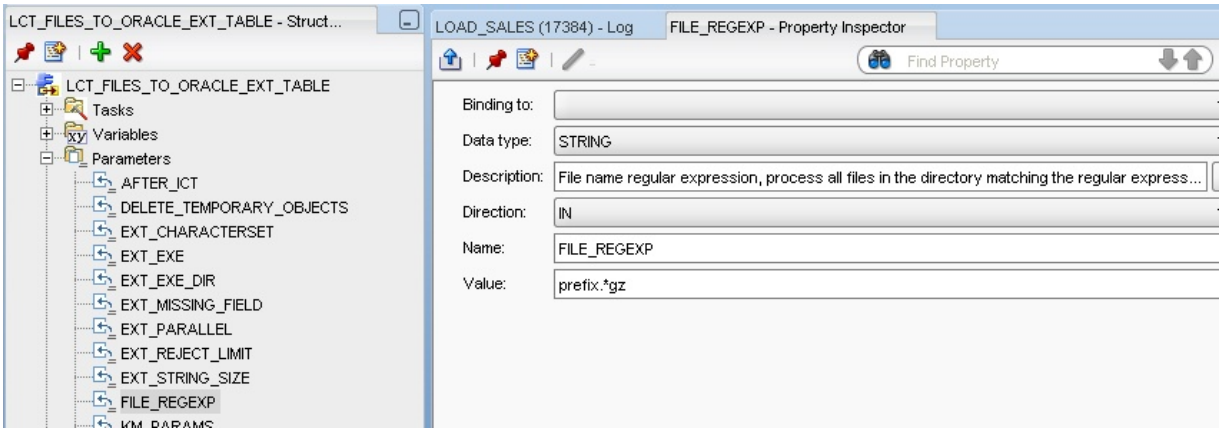

Figure 7. Parameters to the code template.

Just like in ODI, OWB can add pieces of Java into the template and use them from the template. For example let's add some java that returns a comma separated string of all of the file names that match a regular expression (that we will get from a code template parameter, just like an ODI option). Here is a very basic piece of java code that will return a list of file names that we can use in the external table definition.

LOAD_SALES (17384) - Log | Property Inspector | **Task Editor**

Description | Source | Target

```
1  <? public class extTabFiles {
2   public static String getFileList( String dire, String mtx ) {
3            java.io.File dir = new java.io.File(dire);
4            String[] children = null;
5            java.io.FilenameFilter filter = new java.io.FilenameFilter() {
6              public boolean accept(File fil, String name) {
7                return name.matches(mtx);
8              }
9            };
10           children = dir.list(filter);
11           String retString="";
12           for (int i=0; i < children.length;i++) {
13             if (i > 0) retString +=",";
14             retString += "'"+children[i]+"'";
15           }
16         return retString;
17       }
18 } ?>
```

Figure 8. Snippet of java to return comma separated list of file names.

This java method getFileList can then be used in the code template for the external table definition. Below see how we can then use this when listing the data file names for the external table.

```
LOCATION (<@=extTabFiles.getFileList("<%=odiRef.getSrcTablesList("", "[SCHEMA]", "", "")%>", "<%=odiRef.getOption("FILE_REGEXP")%>")@>)<%)%>
```

Figure 9. The code template task for data file names.

Now the code template can be used by the mapping, you will see the parameters to the template exposed in the execution unit view of the mapping designer. Below you can see where we use the code template and supply the regular expression for the files (all files with name *items.*gz*), so we will get all the UK files, all the US files for order items and so on.

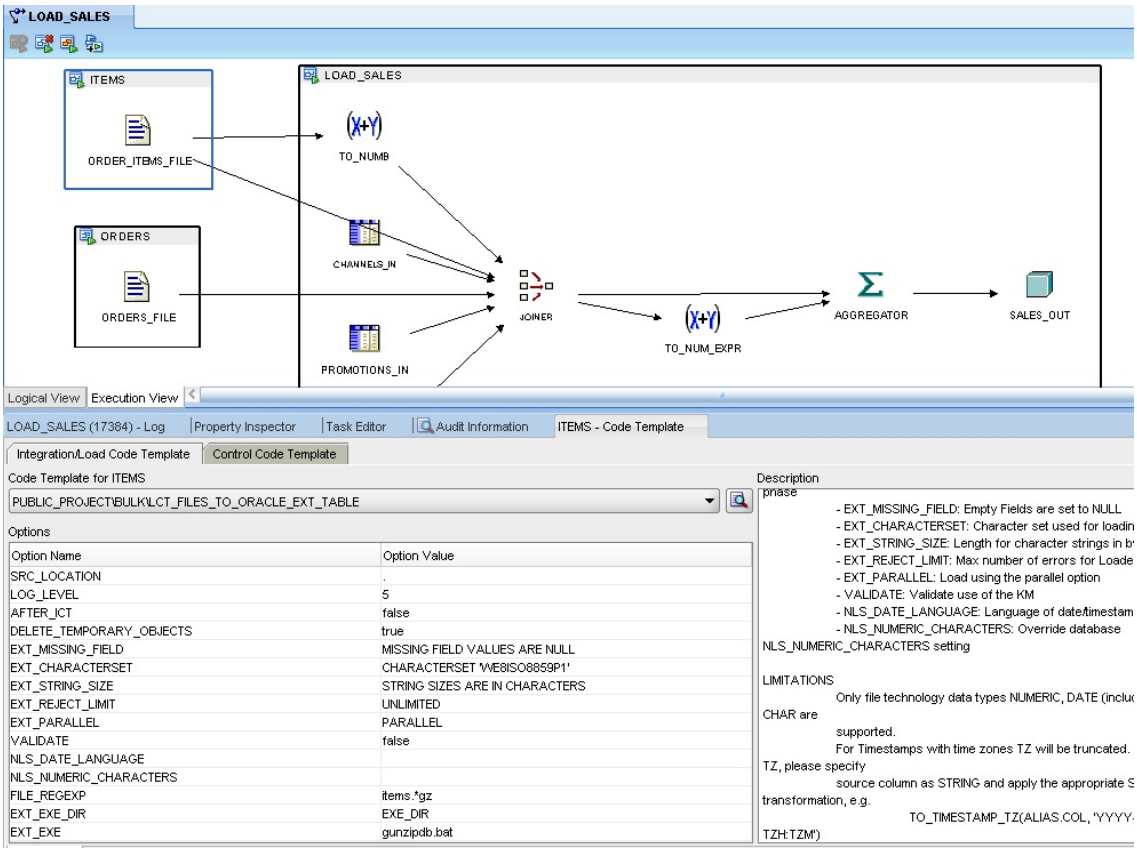

Figure 10. Applying the code template to an execution unit.

