# Arbeiten mit Template Mappings

**Carsten Herbe**
**Metafinanz-Informationssysteme GmbH**
**München**

**Schlüsselworte:**

Oracle Warehousebuilder 11gR2, OWB 11gR2, Code Templates, Template Mappings

## Einleitung

Neu in dem Oracle Warehousebuilder 11gR2 gibt es Template Mappings, basierend auf den Knowledge Modulen des Oracle Data Integrators (ODI). Sie ermöglichen nicht nur die Anbindung verschiedener Datenbanksysteme, sondern ermöglichen auch dass Teile der Mapping-Logik direkt in den Quellsystemen ausgeführt werden bevor die Daten im Zielsystem zusammengeführt, überprüft und gespeichert werden.
Dieser Vortrag stellt das Arbeiten mit den Template Mappings vor und gibt einen Überblick über die verschiedenen zugrundeliegenden Code Templates wie Load, Integration und Control.

## OWB Architektur

Zu den bekannten OWB-Komponenten kommt der Agent. Dies ist ein Java-Prozess, der auf einem beliebigen Rechner (idealerweise auf dem Quell- oder Ziel-Host) laufen kann. Über ihn werden die einzelnen Code Templates ausgeführt. Je nach Code Template laufen Teile des Mappings wieder in den verschiedenen Datenbanken.

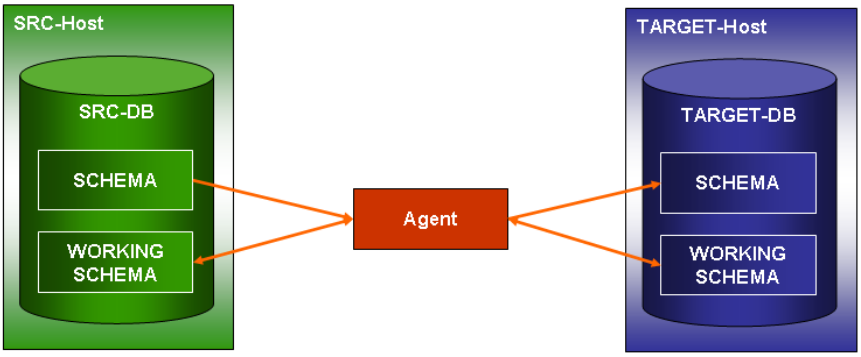


*Abb. 1: OWB Agent*

Der Agent muss extra gestartet werden:

```
> cd $ORACLE_HOME/owb/bin/unix
> ./local_service_login.sh
```

Zum Starten des Agents muss noch ein Passwort eingegeben werden. Dieses hat den Default-Wert „welcome“.

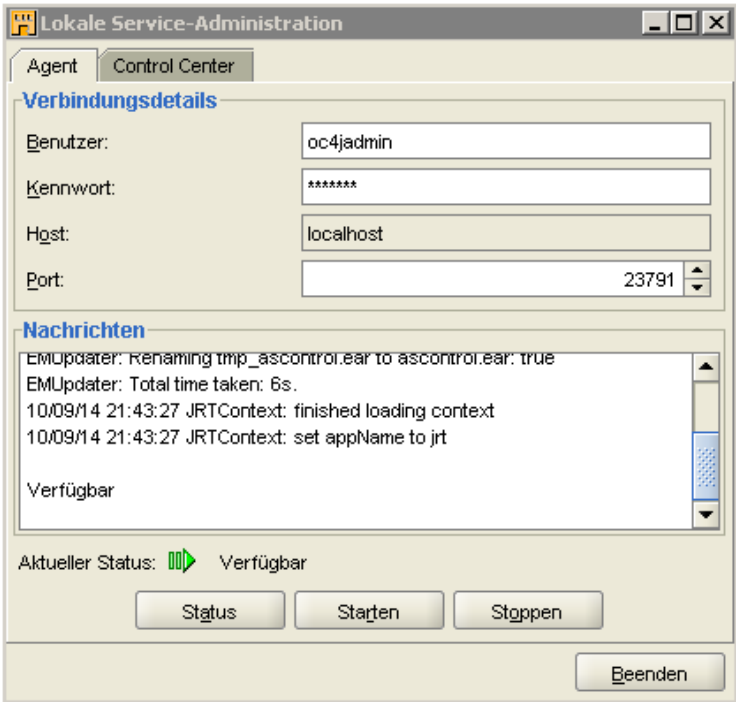


*Abb. 2: Starten des Oracle Warehousebuilder Agents*

Die Agent Location ist im Oracle Warehousebuilder zu konfigurieren:

