An Oracle White Paper
February 2010

# Oracle Warehouse Builder 11gR2: OWB ETL Using ODI Knowledge Modules

ORACLE

# Introduction

This paper discusses how knowledge module technology from Oracle Data Integrator is used in Oracle Warehouse Builder Enterprise ETL 11gR2 (OWB-EE) code template mappings to add fast, flexible, heterogeneous data integration and changed data capture capabilities.

OWB-EE 11gR2 adds ODI-based data integration infrastructure to the most widely used data warehousing tool for Oracle. This delivers the benefits of fast, flexible, heterogeneous data integration to today's Warehouse Builder customers. Customers can get the benefits of ODI in a familiar form that preserves and builds upon existing investments in OWB designs and skills. Existing designs can be updated to use the new infrastructure with few or no logical-level design changes. OWB-EE 11gR2 lowers TCO, shortens time to value and improves ROI for Oracle data warehouse customers.

A future release, ODI 12g, will merge functionalilty from OWB and deliver a superset of the capabilities of today's products. ODI 12g will offer a smooth migration path for OWB-EE 11gR2 designs that use the ODI infrastructure. Customers who license ODI-EE can implement using OWB-EE or ODI today and know that their strategic investments in data integration will be protected tomorrow.

This paper assumes minimal familiarity with Oracle Data Integrator and familiarity with ETL mappings in OWB 11.1 or earlier.

## Oracle Warehouse Builder and Data Integration Requirements

Created as an end-to-end solution for Oracle data warehousing, Oracle Warehouse Builder (OWB) is the most widely used tool for data warehousing-related ETL targeting the Oracle Database. However, for data integration, Warehouse Builder has mostly depended upon infrastructure provided by the Oracle database:

- Database links, for data movement
- Database gateways for heterogeneous connectivity
- Flat file loading through external tables or SQL Loader
- Process flows running on Oracle Workflow for complex multi-step integration processes, such as data movement with flat files and FTP
- The Control Center Service running in the database for all runtime activities

Data warehousing customers, like other data integration customers, face a number of emerging challenges not adequately addressed by this range of capabilities:

- The shift to operational data warehousing, with more frequent updates to the warehouse (CDC or real-time data integration) and shrinking or non-existent batch windows
- A larger number and variety of heterogeneous data sources
- Rapidly increasing data volumes
- More varied data integration scenarios, such as geographically distributed data sources, where high-performance bulk data movement adds value but also complexity
- Rapid advances in underlying database technology, such as Oracle Exadata, that benefit from novel data movement and integration techniques, and that deliver unprecedented processing power best leveraged by keeping workloads within the Oracle database

Data integration requirements around data warehousing are therefore changing in the face of these new demands and new technological opportunities, and customers need data integration products that can keep up. Oracle's long-term strategy for data integration is centered on Oracle Data Integrator (ODI), because of its flexible, extensible framework for incorporating new data integration technologies and exploiting investments in underlying database processing power.

However, Oracle Database customers who have already made Oracle Warehouse Builder part of their data warehousing strategy could not preserve existing investments in designs and skills. At present OWB designs cannot be migrated directly to ODI, and OWB developer skills do not directly transfer to ODI's ETL design paradigm.

To address the market with existing investments in OWB, Oracle has delivered OWB Enterprise ETL (OWB-EE) 11gR2. OWB-EE has been retrofitted with support for the key technologies in

Oracle Data Integrator, to align it with Oracle's long-term data integration strategy. Specifically, an ODI-based, database-independent data integration infrastructure has been added to OWB alongside the original code generation and data integration capabilities designed around the Oracle database. A natural extension to the familiar Warehouse Builder mapping paradigm allows the creation of mappings that use the new ODI-style data integration infrastructure, and the upgrading of existing mappings to leverage the new features with minimal redesign. Administration of the new capabilities is fully integrated with Oracle Warehouse Builder as well.

OWB-EE 11gR2 provides a way for OWB customers to add advanced data integration techniques to existing solutions, minimizing any switching costs related to training, reimplementation or disruption of long-validated solutions.

This new extension to OWB aligns OWB Enterprise ETL with the overall Oracle Data Integrator Enterprise Edition offering. The functional benefits are substantially the same as those delivered by ODI ETL:

- JDBC-based connectivity for maximum flexibility
- Native support for non-Oracle platforms
- In-database EL-T processing on non-Oracle databases, to push workload to sources and staging systems
- High-performance bulk data movement options
- A framework for designing and executing change data capture processes as ETL mappings

Migration of existing OWB mappings to the new infrastructure is straightforward and generally does not require major logical-level design changes. A high degree of compatibility with ODI knowledge modules allows the use of either tool as appropriate with the same data sources and data movement techniques. Certified knowledge modules and platforms ship with OWB 11gR2 for DB2 and SQL Server, and customers can add new ones.

Warehouse Builder customers who upgrade to OWB 11gR2 with OWB-EE can therefore safely continue to invest in and use OWB for Oracle data warehousing, including new implementations and enhancements to existing ones. They are part of the long-term roadmap for data integration at Oracle, and their data integration requirements around the data warehousing use case will be well served, as always.