Note we are also using the gunzipdb.bat preprocessor executable which un-compresses the data file.

When the mapping is executed, the compressed files will be identified from the java snippet we added into the code template and an external table created to load these files, the preprocessor can uncompress the files and stream the data through the external table. This is a simple yet great capability for those who transport compressed files around their networks and want to load them directly into their database via external tables.

Bulk Data Faster

Pumping Data – Oracle to Oracle

OWB traditionally supported database links as the prime distributed data movement mechanism. This is a good choice for Oracle to Oracle but for certain scenarios alternatives are possible. Now with the open connectivity framework capabilities we can customize map designs to leverage such capabilities as Oracle Data Pump (provides high speed, parallel, bulk data and metadata movement of Oracle database contents).

Logically the map design remains the same, just different realizations of the implementation; let's look at how this is done using code templates. The traditional mappings (PLSQL, SQLLoader, and ABAP for example) are created under the *Mappings* node in the designer tree. The user has no capability to change the code generation template for these cases; it uses the built-in code generators.

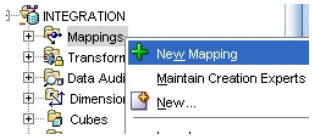

Figure 11. Traditional mapping under Databases node.

A mapping created under Databases - > Mappings has no 'Execution Unit' view that is available for the code template based mappings, the OWB code generators resolves the mapping and generates (for PLSQL) a number of different types of code – set based SQL statement for the entire graph and PLSQL code which is row based and *critical for processing rows in order*.

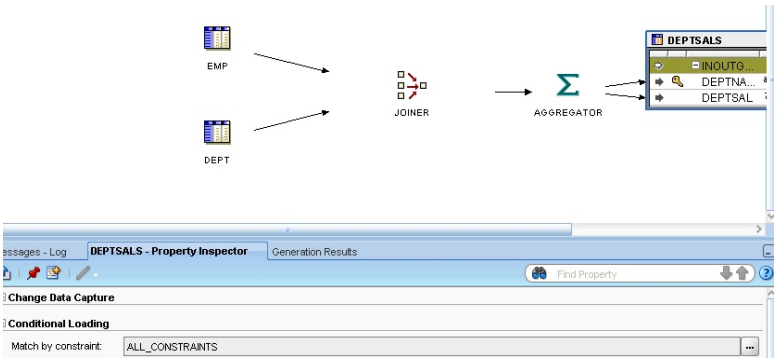

Figure 12. A mapping with no execution unit view.

This map in a distributed scenario where EMP and DEPT are remote will generate SQL using database links.

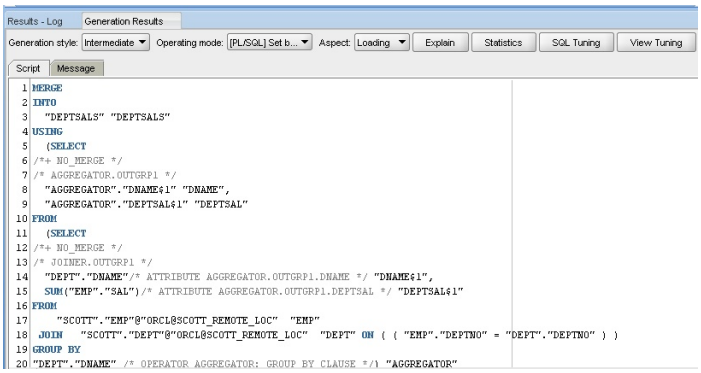

Figure 13. SQL Merge statement generated using database links.

Notice the database link code generated in the mapping, for distributed mappings this used to be the only option.

This mapping design can be taken and its implementation customized to use an alternative data movement mechanism, this is a simple copy-paste exercise, from the mappings under Databases to code template mappings. The mapping is copied from the Database->Mappings node and pasted into a code template mapping folder.

When the mapping is pasted into the code template mapping the code generation will remain the same, the same code generator will be used and a special code template 'Oracle Target' code template assigned. Now though we can change the execution units and put operators into units and change the code template. The figure below illustrates the basic SQL to Oracle template that moves data from any ANSI-92 SQL system to Oracle.

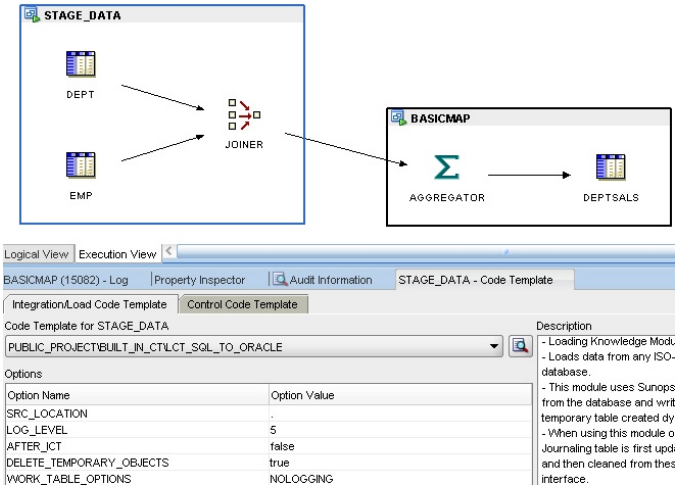

Figure 14. Mapping using simple load code template to debug.

