Dynamisch Unterschiede in Datensätzen auf Feldebene finden

Sven-Olaf Kelbert
MT AG
Ratingen

Schlüsselworte
Differenzen, Dynamisches SQL, PL/SQL, Data Dictionary, pipelined table function.

Einleitung
Um herauszufinden, ob Datensätze in einer Tabelle unterschiedlich sind, bietet Oracle die Mengen-Operation MINUS an. Mit MINUS findet man aber nur heraus, ob es grundsätzlich Unterschiede zwischen den beiden Datensätzen gibt. Was ist aber, wenn man konkret herausfinden will, in welchen Spalten die Unterschiede liegen? Und was macht man, wenn die betroffene Tabelle mehrere Hundert Spalten hat und man nicht einfach alle nacheinander anschauen kann? Und was ist, wenn man die Unterschiede auch noch für alle Detail-Sätze des Datensatzes benötigt und dies auch noch über 100 Tabellen betrifft?
Die Aufgabe ist also, sämtliche Unterschiede in einem beliebigen, zusammenhängenden fachlichen Datenkonstrukt herauszufinden. Das schreit geradezu nach einer dynamischen, flexiblen Lösung.

Die Ausgangslage
Eine Fonds-Gesellschaft benutzt für die Verwaltung ihrer Fonds ein Datenbank-System, den sog. Fonds-Profiler. In diesem System können zu jedem Fonds, der von der Gesellschaft aufgelegt wird, umfangreiche Informationen hinterlegt werden, wie z.B. die Anlagevorschriften für bestimmte Länder, Anlageformen und Börsen, Benchmarks sowie strukturelle und verwaltungstechnische Angaben.

Das Datenmodell
Da diese Informationen so zahlreich sind, ist auch das Datenmodell recht umfangreich. Die Basistabelle für ein Fondsprofil ist dabei die Tabelle FONDSSTAEMME. Ein Fondsstamm kann dabei mehrere Anteilsscheine enthalten, welche wiederum weitere verschiedene Detail-Informationen besitzen.