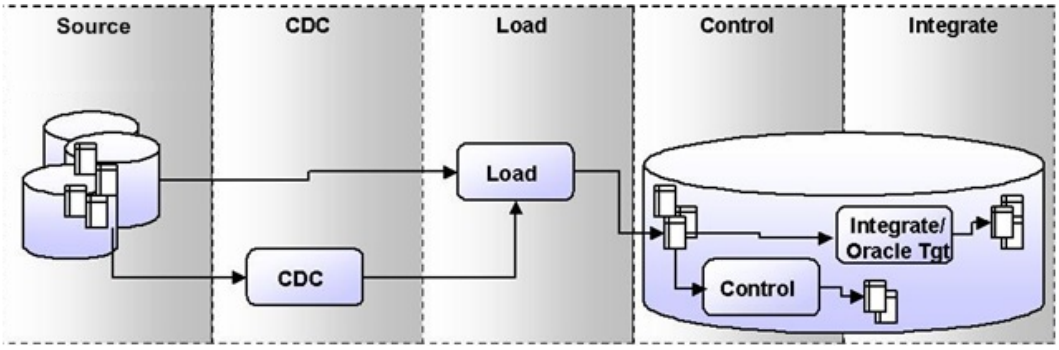
## OWB 11gR2 and Knowledge Module-Based Data Integration

As noted above, OWB's new data integration structure is based on ODI's key technology: knowledge modules. OWB users not familiar with ODI can get the necessary context from the

following discussion, then see how the ODI-based functionality fits into the overall OWB architecture.

## Understanding Code Template-Based Data Integration

ODI Knowledge Modules (KMs) are code templates. Each code template provides a code skeleton for steps of an individual phase in an overall data integration process. The steps in the process are summarized in the following diagram.




The main flow of data movement is:

- The Load phase refers to loading the staging area for the actual integration. Loading extracts data from the source system (possibly after some preliminary processing, such as filters, joins and transformations) and loads it into staging tables, which can be in the target database or an intermediate staging database.
- The Integrate phase takes staged data, either in the staging or target database, performs integration-related transformation, and delivers it to the destination tables in the target database.

Optional phases include:

- For changed data capture, processing to identify changed rows in the source happens before the Load phase.
- For inline data quality management, staged data is tested against data constraints during the Control phase, and only compliant data is passed to the target.

Each phase of this process requires one or more tasks, which may run at the source, in a staging area, or at the final target. Each code template describes the tasks required for a phase of the process.

The code in each template appears in nearly the form that it will be executed, except that it includes Oracle Data Integrator (ODI) substitution methods enabling it to be used generically by many different integration jobs. The code that is generated and executed is derived from the

declarative rules and metadata defined in the OWB mappings where the code templates are used. The generated code for the ETL job is deployed to a J2EE runtime agent for execution; from that agent, in turn, tasks are dispatched to or run against sources and targets as needed.

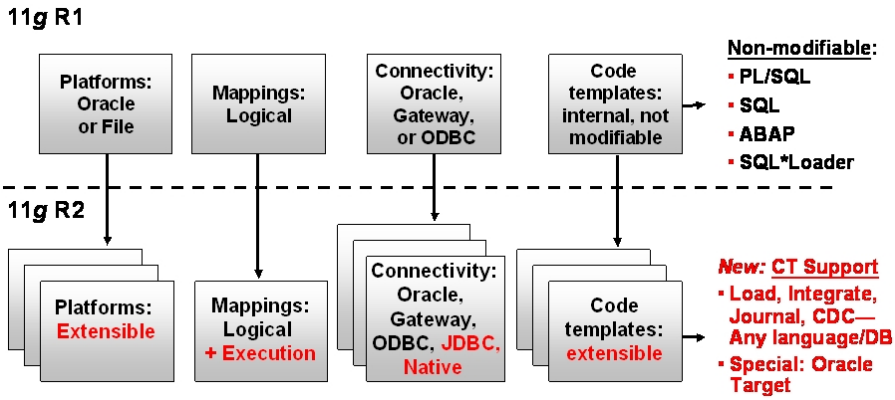
Templates may be generic, for databases accessed by JDBC, or specific to a given source platform, target platform, and/or data access and movement technology. Data access and movement can be based on any of a number of mechanisms—for example, on Oracle, Data Pump could be used to extract data in bulk, FTP to move the data, and Data Pump on the target could handle the loading. The more tailored the template is to source and target and the operation being performed, the more effective it can be. Out of the box, certified templates for common strategies on widely used platforms are provided. Customers can also create more templates to implement optimal data movement for their specific platforms and environments.

In general, code is generated to run in the native language of the source or target, and uses native data types on each platform. The code generation and execution process reconciles differences among data types on the source and target.

## OWB Architecture Extensions for Code Templates

Oracle Data Integrator provided the original implementation of knowledge module technology, and defined a templating notation and API for use in incorporating metadata such as table and column names from sources and targets in the code generated from the templates. Warehouse Builder now implements the same APIs and templating syntax, and provides a comparable J2EE runtime.

The following diagram summarizes the architectural extensions to OWB related to the new ODI-like data integration capabilities in OWB 11gR2.




