Automatisierung im DWH – Das Leben erleichtern mit dem ODI

Dr. Jens Bleiholder, Marian Strüby
OPITZ CONSULTING Deutschland GmbH
Standort Berlin

Ulf Jeffke
Vodafone Kabel Deutschland GmbH
Standort Berlin

Schlüsselworte

Data Vault; Data Warehouse; Datenbewirtschaftung & ETL; Datenmodellierung; DWH & Datenintegration; Oracle Data Integrator (ODI)

Einleitung

Modularisierung, Standardisierung, Automatisierung. Mit diesen drei Stichworten kann man die Vorteile des ODI 12c als ETL-Tool im eigenen DWH-Projekt auf den Punkt bringen. Der ODI eröffnet hier eine ganze Reihe an Möglichkeiten die Entwicklung eines DWH zu beschleunigen, dem Entwickler Arbeit abzunehmen und gleichzeitig die Entwicklung auch zuverlässiger und fehlerfreier zu machen. Dabei wird die Entwicklung noch wesentlich effizienter, wenn man auch bei Datenmodellierung und Architektur auf Modularisierung, Standardisierung und Automatisierung achtet. Anhand von Beispielen aus verschiedenen Kundenprojekten stellen wir unsere Erfahrungen auf diesem Gebiet vor und zeigen, wie der ODI 12c dabei hilft ein DWH mit "faulen" Entwicklern, also größtmöglich automatisiert, aufzubauen: Generieren von Datenmodellen und zugehörigen Mappings, automatisches Deployment, Anpassung von Knowledge Modulen, etc. Den Kunden freut es, bekommt er nun mehr DWH für sein Geld. Den Entwickler freut es, muss er nun nicht mehr die langweiligen immer gleichen Arbeiten erledigen. Den Endanwender freut es, bleibt nun mehr Zeit übrig, um auf seine Probleme und Businesslogik einzugehen

Standardisierung und Modularisierung

Durch die Schaffung und Einhaltung von Standards lassen sich DWH-Projekte wesentlich einheitlicher entwickeln. Neue Mitarbeiter im DWH-Team kommen schneller ins Projekt hinein, alles ist aus einem Guss und man findet sich schneller zurecht. Das zeigt sich nicht nur in einer Schichtenarchitektur, die verschiedenen Schichten auch verschiedene Aufgaben zuweist und regelt welche Funktionalität in welcher Schicht erledigt wird, sondern auch in Sachen Datenmodellierung. Hier kennt man üblicherweise eine Reihe von Namenskonventionen zur Benennung von Spalten, Indizes und Schlüsselbeziehungen. Trotzdem sehen die Tabellen in einem 3NF-Schema alle verschieden aus und enthalten üblicherweise auch alle drei Aspekte einer Entität, nämlich a) Schlüssel/Identität, b) Eigenschaften und c) Beziehungen. Die Data Vault Modellierungstechnik geht hier noch einen Schritt weiter und standardisiert auch die Tabellen, indem jeder dieser drei Aspekte in einen eigenen Tabellentyp ausgelagert wird. Abbildung 1 zeigt ganz grob die Aufteilung einer 3NF-Tabelle „Kunde“ auf die verschiedenen Hub-, Link- und Satellitentabellen in Data Vault. Jeder Aspekt ist dabei durch eine eigene Farbe gekennzeichnet

