

Karol Hajdu, Trivadis GmbH

Oracle Data Integrator (ODI): elegante und effiziente Umsetzung der Best Practice leicht gemacht!

Der Oracle Data Integrator (ODI, ehemals Sunopsis) erfreut sich immer grösserer Beliebtheit in unternehmensweiten DWH- und EAI-Projekten. Gründe dafür gibt es gleich mehrere. Dieser Artikel beleuchtet einen dieser Gründe: das Konzept der Knowledge Modules. Sie erlauben, die Generierung des Quellcodes weitgehend frei anzupassen. In Werbeatikeln für ODI wird über diesen Flexibilität-Vorteil hinaus noch eine weitere Botschaft transportiert: Durch den Kauf von ODI erwerbe der Kunde kostenfrei auch eine Library der Knowledge Modules, und somit auch die Best Practice Lösungen der üblichen Problemstellungen aus dem DWH- und Datenintegrations-Umfeld. Unsere Erfahrungen zeigen: Was einer «Best Practice» nennt, kann der andere als «schlecht oder ungenügend» einstufen. Lesen Sie weiter und bilden Sie sich anhand des aufgeführten Beispiels Ihre eigene Meinung. Unsere Erfahrung: ODI ist ein geeignetes Tool, um vorhandene Best Practices zu automatisieren. Im Falle, dass Sie noch keine Best Practice haben: Die frei verfügbaren Knowledge Modules von Oracle stellen meistens leider noch keine fertige Lösung dar.

Dieser Artikel geht davon aus, dass der Leser mit den Grundkonzepten eines Data Warehouses vertraut ist.

Was ist ein Knowledge Module (KM) und wozu ist es in ODI da?

Im ODI wurde das Konzept der deklarativen SW-Entwicklung recht konsequent und erfolgreich umgesetzt: Die Definitionen davon, WAS gemacht werden soll, WO und WIE es gemacht werden soll, werden weitgehend getrennt festgelegt. Die Transformationslogik (WAS) wird in Form von Eigenschaften und Ausdrücken (Expressions) im sog. Datenmodell (Metadaten über Datenstrukturen) und in sog. ODI Interfaces (Mappings) spezifiziert. Anschliessend kann für diese Objekte die Ausführungslokation (das WO, sog. Execution Location) zugeordnet werden. Ein durchgedachtes, im ODI eingebautes Regelwerk leitet daraus die sog. Daten-

flüsse ab. WIE diese Datenflüsse (d.h. mit welcher Strategie und mit welchen Methoden) in bestimmter Technologie umgesetzt werden sollen, dies wird durch die Zuordnung eines sog. Knowledge Modules wiederum deklarativ festgelegt.

Hier ein **Beispiel**, wie die Aspekte WAS, WO und WIE in einem ODI Interface zusammenspielen: Die Mappings im ODI Interface spezifizieren, dass z.B. zwei Tabellen aus einer Datenbank mit einem JOIN verknüpft werden sollen und dass das Resultat in eine Ziel-tabelle in anderer Datenbank eingefügt werden soll. Die Execution Location des JOINS spezifiziert, in welcher Datenbank der JOIN ausgeführt werden soll. Das Knowledge Module definiert u.a. auch, WIE die Resultatmenge in die Zieltabelle geschrieben wird, z.B.

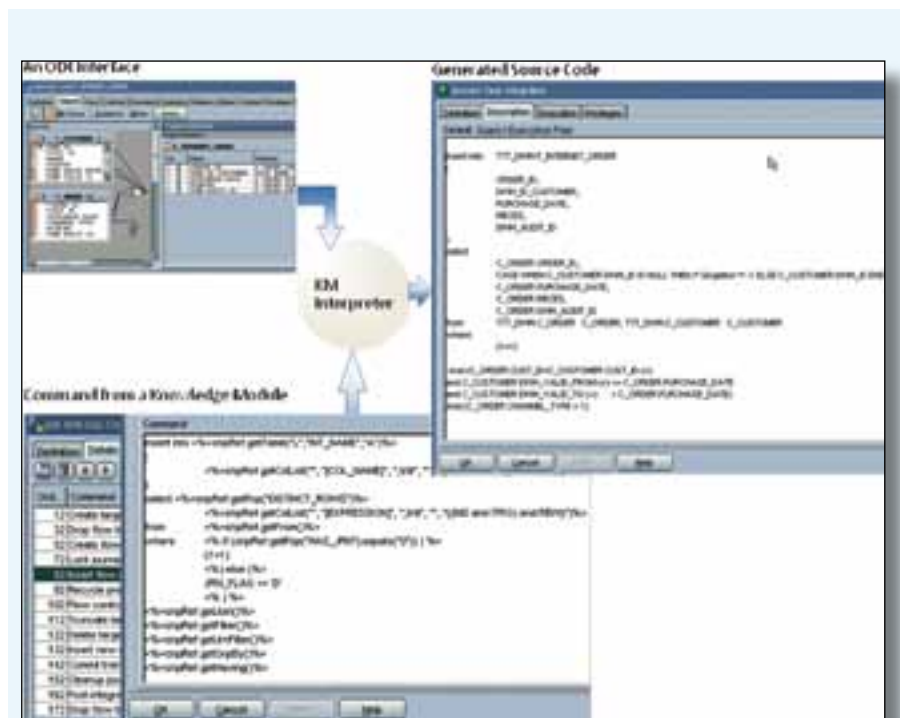


Abb. 1: Prinzip und Verwendungszweck von Knowledge Modules in ODI.

als ein LOOP der Einzel-INSERTs (satzorientiert) oder ein INSERT... SELECT Befehl (mengenorientiert).

Ein Knowledge Module ist eine Sequenz von Schritten (sog. Commands). Diese Schritte definieren, wie der Quellcode für einen Datenfluss im ODI Interface aufgebaut (generiert) werden soll. Die Aufgabe des Knowledge Modules wird in der Abbildung 1 schematisch dargestellt.