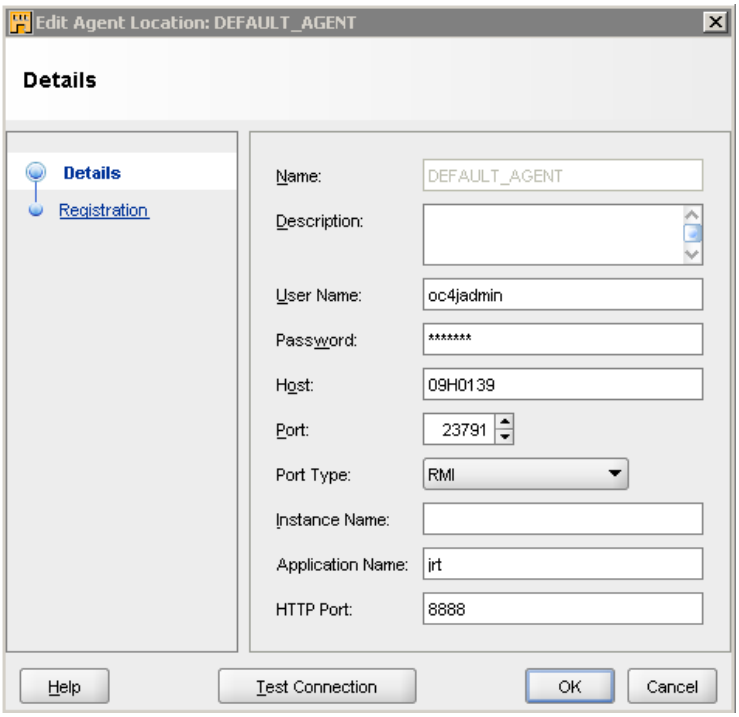


*Abb. 3: Agent Location*

Neuerungen gibt es auch bei den Location. Hier ist neben User und Schema auch das Work Schema zu setzen. In diesem Schema legt der OWB temporäre Objekte an. Diese werden in der Regel beim Start des Mappings angelegt und am Ende gelöscht.

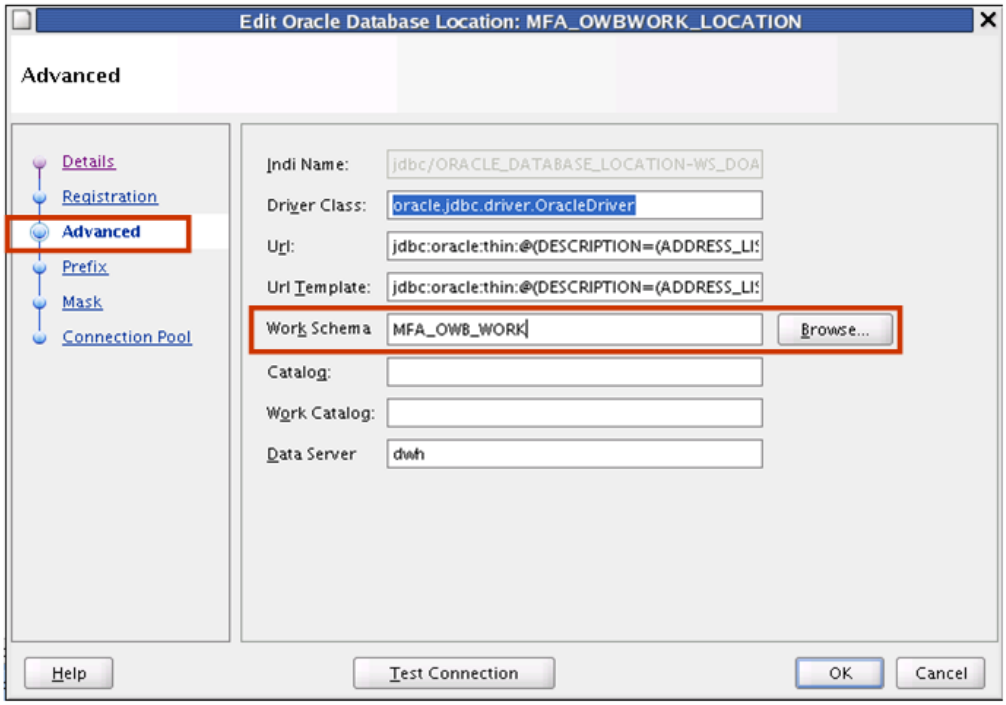


*Abb. 3: Work Schema einer Location*

Wie sind nun User, Schema und Work Schema am besten zu setzen? Hier ist nämlich zu beachteten, dass der User mit dem man sich verbindet in dem Work Schema Objekte anlegen darf und in auf die Objekte des Schemas zugreifen können muss. Es empfiehlt sich folgende Einstellung:

```
User = Work-Schema != Schema
```

Dies bietet mehrere Vorteile:

- Trennung von eigentlichen Schema-Objekten und temporären ETL-Objekten
- ETL User hatte begrenzbaren Zugriff auf Schema-Objekte (diese müssen explizite ger Grant vergeben werden)
- Keine Probleme mit Berechtigungen beim Erstellen der temporären ETL-Objekte

Bei einer Source Location braucht der User Leserechte für die Quellobjekte im Schema, bei einer Target Location braucht er Schreibrechte für die Tabellen im Schema.
Problematisch ist es wenn User und Work Schema unterschiedliche Datenbankuser sind, da in diesem Fall der User versucht Objekte im Work Schema anzulegen.

Eine Ausnahme zu dieser Regel gib es bei der Verwendung des speziellen Code Templates DEFAULT_ORACLE_TARGET_CT. Dazu mehr im Abschnitt „Code Templates“.

**Template Mappings**

Template Mappings sind nicht wie die klassischen Mappings einem Datenbankmodul zugeordnet. Man findet sie daher in dem Ordner „Template Mappings“ direkt unterhalb des Projekt-Ordners.

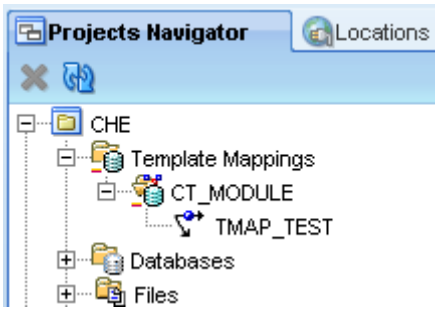

*Abb. 3: Template Mappings im Projekt Ordner*

Jedem Template Mapping Modul ist ein Agent zugeordnet, so wie einem Datenbankmodul eine Datenbank-Location zugeordnet ist.

