Die Crux mit dem Delta – vom Fullload zum Incremental Load

Bodo Clausen
OPITZ CONSULTING Gummersbach GmbH

Schlüsselworte:
Oracle Golden Gate, OGG, Oracle CDC, Asynchronous Distributed Hotlog CDC, incremental load, Delta, Real-Time-DWH

Einleitung

In Datawarehouse Projekten müssen aus verschiedenen operativen Systemen Informationen an das Warehouse geliefert werden. Dazu werden die Daten erst in ein Staging-System geschrieben, ohne dabei geändert zu werden.
Es ist nicht selten dass für Tabellen entschieden wird diese komplett in das Staging-System zu schreiben. Die Gründe dafür können vielfältig sein. Zum einen ist dies die einfachste Möglichkeit und somit sehr schnell implementiert. Zum anderen gibt es Anforderungen, die verlangen die Daten alle 24 Stunden über Nacht zu transferieren. Die Zeit reicht aus, entsprechende Performanz Verschlechterungen durch das Laden fallen in diesen Stunden nicht ins Gewicht, da die Systeme in diesen Zeiten oft kaum genutzt werden.
Aus dieser Grundsituation heraus war die Entscheidung, einen Fullload zu verwenden, durchaus sinnvoll. Doch wenn sich eine der zur Entscheidung führenden Vorbedingungen verändert, wird ein inkrementelles Laden notwendig.
Unter inkrementellem Laden oder Delta-Load wird das ausschließliche Laden von Datenänderungen verstanden. Alle Daten, die sich nicht verändert haben werden nicht übertragen.

Eine sich ändernde Vorbedingung könnte sein, dass die nächtlichen Ladevorgänge kaum noch in der vorgegebenen Zeit beendet werden können, da immer mehr Daten verarbeitet werden müssen. Des Weiteren könnten sich die Anforderungen ändern und das Management benötigt jetzt aktuellere Informationen. Dadurch wird der zu Beginn sinnvolle Fullload schnell zum Problem. Denn viele Bedingungen die den Fullload möglich machten, fallen durch kleinere Intervalle weg. Wenn zum Beispiel von einem Ladevorgang alle 24 Stunden nur auf alle 12 Stunden gewechselt wird, ist das Problem schnell zu erkennen. Wenn der Fullload in der Nacht drei Stunden in Anspruch genommen hat und um Mitternacht gestartet wurde, waren ab 3 Uhr morgens die Daten verfügbar. Wenn alle Daten, die sich bis 12 Uhr geändert haben, Teil des zweiten Ladevorgangs sein müssen, würde dieser um 12 Uhr mittags gestartet und bei gleicher Laufzeit um 15 Uhr beendet. Dies liegt in der Kernarbeitszeit. Da in dieser Zeit die größte Last auf den operativen Systemen besteht und durch diesen Ladevorgang weitere Last erzeugt wird, kann dies zu Engpässen und somit Performanzproblemen führen. Das hat zur Folge, dass die Daten des Vormittags erst am Nachmittag zur Verfügung stehen. In diesen Situationen wird nach einer intelligenteren Methode gesucht, um die Daten mit weniger negativem Einfluss auf das operative System zu laden und die Daten schneller zur Verfügung zu stellen. Es wird die Frage gestellt werden, warum alle Daten geladen werden und nicht nur die deutlich kleinere Menge der Daten, die sich verändert haben, beziehungsweise neu sind oder gelöscht wurden.
Eine weitere Herausforderung kann die Laufzeit werden. Wenn die Laufzeit das Ladeintervall übersteigt, ist das Intervall so nicht mehr zu halten. Bei dem genannten Beispiel mit drei Stunden Ladezeit würde bei der Anforderung, alle zwei Stunden die Daten zu laden, der erste Ladevorgang

noch nicht abgeschlossen sein, wenn der zweite bereits gestartet werden muss. Da in weiteren Schritten beim Laden in Datawarehouses verschiedene Transformationen der Daten vorgenommen werden, ist das Ausführen von parallelen Ladevorgängen auf die gleichen Daten alles andere als trivial.
Das sind Beispiele für Situationen in denen begonnen wird von Fullloads auf inkrementelle Ladevorgänge umzustellen. Anhand eines einfachen Beispiels werde ich drei Möglichkeiten aufzeigen, dies durchzuführen.