Libraries der Knowledge Module: übernehmen und weiterentwickeln

Bei der Installation von ODI wird auch eine Menge von Knowledge Modules für verschiedenste Ziel-Technologien kostenfrei mitgeliefert. Einen Auszug davon finden Sie in der Abbildung 2.

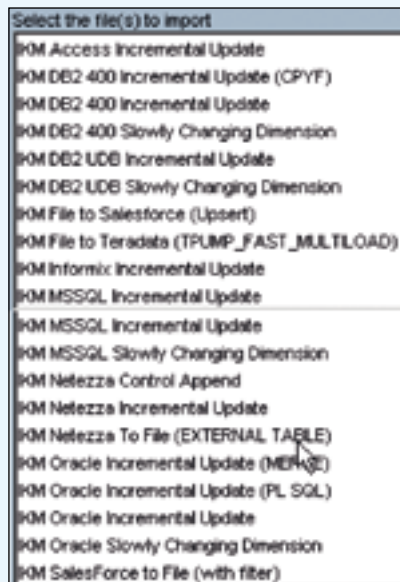


Abb. 2: Auszug aus der Liste der mitinstallierten, frei verfügbaren Knowledge Modules.

Für spezifische Projektbedürfnisse lassen sich diese Knowledge Module einfach erweitern oder auch ganz neue Knowledge Module entwickeln.

Ein Praxisbeispiel: Versionierung der Stammdaten im DWH

Ziel dieses Artikels ist einerseits aufzuzeigen, wie einfach sich im ODI ein bestehendes Knowledge Module erweitern lässt, andererseits aber auch auf die Gefahren hinzuweisen, wenn man mit der Annahme arbeitet, dass die mitgelieferten kostenfreien Knowledge Modules out-of-the-box bereits eine Lösung darstellen.

Wir zeigen dies an einem Beispiel, das in der DWH Praxis sehr verbreitet ist: an der Problemstellung der Versionierung.

Versionierung - das Prinzip

Bei der Versionierung geht es darum, die Änderungen seit dem letzten Datenabgleich zu erkennen, und dieses

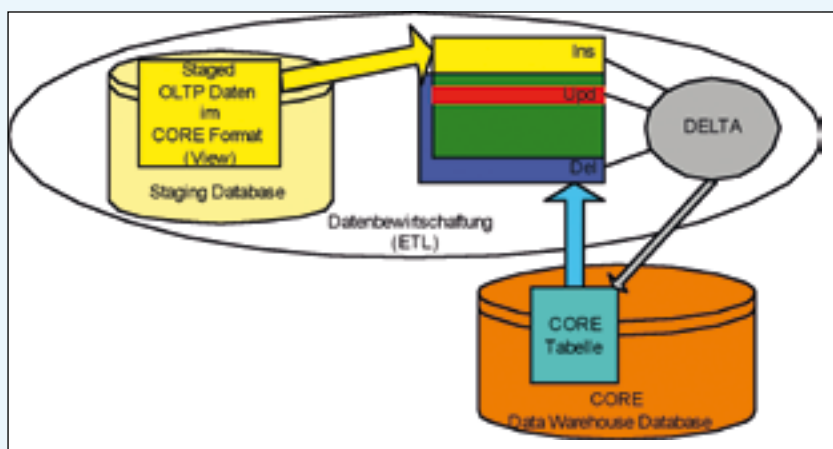


Abb. 3: Prinzip der Versionierung der Stammdaten.

Delta in geeigneter Form in das CORE Data Warehouse abzulegen. Die Abbildung 3 stellt diesen Abgleichprozess grafisch dar.

Die Versionierung werden wir jetzt anhand von ein paar Beispieldaten erklären.

Nehmen wir mal an, dass es in der Staging Area eine Tabelle SA_CUSTOMER gibt. Ihre Struktur und ihr Inhalt ist in Abbildung 4 dargestellt.

Table SA_CUSTOMER		
CUST_ID	NAME	STATUS
4711	John_4711	active
4712	Thomas_4712	suspended
4713	Lisa_4713	active
4714	Simon_4714	inactive
4715	Robert_4715	inactive

Abb. 4: Struktur und Inhalt der Staging Tabelle SA_CUSTOMER.

Diese Staging Tabelle enthält einen Abzug (Snapshot) der Daten aus einem OLTP System für einen bestimmten Zeitpunkt (z.B. den 31.01.2009).

Die Spalte CUST_ID bildet den Primärschlüssel dieser Tabelle.

Weiter nehmen wir als Beispiel an, im CORE Data Warehouse gibt es eine Tabelle C_CUSTOMER. Ihre Struktur und ihr Inhalt zum Zeitpunkt vor dem Versionierungslauf sind in Abbildung 5 dargestellt.

C_CUSTOMER					
CUST_ID	CUST_ID	ValidFrom	ValidTo	NAME	STATUS
-1000	4710	05.01.2009	01.01.2400	Bill_4710	suspended
-1001	4711	01.01.2009	01.01.2400	John_4711	active
-1002	4712	01.01.2009	10.01.2009	ThoSmas_4712	active
-1003	4712	10.01.2009	01.01.2400	ThoSmas_4712	suspended
-1004	4714	10.01.2009	01.01.2400	SiQmon_4714	active
-1005	4715	08.01.2009	01.01.2400	Robert_4715	active

Abb. 5: Struktur und Inhalt der CORE DWH Tabelle C_CUSTOMER (vor dem Versionierungslauf).