The mapping can be tuned, we can choose different templates. For example the Oracle to Oracle DataPump load code template creates an external table on the source and the SQL query

represented by the source execution unit is data pumped to a compressed file when the map is executed, then the file is transferred using DBMS_FILE_TRANSFER to the stage/target where an external table on the data pump file is created.

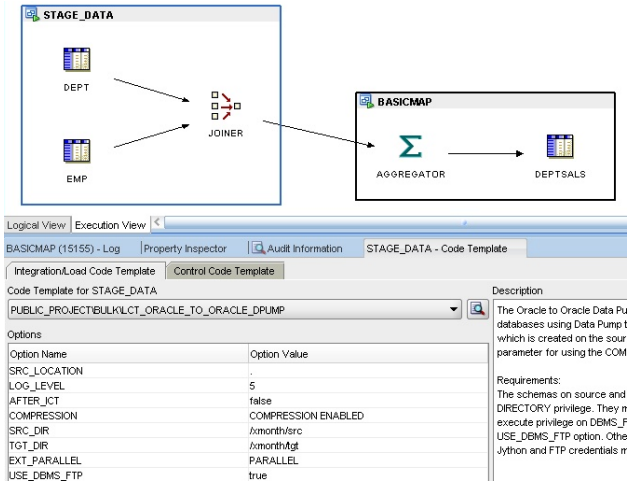

Figure 15. Code template mapping with Data Pump.

With this approach mapping designers are shielded from changing the designs of the mappings and it's a configuration, tuning activity to define the timing of how to integrate the data. They can get fast parallel extraction where the data resides, compressed data transport across the network and efficient staging using Oracle Data Pump.

Heterogeneous Bulk Data Faster

The code template architecture in OWB leverages the ODI 10g Substitution Reference and the Tools Reference, new templates can be constructed to leverage practices that are suited to you, to leverage the technology and tools you already have. To illustrate a heterogeneous batch data movement pattern has been implemented that leverages existing bulk unload utilities from the likes of DB2 or SQLServer for example. The logical map design remains the same just the physical configuration of the solution changes.

For example with Microsoft SQLServer, the bulk unload could be performed by using the *bcp* utility with the *queryout* primitive.

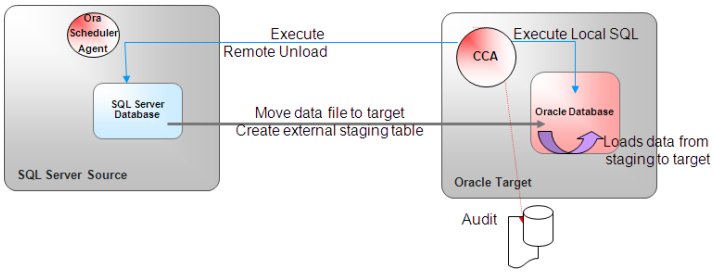

Figure 16. Bulk extract and load using SQLServer and Oracle.

By pairing the system specific unload with a generic staging approach a pattern is developed where new unload utilities can be plugged-in that pair with the staging design. So for Oracle external table on an export file is a great staging point. The Oracle Scheduler Agent is a lightweight remote program that provides remote job execution and file transfer capabilities that works great with the Oracle database, and ideal choice for the task.

This means we can isolate the extract part and plug in extracts for DB2 UDB, MySQL etc and perform bulk extract and load very well for Oracle, so by replace SQLServer with some other system we can do unloads using command line utilities like bcp with SQLServer or with SQL such as the 'SELECT … INTO FILE…' from MySQL.

For example in the mapping below the mapping remains the same whether the data is moved via basic connectivity or bulk data movement, simple a different code template is assigned.

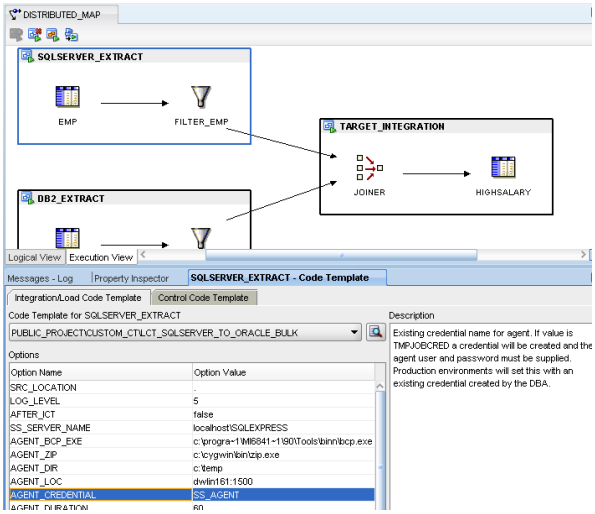

Figure 17. Example mapping moving bulk from SQLServer to Oracle.

The code template parameters or options can be set, and in the graphical pane the operators can be moved from one execution unit to another, so it's possible to drastically change the execution plan by moving them from one unit to another. *How* the data is extracted and moved is determined by the code template. You can see that these techniques will help you design the mapping logically, and tune, configure to meet your requirements. The best part of all is the ability to integrate and leverage the existing tools and technologies you *already* own.

Change Data Capture (CDC) Mappings Made Easy

Batch Mode CDC

Building a CDC mapping is much like building any other OWB code template mapping, but involves a few extra steps to select the change movement technology and the CDC tables, and manage the CDC process.

The source module where the tables reside must be assigned a CDC code template. The 'CDC Code Template' tab in the module's editor is where the code template is assigned. If there are any options/parameters for the template they will appear in the bottom section of this panel – for example the log level and validation options are commonly defined here.

