Vergleich zweier Datenbanken

Analyse und Implementierung in CLIPPER

Karel Štípek

ftp://pcnews.at/dsk/5x/54x/541/datbank/

Verwendung des Programms

Es kommt ziemlich häufig vor, daß man den Inhalt von zwei Dateien vergleichen muß; der Vorgang wird auch „Synchronisieren“ genannt.

Es gibt zwei wichtige Anwendungen für das Synchronisieren:

- Verfolgen von Änderungen bei einer wiederholten regelmäßigen Verarbeitung (neue, geänderte, stornierte Einträge)
- Testen der Auswirkung von Bearbeitungen durch ein Programm auf die Daten.

Gehen wir von zwei Dateien **DB1** und **DB2** aus. Die Abkürzung entspricht dem in dBASE und CLIPPER üblichen Begriff „Datenbank“, die eigentlich in neueren Systemen als „Tabelle“ bezeichnet wird. Der präsentierte Algorithmus ist zwar ganz allgemein anwendbar, die konkrete Implementierung erfolgt aber im CLIPPER.

Die meisten Tabellen haben einen eindeutigen Schlüsselwert definiert, der verglichen wird.

Drei Informationsbereiche sind bei dem Vergleich wichtig:

1. Welche Schlüsselwerte sind in DB1 vorhanden, fehlen aber in der DB2
2. das gleiche umgekehrt
3. in Datensätzen, die den gleichen Schlüssel in beiden Tabellen haben, muß man den Inhalt aller restlichen Felder vergleichen und die eventuellen Unterschiede auswerten.

Im vorgelegten Programm wird eine Ergebnistabelle erstellt. Sie hat die gleiche Struktur wie beide verglichenen Tabellen, am Anfang des Satzes wird die Tabellenbezeichnung und Recordnummer hinzugefügt. Die Ergebnistabelle kann noch vor dem Beenden des Programms bearbeitet und später beliebig verwendet werden.

Beispiel

Folgende zwei Tabellen werden mit dem Schlüsselausdruck „**PLZ**“ verglichen.

DB1

Land	Plz	Ort	Vorwahl
A	1050	Wien-Margareten	01
A	1210	Wien Floridsdorf	0221
A	2500	Baden bei Wien	02252
CZ	11000	Praha - Zentrum	02
CZ	25601	Benesov	0301
CZ	25801	Vlasim	0303
D	51375	Leverkusen	0214
D	64291	Darmstadt	06151

DB2

Land	Plz	Ort	Vorwahl
A	1010	Wien-Altstadt	0221
A	1210	Wien Floridsdorf	0221
A	2500	Baden bei Wien	02252
CZ	11000	Praha - Zentrum	2
CZ	25601	Benesov	0301
D	64291	Darstadt	06151

Ergebnistabelle

bezeichnet die Sätze, deren Schlüsselwerte in der ersten Tabelle fehlen, mit DB2 und umgekehrt.

Die nacheinanderfolgenden Datensätze mit dem Eintrag DB1 und DB2 enthalten Datensätze mit dem gleichen Schlüssel, aber mit mindestens einem unterschiedlichen Feld.

Durch Einfügen leerer Datensätze wird die Tabelle übersichtlicher.

Db	Recno	Land	Plz	Ort	Vorwahl
DB2	1	A	1010	Wien-Altstadt	0221
DB1	1	A	1050	Wien-Margareten	01
DB1	4	CZ	11000	Praha - Zentrum	02
DB2	4	CZ	11000	Praha - Zentrum	2
DB1	6	CZ	25801	Vlasim	0303
DB1	7	D	51375	Leverkusen	0214
DB1	8	D	64291	Darmstadt	06151
DB2	6	D	64291	Darstadt	06151