Die Ausgangslage
Ich habe eine Tabelle erstellt, die MY_SALES heißt und in den folgenden Tests verwendet wird. Diese einzelne Tabelle ist die einfachste Möglichkeit, um die Herausforderungen und entsprechende Lösungen darzustellen. Hier folgt die Struktur der Tabelle inklusive einer Sequenz für das Hochzählen des surrogaten Primärschlüssels, einem Insert-Trigger und einem Update-Trigger. Der Insert-Trigger füllt eine fortlaufende eindeutige Zahl in das Feld ID und schreibt das aktuelle Datum inklusive der Zeit in das Feld Modify_Date. Der Update-Trigger schreibt nur das aktuelle Datum inklusive Zeit in das Modify_Date-Feld. Diese Tabelle stellt eine Tabelle in einem operationalen System dar und muss in ein Staging-System übertragen werden.

Die Tabelle:

```
create table my_sales (id number primary key,
                       product_id number not null,
                       customer_id number not null,
                       qty number(22,0) not null,
                       sales_price_unit number (10,2) not null,
                       sales_date date,
                       modify_date date);
```

Die Sequenz und Trigger:

```
create
create sequence seq_my_sales;

CREATE OR REPLACE TRIGGER TR_I_my_sales
BEFORE INSERT ON my_sales
FOR EACH ROW
BEGIN
  :new.id := seq_my_sales.nextval;
  :new.modify_date := sysdate;
END;

CREATE OR REPLACE TRIGGER TR_U_my_sales
BEFORE UPDATE ON my_sales
FOR EACH ROW
BEGIN
  :new.modify_date := sysdate;
END;
```

Die Daten sind zufällig generiert und enthalten 10000 Zeilen, die Anzahl der verkauften Güter liegt zwischen 1 und 500. Es gibt Produkte mit der ID 1 bis 10. Es gibt Kunden mit der ID 1 bis 25. Das Verkaufsdatum liegt maximal 30 Tage in der Vergangenheit basierend auf der aktuellen Systemzeit bei der Erzeugung der Daten (10.09.2011 ist gerade aktuell).

Ziel ist es, nur die Änderungen im Staging-System zur Verfügung zu stellen, da der weiterverarbeitende Prozess ebenfalls so umgestellt wird, dass nur die veränderten Daten verarbeitet werden. Eine Alternative wäre eine Replikation, die nur eine Repräsentation des Ist-Zustandes der Tabelle im Staging-System darstellt. Dies kann im Einzelfall sinnvoll sein. In der Regel wird ein Prozess, welcher die Änderungen im operationalen Quellsystem feststellen kann, so angepasst, dass die nachfolgenden Prozessschritte auch nur die Änderungen weiterverarbeiten. Dadurch werden die nachfolgenden Transformationen ebenfalls optimiert und die Laufzeit reduziert.

Der einfache SQL basierte Ansatz
Hier wird durch SQL-Abfragen nur die Menge an Daten abgefragt, die basierend auf dem Modify_Date geändert wurden. Dazu ist es notwendig, dass das System weiß, bis zu welchem Modify_Date der letzte Ladevorgang durchgeführt wurde.
Im zweiten Schritt wird dieses Datum verwendet, um alle Daten abzufragen, die danach geändert wurden. Das Einfügen neuer Daten stellt dabei ebenfalls eine Änderung dar.

Der Initiale Ladevorgang könnte so aussehen.
1.) Feststellen was das zur Zeit maximale Modify_date auf der Tabelle ist
```
SELECT MAX(MODIFY_DATE) FROM MY_SALES;
```
2.) Den Wert in ein entsprechendes Select Statement einbauen
```
SELECT * FROM MY_SALES@ORCL WHERE MODIFY_DATE <= 'das gefundene
Datum';
```

Hier habe ich als Beispiel einen Datenbanklink und ein SQL verwendet. Dies ist natürlich auch durch ETL Tools, wie Informatica PowerCenter, durchführbar. Es geht hier nur darum die Daten entsprechend abzufragen.
Wenn das initiale Laden abgeschlossen ist, kann der inkrementelle Ladevorgang beginnen. Dabei muss das zuletzt geladene Modify_Date verwendet werden, dies muss also entsprechend gespeichert werden.
Dann kann mit folgendem Statement das Delta identifiziert und geladen werden.

