Data Quality Layer mit PL/SQL wirkungsvoll einsetzen

Dr. Markus Vincon
areto consulting GmbH
Köln

Schlüsselworte
Data Quality, PL/SQL, Metadaten, Prozess-Überwachung

Einleitung

In diesem Vortrag soll anhand eines konkreten Beispieles aufgezeigt werden, wie man verschiedene für ein DWH wichtige Aspekte mit PL/SQL umsetzen und integrieren kann. Dies fängt mit einer einfachen Prozess-Überwachung an und geht über eine Metadaten-Verwaltung bis hin zur Integration von verschiedenen Data-Quality-Aspekten.

Historie und technischer Aufbau

Angefangen hatte alles in einem konkreten DWH mit einer Prozess-Überwachung, d.h. im Grunde vier Methoden für job_start, job_ende, workflow_start und workflow_ende. Im Folgenden möchte ich bei der Benamung mit Jobs und Workflows bleiben, ein Workflow ist dabei ein einzelner ETL-Prozess, der Daten von ein oder mehreren Quellen zu ein oder mehreren Zielen transportiert und transformiert. Ein Job kann aus vielen Workflows bestehen und nur ein Job kann auch explizit ausgeführt werden.

Alleine diese vier einfachen Methoden eröffneten uns die Möglichkeit, dass wir die ablaufenden Prozesse im DWH überwachen konnten. Wir haben das Ganze noch um das Abspeichern von Fehlermeldungen sowie eine Zählung von neu erzeugten Zeilen (sowie Größe der einzelnen Tabellen erweitert). Zur Identifikation wurden den laufenden Jobs und Workflow aufsteigend vergebene IDs zugeordnet, die auch in allen Tabellen (STAGE, CORE, MART) gespeichert wurden.

Die Organisation des DWH war im Kern eine relational historisierte CORE-Schicht (d.h. historisiert, aber es war im Wesentlichen noch das relationale Schema der Quellsysteme sichtbar). An der Stelle hat uns unser Metadaten-Framework schon in der Vergangenheit sehr geholfen, bei der Analyse von Ungereimtheiten oder Fehlern. Es war uns sogar zweimal möglich (aufgrund von Fehlern), dass wir einen Teil des CORE (einmal 20 Tagesbeladungen und einmal 45 Tagesbeladungen) zurücksetzen/löschen, dann etwas korrigieren und die Verarbeitung von STAGE nach CORE wiederholen konnten.

Das erste Mal, dass dieses Zurücksetzen nötig wurde, war relativ banal. Nach einem willkürlichen Prozess-Abbruch (die 32-Bit-Version des ETL-Tools hatte manchmal Probleme mit der Speicher-Allokation) sollte nur ein „halber“ CORE-Lauf ausgeführt werden. Dieser lief erfolgreich, allerdings wurde danach vergessen, auf Produktion wieder den „vollen“ CORE-Lauf zu deployen.

Es brauchte 20 Tage, bis die DWH-Anwender in ihren Berichten Unstimmigkeiten sahen. Dies brachte uns zu der Idee, dass wenn wir z.B. schon die in einem Lauf neu erzeugten Zeilen zählen, auch eine Art Aktivitätsüberwachung integrieren könnten. Es ist natürlich sehr diffizil, zu wissen, was pro Datenbank-Tabelle eine gute obere oder untere Schranke ist, um je nach der Anzahl der erzeugten

Zeilen eine Mail zu verschicken, aber auch das hat sich in der Praxis bewährt (solange man ggfs. drauf achtet, dass nicht eine Mail pro Tabelle sondern eine Zusammenfassung verschickt wird).

Das mit der Zusammenfassung (auch für Fehlermeldungen) war aber kein Problem, da wir entsprechende Prüfungen immer erst am Ende eines Jobs vorgenommen haben.
Nachdem wir bis dahin wirklich gute Erfahrungen mit unserem Metadaten-Framework gemacht haben, wollten wir das Ganze noch um weitere Aspekte der Data Quality ergänzen.

Das betrifft zum einen die Möglichkeit, ein recht allgemeines SQL selbst zu definieren, welches dann ebenfalls nach einem Job-Lauf ausgeführt werden kann und dessen Ergebnisse in die Aktivitätenüberwachung (plus Mail-Versand) integriert wurde. Zum anderen wollten wir die Möglichkeit schaffen, für ein paar wiederkehrende und standarisierbare DQ-Überprüfungen, wie sie in der Praxis häufiger auftreten (Suche nach Ausreissern, Fremd-Schlüssel-Verletzungen). Auch deren Ergebnisse konnten in die Aktivitätenüberwachung integriert werden.

Dabei war es uns auch wichtig soweit es möglich war, dass wir die Ergebnisse der Prüfungen den ETL-Prozessen selbst wieder zugänglich machten (in Form einer zusätzlichen Tabelle, die die Schlüssel der Tabellen mit einem Ergebnis-Code der Überprüfungen möglich machte).