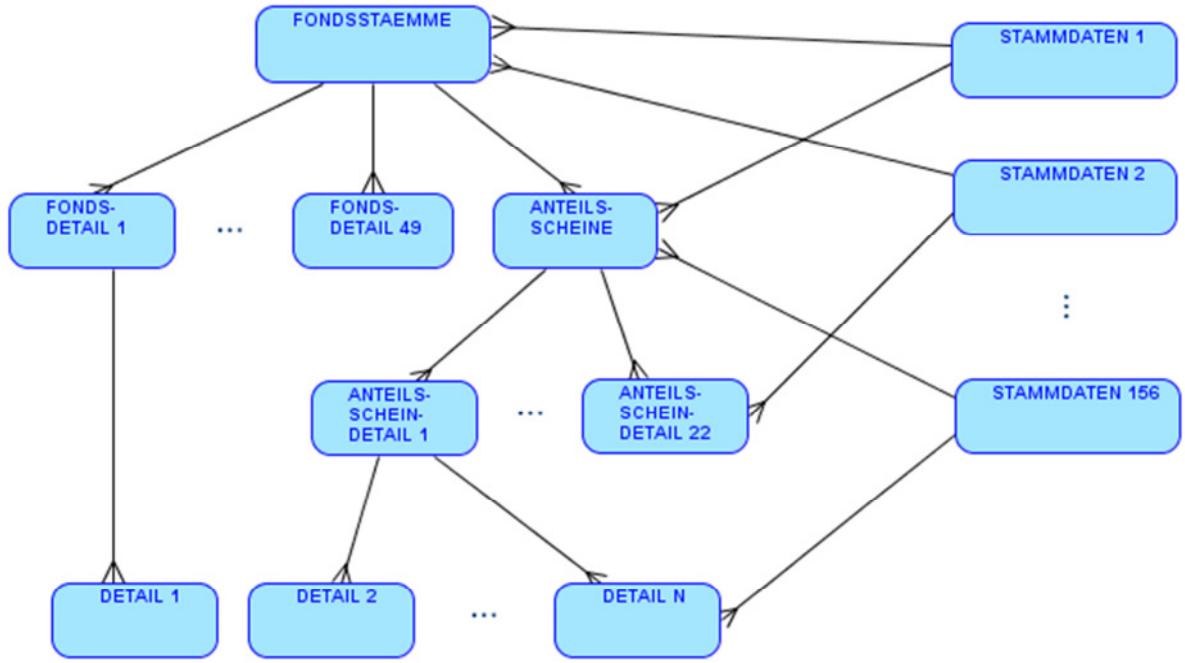

Abb. 1: Überblick über das Datenmodell

Ein Fondsstamm hat Verbindungen zu 50 Tabellen, die Detail-Inhalte enthalten können. Ein Anteilsschein hat Verbindungen zu weiteren 22 Tabellen mit Detail-Inhalten. Und viele dieser Detail-Tabellen haben weitere Detail-Tabellen mit weiteren Detaillierungen über mehrere Ebenen. Außerdem enthält das System 156 Stammdaten-Tabellen, deren Inhalte aus den meisten Tabellen referenziert werden. Insgesamt enthält das Datenmodell 344 Tabellen mit 7307 Spalten.

Der Wunsch

Im Fondsprofiler ist es möglich, ein komplettes Fondsprofil mit allen Detail-Informationen zu kopieren. Dies ist einerseits nötig, um neue Versionen eines Fondsprofils anlegen zu können, wenn sich z.B. Anlagerestriktionen ändern sollen. Aufgrund der Revisionssicherheit muss die Original-Version eines Fondsprofils natürlich erhalten bleiben. Außerdem soll es möglich sein, neue Fonds auf Basis von bestehenden anzulegen, um nicht bei jeder Fondsanlage sämtliche Daten komplett neu ausfüllen zu müssen.
Wurde von einem Fondsprofil eine neue Version erzeugt, besteht der Wunsch, die beiden Versionen mit Hilfe eines sogenannten Delta-Reports miteinander zu vergleichen, um die Unterschiede zwischen den Versionen festzustellen. Dies dient einerseits der Identifizierung von Eingabefehlern und somit der Sicherheit des Fondsmanagers. Andererseits soll es so aber auch möglich sein, falls mehrere neue Versionen eines Fondsprofils erzeugt wurden, Änderungen von einer Version auf eine weitere Version zu übertragen, solange diese Version noch nicht finalisiert, also aufgelegt, wurde.
Da ein manueller Vergleich aufgrund der hohen Anzahl von zu vergleichenden Daten in ca. 150 Masken nicht möglich ist, muss dieser Vergleich programmatisch umgesetzt werden.

Die Umsetzung – allgemeiner Teil

Aufgrund der Anzahl der zu vergleichenden Datenbank-Felder bot es sich nicht an, die einzelnen Felder explizit und hart verdrahtet miteinander zu vergleichen. Außerdem hätte man bei so einem Konstrukt bei jeder weiteren Datenmodell-Änderung daran denken müssen, auch den Delta-Report anzupassen.

Somit wurde eine dynamische Lösung entwickelt, die einerseits übersichtlich sein sollte und des Weiteren bei zukünftigen System-Anpassungen automatisch weiter funktionierte.

Der Einstieg in den Delta-Report erfolgt über die Tabelle FONDSSTAEMME. Hier wird zuerst ermittelt, welche weiteren Versionen es zu diesem Fondsprofil gibt. Diese verschiedenen Versionen werden dann durchlaufen, um sämtliche Änderungen zwischen allen Versionen eines Fondsprofils aufzeigen zu können.