Den Wert in ein entsprechendes Select Statement einbauen
```
SELECT * FROM MY_SALES@ORCL WHERE MODIFY_DATE > 'letztes Loaddate' and
MODIFY_DATE <= 'aktuelles Loaddate z.B. sysdate'
```

Dieses Vorgehen hat zwei besondere Schwachstellen. Erstens werden so zwar Insert und Update Aktionen durch das Modify_date in das Staging-System geladen, die Deletes gehen dabei allerdings verloren. In einem Projekt habe ich schon gesehen, dass das Abarbeiten der Deletes durch eine trigger-basierte Historisierung umgesetzt wurde. Diese war bereits vorher in dem operationalen System vorhanden und es mussten nur noch aus der Historisierungstabelle die Datensätze mit dem Delete-Flag selektiert werden. Dies würde ich nicht in ein solches System einbauen, um die gelöschten Datensätze zu erhalten. Solch eine Historisierung verringert die Performanz dramatisch, da die Datenbank die doppelte Menge an Daten schreiben muss. Es wird jeweils ein Datensatz in die Historisierungstabelle geschrieben, es wird also auch der doppelte Speicherplatz in der Datenbank benötigt.
Die zweite Schwachstelle ist das hier thematisierte Problem, die Möglichkeit, dass nicht alle Daten in das Staging-System übernommen werden könnten. Diese Art der Lösung ist nicht transaktionssicher. Das bedeutet, dass Transaktionen oder Teile von Transaktionen verloren gehen können.
Anhand eines einfachen Beispiels werde ich erläutern, was damit gemeint ist.

Ich werde dies an der bereits vorgestellten Tabelle MY_SALES demonstrieren. Der Datensatz mit der ID 1000 wird hierzu modifiziert.

Das operationale System, auch Quelle genannt, ist die ORCL Datenbankinstanz und das Datenbankschema BCL. Das Staging-System ist die STAGING Datenbankinstanz.
Für die SQL basierte Lösung wird nur dargestellt, welche Daten auf der Quelle zu welchem Zeitpunkt selektiert werden. Dies kann mit einem Datenbanklink von der Staging Datenbank durchgeführt werden, um die Daten dort in eine Tabelle zu schreiben. Dieses Schreiben ist für den Test nicht wichtig und wird deshalb nicht gezeigt.

Der aktuelle Datensatz ist bereits in dem Staging-System vorhanden.

```
SELECT * FROM MY_SALES WHERE ID = 1000;
```

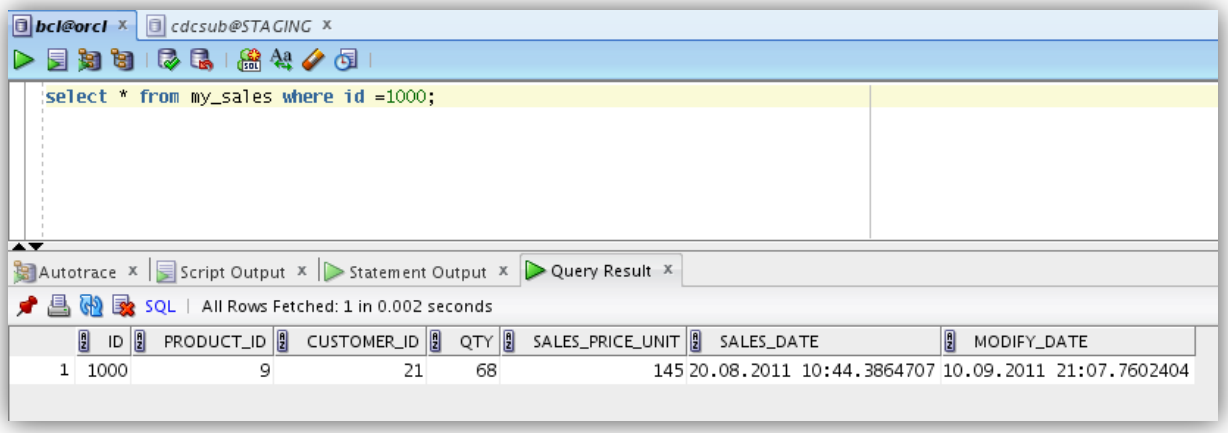

Abb. 1: Der beschrieben Datensatz

In vielen Quellsystemen werden aus anderen Systemen Massendatenänderungen durchgeführt. Das könnten zum Beispiel Massenbestellungen von Kunden sein, die in das Quellsystem hinein geladen werden. Diese können lange Laufzeiten haben. Ein solcher Prozess wird durch das folgende Skript simuliert. Es handelt sich um eine langlaufende Transaktion in welcher entweder alle Daten am Ende bestätigt werden (commit) oder alle Daten verworfen werden (rollback).