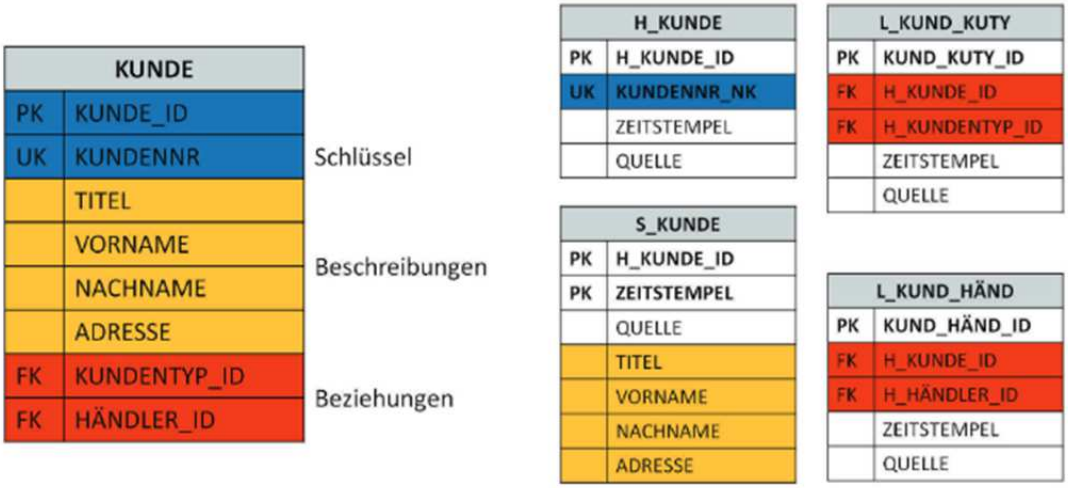

Abbildung 1: Darstellung der Trennung der einzelnen Aspekte (Schlüssel/Identität, Beschreibungen und Beziehungen) einer Entität „Kunde“ bei der Data Vault Modellierung (rechts im Bild) im Gegensatz zur Modellierung in 3NF (im Bild link), bei der alle drei Aspekte in einer Tabelle gemeinsam vorhanden sind.

Durch Standardisierung in der Datenmodellierung ist es möglich standardisierte Beladungsregeln festzulegen, die jeweils für einen Tabellen-„Typ“ gelten und immer gleich sind. In unseren Projekten nutzen wir nicht nur ähnliche Namensregeln, sondern auch standardisierte Beladungsregeln. Diese sind durch Data Vault vorgegeben, können aber immer wieder auch angepasst werden. Beispiele für solche Beladungsregeln sind die Unterscheidung von SCD2-Beladung für Satelliten nach „mit End-Dating“ und „ohne End-Dating“, die Beladung mittels Partition Exchange Loading, etc.

Auch das ETL-Tool, in unserem Fall der Oracle Data Integrator (ODI), sieht einen standardisierten Ablauf bei der Integration von Daten vor, der aus der Abfolge verschiedener Knowledge-Module besteht, die jeweils aktiviert/deaktiviert sowie konfiguriert werden können. Abbildung 2 zeigt den im ODI verwendeten Standardablauf zur Integration von Daten. Üblicherweise werden aber nicht alle Schritte durchlaufen, die am häufigsten verwendeten Knowledge Module sind die Load und die Integration Knowledge Module.

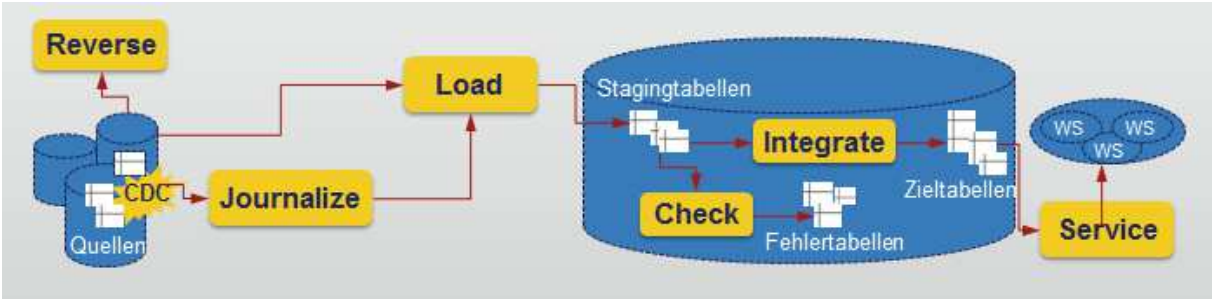

Abbildung 2: Darstellung des Integrations-Workflows im ODI, unter der Verwendung von Knowledge Modulen.