Die Spalte DWH_ID bildet den sog. Surrogate Key, der zugleich auch den Primärschlüssel der Tabelle darstellt. Die Tabelle C_CUSTOMER enthält alle historischen Veränderungen des Attributs STATUS. Das Attribut STATUS ist ein versionierungsrelevantes Attribut (sog. SCD2 Attribut). Die technischen Attribute ValidFrom und ValidTo schränken die Gültigkeit der einzelnen Versionen ein (der spezielle Wert 01.01.2400 bedeutet, die Version ist gültig «for ever»). Das Attribut NAME ist kein versionierungsrelevantes Attribut (sog. SCD1 Attribut). Das Attribut CUST_ID ist ein sog. Natural Key. Kombiniert mit einer Zeiteinschränkung ermöglicht es

die eindeutige Identifikation einer bestimmten Kundenversion.

Die Abbildung 6 zeigt den Inhalt der Tabelle C_CUSTOMER nach dem Versionierungslauf.

C_CUSTOMER					
DWH_ID	CUST_ID	ValidFrom	ValidTo	NAME	STATUS
-1000	4710	05.01.2009	31.01.2009	Bill_4710	suspended
-1006	4713	31.01.2009	01.01.2400	Lisa_4713	active
-1001	4711	01.01.2009	01.01.2400	John_4711	active
-1002	4712	01.01.2009	10.01.2009	Thomas_4712	active
-1003	4712	10.01.2009	01.01.2400	Thomas_4712	suspended
-1004	4714	10.01.2009	31.01.2009	Simon_4714	active
-1007	4714	31.01.2009	01.01.2400	Simon_4714	inactive
-1005	4715	08.01.2009	31.01.2009	Robert_4715	active
-1008	4715	31.01.2009	01.01.2400	Robert_4715	inactive

Abb. 6: Inhalt der CORE DWH Tabelle nach dem Versionierungslauf.

Die Version mit DWH_ID=-1000 wurde abgeschlossen (ValidTo auf 31.01.2009 gesetzt), denn der Eintrag mit CUST_ID=4710 kommt in der SA-Tabelle nicht mehr vor (wurde im OLTP System wahrscheinlich gelöscht).

Die Version mit DWH_ID=-1006 wurde neu hinzugefügt, denn es gab in der CORE Tabelle noch keine Version für CUST_ID = 4713.

Die Version mit DWH_ID=-1004 wurde abgeschlossen (ValidTo auf 31.01.2009 gesetzt), denn beim CUST_ID = 4714 hat sich der STATUS geändert (von «active» auf «inactive») und diese Änderung führt zur Erstellung einer neuen Version (mit DWH_ID=-1007).

Das Gleiche geschah auch beim Kunden mit CUST_ID = 4715. Dort hat sich der STATUS geändert (von «active» auf «inactive»). Die Version mit DWH_ID=-1005 wurde abgeschlossen (ValidTo auf 31.01.2009 gesetzt) und eine neuen Version (mit DWH_ID=-1008) erstellt.

Beim Kunden mit CUST_ID = 4712 hat sich der Name geändert (vom «ThoSmas_4712» auf «Thomas_4712»). Die Änderung an diesem Attribut ist jedoch nicht versionierungsrelevant, es wurde also keine neue Version gebildet. Alle bestehenden Versionen werden mit dem neuen Wert angepasst.

Das Gleiche geschah auch beim Kunden mit CUST_ID = 4714. Dort hat sich der Name geändert (von «SiQmon_4714» auf «Simon_4714»).

Versionierung - die möglichen Umsetzungen

Es gibt mehrere Wege, wie die Versionierung umgesetzt werden kann.

Die Umsetzungen unterscheiden sich meistens darin, wie gut sie folgende Ansprüche erfüllen:

- Die Umsetzung implementiert die Zielsetzung korrekt, d.h. enthält keine Denkfehler.
 - Sie ist robust genug, so dass auch Randfälle und komplexe Fälle richtig behandelt werden.
- Sie ist in der gegebenen Technologie (z.B. Oracle11g DBMS) lauffähig, d.h. enthält keine Syntaxfehler.
- Sie berücksichtigt die Gegebenheiten der Technologie (Stärken, Limitierungen, Bugs etc.).
 - Sie ist effizient im Sinne ihres Verwendungszwecks (Durchsatz bei Mengenverarbeitung; Latenzzeit bei Einzelverarbeitung).
 - Sie führt keinen künstlichen Flaschenhals in die Verarbeitung ein, der die Skalierbarkeit blockieren würde.

In diesem Artikel werden zwei Umsetzungen der Versionierung in der Zieltechnologie Oracle 11g vorgestellt:

- a) Die Umsetzung anhand des Standard Knowledge Modules «IKM Oracle Slowly Changing Dimension», das von Oracle mit der ODI Standard-Installation mitgeliefert wird.
- b) Die Best Practice der Trivadis.

Aus Gründen der besseren Verständlichkeit beginnen wir mit der Vorstellung der Best Practice der Trivadis.

Versionierung - Best Practice der Trivadis

Listing 1 und Listing 2 zeigen die Best Practice für die Umsetzung der Versionierung in der Zieltechnologie Oracle11g DBMS.

- Sie verwendet den FULL OUTER JOIN, um das Delta zu finden.
- Sie minimiert die Anzahl der Zugriffe auf die Tabellen.
- Alle Operationen der generierten SQL Befehle sind voll parallelisierbar.
- Die Schreiboperationen mit temporärem Inhalt verursachen nur ein minimales Undo- und Redo-Aufkommen (I/O Operationen).

porärem Inhalt verursachen nur ein minimales Undo- und Redo-Aufkommen (I/O Operationen).