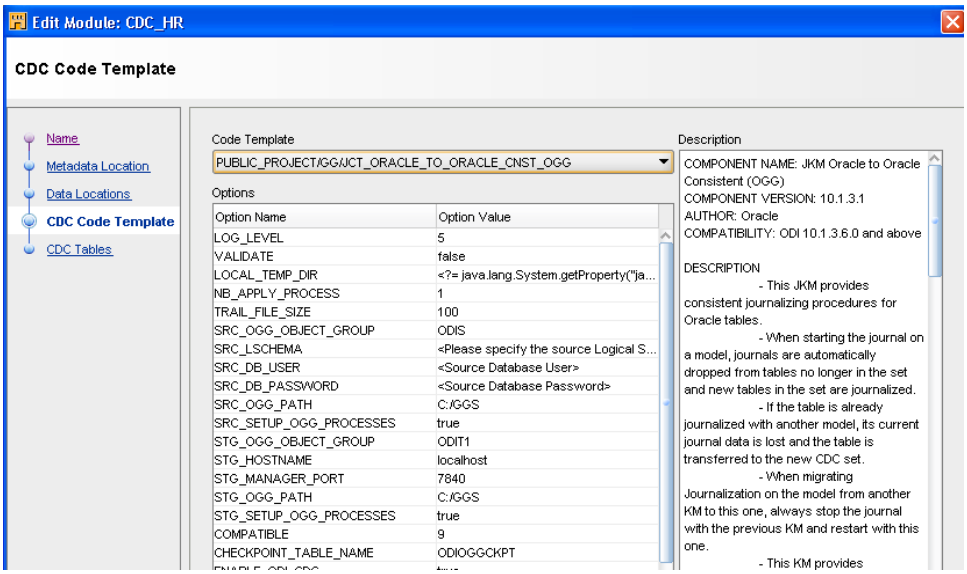

Figure 18. Applying the Oracle GoldenGate CDC template.

After the template has been selected (this is the *how*), the next phase is to define *what* we want to capture. This is done again in the module editor, using the 'CDC Tables' tab, below you can see we are tracking changes for the JOB_HISTORY table in the HR schema which is represented by the CDC_SRC module in OWB below.

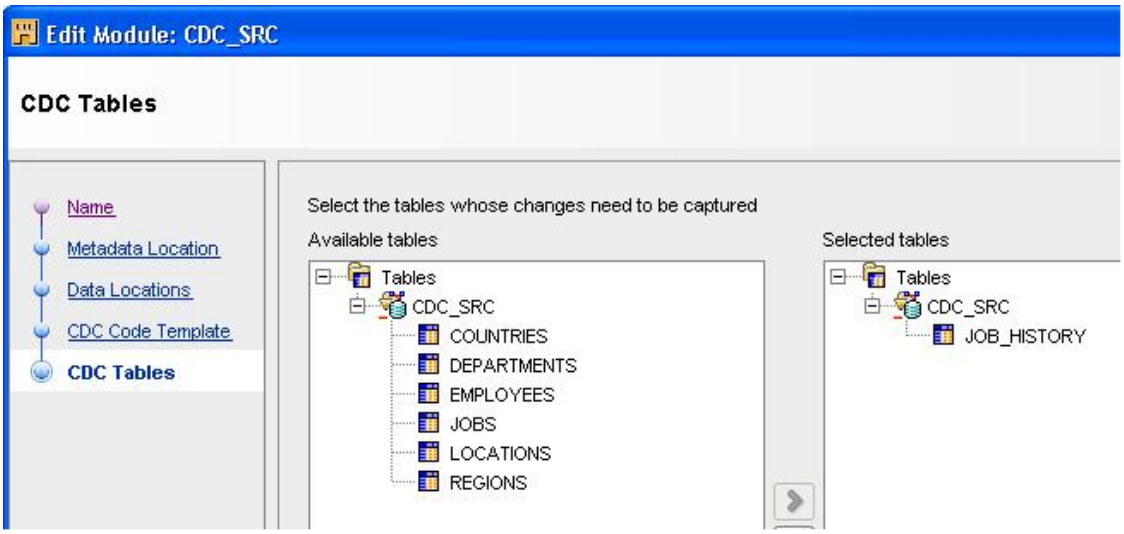

Figure 19. Select the tables to capture changes.

When a CDC code template is assigned to a module, the module has new capabilities available; the CDC start/stop, subscribe/unsubscribe, purge, extend and subscribe lock/unlock CDC capabilities are enabled on the context menu of the module. To incorporate the CDC operations

in a production system, obviously operating from the OWB designer is not an option, so the operations can be exposed as web services and utilized, for example, from a process flow, BPEL flow or any web service consumer.

Given the task involved there is a big difference in the prerequisites for a basic trigger-based template compared to LogMiner or GoldenGate. For example the trigger based example is very simple and works out of the box with minimal administrative operation. Log miner and Oracle GoldenGate have more administrative interaction. For example when the start CDC is executed for the Oracle GoldenGate component, obey files are created for the source and staging system that will be executed using the Oracle GoldenGate command interpreter, and you may have to deploy these manually on the source and target systems.

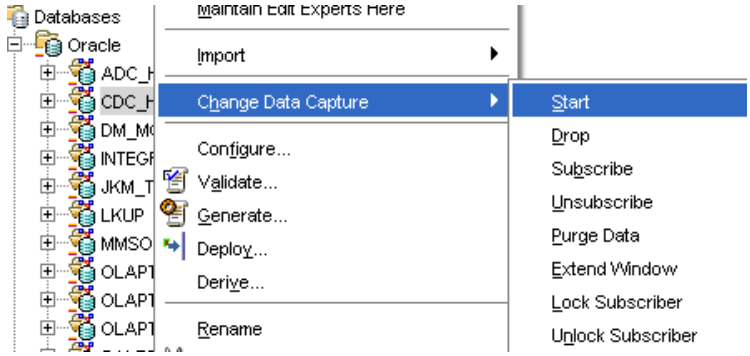

Figure 20. Starting the CDC capture from the module.

Subscribing to changes and starting the CDC capture from OWB (see Figure 20) for some tables using the Oracle GoldenGate code template generates a series of obey files just like with ODI. These files need to be taken to the source/stage environments where the GoldenGate processes will be run.

When the CDC process has been started, now set up subscribers - a change can be consumed my multiple subscribers. The figure below illustrates adding a subscriber named JOBCHANGES; this name will be used when we build a mapping.

Figure 21. Add a subscriber.

Simply type in the name for the subscriber in the table and press OK, to define the subscriber.

