

# Powerful PL/SQL: Collections indizieren mit VARCHAR2- Indizes – ein Praxisbeispiel

Autor: Klaus Friemelt, MT AG

## Schlagworte

dynamisches BULK SQL, VARCHAR2-indizierte PL/SQL-Tabellen

## Einleitung

Mit den letzten Oracle Datenbankversionen wurde auch die PL/SQL-Engine der Datenbank um mächtige Funktionalitäten erweitert, wie z.B. *BULK SQL* (seit 8i) und *alphanumerische Schlüssel* für PL/SQL-Tabellen (seit 9i Release 2). Damit lassen sich für manche Aufgabenstellungen enorme Performancegewinne erzielen, bzw. ganz neue Lösungswege erschließen. Das hier vorgestellte Beispiel basiert auf der Datenbankversion 10g.

## Aufgabe

Im vorliegenden Szenario sollen knapp 33.000 Datensätze mit dem Oracle Warehouse Builder (OWB) in eine Staging-Tabelle geladen werden. Dort sollen sie logisch validiert werden, indem die Inhalte diverser Spalten der Staging-Tabelle gegen Wertebereiche abgeglichen werden sollen. Fehlerhafte Sätze erhalten in der Staging-Tabelle ein Kennzeichnungsflag, korrekte Sätze werden später durch den OWB in der Zieltabelle abgelegt.

Eine „harte“ Validierung durch Check-Constraints auf der Datenbank oder durch Trigger ist aus zwei Gründen ausgeschlossen: zum einen sollen die fehlerhaften Sätze trotzdem geladen werden, um sie nach dem Laden korrigieren zu können, zum anderen soll die Validierung performant sein.

## Lösung

Die Lösung basiert auf einer PL/SQL-Tabelle T\_LOOKUP, die nicht wie üblich mit BINARY\_INTEGER, sondern mit einem alphanumerischen Schlüssel indiziert wird:

*Beginn Programmcode*

```
Type t_vc100_ib_vc1 is table of VARCHAR2(1) index by VARCHAR2(100);  
T_LOOKUP t_vc100_ib_vc1;
```

*Ende Programmcode*

Die Tabelle T\_LOOKUP wird mit den möglichen Werten aus der Referenztable indiziert und erhält als Eintrag lediglich ein Flag.

Die Daten aus der Spalte der Staging-Tabelle werden in zwei „klassisch“ indizierte PL/SQL-Tabellen T\_RID und T\_VALUE geladen; T\_RID nimmt die ROWID auf, T\_VALUE den Spalteninhalt der Staging-Tabelle:

*Beginn Programmcode*

```
type t_rowid is table of ROWID index by BINARY_INTEGER;  
T_RID t_rowid;
```

```
type t_vc100_ib_bi is table of VARCHAR2(100) index by BINARY_INTEGER;  
T_VALUE t_vc100_ib_bi;
```

*Ende Programmcode*

Damit eröffnet sich nun die Möglichkeit, für alle 35.000 Sätze der Staging-Tabelle (hier DATA\_STAGE) die zu überprüfende Spalte (hier „CURRENCY“) per BULK COLLECT mit einem einzigen Statement in diese PL/SQL-Tabellen zu laden:

*Beginn Programmcode*

```
SELECT ROWID, CURRENCY BULK COLLECT INTO T_RID, T_VALUE FROM DATA_STAGE
```

*Ende Programmcode*

Jetzt müssen in einer Schleife für alle Einträge der Tabelle **T\_VALUE** die Existenz in der Tabelle **T\_LOOKUP** verifiziert werden. Die fehlerhaften Sätze erhalten einen Eintrag in einer weiteren PL/SQL-Tabelle **T\_UPDATE\_F** vom Typ **t\_rowid**:

*Beginn Programmcode*

```
for i in 1..T_RID.COUNT
loop
  if NOT T_LOOKUP.EXISTS( T_VALUE(i) ) then
    -- illegaler Wert, eintragen zum Update
    T_UPDATE_F(j) := T_RID(i);
    j := j +1;
  end if;
end loop;
```

*Ende Programmcode*

Als letztes werden für alle auffälligen Sätze in der Tabelle **T\_UPDATE\_F** die entsprechenden Sätze in der Staging-Tabelle mit dem Fehlerflag F versehen. Auch dies lässt sich elegant in einem einzigen SQL-Statement erledigen:

*Beginn Programmcode*

```
FORALL j IN 1..T_UPDATE_F.COUNT
  UPDATE DATA_STAGE
    SET PROCESSING_IND = 'F'
  WHERE ROWID          = T_UPDATE_F(j);
```

*Ende Programmcode*

So können (im schlimmsten Fall) 35000 fehlerhafte Sätze auf einen Schlag in der Datenbank aktualisiert werden!

