Besser werden durch Austausch eines ETL-Tools mit PL/SQL

Christian Eberhardt
b.telligent GmbH & Co. KG
München

Schlüsselworte
Ablösung Tool, PL/SQL, Talend

Einleitung

BI ist und bleibt ein Boom-Thema. Trotzdem sollen auch im BI-Umfeld Kosten reduziert werden. In einer nicht zu heterogenen Unternehmens-IT-Landschaft ist es ein Ansatzpunkt, ein vorhandenes ETL-Tool durch ein datenbankinternes Beladungs-Framework zu ersetzen (ELT). So wird die Komplexität der Systemlandschaft reduziert, Lizenzkosten gespart, das Datenbank-Knowhow gesteigert und das Testing vereinfacht. Dieser Vortrag zeigt, wie über alle Schichten eines DWH hinweg mit Hilfe von PL/SQL ein ETL-Tool ersetzt werden kann.
Das Beispiel basiert auf der Ablösung von Talend 4 durch PL/SQL auf Oracle 11.2.

Die Situation

Im hier betrachteten Fall liegt eine im Vergleich sehr einfache Konfiguration der Produktionsumgebung vor. Ein einzelnes OLTP-System dient als alleinige Datenquelle für ein geringvolumiges DWH. Das OLTP-System bietet Echtzeitzugriff mit etwa 50 eigenen Schnittstellentabellen. Beim Abzug der Daten ins DWH muss also keine separate OLTP-Standby-Maschine aktualisiert werden, es muss ebenso wenig ein zeitliches Extraktionsfenster beachtet werden. Ganz im Gegenteil, das produktive OLTP-System darf in Echtzeit belastet werden.

Das DWH wiederum wurde als einfaches historisiertes Datenlager konzipiert. Es gibt genau zwei Schichten: Stage und Star. Stage beinhaltet den aktuellen Datenabzug des OLTP-Systems und zusätzlich eine dauerhafte Historie aller jemals geladenen Daten (das sog. Archiv). In der Star-Schicht sind Bewegungsdaten als Fakten abgelegt, Stammdaten hingegen meistens als SCD2-Dimension modelliert. Die Bewirtschaftungslogik vom OLTP-System über Stage nach Star ist hierbei nahezu geschäftslogikfrei. Die Anzahl der aus dem OLTP-System ins DWH zu ladenden Datensätze pro Tag beträgt etwa 1 Million.

Es zwickt an vielen Ecken

So einfach sich die Situation darstellt, so groß waren die Probleme im DWH. Die Entwicklungskosten waren hoch und die Ursachen dafür vielfältig. Der extrem hohe manuelle und damit zeitliche Aufwand bei der Entwicklung mit Talend war dabei das Hauptproblem.
Dies wird am Beispiel eines Jobs deutlich. Der Prozess zur Erstellung eines einfachen Mappings (d.h. eines ETL-Jobs) ohne komplexe Geschäftslogik besteht im vorliegenden Fall aus mehreren Schritten, wobei z.B. für die gesamte Übernahme einer neuen Tabelle ½ bis 1 Tag gerechnet werden muss.

Schritt	Aktion	Zeit für die Erledigung [Min]
1	Übernahme des Datenmodells aus dem OLTP-System	30
2	Manuelle Anlage von Tabellenmetadaten in Talend	30-120
3	Kopieren eines Templates	5
4	Implementierung des Jobs	Unterschiedlich
5	Manuelle Fehlerlogik (teilweise manuell in	30-60

	Java)	
6	Eintragen des neuen Jobs in diverse Konfigurationsdateien des Buildsystems	30
7	Erstellung und Konfiguration von Schedulerjobs	30
8	Test	90-180 Minuten

Bei Schritt 2 fällt auf, dass die Metadaten nicht automatisch ins Repository importiert werden, sondern manuell angelegt werden. Dies ist aufgrund eines internen Fehlers in der Verarbeitung des Datentyps TIMESTAMP WITH TIME ZONE notwendig. Andernfalls würde bei jedem neuen Metadatenabgleich (z.B. Aktualisierung von Spaltenkommentaren) das Repository wieder zum Falschen hin verändert werden.
Weitere technische Probleme treten z.B. bei Jobs auf, welche durch die Verwendung der Komponente tParallel unzuverlässig werden. Ähnlich verhält es sich mit den Oracle-ELT-Komponenten, die zwar vorhanden sind, aber nicht verlässlich funktionieren.