In dem Template Mapping Editor gibt es zwei Sichten auf das Mapping, den Logical View und den Execution View.

Im Logical View wird die Mapping Logik wie bei klassischen OWB Mappings entworfen. Hier werden die logischen Transformationen definiert, unabhängig von der Datenbankplattform oder der physikalischen Verteilung der Tabellen auf unterschiedliche Server:

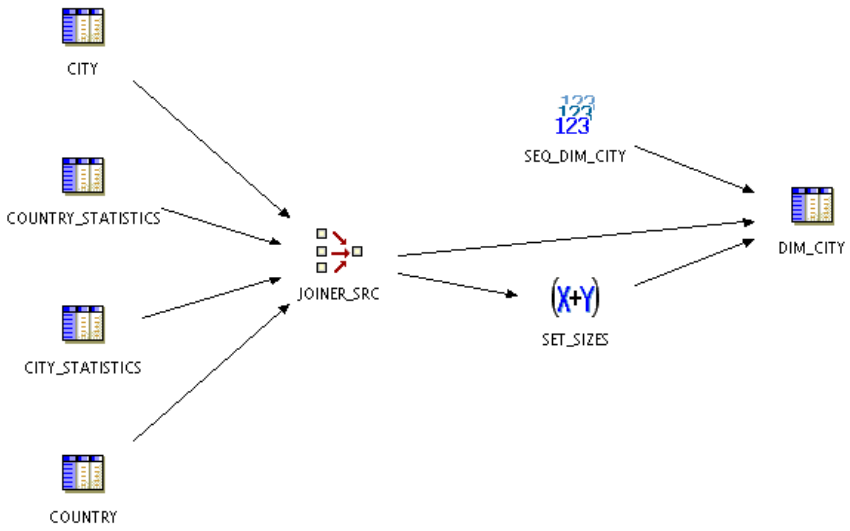

*Abb. 4: Logical View eines Template Mappings*

In der zweiten Sicht, dem Execution View, werden Teile des Mappings zu Execution Units zusammengefasst. Ein Mapping kann aus einer einzigen oder mehreren Execution Units bestehen. Es kann getrennte Execution Units für Datenextraktion, Datenprüfung und Schreiben von Daten geben.

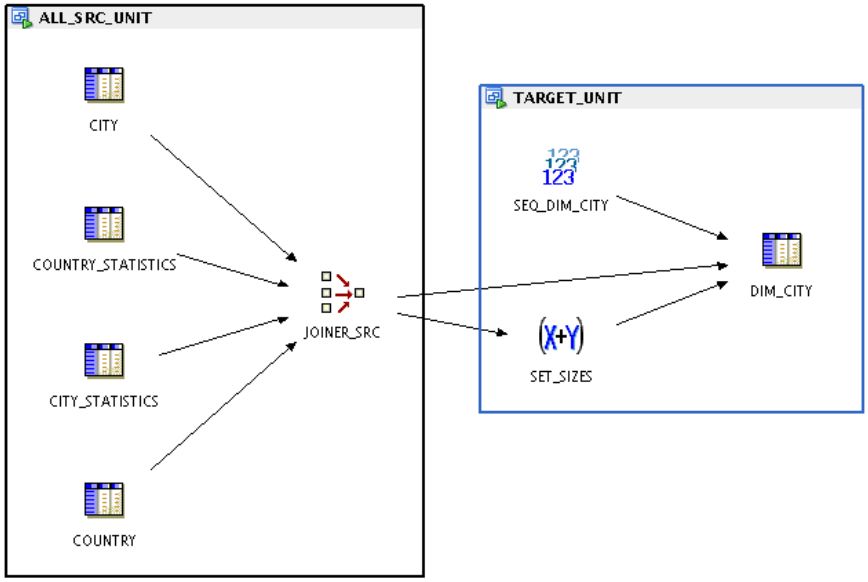


*Abb. 5: Execution View eines Template Mappings*

Selbst mehrere Execution Units für Datenextraktion sind sinnvoll, wenn die Quelldaten auf verschiedene Datenbanken verteilt sind. Dazu kommt am Ende noch ein Beispiel.

Jeder Execution Unit wird ein Code Template zugeordnet. Diese entsprechen den Knowledge Modulen im Oracle Data Integrator (ODI). Ein Code Template gehört zu einer der Gruppen

- Load: Extrahieren von Daten
- Integration: Schreiben von Daten
- Control: Errorlogging
- Change Data Capture

Innerhalb dieser Gruppen gibt es verschiedene Code Templates, z.B. für verschiedene Datenbankplattformen oder unterschiedliche funktionale Implementierungen.