Grundlegende Techniken und Tabellen
Wichtig für das Funktionieren des Metadaten-Frameworks waren zwei Dinge:

- Ein ETL-Prozess (Job und Workflow) kann sich gegenüber dem Framework identifizieren (z.B. durch eine Methode job_name() oder workflow_name() des ETL-Tools)
- Ein ETL-Prozess (Job und Workflow) muss die Möglichkeit haben, dass er die vom Metadaten-Framework für ihn erzeugte ID zwischenspeichern kann

Grundlegend sind dann die beiden folgenden Tabellen, die letztendlich die entsprechenden Lauf-Ids definieren. Die in beiden Tabellen enthaltenen Felder für Start und Ende des Extrahierens werden nur bei STAGE-Tabellen genutzt. So bietet das Framework die Möglichkeit, dass bei Delta-Beladungen in der STAGE insbesondere jeder Workflow immer weiss, wann er beginnen muss.

```
CREATE TABLE T_MD_JOB_LAUF (
job_lauf_id number NOT NULL,
status_fk number NOT NULL,
job_fk varchar(50) NOT NULL,
start_tstamp date,
ende_tstamp date,
extrakt_start_datum date,
extrakt_ende_datum date);

CREATE TABLE T_MD_JOB_WF_LAUF(
job_wf_lauf_id number NOT NULL,
workflow_fk varchar(100) NOT NULL,
job_fk varchar(50) NOT NULL,
status_fk number NOT NULL,
job_lauf_fk number NOT NULL,
start_tstamp date,
ende_tstamp date,
extrakt_start_datum date,
extrakt_ende_datum date);
```

Die vorliegenden Tabellen können auch dazu benutzt werden, um sicherzustellen, dass nie mehr als ein Job gleichzeitig im DWH läuft (oder zumindest nicht zweimal der gleiche Job). Daneben gibt es eine Tabelle, die sogar sicherstellt, dass bestimmte Workflows nur in bestimmten Jobs laufen dürfen (bei der Entwicklung manches Mal nervig, da man die Workflows auch erst seinem eigenen Entwicklungs- bzw. Test-Job zuordnen muss, aber wir gehen da lieber auf Nummer sicher).

```
CREATE TABLE T_MD_JOB_WF(
workflow_fk varchar(100) NOT NULL,
job_fk varchar(50) NOT NULL);
```

Daneben gibt es eine zentrale Tabelle für die Steuerung oder besser Abstimmung der Workflows zwischen STAGE und CORE. So kann für jeden Workflow, der Daten von der STAGE zum CORE transportiert werden, bestimmt werden, welche Portion an Daten transportiert werden muss. Ohne äusseren Eingriff ist das immer der letzte Lauf, der zu dem Workflow der STAGE gehört, es gibt aber auch die Möglichkeit einen bestimmten Job-Lauf fest zu definieren, so dass nur der Workflow-Lauf aus diesem Job-Lauf berücksichtigt wird. Dieser Mechanismus wird benötigt, wenn man wie am Anfang beschrieben vor dem Problem steht, dass man beispielsweise 20 CORE-Beladungen zurücknehmen muss, um diese dann (beginnend bei dem ältesten) später wieder von der STAGE-Schicht in die CORE-Schicht zu transportieren.

```
CREATE TABLE T_MD_WF_HISTORISIERUNG(
workflow_stage_fk varchar(100) NOT NULL,
workflow_historisierung_fk varchar(100) NOT NULL);
```

Die grundlegenden Tabellen für die Fehlerverwaltung oder auch Job-spezifische Parameter oder den Mail-Versand unterschlage ich hier mal (wir wollten so viel wie möglich konfigurierbar in der Datenbank). Als nächstes die Tabellen für die Aktivitäts-Schranken und die Aktivitätsüberwachung. Am Ende eines Jobs kann bestimmt werden, welche Überwachungen ausgeführt werden sollen. Dabei werden in einem ersten Schritt die Schranken noch nicht berücksichtigt, d.h. es wird auf jeden Fall gezählt. Erst wenn alle Überwachungen und auch DQ-Prüfungen (die ja auch in die Aktivitätsüberwachung einfließen) ausgeführt sind, werden die Zählungen mit den Schranken in einer View so zusammengefasst, dass dort ersichtlich ist, ob wirklich relevante Nachrichten erzeugt worden sind oder werden müssen.

Die folgenden Tabellen sind dabei möglichst flexibel, so kann auf die untere oder obere Schranke verzichtet werden (bitte nicht auf beide gleichzeitig). Über die zweite Tabelle ist steuerbar, ob die Überwachung sich nur auf eine bestimmte Spalte (möglicherweise auch mit einem besonders zu prüfendem Inhalt) oder auf den gesamten Tabelleninhalt (bezogen auf den aktuellen Job) ausgeführt werden soll.

```
CREATE TABLE T_MD_JOB_WF_AKTIVITAET_S(
aktivitaet_schranke_id number NOT NULL,
untere_schranke number,
obere_schranke number);

CREATE TABLE T_MD_JOB_WF_AKTIVITAET_U(
aktivitaet_ueberwachung_id number NOT NULL,
job_fk varchar(100) NOT NULL,
workflow_fk varchar(100) NOT NULL,
table_name varchar(100) NOT NULL,
aktivitaet_art varchar(20) NOT NULL,
spalte_pruefung varchar(100),
inhalt_pruefung varchar(100),
```

```
aktivitaet_schranke_fk number NOT NULL);
```