Im ersten Schritt (Listing 1) wird das Delta ermittelt und in einer temporären Tabelle festgehalten, weil auf diese Resultatmenge mehrmals zugegriffen werden muss. Die Flags (Attribute der Delta-Tabelle) halten fest, um welche Art der Änderung es sich handelte (DML_I, DML_D, DML_U_SCD2, DML_U_SCD1).

```
-- Step 1
-- insert changes (Delta) into I$(FULL OUTER J)

insert /** APPEND NOLOGGING */ into TTT_DWH_ETL_RUN.I$_C_CUSTOMER
(
    DWH_ID ,
    CUST_ID ,
    NAME ,
    STATUS ,
    DML_D ,
    DML_I ,
    DML_U_SCD1 ,
    DML_U_SCD2
)
select
    DWH_ID,
    CASE WHEN DML_D=1 THEN OLD_CUST_ID ELSE CUST_ID END AS CUST_ID,
    CASE WHEN DML_D=1 THEN OLD_NAME ELSE NAME END AS NAME,
    CASE WHEN DML_D=1 THEN OLD_STATUS ELSE STATUS END AS STATUS,
    DML_D ,
    DML_I ,
    DML_U_SCD1 ,
    DML_U_SCD2
from ( select
    C.DWH_ID,
    SA.CUST_ID,
    SA.NAME,
    SA.STATUS,
    C.CUST_ID AS OLD_CUST_ID,
    C.NAME AS OLD_NAME,
    C.STATUS AS OLD_STATUS
, CASE WHEN SA.CUST_ID IS NULL THEN 1 ELSE 0 END AS DML_D
, CASE WHEN C.CUST_ID IS NULL THEN 1 ELSE 0 END AS DML_I
, CASE WHEN C.CUST_ID IS NOT NULL
    AND SA.CUST_ID IS NOT NULL
    AND ( C.NAME <> SA.NAME )
    THEN 1 ELSE 0 END AS DML_U_SCD1
, CASE WHEN C.CUST_ID IS NOT NULL
    AND SA.CUST_ID IS NOT NULL
    AND ( C.STATUS <> SA.STATUS )
    THEN 1 ELSE 0 END AS DML_U_SCD2
from (SELECT
    SA_CUSTOMER.CUST_ID AS CUST_ID,
    SA_CUSTOMER.NAME AS NAME,
    SA_CUSTOMER.STATUS AS STATUS
from TTT_DWH_SA.SA_CUSTOMER SA_CUSTOMER
) SA
full outer join
(SELECT *
from TTT_DWH.C_CUSTOMER
where DWH_VALID_TO = to_date ('01-01-2400', 'mm-dd-yyyy')
) C
on
SA.CUST_ID = C.CUST_ID
)
WHERE DML_I+DML_U_SCD1+DML_U_SCD2+DML_D > 0
;
```

Listing 1: Abgleich und Erkennen der Differenzen zwischen der Stage Tabelle und der CORE Tabelle.

Anschließend (Listing 2) werden entsprechende DML Befehle gegen die CORE Tabelle abgesetzt.

```
-- Step 2
-- update target with SCD1 changes
MERGE
INTO -- target table
    TTT_DWH.C_CUSTOMER T
USING
    (SELECT * FROM TTT_DWH_ETL_RUN.IS_C_CUSTOMER WHERE DML_U_SCD1 = 1) D
ON (T.CUST_ID = D.CUST_ID )
WHEN MATCHED THEN
    UPDATE
        SET
            T.NAME = D.NAME
;

-- Step 3
-- close versions of SCD2
MERGE
INTO -- target table
    TTT_DWH.C_CUSTOMER T
USING
    (SELECT * FROM TTT_DWH_ETL_RUN.IS_C_CUSTOMER WHERE DML_U_SCD2 = 1 OR DML_D = 1) D
ON (T.DWH_ID = D.DWH_ID )
WHEN MATCHED THEN
    UPDATE
        SET
            T.DWH_VALID_TO = to_date('31-01-2009', 'dd-mm-yyyy')
                          /* expression for logical_snapshot_date */ ;

-- Step 4
-- insert new rows (SCD1+SCD2)
INSERT INTO TTT_DWH.C_CUSTOMER
(
    DWH_ID ,
    CUST_ID ,
    NAME ,
    STATUS ,
    DWH_VALID_FROM ,
    DWH_VALID_TO ,
    DWH_AUDIT_ID
)
SELECT
    TTT_DWH.C_CUSTOMER_S.nextval,
    CUST_ID ,
    NAME ,
    STATUS ,
    to_date('31-01-2009', 'dd-mm-yyyy') /* expression for logical_snapshot_date */,
    to_date ('01-01-2400', 'mm-dd-yyyy'),
    1
FROM TTT_DWH_ETL_RUN.IS_C_CUSTOMER D
WHERE DML_U_SCD2 = 1 OR DML_I = 1
;
```

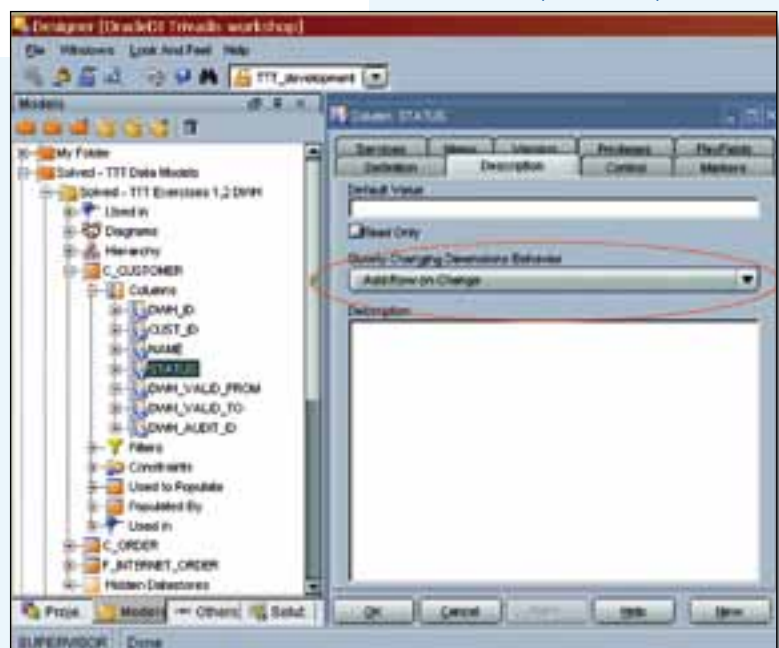
Listing 2: Durchführung der DML Befehle, welche die Differenzen zwischen Stage und CORE Tabellen beseitigen.

