

4. Versuch: PL/SQL, Prozeduren, Funktionen, Trigger

Verwenden Sie für die folgenden Aufgaben eine Datenbasis *ohne* Referentielle Integritätsbedingungen; d. h. lassen Sie `create.sql` einmal laufen.

Aufgabe 4.1 (Rekursive Funktion; 10 P)

Schreiben Sie eine rekursive Funktion, die die Länge eines Flusssystemes (d.h. eines Flusses sowie aller Flüsse, die direkt oder indirekt in ihn hineinfließen) berechnet.

- Berücksichtigen Sie dabei erst einmal Seen *nicht* (also ignorieren Sie, dass ein Fluss in einen See fließt, dessen Abfluss ein anderer Fluss ist).
- Überlegen Sie, wie Sie Ihre Lösung am besten auf Flüsse, die in einen See fließen, dessen Abfluss ein anderer Fluss ist, erweitern. Setzen Sie Ihre Überlegung praktisch um.

Aufgabe 4.2 (Cursor; 10 P.)

Berechnen Sie, welcher Anteil der Weltbevölkerung mindestens notwendig ist, um 50% des Welt-Bruttoinlandsproduktes zu erzeugen. Nehmen Sie dabei an, dass sich das Bruttoinlandsprodukt eines Landes gleichmäßig auf alle Einwohner verteilt. Geben Sie an, wieviele Einwohner aus welchen Ländern beteiligt sind. Verwenden Sie einen Cursor über ein geeignet gewähltes `SELECT`-Statement.

Aufgabe 4.3 (Trigger)

In Aufgabe 3.2 wurden bei der Löschung eines Flusses/Sees/Meeres die entsprechenden Tupel in `geo_River/geo_Lake/geo_Sea` ebenfalls gelöscht. Überlegen Sie sich, welche referentielle Aktion bei Löschung eines Flusses/Sees/Meeres bezüglich der Relation `located` sinnvoll ist und implementieren Sie diese durch einen Trigger.

Aufgabe 4.4 (Referentielle Aktionen über Trigger; 10 P.)

In Aufgabe 3.2 haben Sie referentielle Integritätsbedingungen und entsprechende löschende Aktionen über Constraints implementiert. Schreiben Sie nun Trigger, die bei der Umbenennung eines Landes (`Country.Code`) oder einer Provinz (`Province.Name`) diese Umbenennungen in der gesamten Datenbasis ausführen.

Aufgabe 4.5 (Flugstreckennetz; 50 P.)

Sie besitzen ein Flugzeug mit x km Reichweite.

- Schreiben Sie eine Prozedur (mit Aufrufparametern x für die Reichweite sowie $s1$ und $s2$ für zwei Städte (inklusive Namen und Code), die folgendes berechnet und in eine von Ihnen vorher erstellte Hilfstabelle schreibt:
 - Die Länge des kürzesten Reiseweges in km von $s1$ nach $s2$,
 - Die Länge desjenigen Weges in km, der die wenigsten Zwischenlandungen benötigt.Schreiben Sie alle Zwischenlandungen auf den berechneten Wegen in jeweils eine Tabelle.
- Nehmen Sie zwei Parameter hinzu:
 - $t1$: Zeit, die Sie pro Zwischenlandung benötigen,
 - v : Reisegeschwindigkeit Ihres Flugzeugesund berechnen Sie unter diesen Bedingungen jeweils den *schnellsten* Weg von $s1$ nach $s2$.
- Erweitern Sie die Berechnung um einen weiteren Parameter:
 - tc : Zollabfertigungszeit bei grenzüberschreitenden Flügen (Annahme: Bei jedem Zwischenstop in einem anderen Land sind zeitraubende Formalitäten zu erledigen)und untersuchen Sie, ob sich die berechneten Wege ändern.

Experimentieren Sie mit unterschiedlichen Start-Ziel-Kombinationen und Reichweiten, z.B. um zu sehen, wie sich die Strecke aufgrund geringerer Reichweite ändert, oder wie sich die optimale Strecke aufgrund unterschiedlicher Geschwindigkeiten ändert.

Wie verhält sich Ihr Algorithmus, wenn die Reichweite zu klein ist, um das Ziel überhaupt zu erreichen (z.B. Honolulu)?

Nehmen Sie an, dass Sie in jeder beliebigen Stadt zwischenlanden und nachtanken können. Verwenden Sie dazu die Tabelle `dbis.Distances`, die public lesbar sein sollte und zu der es bereits ein systemweites Synonym `distances` gibt. Diese wurde folgendermaßen erzeugt:

```
CREATE TABLE Distances
(City1    VARCHAR2(40),
 Country1 VARCHAR2(4),
 City2    VARCHAR2(40),
 Country2 VARCHAR2(4),
 dist     NUMBER);
CREATE INDEX Dist_dist ON Distances (dist);
CREATE INDEX Dist_City1 ON Distances (City1, Country1);
CREATE INDEX Dist_City2 ON Distances (City2, Country2);
CREATE PUBLIC SYNONYM DISTANCES FOR dbis.distances;
```

Verwenden Sie auf Tabellen, die Sie selbst erstellen, ggf. ebenfalls Indexe.

Abgabe bis 20.6.2014