Die Funktion mit dem wesentlichen Programminhalt hat dann folgende Signatur:

```
function get_differenzen_table
  (p_table_name in varchar2,
   p_pk_constraint_name in varchar2,
   p_pk_column_name in varchar2,
   p_pk_id_akt in number,
   p_pk_id_hist in number,
   p_fond_version in number,
   p_master_detail_kz in varchar2)
  return t_differenzen
  pipelined;
```

Die Parameter haben dabei folgende Inhalte:

- p_table_name: Tabellen-Name der aktuell zu vergleichenden Tabelle, beim Start also ‚FONDSSTAEMME'
- p_pk_constraint_name: Der Name des PK-Constraints der aktuell zu vergleichenden Tabelle
- p_pk_column_name: Der Name der PK-Spalte der aktuell zu vergleichenden Tabelle
- p_pk_id_akt: Der Wert der PK-Spalte des Datensatzes der Ausgangs-Version
- p_pk_id_hist: Der Wert der PK-Spalte des Datensatzes der Vergleichs-Version
- p_fond_version: Versions-Nummer der aktuell betrachteten Fondsprofil-Version
- p_master_detail_kz: Kennzeichen für die weitere Verzweigungsrichtung

Die Funktion ist eine pipelined table function und gibt einen Objekt-Typ zurück, in welchem alle Informationen zu Unterschieden enthalten sind:

- Name der Tabelle und Spalte, in welcher die Differenz auftritt
- Name der Maske sowie des Feldes, in welchem die Differenz zu sehen ist
- Werte, die das Feld in den beiden Versionen annimmt
- diverse Hilfsdaten für den Abgleich, wie z.B. Wert und Name der PK-Spalten, Wert und Name der FK-Spalten für Stammdatentabellen
- Informationen, wer die Änderung wann durchgeführt hat

Die Funktion ruft sich rekursiv selbst auf und durchläuft auf diese Art den Teil des Datenmodells, der mit der Tabelle FONDSSTAEMME über Constraints zusammenhängt.

Die Umsetzung im Detail
Wie funktioniert nun die Funktion get_differenzen_table konkret? Als erstes werden zu der übergebenen Tabelle alle Spalten aus dem Data Dictionary herausgesucht und per Loop durchlaufen, um Differenzen in jeder Spalte finden zu können:

```
select column_name,
       data_type
  from user_tab_columns
```

```
 where table_name = p_table_name
```

Nun muss unterschieden werden, ob es sich bei der aktuell zu betrachtenden Spalte um eine normale Spalte handelt oder um eine FK-Spalte, also eine ID-Spalte, die auf einen Masterdatensatz in einer Stammdatentabelle verweist. Denn bei einer solchen FK-Spalte nützt es dem Anwender nichts, wenn ein Unterschied zwischen zwei IDs angezeigt wird, da der Anwender die internen IDs nicht kennt. Hier muss der zugehörige Masterdatensatz aus der Stammdatentabelle geholt werden und das Beschreibungsfeld, welches zu der ID gehört, dem Anwender angezeigt werden.
Da der Funktion get_differenzen_table die IDs der beiden zu vergleichenden Datensätze schon mitgegeben werden, kann man nun direkt die einzelnen Spalten, die ja per Loop durchlaufen werden, miteinander vergleichen. Dies erfolgt mit dynamischem SQL:

```
execute immediate
  'select tab_akt.'||rec.column_name||', '||
  '       tab_hist.'||rec.column_name||
  '  from '||p_table_name||' tab_akt, '||p_table_name||' tab_hist '||
  ' where tab_akt.'||p_pk_column_name||' =
                        nvl('||to_char(n_id1)||', '||to_char(n_id2)||')'||
  '   and tab_hist.'||p_pk_column_name||' =
                        nvl('||to_char(n_id2)||', '||to_char(n_id1)||')'
into v_wert_akt, v_wert_hist;
```

Ist die Spalte vom Typ DATE, muss noch eine Formatierung ergänzt werden:

```
execute immediate
  'select to_char(tab_akt.'||rec.column_name||',
                                              ''DD.MM.YYYY HH24:MI:SS''),'||
  '       to_char(tab_hist.'||rec.column_name||',
                                              ''DD.MM.YYYY HH24:MI:SS'')'||
  '  from '||p_table_name||' tab_akt, '||p_table_name||' tab_hist '||
```