Neben technischen Unzulänglichkeiten gibt es auch benutzungsbedingte Schwächen. So ist es z.B. schwierig, robuste und wiederverwendbare Templates zu gestalten. Die grobe Steuerung von Datenbankverbindungen, Logging und gewisse allgemeine Fehlerfälle können in ein solches Template aufgenommen werden. Auch der hohe Testaufwand (Schritt 8) ist ein benutzungsbedingtes Problem. Die manuelle Ausführung eines Talend-Jobs mit Unittest-Daten (wenige fachliche Datensätze) dauert bis zu einer halben Minute. Soll darüber hinaus ein Unittest-Werkzeug verwendet werden, muss die Talend CommandLine herangezogen werden: dies ist ein Tool zur Steuerung des Talend-Repositorys per Kommandozeile. Auch dieser Weg bringt keine zeitliche Verbesserung. Der Grund für die schlechten Unittest-Laufzeiten ist die langwierige Vorbereitungs- und Initialisierungszeit eines Jobs. Ein einfaches 1:1-Mappings besteht aus immerhin fast einhunderttausenden Zeilen Java-Code.
Des Weiteren ist die Wahl des Paradigmas nicht einfach – also ETL versus ELT. Auf der einen Seite wird ein Tool ja gerade deswegen angeschafft, um in komplexen Umgebungen schneller und nachhaltiger zu produktiven Ergebnissen zu gelangen. Auf der anderen Seite wird der Vorteil der Datenquellenabstraktion oft mit mittelmäßiger Performance erkauft. Die Erfahrung zeigt, dass kritische Prozesse oft so datenbanknah wie möglich implementiert werden müssen. Für Talend bedeutet das, dass performancekritische Abschnitte direkt per SQL umgesetzt werden.

Die Infrastruktur muss ebenfalls kritisch betrachtet werden. So ist der TAC-Server (Talend Administration Center) nur für die Entwicklungsumgebungen installiert. Dieser Service bietet eine graphische Oberfläche für weitere interne Dienste an, z.B. einen Job-Ausführungsdienst, Kontrolle über SVN (Versionsverwaltung), Benutzerverwaltung etc.
Ein solcher Dienst existiert in anderen Umgebungen wie z.B. Systemintegration oder Produktion nicht. Aus diesem Grund muss jeder Job stets mit allen Bibliotheken exportiert werden, so dass er selbstständig lauffähig ist. Für die Ausführung wird dann aber auch nur eine einfache Java Laufzeitumgebung benötigt. Dies führt jedoch wiederum dazu, dass ein einzelner Job schnell zu vielen Megabytes aufgebläht wird. Das komplette ETL-Deployment-Artefakt mit allen Jobs, welches im Übrigen zu versionieren ist, umfasst in seiner Gesamtheit schnell mehr als ein Gigabyte.

Der Multiplikator / Die Initialzündung
Die im vorherigen Abschnitt aufgeführten Probleme gingen in einem Maßnahmenkatalog auf, dem folgende Wünsche zugrunde lagen.

- Alle Prozesse sollten mit möglichst geringem manuellem Aufwand erstellt werden.
- Gleichartige Prozesse sollten, so die Vision, einmal verstanden und mittels dynamischen SQL generiert werden können.
- Mit dem Wissen und den Tools auch angrenzender Teams sollten durch die Nutzung erprobter Lösungen Synergien genutzt und damit Aufwand und Kosten reduziert werden.
- Datenbankseitig sollten die nicht ausgeschöpften Ressourcen genutzt werden, also parallele Ausführung, Komprimierung und Partitionierung, korrekte Verwendung von Daten des Typs TIMESTAMP WITH TIME ZONE, geringere Artefaktgröße etc.