Versionierung - Umsetzung mit «IKM Oracle Slowly Changing Dimension»

In diesem Abschnitt werden wir die Umsetzung der Versionierung im ODI zeigen. Für die Generierung werden wir das frei mitgelieferte Knowledge Module «IKM Oracle Slowly Changing Dimension» verwenden.

Als Erstes definieren wir im ODI Designer Tool das sog. SCD Verhalten: Für jede Spalte der Tabelle C_CUSTOMER wird definiert, wie sich die Spalte in Bezug auf die Versionierung verhält, bzw. welche Rolle sie hier übernimmt (siehe Abbildung 7).

Abb. 7: Das SCD-Verhalten der Spalten lässt sich mit dem Designer Tool im Model (Metadaten) definieren.



Das SCD-Verhalten für Spalten unserer Beispielstabelle C_CUSTOMER haben wir entsprechend der Ausführungen in 3.1 konfiguriert (siehe Abbildung 8).

Column	SCD Behavior
DWH_ID	Surrogate Key
CUST_ID	Natural Key
NAME	Overwrite on Change
STATUS	Add Row on Change
DWH_VALID_FROM	Starting Timestamp
DWH_VALID_TO	Ending Timestamp

Abb. 8: Im ODI wurde den Spalten der Tabelle C_CUSTOMER das entsprechende SCD-Verhalten zugeordnet.

Diese Angaben werden im Repository des ODI gespeichert. Das Knowledge Module, das für die Generierung des Quellcodes zuständig ist, kann diese Angaben aus dem Repository herauslesen und bei der Generierung berücksichtigen.

Im ODI erstellen wir weiter ein einfaches Interface¹, das die Daten aus der Tabelle SA_CUSTOMER herausliest, mit dem Inhalt in C_CUSTOMER abgleicht und entsprechend den erkannten Differenzen anpasst.

Die Transformationen des Abgleichs und der Versionierung (z.B. der Lesezugriff auf die CORE Tabelle und der Vergleich der Spaltenwerte) werden nicht direkt im ODI Interface gezeichnet. Da dies in jedem ODI Interface für Versionierung gleich (generisch) gehalten werden soll, werden diese im Knowledge Module definiert. Im ODI Interface erfolgt nur die Zuordnung des entsprechenden Knowledge Modules, das die Versionierung implementiert – also die indirekte Definition.

¹ Interface: Synonym für Mapping in anderen Umgebungen wie Oracle Warehouse Builder oder Informatica's PowerCenter.

Das ODI Interface ist in diesem Fall also sehr trivial – siehe Abbildung 9.

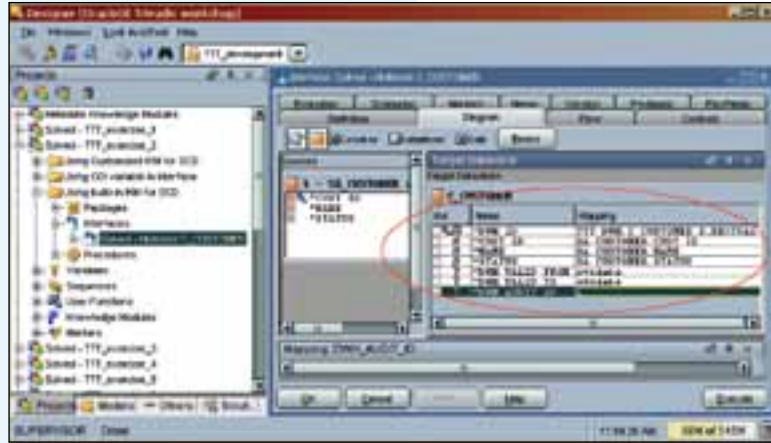


Abb. 9: Die Versionierung wird im ODI mit einem simplen Interface definiert.

Wie bereits erwähnt: Im ODI Interface erfolgt die Zuordnung des Knowledge Modules, das die Versionierung implementiert.

Wir ordnen hier das von Oracle kostenfrei zur Verfügung gestellte Knowledge Module «IKM Oracle Slowly Changing Dimension» zu (Abbildung 10).

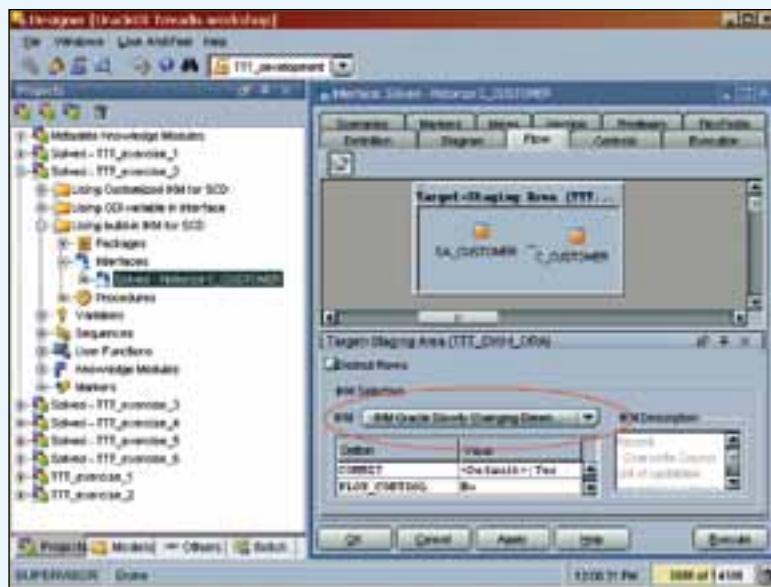


Abb. 10: Dem Datenfluss wird das Standard Knowledge Module «IKM Oracle Slowly Changing Dimensions» zugeordnet.