When a table is identified as being a table for capturing changes it can then be used in multiple modes within the mapping editor. When the table is added into the mapping it is added as a regular table with its columns. The table operator has a property 'Change Data Capture -> Enabled' that when switched on will add a number of pseudo columns containing information about the change (the subscriber, the change date and the flag for insert, update or delete). When the table is used in this mode only the changes will be processed by the mapping. The JKM framework stages changes in tables with the prefix J$ by default. In figure 22 you can see where we enable the capture for the table operator JOB_HISTORY and also define what the subscriber filter is.

Figure 22. Enabling CDC on the table operator.

The table is used just like any other operator; the framework treats it quite differently since it processes the changes being capture by the CDC process we initiated.

Here we see the table operator with CDC enabled and the pseudo columns for the date, the flag indicating type of change and the subscriber name.

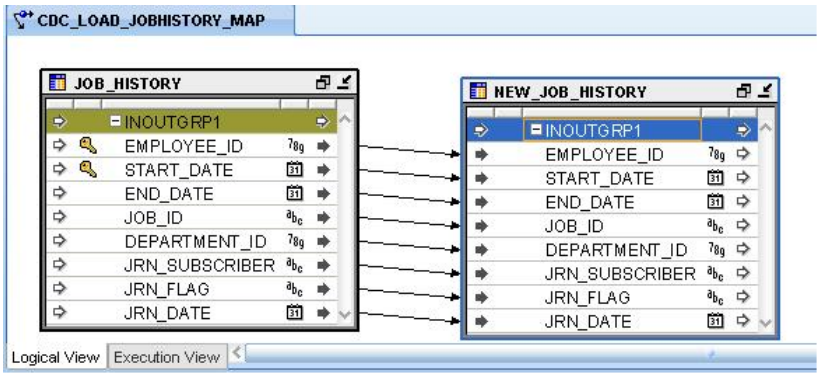

Figure 23. Defining the mapping, how to load and transform the data.

In this example, we have simply mapped columns from table to table but we could use any transformation available in OWB here.

The execution units below show the extraction from the changes and the integration to the target. Each execution unit is assigned a code template. By picking a different template, or building your own, you can customize anything from how the data is extracted and moved to how it is integrated, or anything else you can dream up in between.

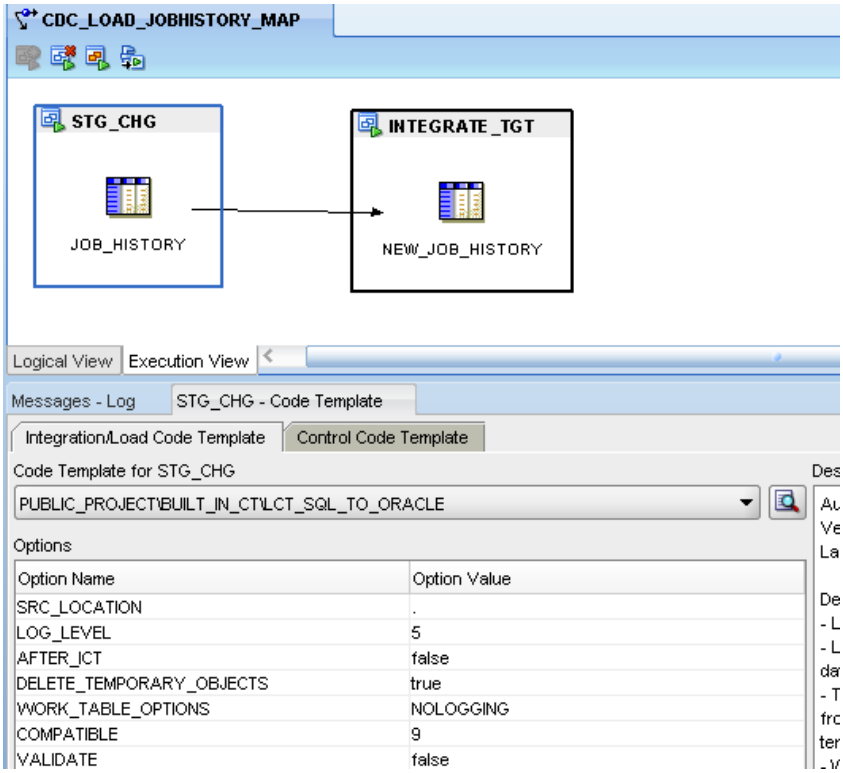

Figure 24. Assign code templates to the execution units.

To invoke the CDC related calls required in a production script/program you need the CDC web services for the module. If you click on the module's context menu you can create web services for the module, this option is available after a CDC code template has been assigned to the module - not exactly obvious but essential and very useful. These web services will let you setup the CDC via process flow or Oracle BPEL for example.

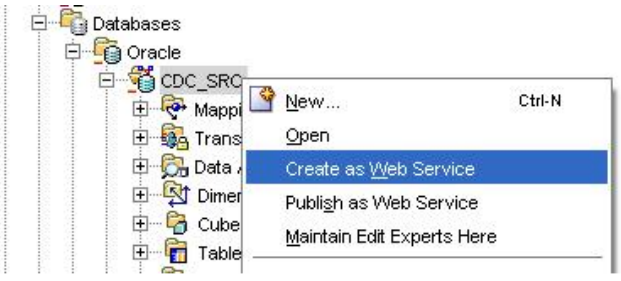

Figure 25. Create web service for CDC operations on module.

You can select a J2EE container such as the default control center agent and use this as the container where the web services will be deployed.

The web services are created for managing the CDC operations.

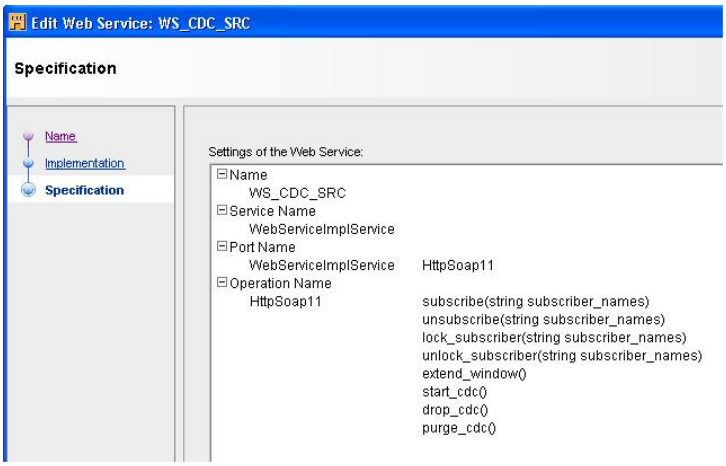

Figure 26. CDC web services can be deployed to a J2EE container.