Sind nun die Werte der Spalte herausgefunden, muss noch unterschieden werden, ob überhaupt zwei Datensätze zum Vergleichen da waren oder nicht. Es könnte z.B. sein, dass nach der Kopie eines Fondsprofils in der neuen Version ein Detail-Datensatz komplett gelöscht wurde. Oder dass ein Detail-Datensatz hinzugefügt wurde, wo es vorher noch keinen gab. Das erkennt man daran, dass der Funktion get_differenzen_table in einem der Parameter p_pk_id_akt oder p_pk_id_hist der Wert null übergeben wurde. In diesem Fall würde der Delta-Report folgendes Ergebnis zurückliefern:

Ausgangs-Version	Vergleichs-Version
<kein Datensatz vorhanden>	xyz

In dem Fall, dass beide Datensätze vorhanden sind, werden die unterschiedlichen Werte ausgegeben:

Ausgangs-Version	Vergleichs-Version
abc	xyz

Die Ausgabe erfolgt hierbei, da es sich ja um eine pipelined table function handelt, indem die auszugebenden Werte einer Variablen zugewiesen werden, die vom Typ des zurückzugebenden Object Types ist. Anschließend wird diese Variable mit pipe row ausgegeben:

```
v_differenz := t_differenz (p_table_name,
                            rec.column_name,
```

```
                                v_wert_akt,
                                v_wert_hist,
                                ...);
pipe row (v_differenz);
```

Dies war das Vorgehen für den Fall, dass wir eine normale Spalte betrachten. Haben wir aber gerade eine FK-Spalte im Zugriff, muss ja in den Master-Datensatz der zugehörigen Stammdatentabelle verzweigt werden. Die verschiedenen Spalten des Master-Datensatzes mit ihren Werten bekommen wir heraus, indem wir die Funktion get_differenzen_table rekursiv für den Master-Datensatz aufrufen. Da die Funktion eine pipelined table function ist, geht dies nicht über einen direkten Aufruf, sondern über ein Select. Das folgende Statement zeigt, wie eine pipelined table function aufgerufen werden kann:

```
select *
  from table(get_differenzen_table(v_master_table_name,
                                   v_master_constraint_name,
                                   v_pk_column_name,
                                   to_number(v_wert_akt),
                                   to_number(v_wert_hist),
                                   p_fond_version,
                                   'M'))
```

Das Select enthält quasi einen Delta-Report für diese Master-Tabelle. Die zurückgelieferten Datensätze können dann in den übergeordneten Delta-Report übernommen werden.

Sind nun alle Spalten der aktuellen Tabellen inklusive ihrer Masterdaten aus den benötigten Stammdatentabellen abgearbeitet, geht es daran, in sämtliche Detail-Tabellen der aktuellen Tabelle herabzusteigen. Der aktuell gerade abgearbeitete Datensatz wird im weiteren Verlauf nun der besseren Unterscheidung wegen als Master-Datensatz bezeichnet.

Als erstes wird jetzt anhand des Data Dictionaries geprüft, welche Tabellen einen FK-Constraint auf den PK des gerade betrachteten Master-Datensatzes der aktuellen Tabellen haben. Diese Detail-Tabellen müssen nacheinander per Rekursion durchlaufen werden, um auch dort die Differenzen herauszufinden.

Für die Überprüfung der Detail-Tabellen ist es wichtig zu wissen, wie viele Detail-Datensätze es jeweils für die beiden zu vergleichenden Master-Datensätze gibt. Gibt es jeweils gar keinen Detail-Datensatz, braucht nichts verglichen zu werden. Gibt es jeweils genau einen Detail-Datensatz, können diese beiden Datensätze direkt miteinander verglichen werden. Kompliziert wird es, wenn zu einem Datensatz mehrere Detail-Datensätze vorhanden sind.