The major changes include:

- An extensible platform framework, for representing native data types from heterogeneous databases

- An extension to the mapping framework, abstracting the logical description of data transformation and movement from execution-level details
- Support for native connectivity using JDBC, as well as other data extraction and movement options
- A new code template-based code generator, for generating native EL-T code in SQL and other languages, to run on heterogeneous platforms instead of just Oracle
- A special code template type, the Oracle Target CT, used to mix Oracle-specific code generation capabilities with the new heterogeneous code template mappings

These will be discussed more in the following sections.

Note that in OWB-EE 11gR2, code template-based code generation has been added alongside the existing "classic" code generation style. While most data integration-related extensions to OWB in the future will build on the new framework (and will only be available to OWB-EE customers), existing implementations based on gateways or Oracle-to-Oracle connectivity will continue to function for the foreseeable future.

OWB-EE 11gR2 comes with certified code templates and platform definitions for common platforms such as DB2 and SQL Server. Customers can define their own as well.

## Understanding OWB 11.2 Code Template-Based ETL

A new type of mapping, code template mappings (or simply template mappings), uses the new code template framework for code generation. The OWB mapping editor has been extended to allow creation of code template mappings in much the same manner as classic mappings.

OWB code templates are analogous to ODI knowledge modules, as described below.

| Template type | Role | ODI Equivalent |
|---|---|---|
| Load CT | Defines steps to extract needed data from the source and move it to the staging tables | Load KM (LKM) |
| Integration CT | Transforms staged data and delivering the results to the target table | Integration KM (IKM) |
| Control CT | Checks rows against data constraints and blocks non-compliant data from moving to staging area | Check KM (CKM) |
| Journaling CT | Selects changed rows to be moved, in CDC mappings | Journaling KM (JKM) |

The template steps usually describe SQL commands for a source or target, but can include other languages as well, including shell scripts, Java logic or other arbitrary code.

Note that some portions of the ODI code template have been omitted, by design, where they were not relevant in the OWB context. For example, OWB does not support ODI Reverse-engineering KMs or Service KMs because OWB uses different mechanisms for reverse-engineering metadata from sources and supporting Web Services. In practice, these differences do not impede the use of code template technology in OWB. The OWB documentation and the OWB SDK on OTN detail which portions of the ODI code templating APIs are supported.

## OWB Data Types and Extensible Platforms

OWB platform definitions play a role similar to ODI's technologies. OWB's system of data types has been extended to allow representation of the native data types of non-Oracle databases. A set of platform-neutral core data types is used internally. A platform definition for a database such as DB2 or SQL Server describes how the native types of the platform map to these OWB core types.

In an ETL mapping that moves data across platforms using the new framework, OWB uses these type mappings at code generation time, to translate data types between source and target platforms. Certified platform definitions for DB2 and SQL Server ship with OWB 11gR2, and customers can define new ones as needed.

## OWB Code Template Mappings vs. OWB Classic Mappings

OWB code template mappings follow the familiar paradigm from "classic" OWB mappings, but with extra details added regarding code generation, deployment and execution.

Consider the idea of declarative design in OWB classic mappings. A mapping is a high-level logical-level description of data movement and transformation. It does not, however, require the developer fully describe every details of the data movement and integration process. The ETL developer declares the desired data flow and transformations. OWB's classic code generation then fills in a lot of detail about the best way to accomplish those results given the source and target databases, connectivity, supported versions of SQL and so on.

Especially in the case of OWB's more complex operators, such as cube and dimension loading and match-merge, a lot of complex and often row-based processing logic is hidden behind the simple data flow represented in a mapping.

Some of these assumptions were implicit in the mapping, given the infrastructure provided in previous releases:

- Data was stored in Oracle databases, databases accessed through gateways (that therefore looked like Oracle databases), or flat files.

- Execution took place entirely within the database where the module hosting the mapping was contained; to distribute workload to different systems would require multiple mappings orchestrated by process flows.
- Data movement across databases was always based on database links, and gateways were used for non-Oracle database access.
- Data types were essentially always Oracle data types, with gateways hiding the details of non-Oracle systems.
- Internally, OWB code generation used a form of code template, but this was never directly exposed to users. Users' primary control over code generation was primarily in selecting a mapping type (ABAP vs. SQL and PL/SQL vs. SQL Loader) and target database version (e.g. 10.1 vs. 10.2 vs. 11.1).

The new mapping framework extends the idea of declarative design, by OWB ETL mappings more abstract. Logical-level data flow and transformation is less tied to physical-level implementation details. Code template-based code generation allows for alternatives to all of the above implicit choices.

The result is more flexible execution options, such as spreading different parts of the workload to different source and target systems, generating native SQL for non-Oracle databases and using their native data types, and moving data by new connectivity technologies.

