Was passt in ein varchar2(10) Feld?

Autor: Martin Busik, WLP Systems GmbH

Die Verwendung von UTF-8 als Datenbankzeichensatz kann Komplikationen mit sich bringen. Manche der Probleme sind nicht offensichtlich und werden deshalb gerne als „Voodoo“ bezeichnet. Dieser Artikel geht auf die Ursachen ein und zeigt, wie man die Probleme vermeidet.

Lange Zeit war die Frage, welchen Zeichensatz eine Datenbank verwendet, nachrangig: Solange Umlaute und das Eszett unterstützt wurden, war alles in Ordnung. Dann kamen die Euro-Einführung und der flächendeckende Einsatz von unicode-basierten Client-Technologien (Windows NT, Java). Unicode-basierte Zeichensätze wurden vermehrt in der Datenbank eingesetzt, auch dann, wenn keine explizite Globalisierungsanforderung dahinter stand.

Ab der Oracle Version 8i stand der Zeichensatz UTF-8 zur Verfügung. Manch einer, der gleichzeitig mit dem Versionswechsel auch den Datenbankzeichensatz geändert hatte, erlebte eine böse Überraschung, als sich seine Exportdatei in die UTF-8 Datenbank nicht mehr importieren ließ. Der Fehler ORA-1899 (value too large for column) trat auf. Er entsteht durch das Aneinandertreffen von zwei Aspekten: UTF-8 und Byte-Semantik für Größenangaben. UTF-8 verwendet eine Codierung mit variabler Länge. Standard-ASCII-Zeichen belegen 1 Byte, deutsche Umlaute belegen 2 Byte und das Euro-Zeichen deren 3. Die Angabe varchar2(10) bezieht sich (unter Oracle 8i) auf Byte, nicht auf Zeichen!

Damit also in ein Datenbankfeld (einer UTF-8 Datenbank) 10 ü-s eintragen werden können, muss dieses mit varchar2(20) deklariert sein. Sollen 10 Euro-Zeichen hineinpassen, muss varchar2(30) in der Deklaration stehen.

In der Praxis hatte dies zur Folge, dass Felder, die Freitexte enthalten sollten, in ihrer Größe mit Faktor 3 angelegt wurden. Die Folgen waren offensichtlich: Die Prüfungen, die sonst die Datenbank gemacht hätte, mussten von Hand ausprogrammiert werden. Problematisch war es um Felder bestellt, die bereits am Größenlimit für Varchar2 angelangt sind und nicht weiter vergrößert werden konnten.

Um Abhilfe zu schaffen, hat Oracle die Größenangabe bei Felddeklarationen um deren Semantik erweitert. Seit Version 9i kann man varchar2-Felder wie folgt deklarieren: varchar2(10 byte) bzw. varchar2(10 char). Damit sollte eine zeichensatzunabhängige Deklaration ermöglicht werden. Während Sie diese Zeilen lesen, stellen Sie sich vielleicht die Frage, was wohl der Defaultwert sein mag: Bytes oder Char?

Die Antwort lautet: Es kommt drauf an. Der Defaultwert kann instanz- oder sessionweit eingestellt - und somit leider auch verstellt - werden . Was ihr System aktuell nutzt, können Sie mit den folgenden Befehlen ermitteln:

```
SHOW PARAMETER nls_length_semantics
```

In der aktuellen Session können Sie den Parameter mittels:

```
alter session set nls_length_semantics =
char;
```

verändern. Instanzweit geht dies mittels:

```
alter system set nls_length_semantics =
char scope=both;
```

Mit den oben genannten Befehlen stellen Sie sicher, dass zukünftige DDL-Statements das gewünschte Ergebnis liefern. Wie aber ermitteln Sie, wie die bereits vorhandenen Felder angelegt wurden?

Lassen Sie uns folgenden Befehl absetzen:

```
CREATE TABLE utf8test (
 F1 varchar2(10),
 F2 varchar2(10 byte),
 F3 varchar2(10 char)
);
```

Wir erwarten, dass ein DESC utf8test die ensprechenden Angaben liefert. Nachfolgend eine Ausgabe von einer 9i Datenbank (und einem 9i Client!):

```
SQL> desc utf8test
 Name     Type
 -------- ----------------------------
 F1       VARCHAR2(10)
 F2       VARCHAR2(10)
 F3       VARCHAR2(10 CHAR)
```

Die Ausgabe von DESC ist also mit einem 9i-Client richtig (Default-Semantik war Byte). Was gibt ein 8i-Client aus? – Sie hätten es richtig erkannt: der 8i Client gibt alle drei Spalten als varchar2(10) aus. Keine Unterscheidung nach Byte und Char.