These services can then be used in a process flow to orchestrate the CDC extend window, lock/unlock and purge calls before and after the mapping.

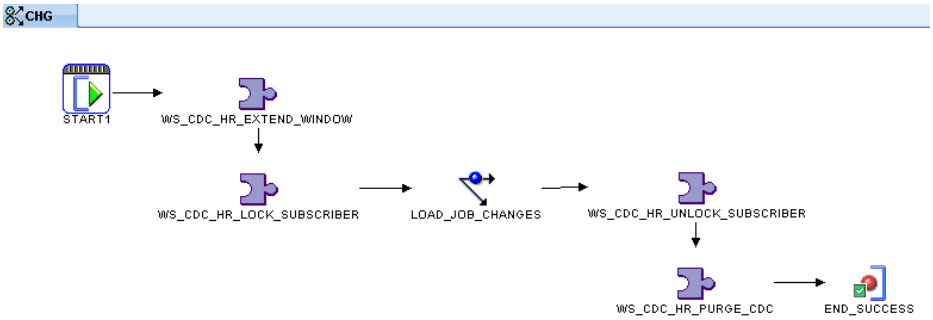

Figure 27. Build a process flow controlling CDC window and record processing.

The web service support in process flow is new in OWB 11gR2 and allows process flows to both consume web services and be exposed as web services for integration in a SOA architecture. In the figure below you can see how the process flow can have a variable defined and the result of the lock subscriber activity is assigned to the variable, the subscriber name parameter is set with the value JOBCHANGES, which we defined earlier.

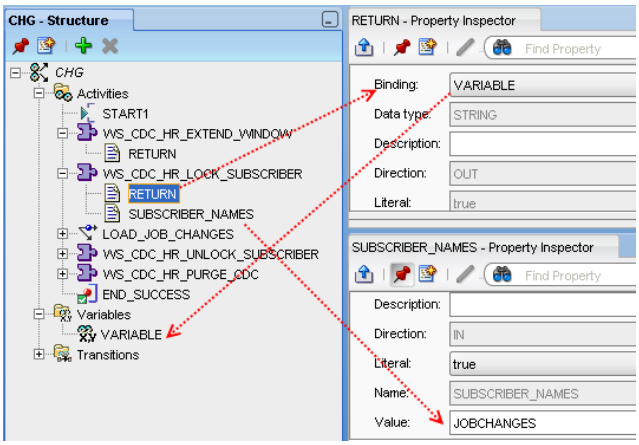

Figure 28. Assign the web service results to local variables.

Now to get to the fun stuff, if we insert some changes into the job history tables, these will be captured by whatever CDC implementation we have set up; GoldenGate, database logs, triggers and so on.

```
insert into job_history values (200,'01-JUN-2004','05-AUG-2004','SA_MAN',90);

insert into job_history values (200,'01-JUN-2005','05-AUG-2005','SA_MAN',90);

commit;
```

If we start the process flow we can see what happens. When the process flow is executed the log window from the designer will show the processing as it executes, in figure 29 below we can see the log window, this gives is a progress panel we can monitor execution from the designer.

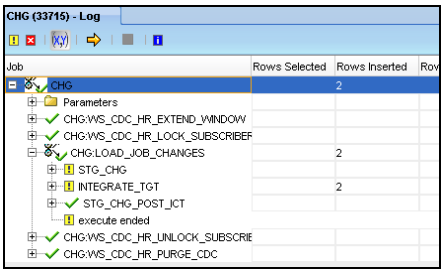

Figure 29. OWB Designer log window.

The place most time is spent is in the Audit Information panel is the place to go see detail of individual steps - just like the ODI Operator is displays the individual commands in template form and substituted form, so you can see the internals of what is done and when.

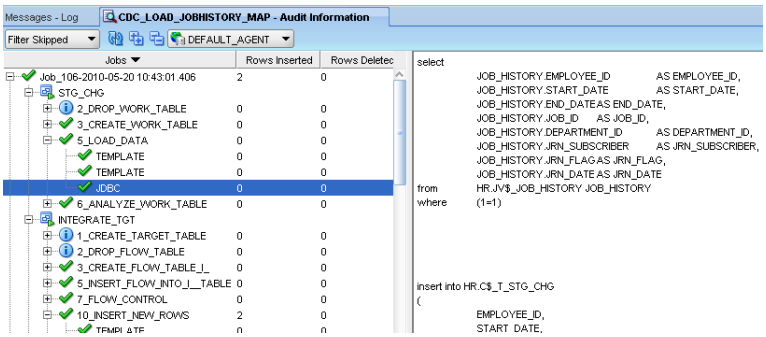

Figure 30. OWB Designer audit window.

We have seen how to setup Change Data Capture in the OWB designer and how simple this has now become. We have everything we need to monitor the execution within the Designer. For production environments this information can be queried from the public runtime audit views or the runtime browser, below we see the details for the process flow within the browser, where we can monitor, start and manage the jobs in the system.

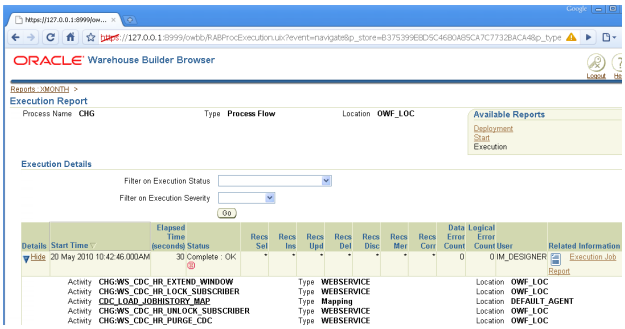

Figure 31. OWB Browser execution details for process flow.