Betrachten wir aber zunächst den Fall, dass zu jedem Datensatz genau ein Detail-Datensatz vorhanden ist.

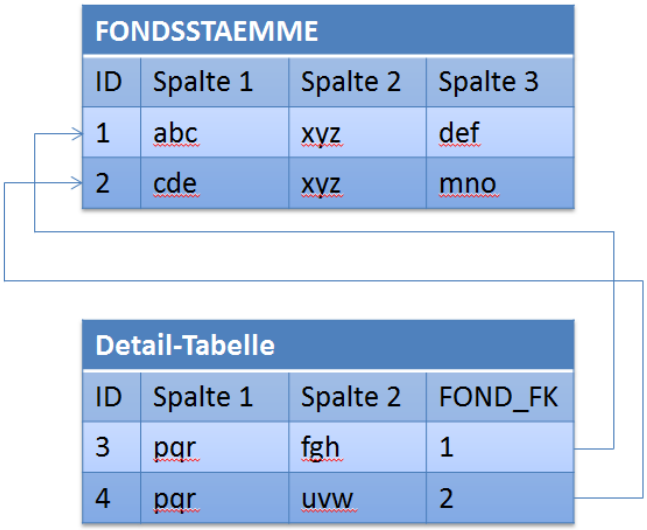

Abb. 2: Szenario mit einem Detail-Satz für jeden Master-Datensatz

Hier wird wieder ein Rekursions-Schritt gemacht, indem get_differenzen_table statt mit den beiden Haupt-Datensätzen aus der übergeordneten Tabelle nun mit den beiden Detail-Datensätzen aus der Detail-Tabelle aufgerufen wird. Die Ergebnisse, also die Differenzen zwischen den beiden Datensätzen, die aus der rekursiv aufgerufenen Funktion zurück geliefert werden, müssen anschließend noch in das Ergebnis der aufrufenden Funktion übernommen werden.

Existieren zu jedem Haupt-Datensatz mehrere Detail-Datensätze, wird es um einiges komplizierter.

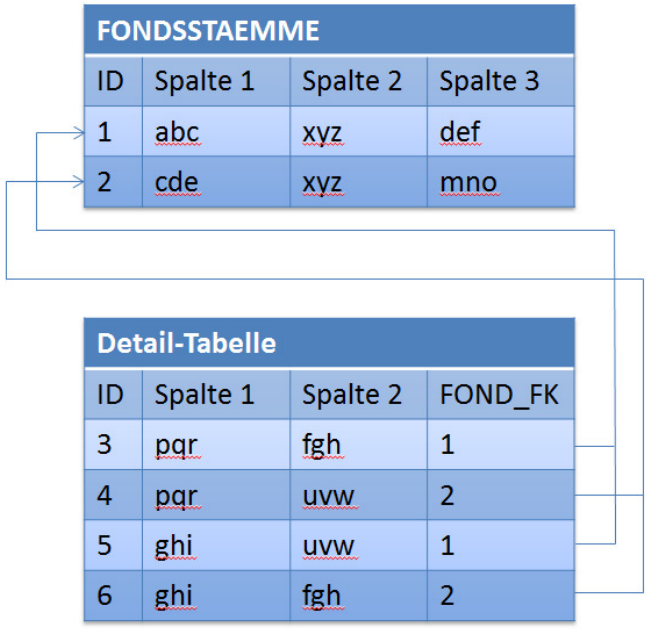

Abb. 3: Szenario mit mehreren Detail-Sätzen für jeden Master-Datensatz

Denn in diesem Fall kann nicht direkt geklärt werden, welche Datensätze miteinander verglichen werden müssen. In obigem Beispiel könnten so entweder die Datensätze 3/4 und 5/6 verglichen werden oder die Datensätze 3/6 und 4/5. Je nach Vergleich fällt dann das Ergebnis des Delta-Reports unterschiedlich aus.