Analyse des Programms

Algorithmus des Dateienvergleichs

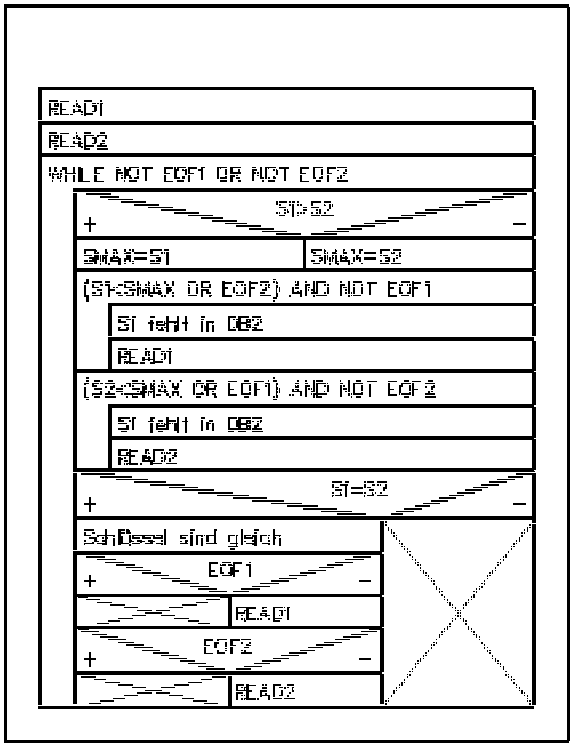
Voraussetzung: Beide Dateien sind aufsteigend sortiert.

Das Programm fängt damit an, daß aus jeder Datei der erste Satz gelesen wird - Anweisungen **read1, read2**.

Die Anzahl der Sätze in beiden Dateien kann unterschiedlich sein, es müssen aber jedenfalls alle Sätze aus beiden Dateien bearbeitet werden. Ende der Datei wird mit **eof1**, bzw. **eof2** bezeichnet.

Schlüsselwerte in den Dateien werden mit **S1**, bzw. **S2** bezeichnet. Damit die fehlenden Schlüssel in einer Datei ausgewertet werden können, wird am Anfang der Hauptschleife der größere Schlüssel in die Variable **SMAX** gespeichert.

Wenn z.B. **SMAX = S1**, dann sind die Werte, die in der 2. Datei kleiner als **SMAX**, in der ersten Datei fehlen und umgekehrt. Deswegen werden beide Dateien bis zu **SMAX** bearbeitet.



Es muß auch berücksichtigt werden, daß die Dateien unterschiedliche Anzahl von Datensätzen haben können, jede Datei kann an einer beliebigen Position des Satzzeigers am Ende der Datei sein. Die Sätze, die in der anderen Datei übrig bleiben, müssen als nicht gefundene Schlüssel in der kürzeren Datei behandelt werden. Passen Sie bitte auf die Klammern in diesen Ausdrücken auf.

Wenn nach diesen beiden „Synchronisierungsschleifen“ die Schlüsselwerte in beiden Dateien gleich sind, wird der Block „Schlüssel gleich“ durchgeführt. Anschließend müssen neue Sätze gelesen werden, aber nur in der Datei, die noch nicht am Ende ist.

Implementierung in CLIPPER

Das Programm vergleicht zwei dBASE-kompatible Dateien, die die gleiche Struktur haben.

Zwei Methoden sind möglich:

- Wenn ein Schlüsselausdruck angegeben wird, werden beide Tabellen indiziert und der oben beschriebene Algorithmus angewendet. In den Sätzen mit gleichen Schlüsseln werden die Inhalte aller Felder verglichen.
- Wenn kein Schlüsselausdruck angegeben wird, werden beide Tabellen sequentiell bearbeitet und in jedem Satz die Inhalte aller Felder verglichen.

Das Programm kann mit vier Parametern aufgerufen werden.

- Die ersten zwei müssen angegeben werden und geben die Namen der verglichenen Tabellen an. In der Ergebnistabelle wird dann der Satz aus der ersten Tabelle mit „DB1“, aus der zweiten mit „DB2“ bezeichnet.
- Der dritte Parameter ist der Schlüsselausdruck. Eingabe dieses Parameters bestimmt die Methode, wie die Tabellen verglichen werden.
- Der letzte Parameter ist der Name der Ergebnistabelle. Wird kein angegeben, wird die Tabelle COMPDB.DBF benannt.

```
function main(dbname1, dbname2, keyexpr, compdbname)
*************************************************
local str1:={}, str2:={} // Strukturen der
                         // verglichenen Tabellen
local compstr:={}        // Struktur der Ergebnistabelle
local keyerror:=.f.      // Schlüssel nicht gefunden
local maxkey             // der aktuelle maximale
Schlüssel
local i:=0
local ok:=.f.
```

Wenn nicht mindestens 2 Parameter angegeben werden, wird ein Hilfetext angezeigt und das Programm beendet.

```
if dbname2 == nil
  ? “Syntax: COMPDB DBname1 DBname2 ” + :
    “[Schlüsselausdruck] [ErgebnisDBname]”
  quit
endif
```

Die gelöschten Sätze werden ignoriert.