Building an OWB Trickle Feed Mapping

The 11gR2 release of OWB includes a new trickle feed mapping mode where the mapping is effectively running as a listener on an Oracle Advanced Queue, the mappings can also execute in a batch to mode extra en masse from the queue. The basic AQ operator let's you read (and write) the structured payload from the queue, and there is special handling for Streams capture process message formats, so if you need to get down to this level of replication and process messages in order then this is the ideal technology.

The OWB database import has been enhanced to import database queues and queue tables, these can also be designed within the tool. When a Streams (for example) queue is reverse engineered, it is imported with the SYS.ANYDATA payload type. When you add a queue into a map, in the wizard you must select whether the queue will be processed in batch or in real-time. Then if real-time, you can either have a user-defined payload/message format or a capture process message format (see Figure 32).

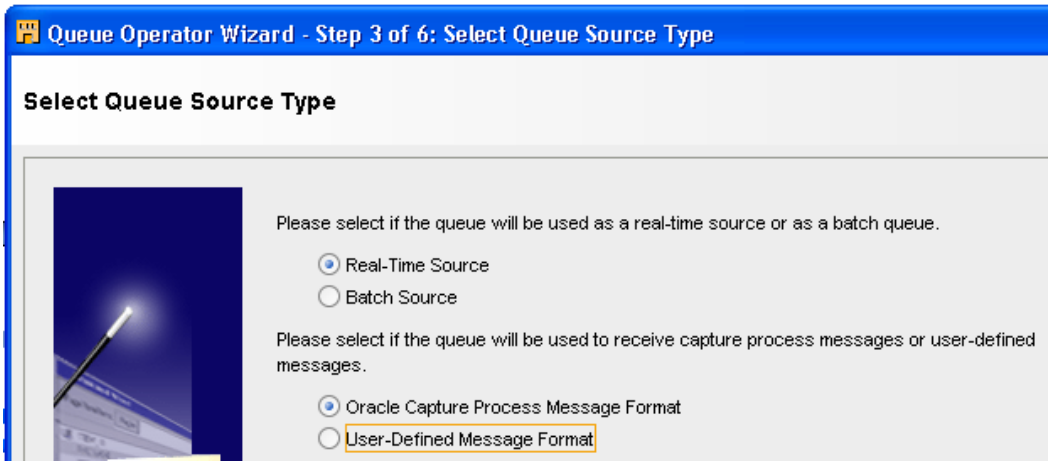

Figure 32. Queue wizard, selected real-time and LCR.

If you select that the payload of the queue is an '*Oracle Capture Process Message Format*' then the next step is to specify the table definition that the message is based on (see Figure 33). The message will be a well formed message (an LCR, logical change record).

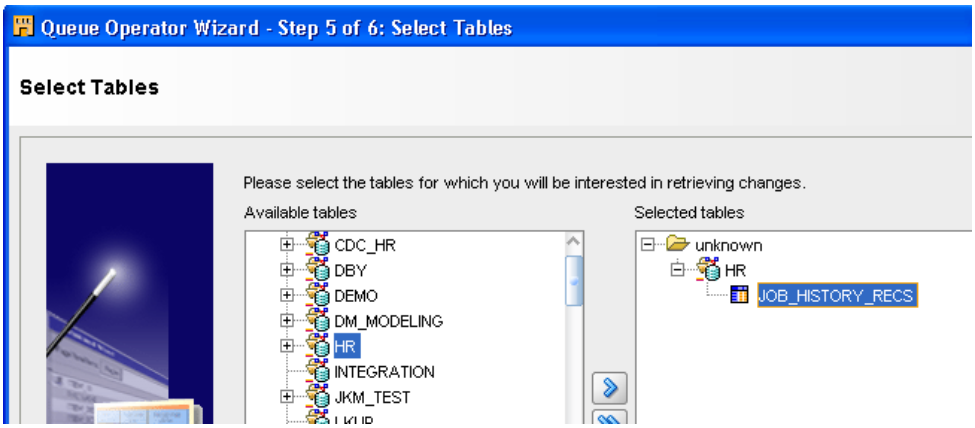

Figure 33. Selecting the source table for tracking change.

Of a user-defined message format was selected you can pick a scalar, object type, any data or even XML for example, then process this within the mapping.

Then for each table selected, define which changes to capture; inserts, updates or deletions. This will impact the operator definition. If all three are selected, then the resultant queue operator will have a group for inserts, a group for updates and a group for deletes.

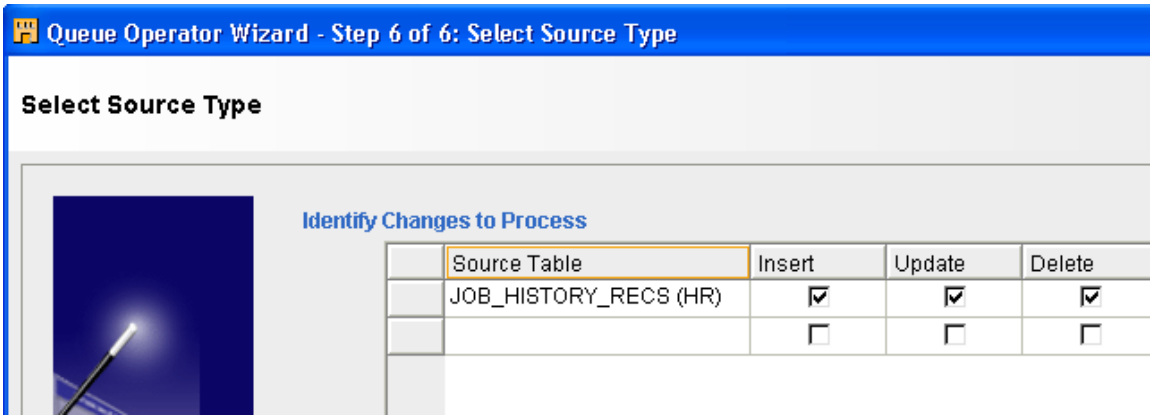

Figure 34. Select the changes to process.