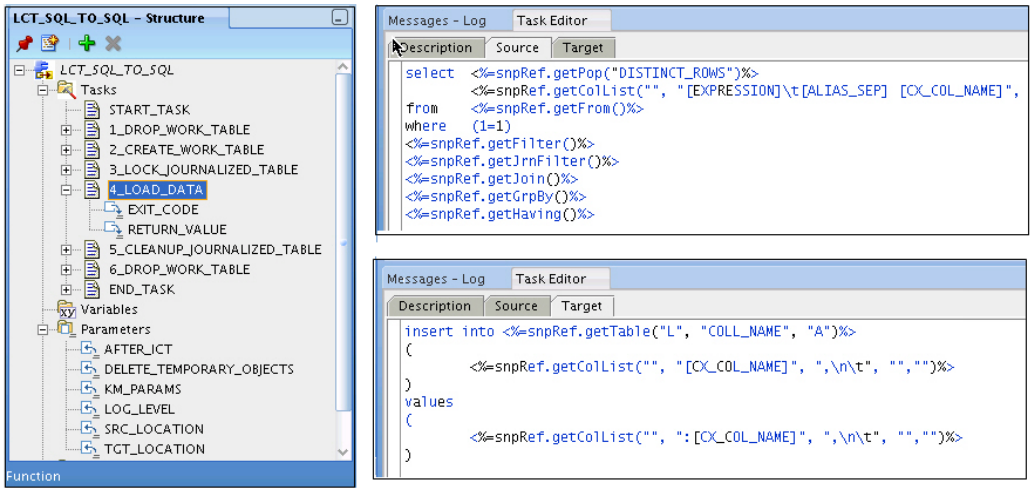
## OWB 11.2 Connectivity and Data Movement Technologies

As noted already, OWB relied on gateways for connectivity in previous releases. In 11gR2, new options are available: JDBC-based data movement (which uses the Control Center Agent to run Java and JDBC-based extraction and loading processing) and other data movement techniques based on the source and target platform. The most significant alternative is bulk data movement, based on flat files and the native bulk unloaders and loaders for the source and destination: extract data to a flat file on a source system, moving it to the target, and loading it into the target. Gateway-based movement also continues to be an option. Regardless of the data movement method chosen, the logical mapping design is no different.

## Working with OWB Code Templates

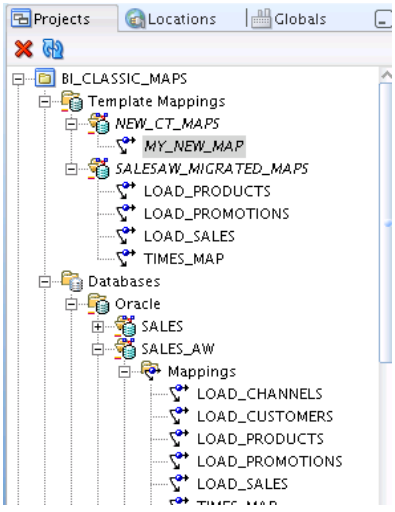
The OWB Design Center now includes a Code Template Editor. Like many other OWB objects, code templates can be defined for an individual project or stored as global objects for use across an entire workspace.

The Code Template Editor is very similar to the OWB Expert Editor. A code template in the editor appears as a series of tasks. The most efficient way to manipulate these steps is in the Structure View of the code template editor.




The Structure View shows the steps in a typical Load CT, in this case the generic SQL to generic SQL code template that relies on JDBC. The template code for the LOAD_DATA step is shown On the right are two views of the task editor window: one showing the skeleton for code to run on the source (the SELECT statement that performs the extraction) and the INSERT statement that puts the rows from the SELECT into a staging table on the target.

## Working with OWB Code Template Mappings




Working with code template mappings builds on what you already know about building classic mappings.

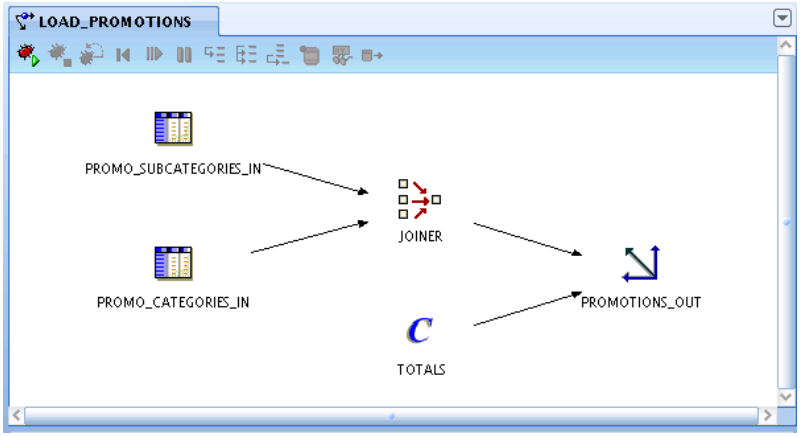
### The Code Template Mapping UI

The OWB Projects Navigator tree contains a new node called Template Mappings where you define template mapping modules to contain template mappings. Template mappings, unlike classic mappings, can involve EL-T code being deployed to both sources and targets, including multiple databases, and so it made more sense to separate them from classic mappings and not attach them to an individual database module.

The example here shows a project with a database module SALES_AW, as well as two code template mapping modules: NEW_CT_MAPS, which contains a new mapping MY_NEW_MAP, and SALESAW_MIGRATED_MAPS, which contains a few maps migrated from the classic mappings in SALES_AW.