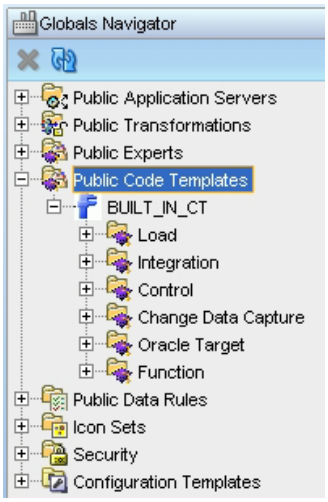


*Abb. 6: Public Code Template Folders*

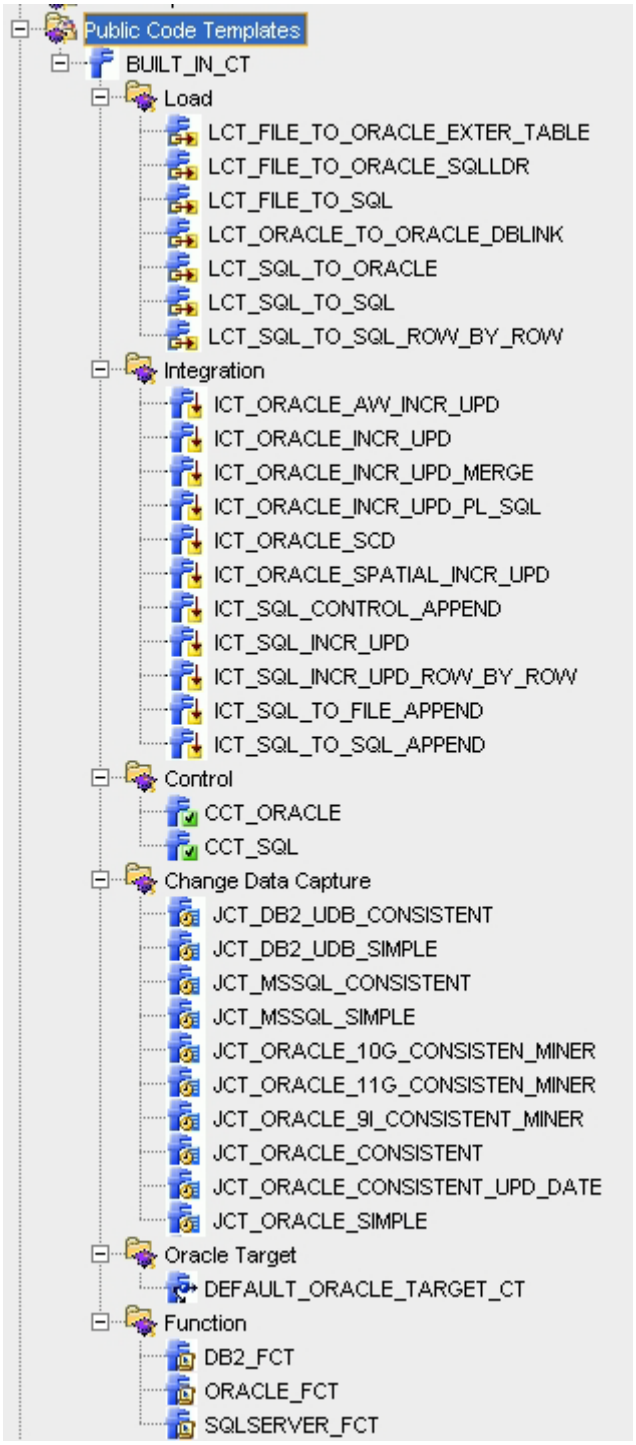

*Abb. 7: Public Code Templates*

Eine besondere Rolle spielt das Code Template DEFAULT_ORACLE_TARGET_CT. Dieses generiert ein PL/SQL Package wie ein klassisches Mapping. Das Package muss in dem Target Schema deployt werden, d.h. in der Location ist Schema = Work Schema ist zu setzen.

**Code Templates**

Code Templates sind keine Blackbox. Man kann sie im Code Template Editor öffnen. Auf diesem Weg kann man auch eigene Code Templates erstellen oder bestehende Code Templates ändern. Ein Code Template kann aus folgenden Tasks bestehen:

- Jacl: Java Command Language (JACL) ist ein in Java geschriebener Tcl-Interpreter
- JDBC: Führt Datenbank-Befehle (SQL) aus.
- Jython: Jython (früher: JPython) ist eine reine Java-Implementierung von Python
- OS: Aufruf von Betriebssystem Befehlen
- Runtime API: Aufruf der Runtime API

Der Code Template Editor ähnelt stark dem Expert Editor:

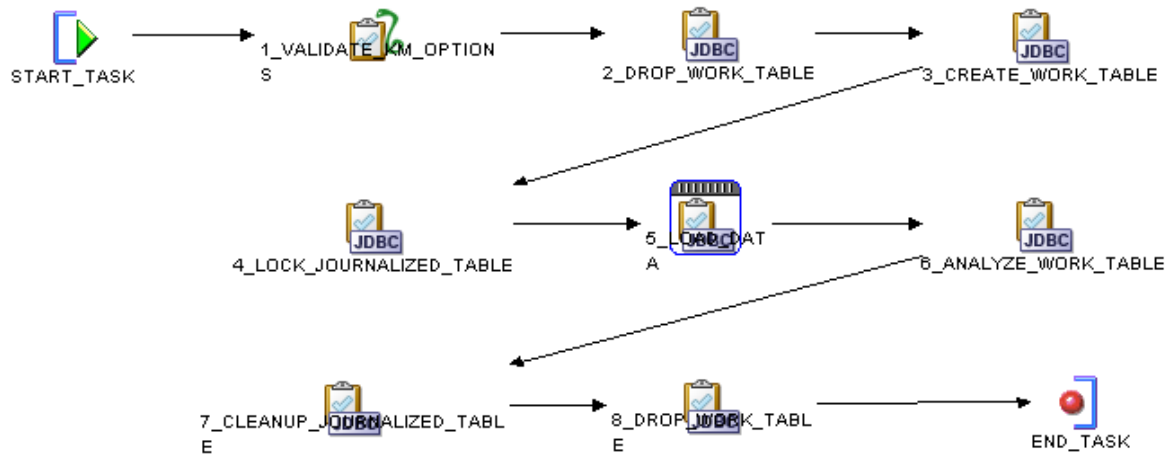


*Abb. 8: Execution View eines Template Mappings*

Für jeden Task gibt es einen Source- und Target-Editor:

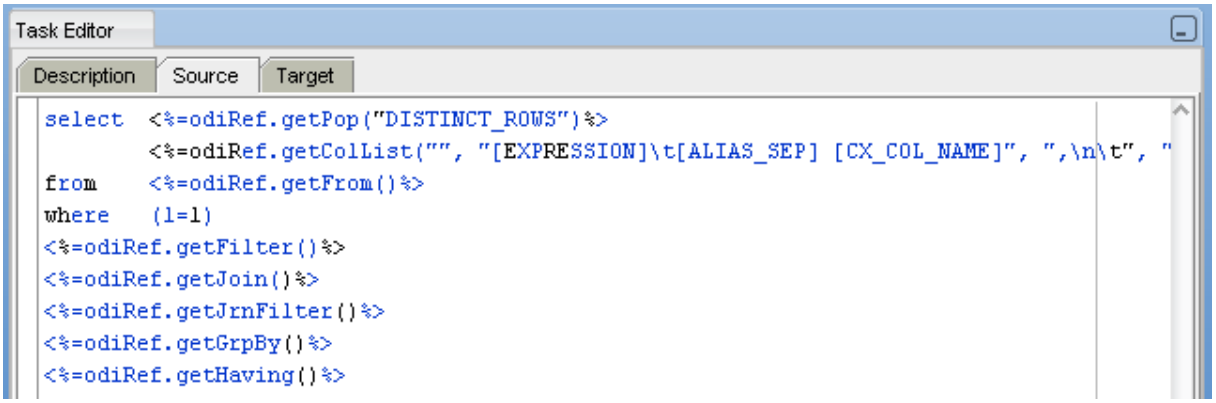


*Abb. 9: Task Editor: Source*

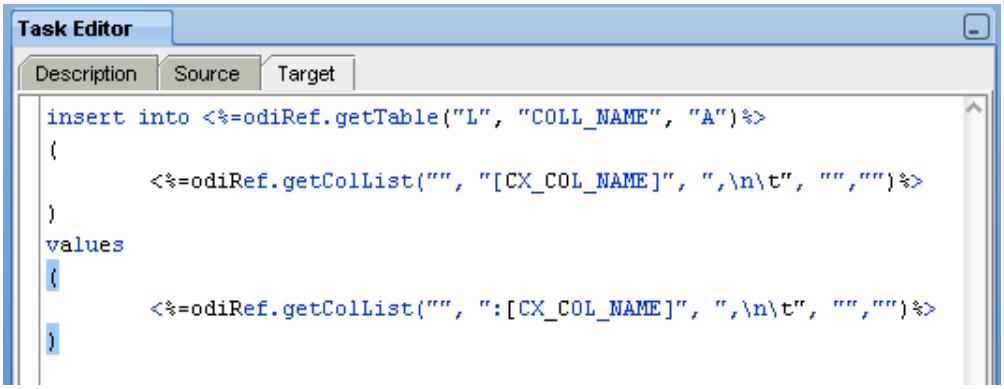


*Abb. 10: Task Editor: Target*

**Deployment und Ausführung von Template Mappings**

Template Mappings werden wie klassische Mappings deployt, nur wird beim Deployment kein Code in der Datenbank erzeugt sondern der Code wird auf einen Agent deployt. Erst bei der Ausführung werden Objekte (meistens temporär) in den Datenbanken angelegt. Beim ersten Deployment wird zusätzlich zu der eigentlichen Mapping Logik noch das Code Template deployt.

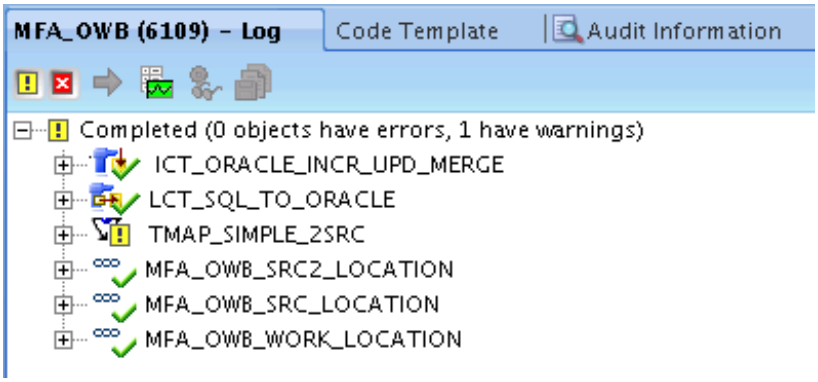


*Abb. 11: Execution View eines Template Mappings*

Der Agent dient hier im Fall von zwei Oracle Datenbanken nur zur Steuerung der einzelnen Execution Units, der Datenfluss wird über Datenbanklinks realisiert. Auch wenn die Mappings von dem Agent gesteuert werden, bedeutet dies nicht automatisch auch dass die Daten über den Agent laufen.