Jetzt wird der erwähnte Datensatz in einer simulierten langlaufenden Transaktion verändert.

```
Session 1: Massenänderungen in einer Batch Demo

prompt sysdate beim Start
select sysdate from dual;
prompt die Änderung
update my_sales set qty = 168 where id = 1000;
prompt simuliere weitere Änderungen durch 10 Minuten Wartezeit
exec dbms_lock.sleep(600);
prompt Commit
commit;
prompt sysdate nach dem Commit
select sysdate from dual;
```

Es wird die Zeit zu Beginn der Ausführung selektiert, danach folgt die eigentliche Änderung, in der die Anzahl der verkauften Produkte von 68 auf 168 gesetzt wird.
Danach wird das DBMS_LOCK Paket mit der Sleep-Prozedur aufgerufen. Die Zahl entspricht den Sekunden, die jetzt gewartet werden. Danach wird diese Transaktion mit einem Commit abgeschlossen.
Erst nach dem Commit sind die Änderungen für andere Datenbanksession sichtbar.
Hiermit wird ein Batch-Job simuliert, der 10 Minuten läuft und die Daten innerhalb einer Transaktion verarbeitet. Dies ist ein moderater Wert, da solche Jobs durchaus deutlich länger als eine Stunde ausgeführt werden könnten.
Der Screenshot zeigt das Ergebnis des Skriptes.

Abb. 2: Datensätze, die zwischen 17:20 Uhr und 18:20 Uhr geändert wurden

Dieser Test zeigt, dass um 18:20 Uhr keine Änderungen an den Daten zwischen 17:20 Uhr und 18:20 Uhr festgestellt werden konnte und somit keine Daten zum Staging-System geschickt werden.

```
Session 2: Inkrementeller Ladevorgang Demo

prompt sysdate beim Start des inkremetellen SQL Loads
select sysdate from dual;
prompt Wieviele Datensätze wurden zwischen 17:20 und 18:20 verändert
select count(*) from my_sales where modify_date >= to_date('11.09.2011
17:20','DD.MM.YYYY HH24:MI') and modify_date < to_date('11.09.2011
18:20','DD.MM.YYYY HH24:MI');
```