### LOAD_PROMOTIONS: A Typical Classic Mapping

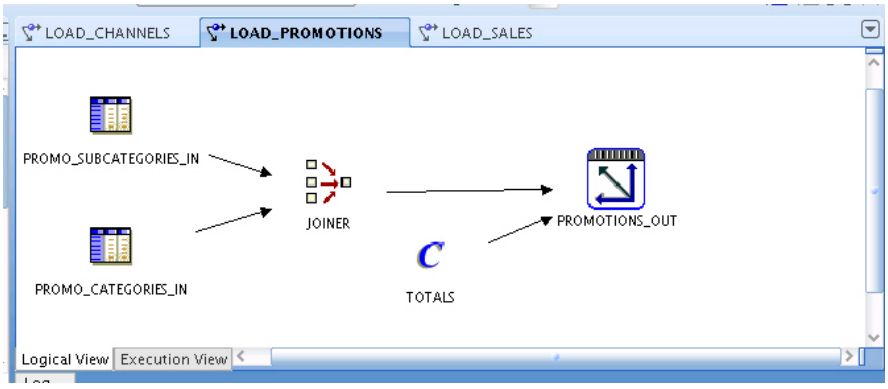
This typical example of a classic Warehouse Builder mapping loads a dimension PROMOTIONS from two source tables, CATEGORIES and SUBCATEGORIES. (Assume the source tables are in a separate database.)




The subsequent examples in this paper will all reference variations on this mapping using the new code template technology.

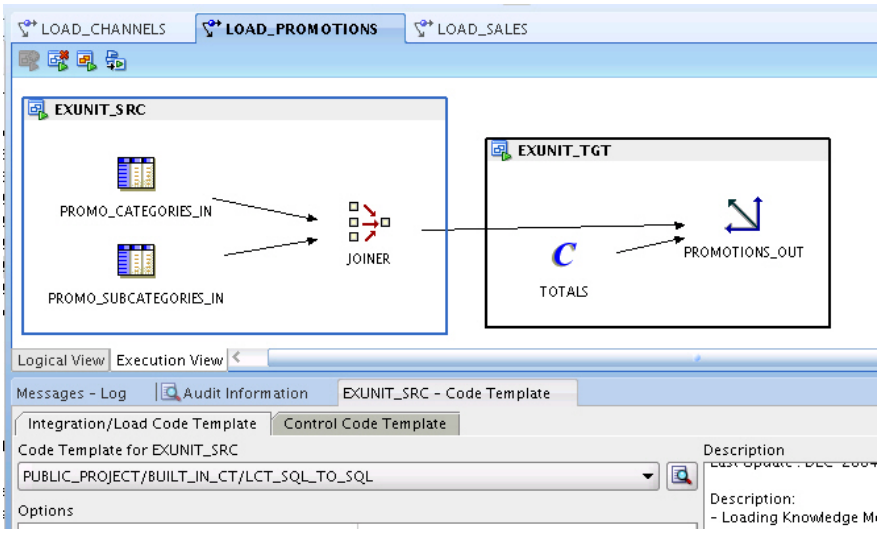
### LOAD_PROMOTIONS as a Code Template Mapping

The mapping editor for code template mappings is much like the classic mapping editor, but it presents two views of the mapping: the Logical View and the Execution View.



The Logical View is the same as the familiar editor for classic mappings, and describes the logic of how data is to be integrated. In this instance, there are still two source tables, a join, and the loading of a dimension.

What's new is the "Execution View" tab at the bottom. Click it to see details:



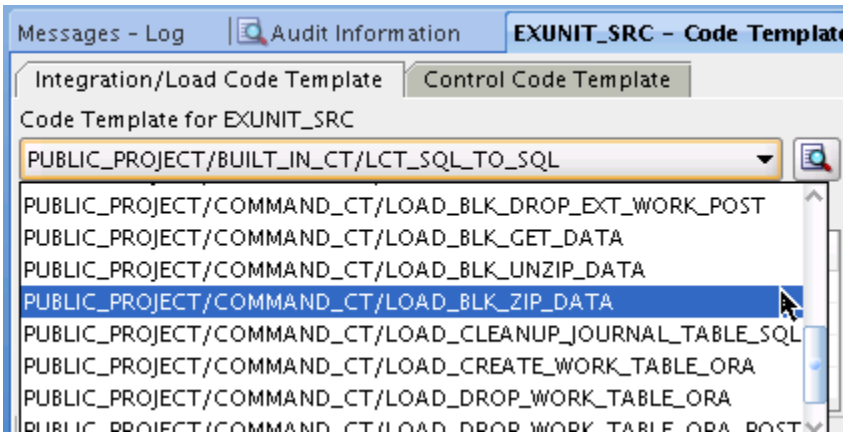

The boxes EXUNIT_SRC and EXUNIT_TGT group the mapping operators together into execution units, which identify a set of operators for which code will be generated together. Each execution unit is associated with one or more code templates that are used as the basis for code generation for that execution unit.

The metaphor of "islands and bridges" has been used to describe OWB code template mappings. Data is processed in the location associated with one execution unit (an "island") and then the results are pushed to staging tables in the next execution unit in the data flow (over a "bridge"). Integration at the next "island" uses the staing tables as sources. (Typically, the code template would also include steps to clean up the staging tables after integration.) Several execution units in a row can be chained together in this fashion, and processing results will move from one island to the next until they are integrated into the final target tables.