Beide Tabellen werden mit Aliasnamen DB1 und DB2 geöffnet. Die Funktion DBSTRUCT() speichert die Tabellenstruktur in einem Array. In den Arrays werden die Namen und Typen aller Felder verglichen.
Werden Unterschiede gefunden, wird das Programm nach einer Fehlermeldung unterbrochen.

```
set deleted on
use (dbname1) alias db1
use (dbname2) alias db2 new

// Strukturen in Arrays speichern und vergleichen
str1:=db1->(dbstruct())
str2:=db2->(dbstruct())
ok:= (len(str1) == len(str2))
i:= 1
while ok .and. (i<=len(str1)) // Feldnamen und -typen
                              // vergleichen
  ok:=str1[i][1]==str2[i][1] .and. str1[i][2]==str2[i][2]
  i++
enddo
if !ok
  ? “Tabellenstrukturen sind nicht gleich”
  quit
endif
```

Wenn ein Schlüsselausdruck beim Programmaufruf angegeben wird, werden beide Tabellen indiziert.

```
// Tabellen bei Bedarf indizieren
if keyexpr <> nil
  select db1
  index on &keyexpr to db1.ntx
  select db2
  index on &keyexpr to db2.ntx
endif
```

Die Ergebnistabelle wird erstellt. Die vorhandene Struktur einer der verglichenen Tabellen wird am Anfang um zwei Felder erweitert. Im ersten Feld wird die Bezeichnung „DB1“ oder „DB2“ gespeichert, in das zweite Feld schreibe ich die Satznummer zur eindeutigen Identifikation des Satzes.

Die mit `DBCREATE()` erstellte Tabelle wird sofort mit dem Aliasnamen `COMPDB` geöffnet.

```
// Ergebnistabelle anlegen
asize (compstr, len(str1))
acopy (str1, compstr)
asize (compstr,len(compstr)+2)
ains (compstr, 1)
compstr[1]:= {“DB”, “C”, 3, 0}
ains (compstr, 2)
compstr[2]:= {“RECNO”, “N”, 6, 0}
compdbname:= if(compdbname == nil, “COMPDB.DBF”,
compdbname)
dbcreate (compdbname, compstr)
use (compdbname) alias compdb new
```

Der weitere Ablauf des Programms entspricht genau dem aufgezeichnetem Ablaufdiagramm wenn ein Schlüsselausdruck angegeben wurde .

Es gibt dBASE-spezifische Techniken, z.B. die ersten readAnweisungen im Diagramm sind überflüssig, weil die ersten Sätze der Tabellen sofort nach dem Öffnen zur Verfügung stehen und zum nächsten Satz springt man mit SKIP, usw.

Der nächste Absatz enthält die Definition der Hauptschleife und die Definition des größeren Schlüssels.

Der Typ der Variablen MAXKEY wird je nach dem Typ des Schlüsselausdrucks erst beim Programmablauf definiert.

```
// alle Datensätze in beiden Tabellen bearbeiten
do while (!db1->(eof())) .or. (!db2->(eof()))
  if keyexpr<>nil                // Tabellen indiziert
    maxkey:= if(db1->(&keyexpr)  db2->(&keyexpr),;
                db1->(&keyexpr), db2->(&keyexpr))
```

Fehlende Schlüssel in der zweiten Tabelle werden gesucht. Für jeden wird ein neuer Satz mit der Bezeichnung „DB1“ in die Ergebnistabelle hinzugefügt und mit der Satznummer und allen Feldern gefüllt.