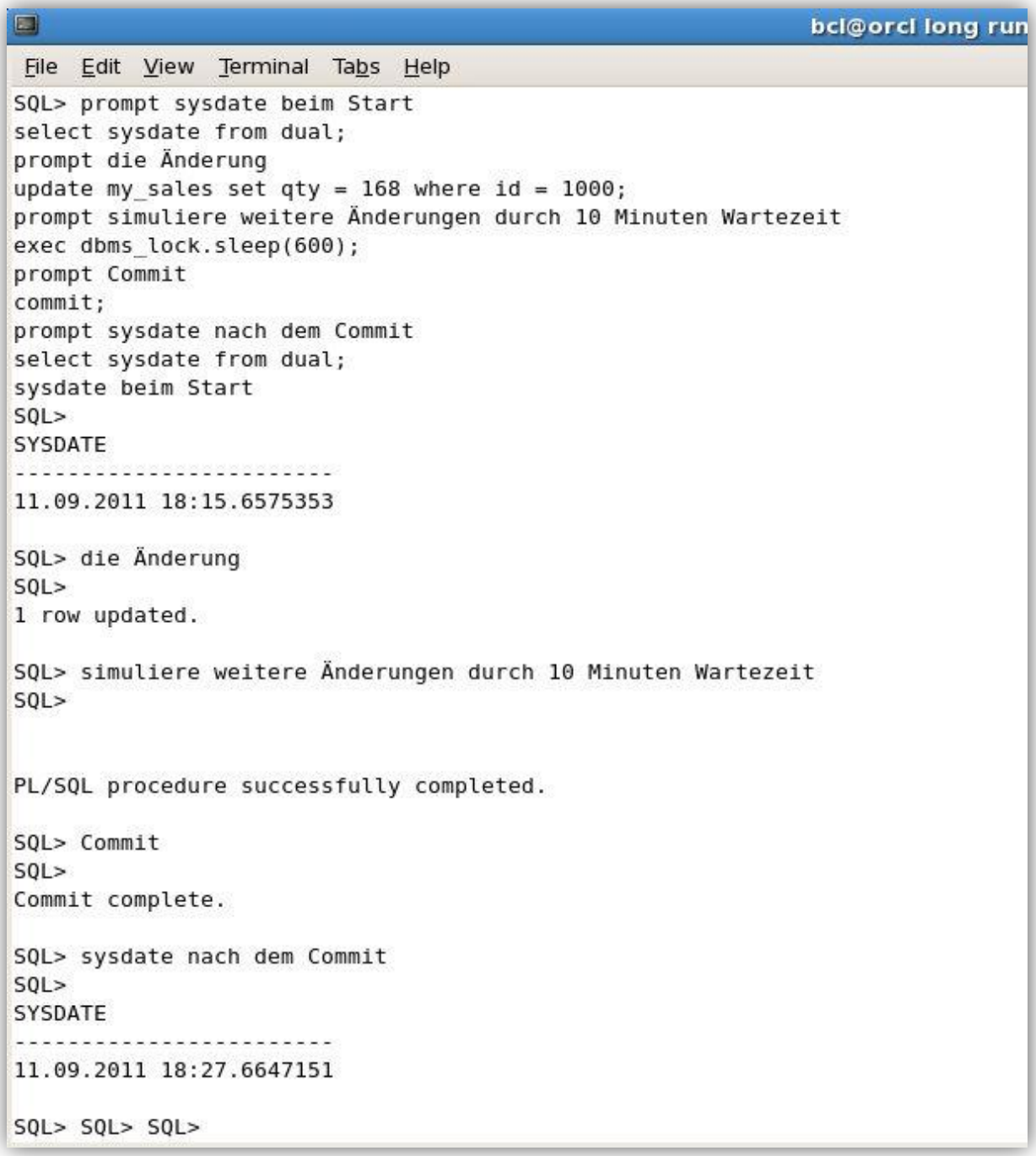

Abb.3: Ausführung des langlaufenden Batch-Jobs

Der Screenshot in Abb.3 zeigt die Ausführung des langlaufenden Batch_Jobs. Dieser Prozess wurde also um 18:15 Uhr gestartet und die Änderung durchgeführt. Beendet wurde diese Transaktion um 18:27 Uhr. Als um 18:20 Uhr der inkrementelle Ladevorgang durchgeführt wurde (siehe Abb. 2), gab es zwischen 17:20 Uhr und 18:20 Uhr keine Änderungen, der nächste Lauf wäre um 19:20 Uhr und würde alle Änderungen zwischen 18:20 Uhr und 19:20 Uhr transferieren.

Dieser muss die Änderung enthalten. Erstaunlicherweise ist dies nicht der Fall, wie Abb. 4 zeigt. Auch in diesem Zeitintervall ist die in dem Test gemachte Änderung nicht enthalten.

```
Nächster Inkrementeller Ladevorgang Demo:

prompt sysdate beim Start des inkremetellen SQL Loads
select sysdate from dual;
prompt Wieviele Datensätze wurden zwischen 18:20 und 19:20 verändert
select count(*) from my_sales where modify_date >= to_date('11.09.2011
18:20','DD.MM.YYYY HH24:MI') and modify_date < to_date('11.09.2011
19:20','DD.MM.YYYY HH24:MI');
```

Abb.4: Datensätze, die zwischen 18:20 Uhr und 19:20 Uhr geändert wurden

In keinem der Ladevorgänge ist die gemachte Änderung an das Staging-System gesendet worden. Da in diesem Test genau feststeht, welcher Datensatz geändert wurde, wird dieser jetzt explizit selektiert, um festzustellen, ob die Änderung überhaupt angewendet wurde.

```
Selektiere den geänderten Datensatz:

prompt sysdate
select sysdate from dual;
prompt selektiere den geänderten Datensatz mit der ID 1000
select * from my_sales where id = 1000;
```