In this example, the EXUNIT_SRC execution unit represents two source tables and a join operation. Because the JOINER is in the execution unit, a join operation will be performed on the source system. The result of the join is moved to staging table on the target database associated with the EXUNIT_TGT execution unit. The dimension loading executes there, using the staging tables as sources. (Typically, the code template would also include steps to clean up the staging tables after integration, which should not be confused with Oracle's temporary tables.)

In the same window, take note of the specific selected load code template in the Load/Integration Code Template tab, which reads LCT_SQL_TO_SQL. This particular code template is generic, moving data from source to staging using JDBC-based connectivity.

Other data movement technologies could be substituted by simply picking a different code template. For example, large volumes of data are most efficiently moved by extracting data at the source to a flat file, moving the flat file from source to target system by FTP or similar means, then loading the flat file into the Oracle database. (Depending upon the speed of file movement, further performance improvements could be achieved by compressing the file before transfer.) To implement this mode of data movement, one need only select a different code template:

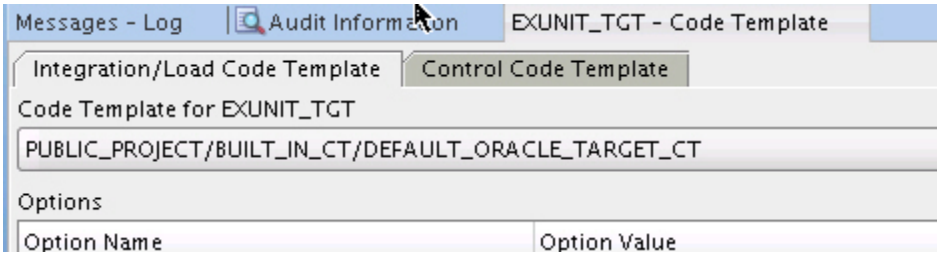



### OWB Classic Operators and the Oracle Target CT

Warehouse Builder's classic mappings include operators whose implementation does not map directly onto ODI's code generation framework. OWB's specific way of implementing features like dimension and cube loading operators, multi-table insert, and pivot and un-pivot do not directly map onto the features provided by ODI's code generation framework. (Cases where PL/SQL code generation is required, such as the OWB match-merge operator, present particular challenges.)

To support such operators in mappings that use the new code template technology, Warehouse Builder 11.2 adds a new type of code template not derived from ODI, called the Oracle Target code template. During code generation, for execution units associated with the Oracle Target code template, OWB uses the Oracle Database-specific code generator used for classic mappings. The same style of Oracle-specific SQL and PL/SQL generated for classic mappings is generated for these execution units.

In the example above, if you select the EXUNIT_TGT execution unit, the mapping editor updates to show that the Oracle Target CT is applied:



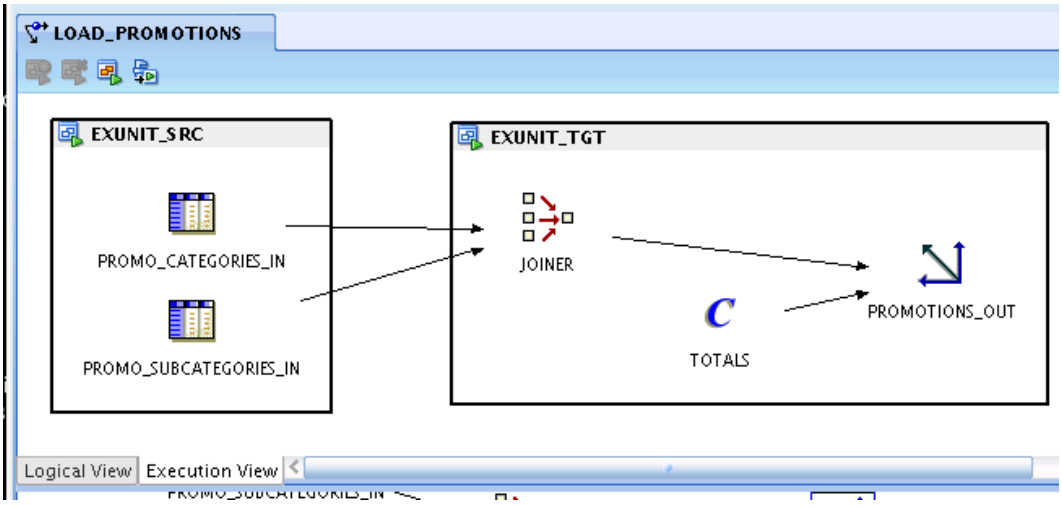

This is why the OWB dimension loading operator is available in a code template mapping.

The resulting mappings divide labor according to the strengths of the underlying technologies:

- ODI-based integration mechanisms fill the roles of flexible connectivity/data movement and pushing parts of the EL-T workload to non-Oracle databases, and ultimately stage the resulting data in temporary Oracle staging tables
- Final integration, which may require complex transformation, row-based processing and dimensional loading, is performed in Oracle, using those staging tables as sources and fully exploiting the Oracle SQL and PL/SQL, underlying database options and so on.