Der nächste in das Framework zu integrierende Punkt kommt eigentlich als Anforderung eines anderen Kunden daher. Dort wurde die Möglichkeit gewünscht, dass SQLs ausgeführt werden können, die auch einen tabellenübergreifenden Bezug herstellen können (beispielsweise eine Überprüfung, ob es Bestellungen gibt, deren Summe von Einzelpositionen nicht der Gesamtsumme in der Bestellung entsprechen).

```
CREATE TABLE T_MD_JOB_SQL_AUSFUEHREN(
sql_ausfuehren_id number NOT NULL,
job_fk varchar(100) NOT NULL,
id_type varchar(1) NOT NULL,
table_name varchar(100) NOT NULL,
sql_text varchar(2048) NOT NULL);
```

Die Herausforderung an der Stelle war, dass für das Ausführen noch ein paar zusätzliche Anforderungen nötig sind:

- Es muss ein Select sein, dass nur eine Menge von IDs zurückgibt
- Die IDs müssen sich auf die Tabelle table_name beziehen
- Es muss angegeben werden, welchen Datentyp die IDs haben (bei uns Number oder Varchar(40))
- Es muss/sollte ein Platzhalter _CURRENT_ im SQL enthalten sein, der durch das Framework mit der aktuellen Job-Lauf-ID ersetzt wird
- Sobald das SQL keine leere Ergebnis-Menge liefert, wird eine Aktivitätsmeldung erzeugt

Neben dem Erzeugen einer Aktivitätsmeldung bei nicht leerer Ergebnis-Menge des SQL-Statements, wird die aktuelle JOB-Lauf-ID, der Tabellenname und die gefundenen IDs in Tabellen (welche genau ist von dem Datentyp der IDs abhängig) gespeichert, so dass diese später durch einen Menschen oder den ETL-Prozess selbst weiterverarbeitet werden können (z.B. Filtern von solchen Bestellungen in besondere Tabellen).

Eigentlich war dann unser nächster Gedanke, dass wir noch Steuerungs-Tabellen für z.b. Fremdschlüssel-Verletzungen oder der Suche nach Ausreissern integrieren, aber bei genauerem Überlegen zeigte sich, dass die Möglichkeit des Ausführens von beliebigem SQL im Grunde schon alle Möglichkeiten bietet. Für Fremdschlüssel-Verletzungen oder Datentyp bzw. Datenlängen-Verletzung hätte man es gut durch weitere sehr einfache Steuer-Tabelle abbilden können, aber die Ausreisser-Suche oder besser schon alleine die Definition, was ein Ausreisser ist, ist sehr komplex (z.B. mehr als x Standardabweichungen vom Mittelwert entfernt oder möglicherweise ein Wert, der in einer Spalte nur einmal vorkommt, während alle anderen Werte zigtausendfach vorkommen oder wenn die Differenz zwischen kleinstem Wert und zweikleinstem Wert mehr als 10 mal so groß ist wie die Differenz von dem zweitkleinsten Wert zum Mittelwert).

Abschluss und Bewertung

Zum Abschluss muss ich sagen, dass wir heutzutage nicht mehr auf das Metadaten-Framework verzichten wollen, auch wenn wir am Anfang nur mit einer eher abstrakten Idee des Nutzens gestartet sind. Etwas anstrengend war, dass wir auch die Möglichkeit der Fehlerbehandlung erst in einem Schritt 1.b eingefügt haben. Dies liess sich zwar gut in unser Entwicklungs-Template für Workflows integrieren, trotzdem mussten die schon vorhandenen Workflows alle einmal angefasst werden.

Von daher hoffe ich, dass ich Sinn und Nutzen von einem solchen Framework aufzeigen konnte, damit in Zukunft möglichst gleich von Anfang an solche Aspekte bei der Entwicklung mit berücksichtigt werden. Viele der Ideen sind dabei nicht von ausgefeiltem PL/SQL abhängig, wir haben in einer anderen Kombination auch die Erfahrung gemacht, dass man mit PL/SQL zu arbeiten auch wiederverwendbaren Inhalt z.B. über Custom Functions an die Schnittstelle zwischen ETL-Tool und Metadaten-Framework bringen kann.

Was noch nicht erwähnt wurde, ist dass wir auch selbst Berichte auf unseren Framework-Tabellen anbieten (z.B. auch Speicherplatz-Auswertungen). Diese könnten ggfs. den Mail-Versand ersetzen, allerdings wollen wir im DWH aktiv mit möglichen Problemen oder Fehlern umgehen.

Kontaktadresse:
Dr. Markus Vincon
Areto consulting GmbH

Carlswerk, Gebäude „Labor 1.7“
Schanzenstraße 6-20
51063 Köln

Telefon: +49 221-66 95 75 0
Fax: +49 221-66 95 75 99
E-Mail markus.vincon@areto-consulting.de
Internet: www.areto-consulting.de