Will man nun noch die Entscheidung, welche der Spalten der Staging-Tabelle gegen welche Lookup-Tabelle zu validieren ist, variabel halten, muss das obige SELECT-Beispiel in dynamischem SQL (NDS) formuliert werden:

*Beginn Programmcode*

```
stm VARCHAR2(2000) := 'SELECT ROWID, ' || col_name || ' BULK COLLECT
                      INTO :B_T_RID, :B_T_VALUE
                      FROM DATA_STAGE';
```

. . .

```
EXECUTE IMMEDIATE stm USING T_RID, T_VALUE;
```

*Ende Programmcode*

Versucht man, eine Prozedur mit diesem Code in der Datenbank anzulegen, erhält man folgende Meldung:

*Beginn Programmcode*

```
PLS-00457: expressions have to be of SQL types
```

*Ende Programmcode*

Definiert man jedoch die Tabellen als Package-Variablen **T\_RID**, **T\_VALUE** und das dynamische Statement als anonymen PL/SQL-Block, kommt man endlich ans Ziel. Im folgenden Listing ist dies im Detail dargestellt:

*Beginn Programmcode*

```
CREATE OR REPLACE package PK_VALIDATION is
```

```

type t_vc1_ib_vc100 is table of VARCHAR2(1)    index by VARCHAR2(100);
type t_rowid        is table of ROWID         index by BINARY_INTEGER;
type t_vc100_ib_bi  is table of VARCHAR2(100) index by BINARY_INTEGER;

-- als Packagevariablen, um als select-into-Ziel im dynamic SQL verwendbar:
T_RID      t_rowid;      -- Tabelle für die ROWID aus DATA_STAGE
T_VALUE    t_vc100_ib_bi; -- Tabelle für den Spaltenwert aus DATA_STAGE
T_UPDATE_F t_rowid;     -- Tabelle für die ROWID der fehlerhaften Sätze
procedure pr_validate(p_table IN VARCHAR2 DEFAULT 'DATA_STAGE');
end PK_VALIDATION;
/

CREATE OR REPLACE package body PK_VALIDATION is
procedure pr_validate(p_table IN VARCHAR2 DEFAULT 'DATA_STAGE') is
begin
declare
i binary_integer;
T_LOOKUP t_vc1_ib_vc100;
stm VARCHAR2(2000);
j NUMBER;
cursor c is select AVN.*, ACN.ACN_NAME
from ALD_VALIDATIONS AVN -- Target und Name der zu validier. Spalten
,ALD_COLUMNS CAN -- alle verfügbaren Spalten der DATA_STAGE
where ACN.ACN_ID = AVN.AVN_ACN_ID;

-- private Funktion um Werte aus Wertebereichstabellen zu ermitteln
function fk_get_domain_values( p_domain IN VARCHAR2 ) return t_vc100_ib_vc1 is
rettab t_vc1_ib_vc100; -- per VARCHAR2 indizierte PL/SQL-Table
begin
for r_c in (select DVE_VALUE
from ALD_DOMAINS don
, ALD_DOMAIN_VALUES dve
where dve.DVE_DON_ID = don.DON_ID
and don.DON_NAME = p_domain)
loop
rettab( r_c.DVE_VALUE ) := 'X'; --Wert als Index der Rückgabetabelle setzen
end loop;
return( rettab );
end;

-- private Funktion, liest Werte aus der beliebigen Spalte einer bel. Tabelle
function fk_get_table_values( p_table IN VARCHAR2, p_column IN VARCHAR2 )
return t_vc1_ib_vc100 is
rettab t_vc100_ib_vc1; -- per VARCHAR2 indizierte PL/SQL-Table
stm VARCHAR2(2000);
TYPE rc is REF CURSOR;
c rc;
val VARCHAR2(100);
begin
stm := 'SELECT DISTINCT ' || p_column || ' FROM ' || p_table || ' ORDER BY 1';
open c for stm;
loop
fetch c into val;
exit when c%notfound;
rettab( val ) := 'X'; -- Wert als Index der Rückgabetabelle setzen
end loop;
close c;
return( rettab );
end;

-- Beginn Hauptprozedur
begin
for r_c in c loop --Schleife über alle zu prüfenden Spalten der DATA_STAGE
T_LOOKUP.DELETE;
PK_VALIDATION.T_UPDATE_F.DELETE;
PK_VALIDATION.T_RID.DELETE;
PK_VALIDATION.PK_VALIDATION.T_VALUE.DELETE;
j := 1;
if r_c.AVN_TARGET = 'D' then -- alle erlaubten Domain-Values laden
T_LOOKUP := fk_get_domain_values( r_c.AVN_COLUMN );
elsif r_c.AVN_TARGET = 'T' then -- aus LookUP-tabelle laden