### One Logical Design, Many Implementations and Topologies

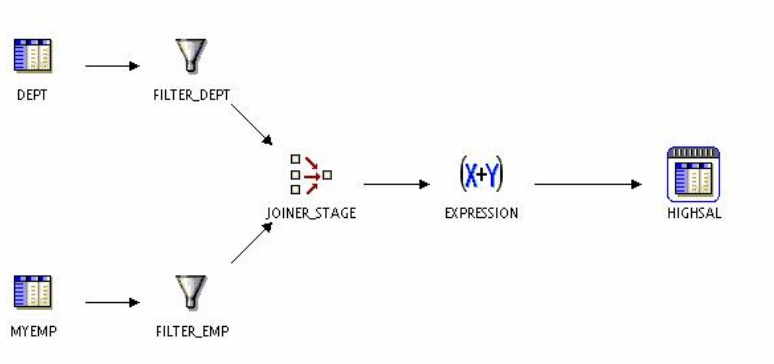
By selecting different execution units, code templates and locations to which code is deployed, you can cause OWB to generate drastically different code to accommodate different scenarios from a single logical mapping design. For example, you could have one configuration that runs the JOIN on the source, then another with different execution design to perform the JOIN on the target:



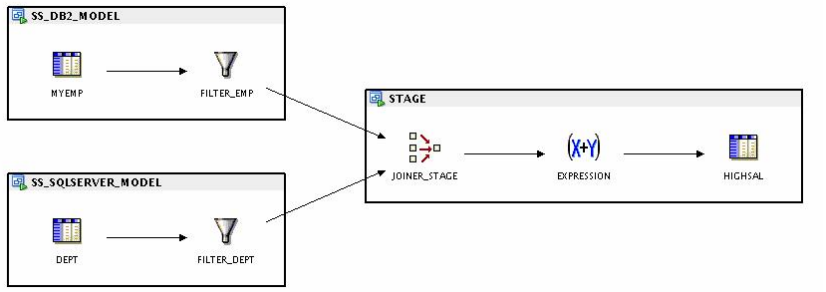
With this execution design, the source table contents will be moved to the Oracle target separately and joined there. The join processing can thus be moved among different databases as needed based on workload, data volumes and so on.

With OWB's multi-configuration feature, for a single code template mapping you can specify completely different execution designs for each configuration—group operators into different execution units, and select different CTs. Thus, when moving from development to test to production, you can prove out your logical mapping design with a simple and convenient execution design and a data movement technique like JDBC, and then use different execution strategies and data movement technologies in test and production.

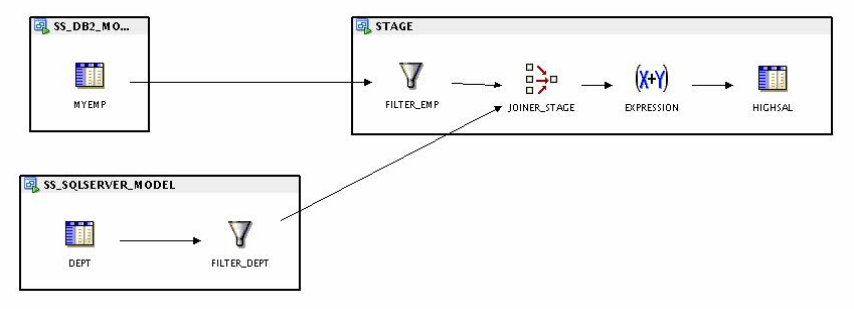
Consider the following logical-level design, in which DEPT is a table on a SQL Server location and MYEMP is on a DB2 location:




Several physical-level execution designs are possible. For example, this version filters data on each of the source systems, then stages the results on the Oracle target for join and further processing:




This version filters the SQL Server data on the source and DB2 data on the target (which is the same as the staging area):



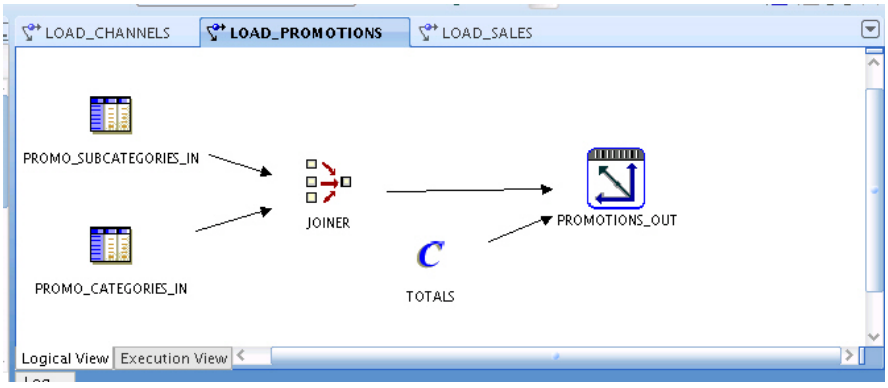

Many more such permutations are possible.

## Migrating Existing Mappings to Code Template Technology

Existing mappings built in previous releases can be migrated to use code template-based generation where these more flexible data movement and processing techniques add value. This allows the incremental introduction of these technologies as needed. You can preserve investments in existing designs while addressing bottlenecks or reducing latency through selective introduction of code template mappings.