Im Design Center sieht man sehr schön, wie die einzelnen Tasks der Code Templates ausgeführt werden:

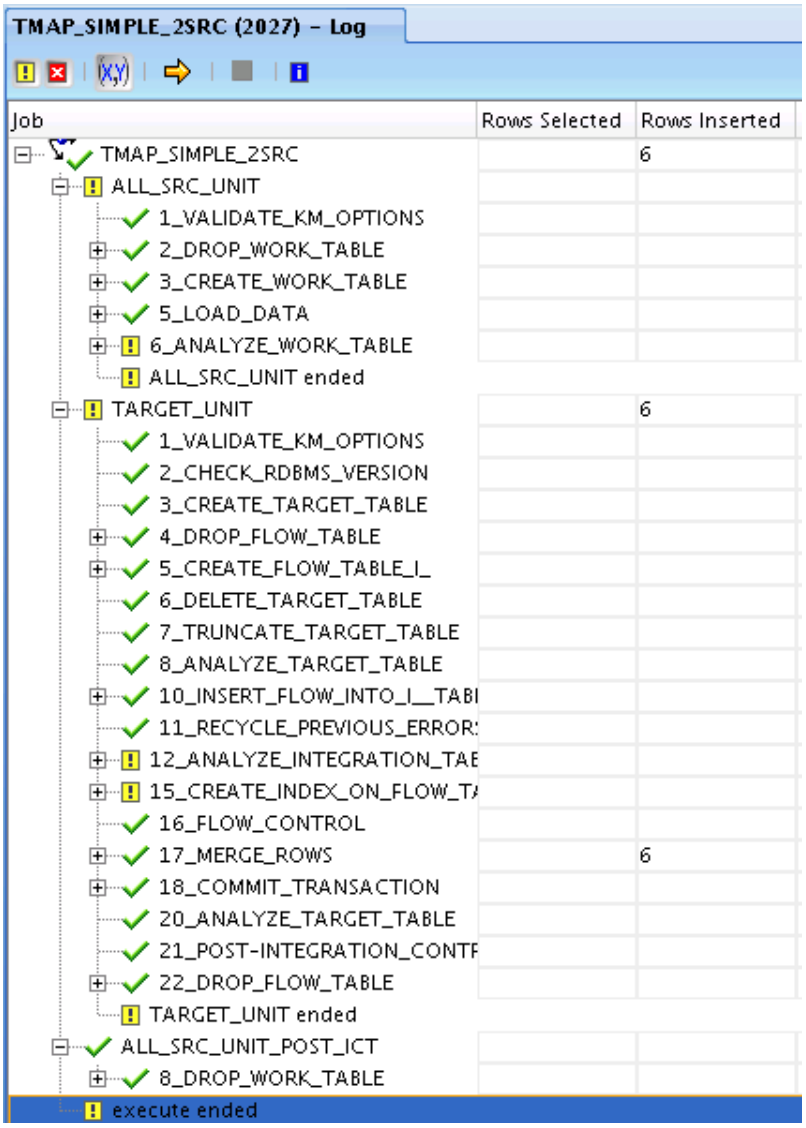

*Abb. 12: Log einer Mapping-Ausführung*

Zu der Log-Ausgabe findet man im Audit Information View zusätzliche Informationen:

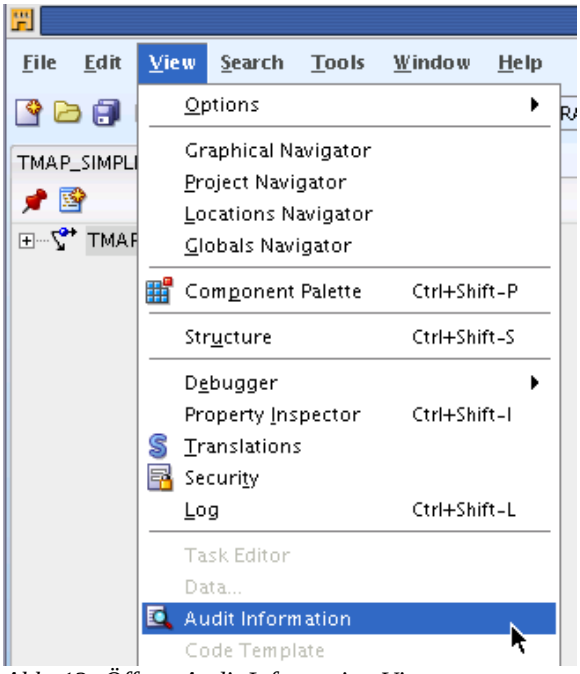


*Abb. 13: Öffnen Audit Information View*

Im Audit Information View kann man zu jedem Task (s. Code Template Editor) den ausgeführten Code sehen:

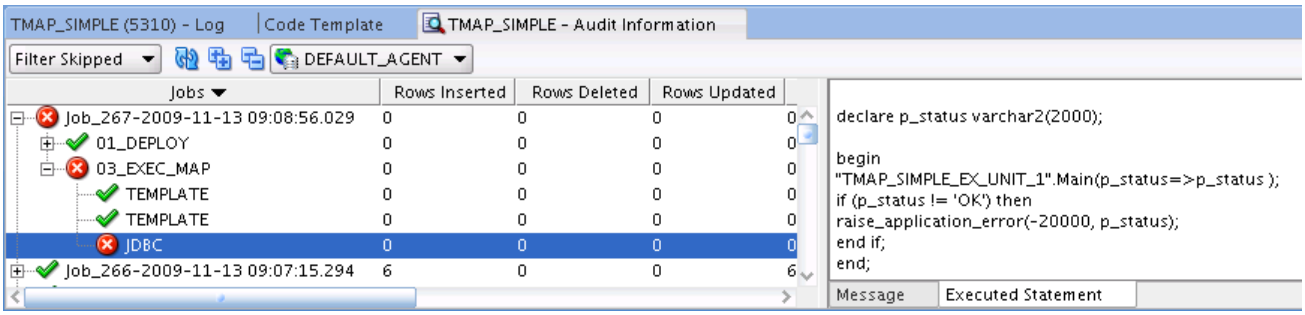


*Abb. 14: Audit Information View*

Hier jetzt im Überblick, wie der Datenfluss aussieht und wo welche temporären Objekte angelegt werden:

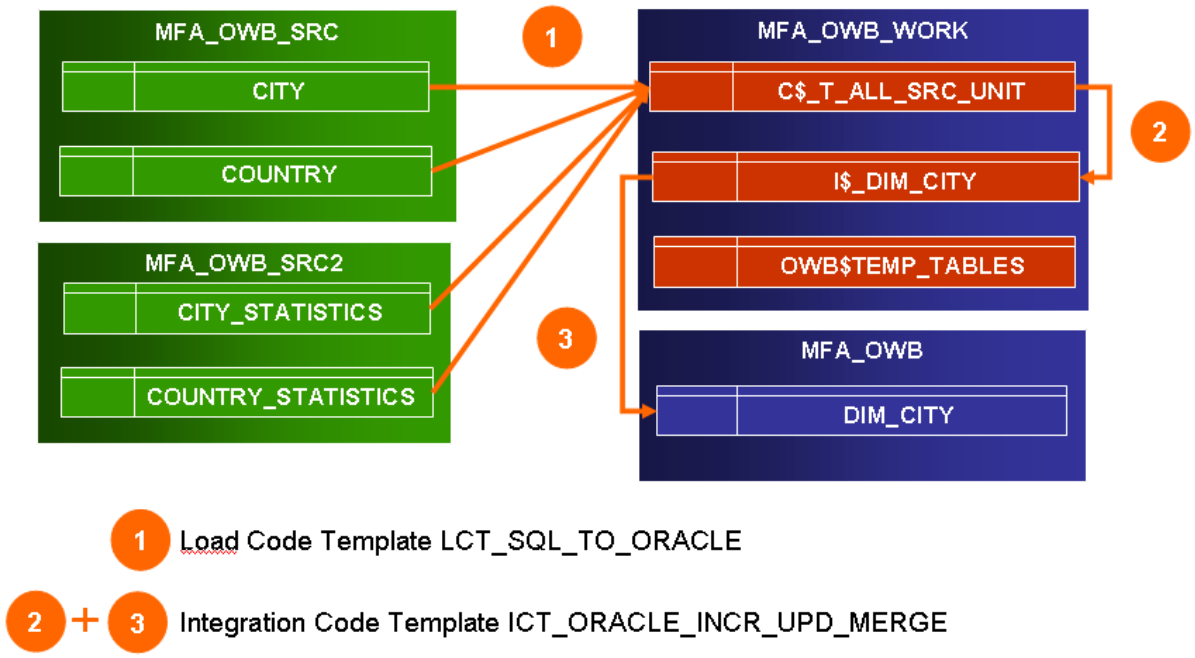


*Abb. 15: Temporäre Objekte in der Target Location während der Mapping Ausführung*

Beispielhaft schauen wir uns noch die Temporäre Tabelle I$_DIM_CITY an:

Columns of MFA_OWB_WORK.I$_DIM_CITY

| Name | Type | Nullable | Default | Comments |
|---|---|---|---|---|
| CITY_ID | NUMBER | Y | | |
| CITY_CODE | VARCHAR2(3) | Y | | |
| CITY_NAME | VARCHAR2(255) | Y | | |
| CITY_SIZE | VARCHAR2(30) | Y | | |
| COUNTRY_CODE | VARCHAR2(3) | Y | | |
| COUNTRY_NAME | VARCHAR2(255) | Y | | |
| COUNTRY_CURRENCY | VARCHAR2(3) | Y | | |
| COUNTRY_SIZE | VARCHAR2(30) | Y | | |
| INSERT_DATE | DATE | Y | | |
| INSERT_AUDIT_ID | NUMBER | Y | | |
| UPDATE_DATE | DATE | Y | | |
| UPDATE_AUDIT_ID | NUMBER | Y | | |

*Abb. 16: Temporäre Tabelle I$_DIM_CITY*

**Mehrere Execution Units**

Zum Abschluss betrachten wir noch mal das gleiche logische Mapping wie zuvor, diesmal allerdings mit zwei Execution Units zum Extrahieren der Daten plus einer Execution Unit zum Schreiben der Daten:

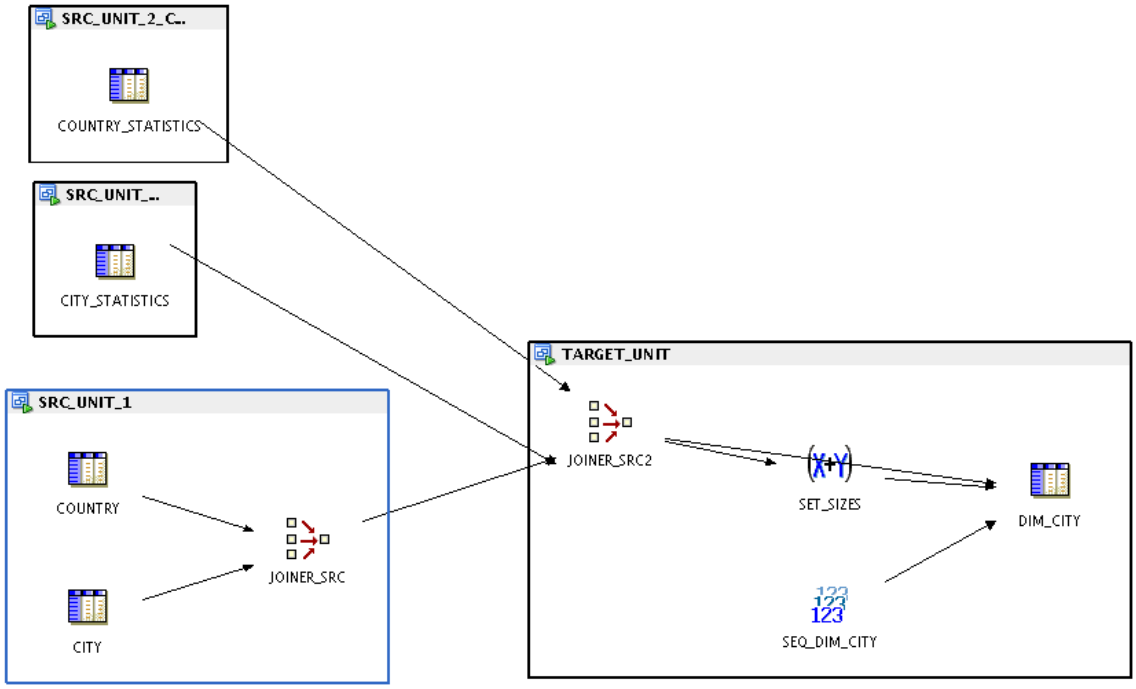


*Abb. 17: Execution View mit drei Units*

Hier werden jetzt auch in dem Work Schema der Target Location mehr Temporäre Objekte erzeugt als in unserem ersten Beispiel:

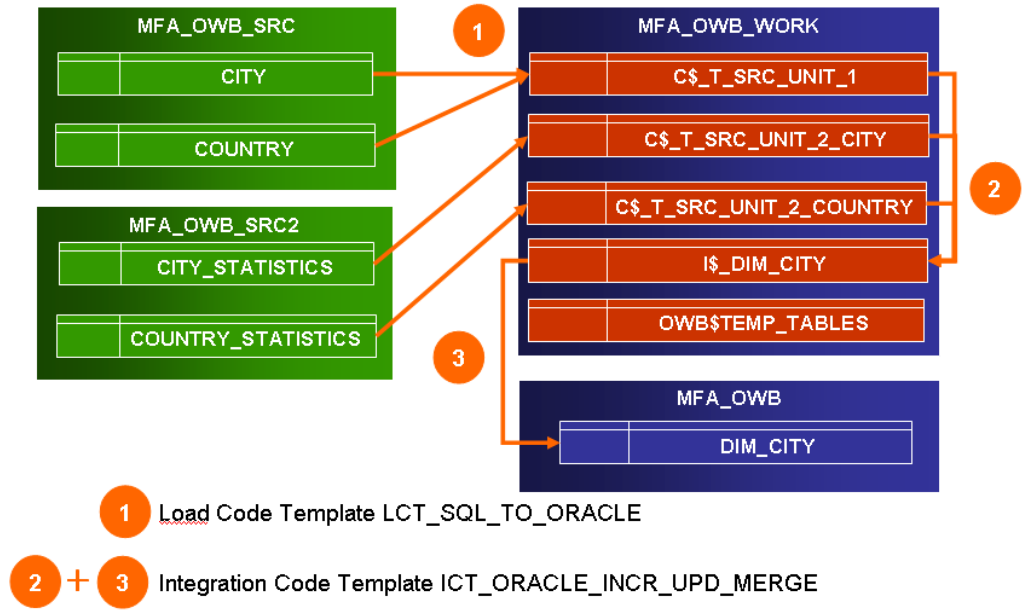


*Abb. 18: Temporäre Objekte in Target Locations während der Mapping Ausführung*

**Fazit**

Mit Template Mappings wird die Mapping Logik (Logical View) von der Art der Implementierung (Code Template) und Ort der Ausführung (Execution View) getrennt. Dadurch gewinnt man deutlich

an Flexibilität. Die Aufteilung in Execution Units will einerseits natürlich gut überlegt sein, andererseits lässt sie sich auch leicht wieder ändern.
Mit den Code Templates wird die Anbindung von nicht-Oracle-Datenbanken erheblich erleichtert. Und mit der Unterstützung für Change Data Capture lässt sich jetzt auch diese Funktionalität komplett in den OWB integrieren.
Einen kleinen Wehmutstropfen gibt es allerdings. Das Generieren des „Intermediate SQLs“ welches man aus den klassischen Mappings kennt, gibt es für Template Mappings (noch?) nicht.
Um die Template Mappings nutzen zu können, ist eine ETL Option Lizenz notwendig, sie sind nicht in der Basis-Version des Oracle Warehousebuilders enthalten.
Alles in allem sind die Template Mappings aber eine sehr mächtige Erweiterung des Oracle Warehousebuilders.

**Kontaktadresse:**

**Carsten Herbe**
Metafinanz-Informationssysteme GmbH
Leopoldstr. 146
D-80804 München

Telefon: +49 (0) 89-360531-5039
Fax: +49 (0) 89-360531-5015
E-Mail carsten.herbe@metafinanz.de
Internet: www.metafinanz.de