Die Initialzündung gelang schließlich mit der schrittweisen Ablösung des ETL-Tools Talend durch ein PL/SQL-Rahmenwerk zusammen mit einem Code-Generator für SCD2-Prozesse.

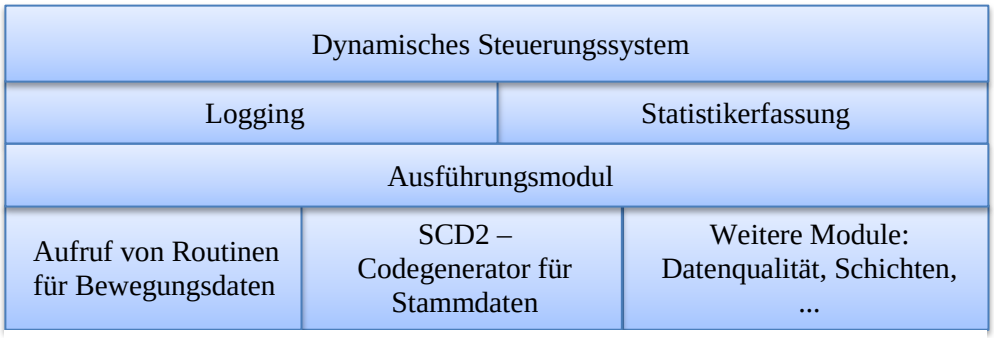

Abbildung 1: PL/SQL-Rahmenwerk

Das dynamische Beladungswerkzeug (Abbildung 1) beinhaltet die interne Steuerung und kapselt die Abläufe und Aufrufe nach außen, in diesem Fall zum Scheduling-Werkzeug. In einer Tabelle wird die Konfiguration jedes Prozesses festgehalten, also z.B. Prozessname, Spalten des fachlichen Schlüssels, Art der Beladung (SCD1, SCD2, einfaches Insert für Bewegungsdaten), Reihenfolge in Abhängigkeit zu anderen Prozessen, Prozentsatz der zu analysierenden Tabellenstatistiken etc. Mit diesen Informationen wird für jeden Prozess die richtige Aktion ausgeführt. Außerdem werden am Anfang jedes Prozesses die Statistiken der Hauptquelltabelle gesammelt. Nach jedem Lauf werden die Statistiken der Zieltabelle aktualisiert. Auch ein einfaches zentrales Logging ist Bestandteil des Rahmenwerks. Darüber hinaus wurde auf Erweiterbarkeit geachtet, so dass auch andere Schichten des DWH oder z.B. Routinen zu Prüfung der Datenqualität aufgerufen und gesteuert werden können.

Da etwa 80% aller Tabellen zu historisierende Stammdaten enthalten, hat der SCD2-Codegenerator die größte Einsparung ermöglicht. Die Liste der Spalten für den fachlichen Schlüssel befindet sich – wie oben bereits angemerkt - in einer Konfigurationstabelle. Alle weiteren Informationen werden im Datenbankkatalog gefunden. Der folgende Code zeigt eine Funktionsskizze:

```
MERGE INTO target tgt USING (
   WITH tmp_scd_types AS (
      SELECT 1 AS scd_type FROM dual
      UNION ALL
      SELECT 2 AS scd_type FROM dual
   )
   SELECT src.* --
     FROM (
        SELECT
              CASE
```

```
                --ganz neue ID: Modus INSERT
                 WHEN tgt.id IS NULL THEN 1
                 --Änderung: Modul Update + Insert
                 WHEN COALESCE(tgt.col1,-99)!=COALESCE(src.col1) OR … THEN 2
               END scd_type
             , <Liste von fachlichen Spalten>
             , <Liste von technischen Spalten>
         FROM stage stg
              LEFT OUTER JOIN target tgt – nur die letzte Version von tgt
              ON <Business Key>
             --WHERE Insert oder Update notwendig
      ) src
         INNER JOIN tmp_scd_types
         ON tmp_scd_types.scd_type <= src.scd_type
)
WHEN MATCHED THEN UPDATE SET
   --technische Felder des neuesten hist. Eintrags öffnen
   tgt.valid_to = src.valid_from,
   <weitere Spalten>
WHEN NOT MATCHED THEN INSERT (
   --neue Daten einfügen (neue Keys und neue Historieneinträge)
   tgt.col1
   <weitere Spalten>
) VALUES (
   src.col1
   <weitere Spalten>
)
```