Each change type will have a group in the queue operator; the group can be read from in the mapping. For example, you can target the inserts to a table with INSERT loading type, and the updates to a table with UPDATE loading type and so on. In Figure 35 below we see the insert group simply has the columns of the table we are capturing changes from, the update group has both the old values and the new values.

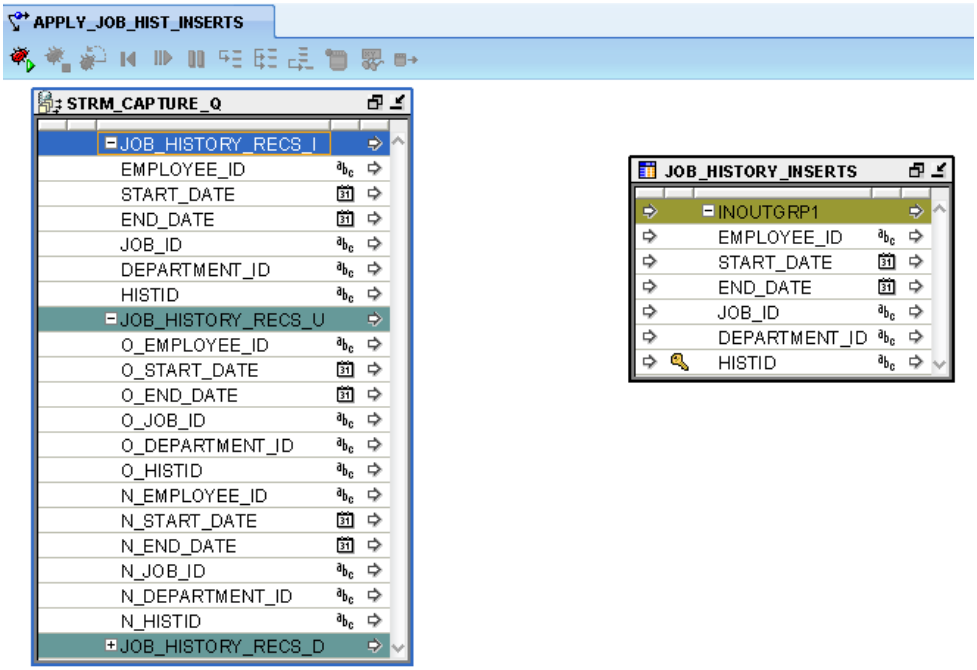

Figure 35. Map the insert group to a target table.

The mapping is deployed just like any other object in OWB, the difference is when the mapping is executed it does not just run and complete, it executes listening on the queue, processing information as it is made available on the queue.

Executing a trickle-feed map will allow an operation to be specified, the operation should be the name of the mapping.

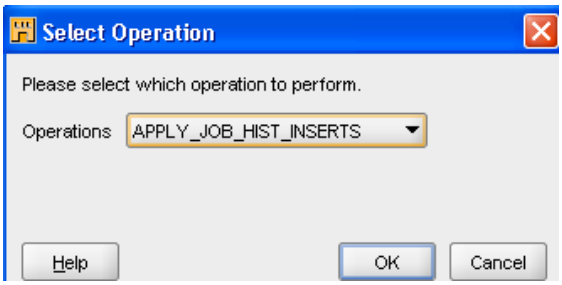

Figure 36. Start the mapping, select the operation with name of map.

When the mapping starts, it changes to a busy running mode, the log window displays a busy running symbol next to the mapping.

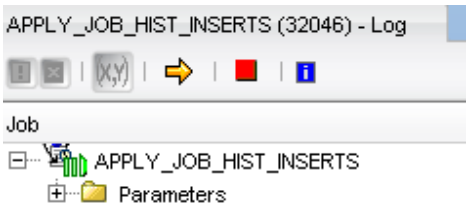

Figure 37. Mapping process is running and listening in trickle-feed mode.

Before we insert into the source where the changes are being tracked, check the target table, currently it has no rows in it;

SQL> select * from histrecs_i;

no rows selected

When we insert a row on the source table that we are capturing changes for using Streams, we can see the row is replicated and applied to our target.

SQL> insert into hr.job_history_recs values (102,'18-AUG-2007','31-DEC-2008','IT_PROG',60,1000);

SQL> commit;

Inserting a row into the source will cause the data to be captured in the database log files, the capture process of Streams will place the changes in a queue, and the OWB trickle feed map in the example will apply the change. Here we see the target table with the replicated row (see Figure 38).

```
SQL> select * from histrecs_i;

EMPLOYEE_ID  START_DAT END_DATE  JOB_ID     DEPARTMENT_I HISTID
-----------  --------- --------- ---------- ------------ ----------
102          18-AUG-07 31-DEC-08 IT_PROG    60           1000
```

Figure 38. The source row has been replicated into the target.

The mapping is constantly running when running in trickle feed mode, apply logic to the changed records. The start and stop execution activities start and stop the listening process and the changes are consumed when they are available on the queue.

Under the hood of the queue operator for Streams capture messages is a pluggable map that uses the LCRSplitter and LCRCast operator. You can also roll your sleeves up and build mappings that just leverage these. The queue operator we looked at in figure 35 simply has 3 output groups and the LCRSplitter splits inserts, deletes and updates. Then the LCRCast builds a column list for each group that is a useful set.

In summary we have seen how queues can be used for consuming data in a trickle feed manner with OWB 11gR2 and that we can consume Streams capture messages as well as custom messages.

Summary

In this paper we have covered various aspects of the Data Integration offerings from Oracle that will help you leverage the newest features for right-time data loading. This covered a spectrum of capabilities from file loading, to Oracle Data Pump to Oracle GoldenGate and Streams. The Oracle Data Integration tools certainly offer a rich open platform and a strong library of technology and components for building and maintaining your data warehouse. As you have read this, you can appreciate how to leverage different code templates and some of the newest ETL features in OWB and ODI to run change data capture and right-time mappings.

Implement Right-Time Data Loading with Oracle Data Integration and OWB 11gR2
October 2010
Author: David Allan

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