Deshalb wurde eine Funktionalität entwickelt, dass bei einer Kopie des Fondsprofils für alle zu kopierenden Datensätze die ID des Ursprungsdatensatzes sowie die ID der daraus entstandenen Kopie in einer Mapping-Tabelle gespeichert werden. Existieren nun mehrere Detail-Datensätze in einer zu vergleichenden Tabelle, wird anhand der Mapping-Tabelle geklärt, welcher Datensatz mit welchem verglichen werden muss. Anschließend wird wieder die Funktion get_differenzen_table rekursiv aufgerufen mit den entsprechenden zu vergleichenden Datensätzen. Die Ergebnisse, also die Differenzen zwischen den beiden Datensätzen, die aus der rekursiv aufgerufenen Funktion zurück geliefert werden, werden anschließend wieder in das Ergebnis der aufrufenden Funktion übernommen.

Schwierigkeiten
Die Verzweigung durch das Datenmodell, um die nächste Detail-Tabelle zu finden bzw. den Masterdatensatz zu einer FK-Spalte, sollte eigentlich dynamisch über das Data Dictionary bestimmt werden über die Tabellen USER_CONSTRAINTS und USER_CONS_COLUMNS. Dies hat sich aber als sehr inperformant erwiesen. Deshalb wurde eine materialized view entwickelt, welche die benötigten Tabellen und Spalten des Data Dictionaries enthält. Diese materialized view wird in jeder Nacht aktualisiert und sieht wie folgt aus:

```
create materialized view CONS_COLUMNS_MV
refresh force on demand
start with to_date('01-01-2012', 'dd-mm-yyyy') next TRUNC(SYSDATE) + 1
as
select detail_ac.table_name detail_table_name,
       detail_pk_ac.constraint_name detail_pk_constraint,
       detail_pk_acc.column_name detail_pk_column_name,
       detail_ac.constraint_name detail_fk_constraint,
       detail_acc.column_name detail_column_name,
       master_ac.table_name master_table_name,
       detail_ac.r_constraint_name master_pk_constraint,
       master_acc.column_name master_column_name
  from all_cons_columns master_acc,
       all_constraints master_ac,
       all_cons_columns detail_acc,
       all_constraints detail_ac,
       all_cons_columns detail_pk_acc,
       all_constraints detail_pk_ac
 where detail_acc.constraint_name = detail_ac.constraint_name
   and detail_ac.constraint_type = 'R'
   and detail_ac.r_constraint_name = master_acc.constraint_name
   and detail_ac.r_constraint_name = master_ac.constraint_name
   and detail_pk_ac.constraint_type = 'P'
   and detail_pk_ac.table_name = detail_ac.table_name
   and detail_pk_acc.constraint_name = detail_pk_ac.constraint_name
   and <Owner in allen Tabellen> = <Owner>;
```

Der Zugriff auf diese materialized view war wesentlich schneller als der direkte Zugriff auf das Data Dictionary.

Vorsicht war weiterhin geboten bei der Kombination von Rekursion mit einer pipelined table function. Bei einer normalen rekursiven Funktion bekommt man von der rekursiv aufgerufenen Funktion ein Ergebnis zurück, welches man anschließend beliebig weiter verarbeiten kann. Eine pipelined table function liefert als Ergebnis allerdings eine Datentabelle zurück. Also kann schon mal der rekursive Aufruf nicht einfach als Zuweisung durchgeführt werden, sondern muss über ein

```
select *
  from table(get_differenzen_table(v_master_table_name,
                                   v_master_constraint_name,
                                   v_pk_column_name,
                                   to_number(v_wert_akt),
                                   to_number(v_wert_hist),
                                   p_fond_version,
                                   'M'))
```

abgebildet werden. Sämtliche Datensätze, die auf diese Art erhalten werden, müssen anschließend weiter gereicht werden in die Datentabelle der aufrufenden Funktion. Es muss also für jeden der Datensätze aus dem Select erneut ein

```
pipe row (v_differenz);
```

durchgeführt werden, da die Datensätze ansonsten verloren gehen. So werden die Ergebnis-Datensätze aus den unteren Ebenen der Rekursion in jeder Ebene wieder per pipe row in das Ergebnis der nächsthöheren Ebene integriert.