Die Idee besteht darin, die zu ladenden Daten – es wird immer ein Delta geliefert – mit den bereits geladenen Daten über den fachlichen Schlüssel hinweg zu verbinden und basierend auf allen anderen Spalten Änderungen zu erkennen. Die zusätzliche Änderungslogik ist notwendig, da vereinzelt auch dann Datenzeilen geliefert werden könnten, selbst wenn keine Änderung gemacht worden sind. Ganz neue Einträge (neuer Business Key) bekommen eine neue Historie in der Zieltabelle (eine Quellzeile bewirkt eine Zeile in der Using-Klausel).
Für bestehende Business Keys werden die neu gelieferten Einträge ans Ende der existierenden SCD2-Historie eingepflegt. Aus Gründen der Einfachheit ist es nicht vorgesehen, dass eine bestehende Historie rückwirkend korrigiert wird. Eine Quellzeile erzeugt aufgrund des Verbundes mit dem Operator „<=“ im Fall einer Änderung zwei Zeilen in der Using-Klausel. Eine Zeile bewirkt die Öffnung des neuesten vorhandenen historischen Eintrags, indem ein Update ausgeführt wird. Die zweite Zeile bewirkt das Eintragen des neuen Datensatzes per Insert.
Bei komplexeren Szenarien zur Datenlieferung kann das Merge beliebig umfangreich werden. Im hier besprochenen Fall wird die erzeugte Anfrage bei einer zehnspaltigen Tabelle etwa 120 Zeilen lang und verwendet Windowing-Klauseln sowie Subselects. Nichtsdestotrotz besteht der Codegenerator aus nicht mehr als 1.500 Zeilen PL/SQL-Code.

Die Prozeduren zur Beladung der Bewegungsdaten liegen in einem separaten Package. Sie werden manuell erstellt und müssen einer Namenskonvention genügen, damit zur Laufzeit abhängig von der Zieltabelle auch die richtige Beladungsprozedur aufgerufen wird. Neben der Schnittstelle zu den Prozeduren für die Beladung von Bewegungsdaten können weitere Schnittstellen auf einfache Art und Weise konfiguriert und mit minimalem Aufwand in das Rahmenwerk eingebracht werden (beispielsweise Routinen zur Qualitätssicherung oder Beladung weiterer Schichten).

Weitere Verbesserungen
Da die Ausführung des Merge-Befehls für große Datenmengen nicht besonders gut skaliert, kann als Verbesserung das bewährte Prinzip von Teile-und-Herrsche umgesetzt werden. So ist es möglich, für jeden Prozess eine gewisse Anzahl von Datenteilen zu definieren (z.B. 20 Teilschritte anstatt eines einzelnen Laufs) und diese nacheinander oder teilparallel auszuführen.
Es wurde die Annahme getroffen, dass schon der erste Teil eines zusammengesetzten fachlichen Schlüssels eine ausreichende Partitionierung zulässt. Es wird nun – ganz klassisch - der Modulo-Operator angewendet um die Gesamtheit der Datensätze in N ähnlich große Teildatensätze zu zerlegen. Ist dieser Modus aktiv, werden anstatt eines einzelnen Merge-Befehls über alle Daten also hintereinander N Merge-Befehle auf jeweils etwa ein zwanzigstel der Daten (wegen 20 Teilschritten) ausgeführt. Dies dauert in Summe zwar etwas länger als eine einzelne Ausführung, garantiert aber die unbedingte Ausführung auch mit sehr vielen Daten. Falls dies nicht der Fall sein sollte, kann der vorher gewählte Wert N vergrößert werden.
Um die Performance und Systemauslastung noch weiter zu erhöhen kann ein Pool von X parallel laufenden Teilprozessen implementiert werden. Nach dem Warteschlangenprinzip können also N Teilprozesse nach und nach auf die X aktiven parallelen Prozesse verteilt werden.