Wir möchten jetzt die Qualität der Umsetzung durch dieses KM untersuchen. Wir nehmen dazu die aktuellste Version von diesem KM – siehe Abbildung 11.

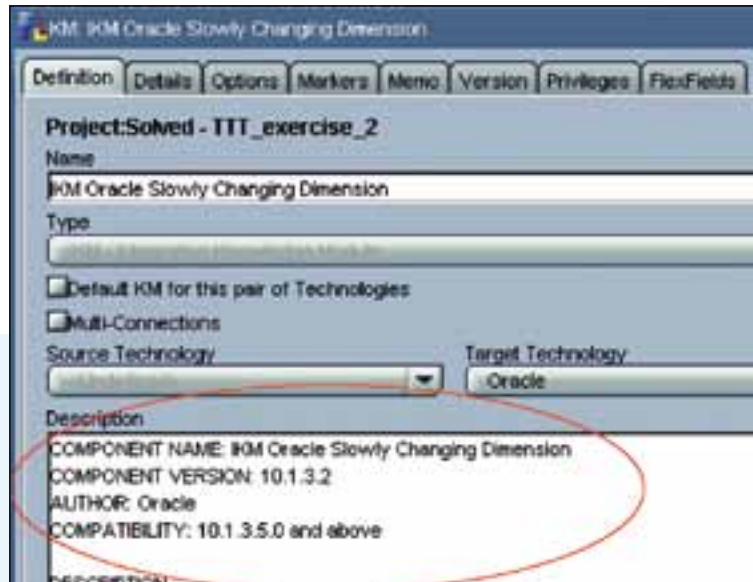


Abb. 11: Zum Zeitpunkt der Erstellung dieses Artikels war von diesem KM die Version 10.1.3.2 verfügbar.

Schauen wir uns die Struktur näher an: Die Schritte, aus welchen dieses Knowledge Module besteht, sind in der Abbildung 12 dargestellt.

In den rot markierten Schritten (Abbildung 12) finden die eigentlichen Datentransformationen der Versionierung statt.

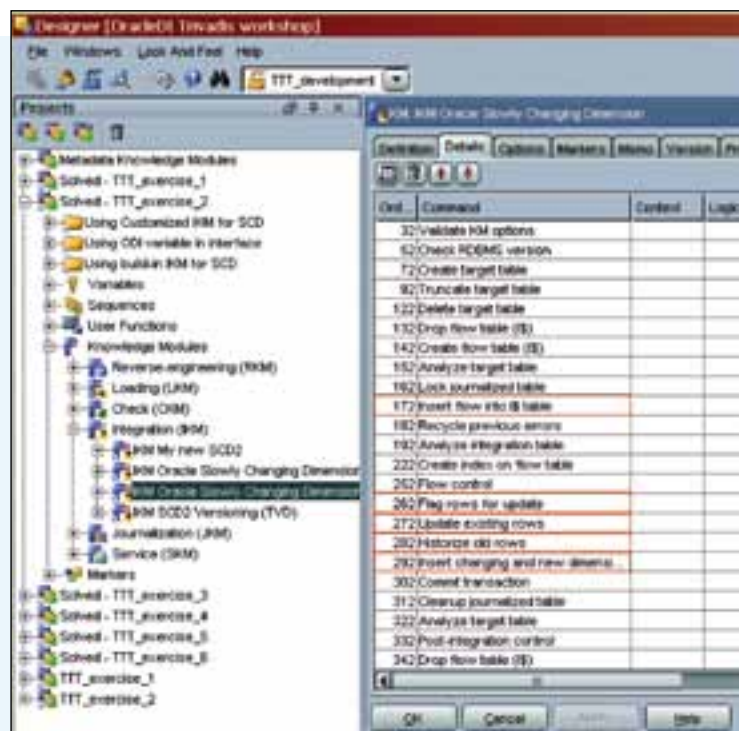


Abb. 12: IKM Oracle Slowly Changing Dimensions besteht aus folgenden Schritten (Commands).

Schauen wir uns jetzt den Quellcode, den diese Schritte generieren, näher an (Listing 3 bis Listing 7).

```

insert /** APPEND */ into TTT_DWH_ETL_RUN.I$_C_CUSTOMER
(
    CUST_ID,
    NAME,
    STATUS,
    DWH_VALID_FROM,
    IND_UPDATE
)
select
    SA_CUSTOMER.CUST_ID,
    SA_CUSTOMER.NAME,
    SA_CUSTOMER.STATUS,
    sysdate,
    'I' IND_UPDATE
from
    TTT_DWH_SA.SA_CUSTOMER SA_CUSTOMER
where
    (1=1)
and NOT EXISTS (
    select ,x'
    from
        TTT_DWH.C_CUSTOMER T
    where
        T.CUST_ID = CUST_ID
    and
        ((SA_CUSTOMER.NAME = T.NAME) or (SA_CUSTOMER.NAME IS NULL and T.NAME IS NULL))
    and
        DWH_VALID_TO = to_date ('01-01-2400', 'mm-dd-yyyy')
)

```

Listing 3: Quellcode, der aus dem Schritt «insert flow into I\$table» generiert wurde.

```

update
    TTT_DWH_ETL_RUN.I$_C_CUSTOMER S
set
    S.IND_UPDATE = 'U'
where
    (S.CUST_ID)
    in
        (
            select
                T.CUST_ID
            from
                TTT_DWH.C_CUSTOMER T
            where
                ((S.STATUS = T.STATUS) or ((S.STATUS is null) and (T.STATUS is null)))
            and
                DWH_VALID_TO = to_date ('01-01-2400', 'mm-dd-yyyy')
        )

```

Listing 4: Quellcode, der aus dem Schritt «Flag rows for update» generiert wurde.

```

update
    TTT_DWH.C_CUSTOMER T
set
    (
        T.NAME
    ) =
        (
            select
                X.NAME
            from
                TTT_DWH_ETL_RUN.I$_C_CUSTOMER X
            where
                X.CUST_ID = T.CUST_ID
            and
                X.IND_UPDATE = 'U'
        )
where
    (T.CUST_ID)
    in
        (
            select
                S.CUST_ID
            from
                TTT_DWH_ETL_RUN.I$_C_CUSTOMER S
            where
                S.IND_UPDATE = 'U'
        )
and
    DWH_VALID_TO = to_date ('01-01-2400', 'mm-dd-yyyy')

```

Listing 5: Quellcode, der aus dem Schritt «Update existing rows» generiert wurde.