Nach der ganzen Gruppe, die in dieser Schleife ausgewertet wird, kommt eine leere Trennungszeile, damit die Ergebnistabelle übersichtlicher wird.

```
    // Fehlende Schlüssel in DB2
    keyerror:= .f.
    do while ((db1->(&keyexpr)<maxkey) .or. db2->(eof()));
             .and. !db1->(eof())
      keyerror:= .t.
      append blank
      fieldput(1, “DB1")
      fieldput(2, db1->(recno()))
      for i:= 1 to len(str1)
        fieldput(i+2, db1->(fieldget(i)))
      next
      skip alias db1
    enddo
    if keyerror
      append blank        // Trennungszeile
    endif
```

Genauso werden die Schlüssel, die in DB1 fehlen, durch Bearbeiten der DB2 bis zum maximalen aktuellen Schlüssel gefunden.

```
    // Fehlende Schlüssel in DB1
    keyerror:= .f.
    do while ((db2->(&keyexpr)<maxkey) .or. db1->(eof()));
             .and. !db2->(eof())
      keyerror:= .t.
      append blank
      fieldput(1, “DB2")
      fieldput(2, db2->(recno()))
      for i:= 1 to len(str1)
        fieldput(i+2, db2->(fieldget(i)))
      next
      skip alias db2
    enddo
    if keyerror
      append blank        // Trennungszeile
    endif
```

Wenn beide Schlüssel gleich sind, vergleicht das Programm mit eigener Funktion COMPFIELDS() die Inhalte aller Felder.

Danach wird zum nächsten Satz gesprungen, aber nur, wenn die Tabelle noch nicht am Ende ist.

```
    // bei gleichen Schlüsseln alle Felder vergleichen
    if (db1->(&keyexpr) == db2->(&keyexpr))
      compfields(str1)
      //--------------
      if (!db1->(eof()))
        skip alias db1
      endif
      if (!db2->(eof()))
        skip alias db2
      endif
    endif
```

Wenn kein Schlüsselausdruck angegeben wurde, werden die Tabellen nicht indiziert und es können auch keine fehlenden Schlüssel ausgewertet werden.

Man kann dann nur die Dateien durchlesen und alle Felder in allen Sätzen vergleichen. Diese Methode kann allerdings nicht oft zu sinnvollen Ergebnissen führen.

```
  else          // nicht indiziert, nur Felder vergleichen
    compfields(str1)
    //--------------
    if (!db1->(eof()))
      skip alias db1
    endif
    if (!db2->(eof()))
      skip alias db2
    endif
  endif
```

Nachdem die Hauptschleife beendet wird, wird BROWSE() auf die Ergebnistabelle aufgerufen, man kann also die Ergebnisse sofort sehen. Leere Tabelle bedeutet natürlich, daß beide verglichene Tabellen gleich sind.

Alle Tabellen werden geschlossen, die Indexdateien gelöscht und das Programm beendet.

```
enddo

go top
browse()          // Ergebnistabelle anschauen
close databases
erase db1.ntx
erase db2.ntx
return nil
```

Die Funktion COMPFIELDS()

Vergleicht alle Felder der aktuellen Sätze von beiden Tabellen. Die Struktur der Tabelle wird als Parameter übergeben.

Wenn in einem beliebigen Feld unterschiedlich Werte gefunden werden, wird der Vergleich sofort abgebrochen und beide Sätze untereinander mit einer Trennungszeile in die Ergebnistabelle hinzugefügt.

```
function compfields(str1)
/*************************
Vergleicht alle Felder eines Satzes
*/
local flderror:= .f.  // ungleiche Felder gefunden
local i:= 1

do while (i<=len(str1)) .and. !flderror
  flderror:= db1->(fieldget(i)) != db2->(fieldget(i))
  i++
enddo

// Wenn Felder ungleich sind, beide Sätze ausgeben
if flderror
  append blank
  fieldput(1, “DB1")
  fieldput(2, db1->(recno()))
  for i:= 1 to len(str1)
    fieldput(i+2, db1->(fieldget(i)))
  next
  append blank
  fieldput(1, “DB2")
  fieldput(2, db2->(recno()))
  for i:= 1 to len(str1)
    fieldput(i+2, db2->(fieldget(i)))
  next
  append blank        // Trennungszeile
endif
return nil
```

Kompilieren und Linken

Das Programm verwendet nur standardmäßige CLIPPER-Bibliotheken und ist mit jeder CLIPPER-Version ab 5.1 kompilierbar. Die von CA gelieferte Batchdatei CL.BAT erledigt die Aufgabe problemlos.