Die gelben Boxen stellen dabei die Knowledge-Module dar, die jeweils in unterschiedlichen Varianten existieren und damit unterschiedlich funktionieren. Sie stellen grob gesagt die unterschiedlichen Arten dar, auf die man Daten laden (Load-Knowledge Module) oder integrieren (Integration Knowledge Module) kann. Durch eine solche Modularisierung kann man die oben genannten Beladungsregeln umsetzen und jederzeit austauschen. So gibt es je ein Integration Knowledge Modul für eine Hub-, eine Link-, und eine Satellitenbeladung. Knowledge Module sorgen im ODI also dafür, dass gleiche Dinge auch immer wieder gleich getan werden.

Nicht nur bei der Datenmodellierung und ETL-Entwicklung, sondern auch beim Testing und beim Deployment-Prozess ist eine Standardisierung möglich. Genau wie bei der ETL-Entwicklung erhöht Standardisierung hier die Zuverlässigkeit, indem sie die Fehler durch manuelles Ausführen reduziert.

Automatisierung

Möglichkeiten zur Automatisierung im DWH-Projekt gibt es prinzipiell mehrere:

- Automatisierung von Tests
- Automatisierung des Deployments
- Automatische Generierung von ETL-Prozessen

Im Vortrag gehen wir genauer auf die beiden letzteren Fälle ein und zeigen auf, wie man diese Arten von Automatisierung mit dem ODI unterstützen kann. Der ODI ermöglicht durch die mitgelieferte Java-API quasi eine „Fernsteuerung“ des ODI. Jede Funktionalität, die auch per Maus über die Oberfläche zugänglich ist, kann auch mittels des Java SDK ausgelöst werden. Alle wesentlichen Objekte des ODI aus dem Repository können damit durch eine Java-Applikation angesprochen und manipuliert werden. Des weiteren lassen sich durch die Skriptsprache Groovy direkt im ODI-Studio Skripte für schnelle Anpassungen der ODI-Objekte erstellen. Zum Beispiel lassen sich so die Attributeigenschaften für die SCD2-Implementierung steuern.

Die Manipulation von ODI-Objekten ist dabei recht einfach erlernbar. Vorausgesetzt werden Kenntnisse der Programmiersprache Java und eine gewisse Einarbeitung in das SDK und die Java API. Als kleines Beispiel zeigt hier Abbildung 3 den Code zu Generierung und Export eines Szenarios:

```
//Encoding Options festlegen
EncodingOptions encode= new EncodingOptions();
// Ausgabe-PFad
String path = "c:\\temp\\ODI\\";
// Package anhand des Namens suchen und das aktuellste Szeanrio dazu
Collection<OdiPackage> cpkg = ((IOdiPackageFinder) tem.getFinder(OdiPackage.class)).findByName(pkgName, project);
OdiPackage pckg = (OdiPackage) cpkg.toArray()[0];
OdiScenario scen = ((IOdiScenarioFinder) tem.getFinder(OdiScenario.class)).findLatestBySourcePackage(pckg.getPackageId(), false);
IOdiScenarioGenerator gene = new OdiScenarioGeneratorImpl(getODIInstance());
// neue Versionsnummer bestimmen
Integer latestVersion = new Integer(scen.getVersion());
latestVersion = latestVersion+1;
// neues Szenario generieren
OdiScenario newScen = gene.generateScenario(pckg, pckg.getName(),String.format("%03d",latestVersion));
System.out.println("SCNENARIO "+newScen.getName()+"_"+ newScen.getVersion() +" generated...");
// Szenario exportieren
ExportServiceImpl expService = new ExportServiceImpl(getODIInstance());
expService.exportToXml(newScen, path, true, true, encode);
```

Abbildung 3: Suchen eines Packages, generieren des Szenarios und Export.