Listing 6: Quellcode, der aus dem Schritt «Historize existing rows» generiert wurde.

```
update   TTT_DWH.C_CUSTOMER T
set      (
          T.DWH_VALID_TO
        ) = (
          select
            S.DWH_VALID_FROM
          from   TTT_DWH_ETL_RUN.I$_C_CUSTOMER S
          where  S.CUST_ID   = T.CUST_ID
          and    S.IND_UPDATE = 'I'
        )
where    (T.CUST_ID)
in       (
          select
            X.CUST_ID
          from   TTT_DWH_ETL_RUN.I$_C_CUSTOMER X
          where  X.IND_UPDATE = 'I'
        )
and      DWH_VALID_TO   = to_date ('01-01-2400', 'mm-dd-yyyy')
```

Listing 7: Quellcode, der aus dem Schritt «Insert changing and new dimensions» generiert wurde.

```
insert into   TTT_DWH.C_CUSTOMER
(
  CUST_ID,
  NAME,
  STATUS,
  ,DWH_ID
  ,DWH_VALID_FROM
  ,DWH_VALID_TO
)
select       CUST_ID,
            NAME,
            STATUS
            ,TTT_DWH.C_CUSTOMER_S.NEXTVAL
            ,DWH_VALID_FROM
            ,to_date ('01-01-2400', 'mm-dd-yyyy')
from         TTT_DWH_ETL_RUN.I$_C_CUSTOMER S
where        S.IND_UPDATE = 'I'
```

Ist diese Umsetzung der Versionierung korrekt?

Schauen wir uns das Resultat an, das einen Ladelauf dieser Umsetzung produziert. Die rot markierten Stellen in der Abbildung 13 weisen auf Stellen hin, wo die Umsetzung fehlerhaft ist:

Befund 1: in Fällen, bei welchen sich der STATUS geändert hat (wie z.B. bei CUST_ID=4715), erzeugt dieser Quellcode keine neue Version (fehlende Version ist in Abbildung 14 kursiv markiert): Denkfehler im Listing 3 (Schritt «insert flow into I\$table»).

Befund 2: Falls sich das Attribut NAME (SCD1=overwriting history) ändert, dann wird nur die letzte Version geändert, nicht aber die volle Geschichte (CUST_ID=4712): Denkfehler im Listing 5 (Schritt «update existing rows»).

Befund 3: Falls der STATUS (SCD2) und gleichzeitig auch der NAME (SCD1) ändert (CUST_ID=4714), dann wird die Änderung am Attribut NAME in vorherigen Versionen nicht übertragen: Denkfehler im Listing 4 (Schritt «Flag rows for update»).

Befund 4: Falls ein Eintrag in der SA-Tabelle nicht mehr vorkommt, führt dies nicht zum Abschliessen der aktuell gültigen Version (CUST_ID=4710). Die Tatsache, dass diese Funktionalität fehlt, ist in dem Knowledge Module nicht dokumentiert.

C_CUSTOMER					
DWH_ID	CUST_ID	ValidFrom	ValidTo	NAME	STATUS
-1000	4710	05.01.2009	01.01.2400	Bill_4710	suspended
-1006	4713	31.01.2009	01.01.2400	Lisa_4713	active
-1001	4711	01.01.2009	01.01.2400	John_4711	active
-1002	4712	01.01.2009	10.01.2009	ThoSmas_4712	active
-1003	4712	10.01.2009	01.01.2400	Thomas_4712	suspended
-1004	4714	10.01.2009	01.01.2400	SiQmon_4714	active
-1007	4714	31.01.2009	01.01.2400	Simon_4714	inactive
-1005	4715	10.01.2009	01.01.2400	Robert_4715	active
	4715	31.01.2009	01.01.2400	Robert_4715	inactive

Abb. 13: Inhalt der C_CUSTOMER nach dem Ladelauf der Versionierung mit «IKM Oracle Slowly Changing Dimension».

Ist diese Umsetzung der Versionierung effizient und skalierbar?

In grösseren DWH Umgebungen ist es üblich, dass einige der Dimensionstabellen recht viele Einträge enthalten (z.B. mehrere Millionen Datensätze). Stellen wir uns vor, unsere Beispielstabelle C_CUSTOMER wäre so eine.

In der aufgeführten Umsetzung wird auf die Tabelle C_CUSTOMER fünfmal zugegriffen (2x lesend, 2x mit Update, 1x mit Insert). Dies lässt sich definitiv effizienter gestalten!

Wer davon ausgeht, dass das von Oracle mitgelieferte «IKM Oracle Slowly Changing Dimension» eine korrekte und effiziente Lösung für die Versionierung darstellt, der kann böse Überraschungen erleben!

Generell empfehlen wir sehr, die Qualität der später zum Einsatz kommenden Knowledge Modules ziemlich genau unter die Lupe zu nehmen, z.B. in Form vom Proof of Concept und anhand repräsentativer Use Cases.

Wenn man mit der Umsetzung der üblichen Fragestellungen bereits eigene Best Practice zur Verfügung hat, dann ist man eventuell schneller am Ziel, wenn man eigene Knowledge Module selber entwickelt.