```

```

        T_LOOKUP := fk_get_table_values( r_c.AVN_TABLE, r_c.AVN_COLUMN );
end if;
if T_LOOKUP.count > 0 then -- nur wenn die LookUp-Werte existieren
    stm := 'begin SELECT ROWID,' || r_c.ACN_NAME || --in die Packagevariablen !
          ' BULK COLLECT INTO PK_VALIDATION.T_RID, PK_VALIDATION.T_VALUE' ||
          ' FROM DATA_STAGE'; end;';

EXECUTE IMMEDIATE stm ; -- alle Werten zu der Spalte laden

for i in 1..PK_VALIDATION.T_RID.COUNT -- über alle Sätze der Spalte loopen
loop
    if NOT T_LOOKUP.EXISTS( PK_VALIDATION.T_VALUE(i) ) then --illegaler Wert!
        T_UPDATE_F(j) := PK_VALIDATION.T_RID(i);
        j := j +1;
    end if;
end loop;
-- In Validierter Tabelle den Verarbeitungsindikator setzen...
stm := 'begin FORALL j IN 1..PK_VALIDATION.T_UPDATE_F.COUNT ' ||
      ' UPDATE DATA_STAGE SET PROCESSING_IND =''F'' ' ||
      ' WHERE ROWID=PK_VALIDATION.T_UPDATE_F(j); end;';
execute immediate stm;
end if; -- von LOOKUP_T.count>0
end loop; -- alle Spalten
end;
end pr_validate;
end PK_VALIDATION;
/

```

*Ende Programmcode*

In Abbildung 1 ist noch ein Beispiel für den Aufruf der Validierung von der Kommandozeile dargestellt. Für insgesamt 128.000 zu validierende Werte und für das Aktualisieren von 3359 fehlerhaften Sätzen wurde gerade eine knappe Sekunde benötigt! (Manche Sätze hatten mehr als eine fehlerhafte Spalte, deshalb ergibt sich  $9 + 3353 + 6 = 3359$ ):

```

c:\ Auswählen 10G_sqlplus
SQL> select processing_ind, count(*) from aladin_data_stage group by processing_ind;
P  COUNT(*)
L      32318
Elapsed: 00:00:00.04
SQL> exec pk_validation.pr_validate;
-----
Zu validierende Spalte:CALLABLE
Validierungsziel D>omain / T>abelle:D
Domain bzw. LookUp-Spalte:CALLABLE
Anzahl Referenzwerte:2
NDS-Statement:
begin SELECT ROWID,CALLABLE BULK COLLECT INTO PK_VALIDATION.T_RID,
PK_VALIDATION.T_VALUE FROM DATA_STAGE; end;
Validierungsfehler:          9/32318
-----
Zu validierende Spalte:CURRENCY
Validierungsziel D>omain / T>abelle:D
Domain bzw. LookUp-Spalte:CURRENCY
Anzahl Referenzwerte:27
NDS-Statement:
begin SELECT ROWID,CURRENCY BULK COLLECT INTO PK_VALIDATION.T_RID,
PK_VALIDATION.T_VALUE FROM DATA_STAGE; end;
Validierungsfehler:        3353/32318
-----
Zu validierende Spalte:INSTTYPE
Validierungsziel D>omain / T>abelle:T
Domain bzw. LookUp-Spalte:PRO_NAME
LookUp-Tabelle:ALD_PRODUCTS
Anzahl Referenzwerte:26
NDS-Statement:
begin SELECT ROWID,INSTTYPE BULK COLLECT INTO PK_VALIDATION.T_RID,
PK_VALIDATION.T_VALUE FROM DATA_STAGE; end;
Validierungsfehler:        6/32318
-----
Zu validierende Spalte:INT_EXT_TYPE
Validierungsziel D>omain / T>abelle:D
Domain bzw. LookUp-Spalte:INT_EXT_TYPE
Anzahl Referenzwerte:2
NDS-Statement:
begin SELECT ROWID,INT_EXT_TYPE BULK COLLECT INTO PK_VALIDATION.T_RID,
PK_VALIDATION.T_VALUE FROM DATA_STAGE; end;
Validierungsfehler:          0/32318
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.96
SQL> select processing_ind, count(*) from aladin_data_stage group by processing_ind;
P  COUNT(*)
F      3359
L     28959
Elapsed: 00:00:00.04
SQL> -

```

Abbildung 1: Beispiel Aufruf Validierung von Kommandozeile

### Fazit

Bei spezifischen Aufgabenstellungen können durch Verwendung von BULK SQL und VARCHAR2-indizierten Tabellen enorme Performancegewinne erzielt werden.

### Kontakt

Klaus Friemelt

klaus.friemelt@mt-ag.com