Nachziehen von Änderungen
Als Zusatz-Funktionalität sollte es möglich sein, die Differenzen, die zwischen zwei Datensätzen festgestellt worden sind, automatisiert anzugleichen, also von der einen in die andere Version des Fondsprofils zu übertragen.

Hier müssen wir zwischen drei verschiedenen Varianten unterscheiden. Fehlt im Delta-Report auf einer Seite des Vergleichs der Datensatz komplett, muss er hinzugefügt werden, sofern er in der zu aktualisierenden Version fehlt. Fehlt er in der ursprünglichen Version, muss er in der zu aktualisierenden Version ebenfalls gelöscht werden. Sind beide Datensätze vorhanden, muss ein Update auf die unterschiedlichen Felder durchgeführt werden.

Muss ein Datensatz gelöscht werden, muss natürlich auch berücksichtigt werden, dass seine sämtlichen Details auch gelöscht werden, bevor der eigentliche Datensatz gelöscht werden kann. Auch dies wurde wieder per Rekursion gelöst und per Durchlauf durch sämtliche Details, die anhand des Data Dictionaries gefunden wurden.

Beim Hinzufügen von Datensätzen muss sichergestellt werden, dass ein Datensatz nur einmal hinzugefügt wird, auch wenn vielleicht mehrere Differenzen im Delta-Report angezeigt werden. Und es muss dynamisch die Liste aller Spalten zusammengestellt werden, damit wirklich der komplette Datensatz übernommen werden kann.

Der einfachste Fall, das Update auf einzelne Spalten, muss nur sicherstellen, dass keine UKs verletzt werden.

Fazit
Mit dynamischen Selects und dem Weg über das Data Dictionary ist es relativ einfach möglich, einen technischen Delta-Report zu erzeugen. Die Probleme stecken allerdings auch hier im Detail. Gibt es

z.B. Tabellen, die Meta-Daten enthalten für die Darstellung in Masken, können diese nicht so einfach derart analysiert werden. Denn dann werden oft Unterschiede angezeigt auf einer Basis, die der Anwender nicht versteht.
Das Problem der Lesbarkeit ist auch sonst gegeben. Der Delta-Report zeigt zwar technisch alle Differenzen zwischen zwei Datensätzen an, kann aber aufgrund der Dynamik nicht beurteilen, ob gewisse Daten den Anwender verwirren oder hilfreich sind. So wurden z.B. gewisse interne Spalten später ausgeblendet und für bestimmte Tabellen hart verdrahtet andere Vorgehensmodelle gewählt. Aber alle diese Anpassungen machen den Programmcode weniger gut lesbar und zerstören letztlich auch die Dynamik.

Im Endeffekt ist dies eine sehr gute Möglichkeit, auf technischer Basis Differenzen zwischen komplexen fachlichen Konstrukten über viele Tabellen und Spalten hinweg herauszufinden. Für einen normalen Fachanwender, der sich mit dem technischen Datenbank-Umfeld seines Programms nicht so gut auskennt, dürfte die Darstellung aber in vielen Fällen Fragen oder Probleme hervorrufen. Hier ist dann eine Abwägung nötig, ob einzelne hart verdrahtete Anpassungen ausreichen, oder ob der ganze Vergleich doch statisch und nicht dynamisch erzeugt wird, weil dort dann bessere Eingriffsmöglichkeiten vorhanden sind.

Kontaktadresse:
Sven-Olaf Kelbert
MT AG
Balcke-Dürr-Allee 9
D-40882 Ratingen

Telefon:	+49 (0) 2102 30961-0
Fax:	+49 (0) 2102 30961-101
E-Mail	sven-olaf.kelbert@mt-ag.com
Internet:	www.mt-ag.com