The subject of migration of existing mappings—when to choose migration, how to do it, how to get the most value out of it—is a broad topic that will be the subject of a future white paper. However, the general process is outlined below. The migration or conversion process is best understood with another example, again based on the initial "classic" LOAD_PROMOTIONs mapping.

Note that the mapping conversion process does not significantly alter the logical-level view of the mapping—all changes affect the execution level.



### Performing the Conversion Steps

The conversion process consists of the following steps:

- If necessary, create a template mappings module in the project tree.
- Copy/paste the original mappings to a code template mapping module.
  - Select the classic mappings to be migrated in the Project Explorer, right-click, and choose "Copy"
  - Right-click the destination template mapping module, and choose "Paste"

  The new code template mappings appear in the project tree.
- In the execution view, make sure the mapping is divided into reasonable execution units
- In the execution view, assign each execution unit code templates based on the desired data movement technology.
- Make sure the mapping is using native data types for the table, view and sequence operators in the mapping, if it wasn't already.

### Defining Execution Units and Assigning Code Templates

Given the metadata about your sources, targets and mapping, such as the locations for your source and target data (in a single Oracle database, multiple Oracle databases, or heterogeneous databases) OWB will divide up the resulting mapping into execution units that determine which parts of the job run on which hosts, and assign default code templates to the execution units. These should be reviewed because the automated assignments are unlikely to yield a best fit for your problem.

### Copying and Pasting Classic to Code Template Mappings

OWB creates template mappings with the same logical design as the original mappings, groups operators into workable execution units, and then attempts to assign default code templates to each unit. Execution units containing operators that only work in classic code generation are assigned the Oracle Target CT. Other execution units are assigned code templates.

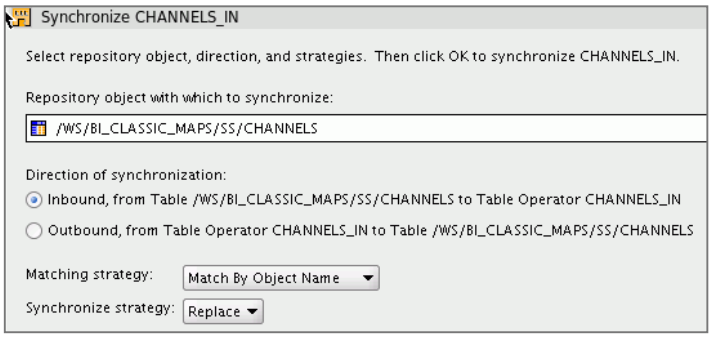
### Updating Table Operators with Native Data Types

Any mapping that had previously used gateway-based connectivity for non-Oracle databases will have Oracle data types for all the table, view etc. operators in the mapping. When converting a mapping to use code template-based data movement, the mapping table, view and sequence operators must be updated to use native data types. The same technique applies if, for example, you want to update a mapping designed around Oracle sources to access non-Oracle sources.

To update a mapping to use native connectivity and data types:

1. Set up new locations (using native connectivity) to connect to the source
2. View the new database objects to make sure there are source tables, views and sequences with same column names as the originals
3. Use the copy and paste process to create a new code template mapping from the original mapping.
4. For each operator to be updated, in the logical view, perform an inbound synchronize operation on the operator to update it with types from the new database object accessed through native data types.
    - Synchronize the operator with the source table or other object in the module associated with the location that uses native connectivity.
    - Make sure the direction is an inbound synchronize
    - Set the matching strategy to "Match by object name" to match up the columns in the operator with the underlying table
    - Set the synchronize strategy to "Replace" to overwrite the old data type information

    Your Synchronize window should look like this:




5. Check the execution unit division and code template assignments on the mapping, to ensure that they are appropriate given the new source and target objects.

The result is a ready-to-use heterogeneous mapping using native connectivity and native data types. Note that the logical-level design of the mapping still looks the same—the only change has been to the data types of the source and target operators. You can then further explore applying different code templates and execution unit groupings to the mapping, to implement different workload partitioning scenarios and data movement methods, add multiple configurations to support multiple different implementations, and so on.

## Conclusion

The availability of code template mapping technology from ODI in OWB-EE 11.2 brings OWB customers much-needed data integration options within the product where they already have designs and skills in place. It also aligns OWB with the long-term Oracle strategy for data integration, building on common tools, frameworks and approaches to produce a tool which, while accessible to the existing customer base, addresses their emerging needs and positions them for the operational data warehouse of the future.

Future whitepapers will continue to explore the new features and behaviors enabled by code template mappings, including such topics as using Change Data Capture in ETL and best practice for applying code template mappings to specific problems in OWB.

## OWB 11.2 Resources

All collateral will be posted on the Oracle Warehouse Builder product page on OTN, here:

http://www.oracle.com/technology/products/warehouse

Advanced applications of code template mappings are frequently discussed on the Warehouse Builder Blog, at:

http://blogs.oracle.com/warehousebuilder

The Oracle Data Integration LinkedIn group is providing a basis for a community of both OWB and ODI users. For details, join the group here:

http://www.linkedin.com/groups?gid=140609

Oracle Warehouse Builder 11.2: OWB ETL
Using ODI Knowledge Modules

February 2010
Author: Antonio Romero

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



0109