Datenbankanwendungsübungen

SS 2003

Wolfgang Hümmer, Maciej Suchomski
Friedrich-Alexander-Universität Erlangen-Nürnberg
IMMD VI (Lehrstuhl für Datenbanksysteme), Martensstr. 3, 91058 Erlangen

Aufgabe 6: Hinweise

In Aufgabe 6 hat der Fehlerteufel gewütet, und manches bleibt etwas unklar. Im folgenden sollen diese Probleme ausgeräumt werden. Ich danke allen aktiven Teilnehmern des Kurses, die mich mit ihren Fragen entsprechend aufgescheucht haben.

1 INSERTs in Aufgabenstellung zu a1.sql

In Aufgabe a1.sql werden zwei INSERT-Befehle angegeben, die den Trigger auslösen sollen. Dazu kommt es leider nicht, weil in beiden Befehlen die selbe `flight_id` (11268) verwendet wird, was den Primärschlüssel von `flight_intervals` verletzt. Korrekterweise müssen die Statements z.B. so aussehen:

```
insert into flight_intervals values(112698, 'AC000', 1, 8,
     to_date('01-FEB-03 07:00 PM', 'DD-MON-YY HH:MI PM'),
     to_date('01-FEB-03 08:00 PM', 'DD-MON-YY HH:MI PM'));

insert into flight_intervals values(112699, 'AC000', 3, 4,
     to_date('01-FEB-03 09:00 PM', 'DD-MON-YY HH:MI PM'),
     to_date('01-FEB-03 10:00 PM', 'DD-MON-YY HH:MI PM'));
```

Außerdem sieht man hier nochmals den Umgang mit Datumswerten (siehe Handout Aufgabe 3).

2 CREATE TYPE in a2.sql

In Aufgabe a2.sql soll zur Übergabe der Flugroute ein Nested Table (siehe Aufgabe 7) mit CREATE TYPE route AS TABLE OF NUMBER angelegt werden. Diese Typ-Definition ist nicht Bestandteil des Triggers, sondern wird z.B. direkt in SQL*Plus eingegeben. Das Ergebnis ist ein neuer Datentyp in eurem Schema. Dieser kann dann als Bestandteil einer Relation oder – wie für die Aufgabe erforderlich – als Typ eines Parameters verwendet werden.

Hints_new.fm: 03.06.03 22:04

3 Verwendung des Nested Table “Route”

Die Verwendung von Nested Tables als Teil einer Relation in SQL wird in Aufgabe 7 gezeigt. Für PL/SQL sind einige zusätzliche Zugriffsmethoden definiert, die für die Aufgabe wichtig sind. Insbesondere muss eine Schleife über den Inhalt des Nested Tables vom Typ ROUTE programmiert werden:

```
CREATE OR REPLACE PACKAGE BODY traffic_pckg AS
 PROCEDURE add_flight (..., flight_route IN ROUTE)
 AS
     ...
 BEGIN
     -- Anzahl der Elemente im Nested Table
     if(flight_route.count > 1 and flight_route.count < 6) then
          -- Position des ersten Elements in flight_route
          c := flight_route.first;
          ...
          -- ap wird der Wert des ersten Elements zugewiesen
          ap := flight_route(c);
          -- Schleife bis letzter Index erreicht
          while c < flight_route.last loop
               c := c+1;
               ...
               -- Zugriff auf Element an der Position, die in c steht
               ap := flight_route(c);
          end loop;
     end if;
 EXCEPTION
     ...
END add_flight;
```

Der an sich ungeordnete Nested Table ist indiziert. Somit kann auf einzelnen Elemente im Table direkt über den Index zugegriffen werden (`flight_route(c)`). Mit `first` und `last` erhält man die Position des ersten bzw. letzten Element, mit `count` die Kardinalität. Genaueres findet sich in Kapitel 5 des *PL/SQL User’s Guide and Reference*, http://www6.informatik.uni-erlangen.de/manuals/oracle/9.2.0/appdev.920/a96624/toc.htm.

4 `old` und `new` in Row-Triggers

Row-Trigger sind Trigger, deren Aktionsteil pro qualifizierenden Tupel der betroffenen Tabelle ausgeführt wird. Für solche Trigger existieren – je nach auslösendem DML-Befehl – die Variablen `old` und/oder `new` zur Verfügung:

- DELETE kennt nur die Variable `old`, da ja kein neuer Wert eingeführt wird, sondern nur ein alter gelöscht. Man kann sich vorstellen, dass das Datenbanksystem in einer Schleife durch die Menge der zu löschenden Tupel geht. Die Attributwerte des aktuellen Tupels stehen in `old` zur Verfügung (und es kann überprüft werden, ob dieses spezielle Tupel gelöscht werden darf).
- INSERT kennt nur die Variable `new`. Bevor also das neue Tupel eingefügt wird, kann über `new` untersucht werden, ob die Werte “passen”. `old` macht hier keinen Sinn, da ja kein alter Wert ersetzt wird.

- UPDATE kenn beide Variablen, da ein existierendes Tupel (`old`) mit neuen Werten (`new`) aktualisiert werden soll. Z.B. kann ein Vergleich zwischen diesen Werten gemacht werden, dass etwa ein Gehalt nicht übermäßig erhöht oder erniedrigt wird.

5 Ablehnen des Events durch den Trigger

Before-Trigger können potenziell das auslösende DML-Kommando ablehnen, schließlich dienen Trigger zur Wahrung der Konsistenz. Das Beispiel im Handout zu Aufgabe 6 zeigt die entsprechende PL/SQL-Anweisung:

```
raise_application_error (-20501, 'Mindestgrenze')
```

Damit wird eine benutzerdefinierte Exception geworfen und das DML-Statement abgeschmettert. Die genaue Bedeutung der Fehler-Codes (-20501) ist nicht relevant (kann natürlich im *PL/SQL User's Guide and Reference*, Abschnitt 7, *http://www6.informatik.uni-erlangen.de/manuals/oracle/9.2.0/appdev.920/a96624/toc.htm*); am besten wird dieser Code einfach beibehalten. Die Fehlermeldung ('Mindestgrenze') sollte allerdings entsprechend angepasst werden.

6 Fehler im ersten Row-Trigger

Besonders peinlich – sowohl für Oracle als auch für uns Betreuer... – ist der Trigger, der sicherstellen soll, dass mindestens ein kanadischer Flughafen im System verbleibt. Konzeptionell ist der Trigger richtig, allerdings verbietet Oracle aus unerklärlichen Gründen in einem Row-Trigger auf einer bestimmten Tabelle der Trigger-Aktion, auf die Tabelle selbst per SQL zuzugreifen! Es ist nur der Zugriff auf `old` erlaubt.

Das Ergebnis ist eine Fehlermeldung im folgenden Stil:

```
ORA-04091: table SIDMKISS.AIRPORT is mutating, trigger/function may not see it
ORA-06512: at "TEST_TRIGGER_BDR", line 3
ORA-04088: error during execution of trigger 'TEST_TRIGGER_BDR'
```

Eine sinnvolle Lösung des Problems müssen wir leider schuldig bleiben. Entweder muss eine Hilfstabelle oder die Applikationslogik herhalten.