Zur Generierung von Mappings greifen wir auf ein Metadatenmodell zurück. Dadurch sind wir in der Lage ganze DV-Modelle automatisiert zu erzeugen. Im Metadatenmodell werden einerseits die Objekte der Quellsysteme (Tabellen und Spalten) und andererseits die Entitäten des Data Vault Modells und deren Beziehung zu einander erfasst. Das Data Vault Modell wird somit „offline“ modelliert und logisch im Metadatenmodell gehalten. Im nächsten Schritt werden die Hubs und Satelliten den Spalten des Quellsystems zugeordnet. Dadurch sind alle Informationen vorhanden, die für den logischen Aufbau des Data Vault Mappings erforderlich sind. Abbildung 4 zeigt ein einfaches Hub-Mapping, das automatisch erzeugt werden kann.

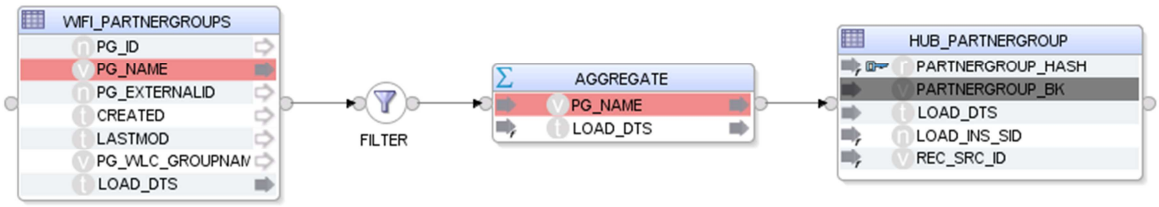

Abbildung 4: Beispiel eines einfachen Hub-Mappings.

Der Ablauf zur Generierung eines einfachen Hub-Mappings stellt sich in etwa folgendermaßen dar:

1. Füge die Quelltabelle ein, die für den gewünschten Hub erforderlich ist.
2. Füge die Filter-Komponente hinzu, um NULL-Werte auszuschließen
3. Füge die Aggregate- Komponente hinzu, um eine eindeutige Liste der Business Keys zu erhalten
4. Füge die Ziel-Hub-Tabelle ein und verbinde den Business Key und setze weitere Data Vault Metainformationen

Das folgende Listing zeigt in Abbildung 5 einen Teil des zugehörigen Java-Codes unter Verwendung der SDK-Objekte des ODI (z.B. OdiDataStore, OdiParameters).

```
public void createSingleMapping() throws Exception {
    OdiTransaction transaction = new OdiTransaction();

    // Generate mapping name
    String mappingName = DataVaultConfiguration.getNamePattern("HUB_MAP")
            .replace("{hub_name}", this.hubTable.getTableName());

    // Start creation of mapping
    Parameters.LOGGER
            .info("Creating hub mapping for single source column with hub sid <"
                    + this.hubTable.getHubSid() + ">.");
    Mapping hubMapping = new Mapping(mappingName,
            super.getOdiProjectFolder());
    OdiEntityManager.getInstance().persist(hubMapping);

    // Add source table
    StagingTable sourceTable = this.sourceTables.get(0);
    Parameters.LOGGER.info("Adding source table <"
            + sourceTable.getTableName() + ">.");
    OdiDataStore sourceDataStore = OdiEntityManager.getInstance()
            .getOdiDataStore(sourceTable.getTableName(),
                    sourceTable.getSchemaName());
    DatastoreComponent sourceComponent = new DatastoreComponent(hubMapping,
            sourceDataStore);

    // Add filter
    Parameters.LOGGER.info("Adding filter <"
            + OdiParameters.ODI_MAP_FILTER_COMP_NAME + ">.");
    FilterComponent filterComponent = new FilterComponent(hubMapping,
            OdiParameters.ODI_MAP_FILTER_COMP_NAME,
            sourceTable.getTableName() + "."
                    + sourceTable.getBusinessKeys().get(0) + " IS NOT NULL");
    sourceComponent.connectTo(filterComponent);
```

Abbildung 5: Codebeispiel zur Generierung eines Hub-Mappings.