Eigene Knowledge Module entwickeln: simpel, vorausgesetzt die Erfahrung mit DWH Umsetzung ist vorhanden

Wie das vorher aufgeführte Beispiel aufgezeigt hat, kommen wir beim ODI Einsatz nicht darum herum, eigene Knowledge Module zu entwickeln.

Und hier kommt die gute Nachricht: Das Entwickeln an sich ist gar nicht so schwierig. Wir zeigen dies jetzt wieder anhand eines Beispiels. Ziel des Beispiels: die Best Practice der Trivadis für die Umsetzung der Versionierung (vorgestellt im Abschnitt 3.3) soll nun in ein neues Knowledge Module übertragen werden.

Wir legen also ein neues Knowledge Module an und benennen es z.B. «IKM SCD Versioning (TVD)». Dann legen wir

die einzelnen Schritte (Commands) an – siehe Abbildung 14. Die Definition der unterstützenden Schritte übernehmen wir einfach vom Standard-IKM, wir ändern nur das Wesentliche: die Schritte, welche den Quellcode für die Daten-transformationen generieren. Diese sind in Abbildung 14 rot markiert.

Um das Prinzip der Entwicklung von Commands der Knowledge Module zu erklären, schauen wir uns mal einen Schritt an: den Schritt «close versions of SCD2».

Im Listing 8 finden Sie den Command für diesen Schritt. Er generiert das zweite MERGE Statement im Listing 2, das für das Schliessen der Versionen zuständig ist (siehe 3.3 Versionierung – Best Practice der Trivadis).

Die Platzhalter «%...%» werden zum Generierungszeitpunkt ausgewertet. Sie entsprechen den Aufrufen von dokumentierten Java-Methoden. Diese Methoden lesen die Attributwerte des

```

MERGE
INTO -- target table
<%=snpRef.getTable(„L“,“TARG_NAME“,“A“) %> T
USING
(SELECT * FROM <%=snpRef.getTable(„L“,“INT_NAME“,“A“) %> WHERE DML_U_SCD2 = 1 OR DML_D = 1) D
ON (<%=snpRef.getColList(““,“T.[COL_NAME] = D.[COL_NAME] “, “ and \n\t“, ““, “SCD_SK“) %>)
WHEN MATCHED THEN
UPDATE
SET T.DWH_VALID_TO = <%=snpRef.getColList(““, “[EXPRESSION]“, “, \n\t“, ““, “SCD_START“) %>
    
```

Listing 8: Command für den Schritt «close versions of SCD2», der den Quellcode des zweiten MERGE-Statements im Listing 2 generiert.

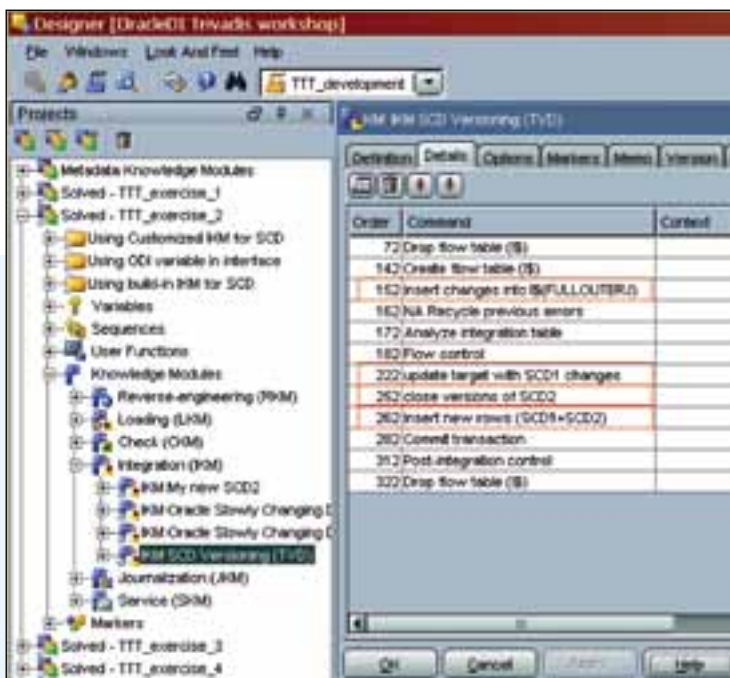


Abb. 14: Struktur des neu angelegten Knowledge Modules «IKM SCD Versioning (TVD)».

jeweiligen, zu generierenden ODI Interfaces (Mappings) aus dem ODI Repository heraus und ermöglichen, dass der Generierungsprozess diese Werte berücksichtigt.

Wer diese dokumentierten Methoden-Aufrufe beherrscht, kann selber eigene Knowledge Module schreiben.

Fazit

Sich das Knowhow anzueignen, **wie** man Knowledge Module entwickelt, ist relativ simpel.

Das Knowhow aufzubauen, **was** man mit den Knowledge Modulen erzielen soll, d.h. wie eine korrekte und effiziente Umsetzung aussehen soll,

dies war und bleibt auch weiterhin komplex. Es braucht vor allem mehrjährige praktische Erfahrung in der Umsetzung von DWH Projekten.

Mit den mitgelieferten KMs von Oracle hat man eine Basis für die Umsetzung, nicht aber eine Lösung.

Sind Sie gerade dabei oder kurz davor, eine Lösung zu bauen, und haben noch Fragen zu den Knowledge Modules? Dann kontaktieren Sie mich einfach: Wahrscheinlich haben wir an ähnlichen Stellen bereits Erfahrungen gesammelt und können Sie recht effizient unterstützen.

Viel Erfolg beim Einsatz des Oracle Data Integrator in Ihrem Data Warehouse wünscht ■

Contact

Trivadis GmbH

Karol Hajdu

E-Mail:

karol.hajdu@trivadis.com