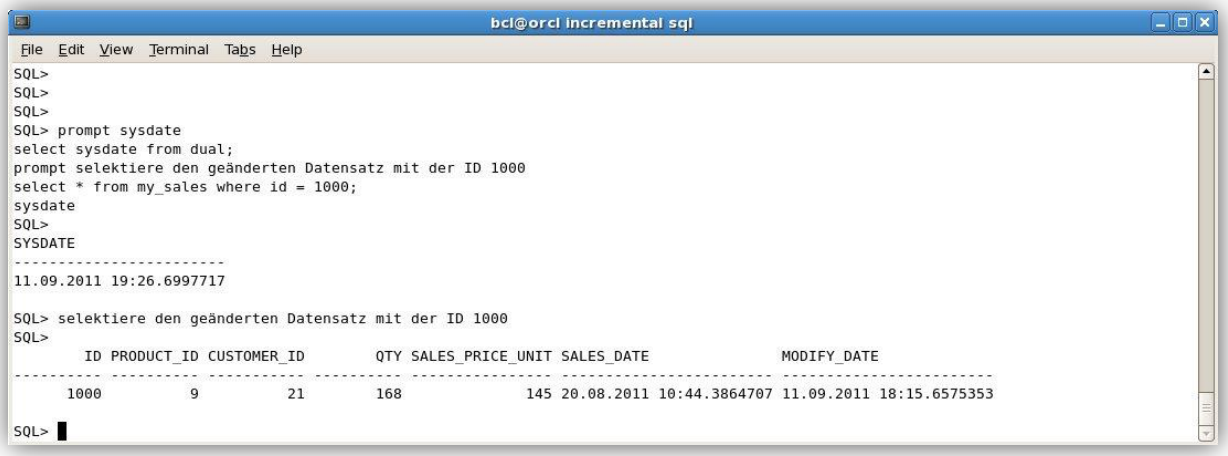

Abb.5: geänderter Datensatz im Quellsystem

Dieses SQL und sein Ergebnis beweist, dass der Datensatz geändert wurde. Hier steht ein Modify_date von 18:15 Uhr. Das war in dem Ladevorgang um 18:20 Uhr noch nicht selektierbar.

Die Folge ist, dass diese Änderung nicht in das Staging-System übernommen und somit auch nicht in das Datawarehouse geladen wird.
Dazu gibt es anzumerken, dass hier kein Fehler im System vorliegt, außer das in dem hier vorgestelltem Design ein logischer Fehler steckt. Die oft verwendeten Trigger setzen das Modify_date am Anfang der Ausführung, dies ist je nach Laufzeit der Transaktion allerdings deutlich vor dem Zeitpunkt, an dem diese Änderung für alle sichtbar wird.

Durch den hier gezeigten Test wurde bewiesen, dass es Situationen gibt, in denen nicht alle Informationen an das Staging-System gesendet werden. Jetzt sollte nicht die Frage nach der Wahrscheinlichkeit, mit der dieses Problem auftritt gestellt werden, sondern nur die Frage, ob der Prozess alle Änderungen garantiert an das Staging-System liefern muss.

Wenn die Daten garantiert geliefert werden müssen, ist dieser Ansatz ohne größere Eingriffe nicht nutzbar.

Hier kommen Change-Data-Capture-Verfahren ins Spiel.

Ausblick auf den DOAG-Konferenzvortrag
Durch den beschriebenen Testfall ist bewiesen worden, dass in bestimmten Konstellationen nicht garantiert werden kann, dass alle Daten aus allen Transaktionen an das Staging-System übertragen werden.
In meinem Vortrag werde ich diesen Testfall mit Oracle CDC exemplarisch lösen.
Als weitere Option wird Oracle Golden Gate verwendet, um dieses Problem ebenfalls zu umgehen.

Kontaktadresse:

Bodo Clausen
OPITZ CONSULTING Gummersbach GmbH
Kirchstraße 6
D-51647 Gummersbach
bodo.clausen@opitz-consulting.com

Telefon: +49 (0) 2261-6001-0
Fax: +49 (0) 2261-6001-4000
Internet: www.opitz-consulting.com