Wenn Sie auf die Data Dictionary Views direkt zugreifen, um die Spaltendeklaration zu ermitteln, sollten sie die Spalten char_used und char_length berücksichtigen:

```
SQL> select column_name, data_length,
char_length, char_used
from user_tab_columns
where table_name = 'UTF8TEST';

COLUMN_NAME DATA_LEN CHAR_LEN CHAR_USED
F1          10       10       B
F2          10       10       B
F3          30       10       C
```

Neben Tabellenspalten wird der Datentyp varchar2 in PL/SQL verwendet. Wie sieht es mit der Semantik der Größe in PL/SQL aus?

Führen Sie den nachfolgenden anonymen PL/SQL Block aus, einmal mit der Byte- und einmal mit der Char-Semantik:

```
set serveroutput on
declare
  v1 utf8test.f3%TYPE;
  v2 varchar2(10);
begin
  v1 := 'ü';
  v2 := 'ü';
  for i in 1..10
  loop
    begin
       v1 := v1 || 'ü';
    exception
       when others then
         dbms_output.put_line('v1: '||i);
    end;
    begin
       v2 := v2 || 'ü';
    exception
       when others then
         dbms_output.put_line('v2: '||i);
    end;
  end loop;
end;
/
```

Der Ausgabe kann man entnehmen, dass auch in PL/SQL der Parameter nls_length_semantics berücksichtigt wird.

Die Semantik der Größenangabe kann explizit angeben werden. Ob das zu portablem und wartbaren Code führt, kann nur im Einzelfall beurteilt werden.

Der vorherige PL/SQL Code ist als anonymer Block angelegt worden, d.h. für die Compilierung und die Ausführung ist dieselbe Semantik verwendet worden. Es stellt sich die Frage, wie das Verhalten bei benannten Objekten, also z.B. Prozeduren und Funktionen, ist, bei welchen Compilierung und Ausführung typischerweise zu zwei verschiedenen Zeitpunkten erfolgen. Um das Beispiel durchzuspielen, wandeln wir den o.g. Code zu einer Prozedur ab und kompilieren sie im Byte-Semantik-Kontext:

```
alter session
 set nls_length_semantics = byte;
```

```
create or replace procedure pp_utf8test …
```

Diese Prozedur führen wir nun im Char-Semantik-Kontext aus:

```
alter session
 set nls_length_semantics = char;
execute pp_utf8test;
```

Der Ausgabe können Sie entnehmen, dass die aktuelle Einstellung von nls_length_semantics bei der Ausführung nicht berücksichtigt wird. Zum Schluss, probieren wir die folgende Befehlssequenz aus:

```
alter session
 set nls_length_semantics = char;
alter procedure pp_utf8test compile;

alter session
 set nls_length_semantics = byte;
execute pp_utf8test;
```

Für Named-PL/SQL-Code wird der Parameter nls_length_semantics offensichtlich zum Kompilierungszeitpunkt verwendet. Diese Tatsache kann zu kaum nachvollziehbaren Ergebnissen führen, insbesondere wenn man berücksichtigt, dass PL/SQL Code bei Bedarf compiliert wird. (Dies ist u.a. dann der Fall, wenn DDL Statements auf Tabellen abgesetzt werden, die im PL/SQL Code referenziert werden)

Das oben genannte Problem lässt sich mit einer instanzweiten Einstellung der Semantik beseitigen. Es gib allerdings ein Problem, für das es keine Lösung gibt: varchar2(4000) Felder. Auch wenn ein Feld als varchar2(4000 char) deklariert wird, können darin nur 4000 Byte und nicht 4000 Zeichen gespeichert werden. Bei Migrationen auf UTF-8 ist zu überprüfen, ob solche Fälle vorkommen und inwieweit der Datentyp CLOB als Alternative in Frage kommt.

■ Fazit

Mit der Char-Semantik steht ein Mittel zur Verfügung, das für die Entwicklung vom portablen Code genutzt werden sollte. Nach Möglichkeit sollte die Char-Semantik instanzweit gesetzt werden. Änderungen auf Session-Ebene sollten vermieden werden, da dies kaum nachvollziehbare Auswirkungen auf PL/SQL Code haben kann.
Für Migrationsszenarien ist die Char-Semantik unerlässlich, allerdings sollte die Nutzung von Feldern, deren Größe bereits nahe am Limit ist, sehr genau analysiert werden.

Kontakt:
Martin Busik
busik@wlp-systems.de