Im physischen ODI-Modell legt der Generator das gewünschte Knowledge Modul und dessen Optionen fest. Dadurch ist sichergestellt, dass alle Mappings den gleichen Aufbau haben.

Die Optimierung der Ladeprozesse im weiteren Projektverlauf konzentriert sich meistens auf das Anpassen der Knowledge Module oder das Ändern von Optionen. Der logische Aufbau eines ODI-Mappings bleibt meistens stabil. Im KM werden die Änderungen vorgenommen. Nach der Neugenerierung der betroffenen Szenarien sind die Änderungen verfügbar.

Hub-Mappings sind der einfachste Fall, aber auch Satelliten- und Link-Mappings, und eingeschränkt auch Dimensions- und Fakten-Mappings sind weitgehend automatisiert erstellbar. Der Schlüssel ist hier wieder die Standardisierung. Aber auch wenn nicht 100% eines Mappings automatisiert erstellt werden können, so helfen auch 80% und führen zu einer schnelleren ETL-Entwicklung.

Im Laufe der Zeit sammeln sich Skripte oder Codeschnipsel für die verschiedensten Dinge an, die man in DWH-Projekten mit dem ODI und seiner Java-API automatisieren kann:

- Das Aktualisieren von Tabellen/Schemata (Re-Engineering) im ODI
- Die Umstellung von Beladungsregeln/Knowledge Modulen
- Konfigurationsänderungen, z.B. Einstellung SCD2

Auch hier wollen wir ein paar Beispiele zeigen.

Fazit – Was bringt es?

Man sieht: die Dinge gehen Hand in Hand. Automatisierung entfaltet seinen vollen Nutzen nur mit Hilfe von Standardisierung, Standardisierung hilft ungemein bei der Automatisierung. Nicht nur mit Microsoft SSIS oder einem PL/SQL-ETL-Framework kann man die Erstellung von ETL-Strecken automatisieren und die Entwicklung erheblich beschleunigen, sondern auch mit dem ODI.

Einige Vorteile der Automatisierung mit dem ODI sind:

- Zuverlässigere Entwicklung, weniger Fehler.
- Das Deployment ist nicht mehr so fehleranfällig
- Automatische Tests sichern die Entwicklung
- Kostenreduktion durch schnellere Entwicklung
- Zufriedenere Entwicklung, da keine stupide Handarbeit mehr gemacht werden muss, durch Automatisierung von „langweiligen“ Arbeiten
- Schlussendlich mehr Zeit für die schwierige fachliche Businesslogik

Das ganze kommt natürlich zu einem Preis: Man muss sich bewusst sein, dass die Entwickler neben SQL und den ganzen Tool-Skills nun auch noch Java programmieren können müssen. Und auch die Generatoren muss man erst einmal erstellen.

Kontaktadresse:

Dr. Jens Bleiholder
OPITZ CONSULTING Deutschland GmbH
Tempelhofer Weg 64
D-12347 Berlin

Telefon: +49 (0) 173 7279426
E-Mail jens.bleiholder@opitz-consulting.com
Internet: www.opitz-consulting.de

Marian Strüby
OPITZ CONSULTING Deutschland GmbH
Tempelhofer Weg 64
D-12347 Berlin

Telefon: +49 (0) 173 7279144
E-Mail marian.strueby@opitz-consulting.com
Internet: www.opitz-consulting.de

Ulf Jeffke
Vodafone Kabel Deutschland GmbH
Germaniastraße 14-17
D-12099 Berlin

Telefon: +49 (0) 172 1591098
E-Mail ulf.jeffke@vodafone.com
Internet: www.vodafone-deutschland.de