Die Wirkung
Das Framework wurde durch einen Entwickler in ca. 4 Wochen umgesetzt. Die Früchte der Arbeit konnten schnell geerntet werden, denn die Produktivität wurde erheblich gesteigert.
Die folgende Tabelle stellt die Entwicklung mit Talend und die Entwicklung basierend auf PL/SQL gegenüber:

Kriterium	Talend	PL/SQL
Infrastruktur (Sind z.B. zusätzliche Dienste oder Netzwerkanpassungen notwendig?)	Komplex	Trivial
Performance	Mäßig	Sehr gut
Größe des Deployment-Artefaktes	Klein – Sehr groß	Sehr klein
Paradigma (ETL/ELT)	ETL (ELT)	ELT
Debuggingfähigkeiten (Kann schrittweise debuggt werden? Gibt es eine graphische Oberfläche?)	Gut	Gut
Wartung (Wie leicht lassen sich neue Features in alte Versionsstände einbringen? Wie einfach lassen sich Fehler beheben?)	Mäßig	Gut
Betrieb (Kann der Betrieb leicht Fehler finden?)	Schlecht	Gut

Neben den Entwicklern ist auch der Betrieb glücklich, denn die Prozesse sind nun wesentlich robuster. Dabei wurde die Verarbeitungszeit für die Daten eines Tages im ersten Schritt von 11 auf 2 Stunden gesenkt, wobei in der ersten Produktivstellung nur die Top-5 Prozesse abgelöst wurden. Außerdem ging der Ressourcenverbrauch des ETL-Servers stark zurück. Die Hauptlast wird inzwischen von der Datenbank getragen.
Auch das Test-Rahmenwerk utPLSQL kann nun genutzt werden. Denn nur so konnte der zentrale SCD2-Codegenerator hinreichend getestet werden. Mehrfach täglich werden alle Unittest-Fälle ausgeführt (Regressionstest) um eventuelle negative Auswirkungen zu erkennen. Unit-Tests sind durch die Zeiteinsparungen kein notwendiges Übel mehr, sondern eine selbstverständliche Versicherung jedes Entwicklers.

Was bleibt
Es hat sich gezeigt, dass die Ersetzung des ETL-Tools durch datenbanknahen Code eine sehr gute Option für homogene Umgebungen ist. Der einfache und wartbare Code unterstützt die Entwicklung, die Wartung und den Betrieb. Die Performance ist auch bei hohem Datenvolumen ausgezeichnet. Datenintegrationswerkzeuge wie Talend können ihre Vorteile ausspielen, wenn Prototypen gefragt sind, nicht sehr viele Daten bewegt werden müssen oder eine komplexe Systemumgebung vorherrscht.

Kontaktadresse:
Christian Eberhardt
b.telligent GmbH & Co. KG
Georg-Brauchle-Ring 54
D-80992 München

Telefon: +49 (89) 122 281 110
Fax: +49 (89) 122 281 130
E-Mail christian.eberhardt@btelligent.net
Internet: www.btelligent.net

b.telligent ist eine Unternehmensberatung, die auf die Einführung und Weiterentwicklung von Business Intelligence, Customer Relationship Management und E-Commerce bei Unternehmen in Massenmärkten spezialisiert ist.

Der Fokus liegt dabei auf der kontinuierlichen Optimierung von Geschäftsprozessen, Kunden- und Lieferantenbeziehungen durch den Erkenntnisgewinn aus der Verdichtung und Analyse von systemübergreifenden Geschäftsdaten. So lassen sich Margen erhöhen, Kosten senken und Risiken besser kontrollieren.

Kunden von b.telligent sind Branchenführer aus den Bereichen Telekommunikation, Finanzdienstleistung, Handel und Industrie.