Databases and Information Systems

6. Übungsblatt

(Testatwoche: 25. - 27. Mai 2010)

Einführung in Datenbanksysteme
Datenbanken für die Bioinformatik

Heinz Schweppe, Katharina Hahn

Aufgabe 1 (Anwendungsarchitektur) *5 Punkte*

Charakterisieren Sie den Unterschied zwischen den in der Vorlesung vorgestellten Client-Server Architektur und Server-seitiger Applikationslogik (*server-side computing*) Architektur. Welche Aufgaben spielt der so genannter Applikationsserver? Geben Sie die zentralen Vor- und Nachteile von *server-side computing* im Vergleich zur Verwendung eines Applikationsservers an.

Bei der Client-Server-Architektur befindet sich die Business-Logic innerhalb des Anwendungsprogramms. Dieses läuft unabhängig vom Datenbankserver.
Bei der Server-Side-Architektur ist die Business-Logic in Stored Procedures vorhanden. Stored Procedures sind dann die Schnittstellen zwischen Applikation und Datenbank. Diese sind in speziellen Host Languages geschrieben (wie z.B. PL-SQL, PL/pgSQL)
Der Applikationsserver liegt auf der mittleren Ebene, die als Middleware zwischen Client und Server steht.
Beim Server-Side-Computing liegt der Hauptvorteil bei einer hohen Effizienz. Ein Applikationsserver bringt Vorteile für die Skalierbarkeit und Kompatibilität der Systeme und sorgt für eine hohe Sicherheit.

Aufgabe 2 (VIEWS) *7 Punkte*

Erstellen Sie in ihrem Schema folgende Sichten (VIEWS) zur Mondial Datenbank. Geben Sie jeweils **gültige** SQL-Statements an.

a) Eine Sicht, die alle Grenzinformationen (*borders*) der Datenbank in symmetrischer Form enthält. D.h., wenn ein Land *c1* benachbart zu *c2* ist, dann müssen die Tupel *(c1, c2, l)* und *(c2, c1, l)* in der Sicht sein, wobei *l* die Länge der Grenze ist.

```
CREATE VIEW Grenzinformationen AS
  ( SELECT country1 AS c1, country2 AS c2, length FROM borders)
 UNION
  ( SELECT country2 AS c1, country1 AS c2, length FROM borders);
-- 610 Tupel
```

b) Eine Sicht, die die Information aller Organisationen (*organization*) enthält, die in einer Hauptstadt angesiedelt sind. Zu jeder Organisation soll zusätzlich die Anzahl der Einwohner der Stadt, in der sie ihren Sitz hat, enthalten sein.

```
CREATE OR REPLACE VIEW Organisationen AS
            ( SELECT o.name, o.abbreviation, o.city, o.country, o.province, o.established, ci.population
  FROM (Organization o  JOIN  Country c ON (o.city=c.capital)) JOIN City ci ON (o.city=ci.name) );
-- 91 Tupel
```

c) Eine Sicht, die alle Informationen aller Länder, die an Meeren liegen, mitsamt der Information über das/die Meer/e, an dem/denen sie liegen, enthält.

```
CREATE VIEW MeerLaender AS
   -- hier: eine Zeile pro Provinz.
   --Auch möglich: SELECT DISTINCT c.name, gs.sea, gs.country, c.capital...
  ( SELECT c.name, gs.sea, gs.country, gs.p2, c.capital, c.area, c.population, s.depth
       FROM (SELECT sea, country, province AS p2 FROM geo_sea) gs
       JOIN Country c ON (gs.country = c.code)
       JOIN (SELECT name AS sea_name, depth FROM Sea) s ON (s.sea_name = gs.sea));
--580 Tupel
```

Aufgabe 3 (Funktionen) *9 Punkte*

Schreiben Sie folgende Funktionen in PL/SQL (Oracle) bzw. PL/pgSQL (PostgreSQL) auf Basis des Schemas der MONDIAL Datenbank.

a) Die Funktion *DENSITY* erhält ein Land als Eingabe und gibt für dieses die Bevölkerungsdichte (Einwohner pro Fläche) zurück

```
CREATE OR REPLACE FUNCTION DENSITY (varchar) RETURNS float8 AS
  'SELECT (population/area)
  FROM Country c
  WHERE (c.name = $1)'
LANGUAGE SQL;

SELECT name, DENSITY(name) FROM Country;
-- 238 Tupel
SELECT name, DENSITY(name) FROM Country WHERE name='Albania';
--1 Tupel
```

b) Die Funktion *BIG_CITIES* erhält ein Land und einen Schwellwert als Eingabe und gibt alle Städte in dem gegebenen Land aus, deren Einwohnerzahl über dem Schwellwert liegt. Für jede solche Stadt soll der Name, die Provinz und die Anzahl der Einwohner ausgegeben werden.

```
CREATE OR REPLACE FUNCTION BIG_CITIES (varchar, int)
   RETURNS TABLE (stadt varchar, provinz varchar, einwohner int) AS
      'SELECT ci.name, ci.province, ci.population
       FROM country c JOIN city ci ON (c.code = ci.country)
       WHERE c.name = $1
          AND ci.population > $2'
LANGUAGE SQL;

SELECT name, BIG_CITIES(name,3000000) FROM Country;
-- 44 Tupel bei 3.000.000 Schwellwert
```

c) Die Funktion *DISTANCE* gibt für 2 Städte ihre Distanz in Kilometern zurück. Beachten Sie, dass eine Stadt nur in Zusammenhang mit der Provinz und dem Land, in dem sie liegt, eindeutig ist. Berechnen Sie die Distanz mit Hilfe der Haversine Formel. Informationen zur Haversine Formel finden Sie z.B. unter *http://www.movable-type.co.uk/scripts/latlong.html*.

Informationen zur Programmierung von PL/SQL bzw. PL/pgSQL finden Sie in den jeweiligen Dokumentationen der beiden Datenbanksysteme.

```
CREATE OR REPLACE FUNCTION distance(city1 IN VARCHAR, province1 IN VARCHAR, country1 IN
VARCHAR, city2 IN VARCHAR, province2 IN VARCHAR, country2 IN VARCHAR) RETURNS double
precision AS'
```

```
DECLARE
        lati1 double precision;
        lati2 double precision;
        longi1 double precision;
        longi2 double precision;
        lat double precision;
        lon double precision;
        a double precision;
        c double precision;
BEGIN
        SELECT latitude INTO lati1 FROM City WHERE name=city1 AND
                province=province1 AND country=country1;
        SELECT latitude INTO lati2 FROM City WHERE name=city2 AND
                province=province2 AND country=country2;
        SELECT longitude INTO longi1 FROM City WHERE name=city1 AND
                province=province1 AND country=country1;
        SELECT longitude INTO longi2 FROM City WHERE name=city2 AND
                province=province2 AND country=country2;
        SELECT radians(lati2-lati1) INTO lat;
        SELECT radians(longi2-longi1) INTO lon;
        SELECT sin(lat/2)*sin(lat/2)+
                cos(radians(lati1))*cos(radians(lati2))*sin(lon/2)*sin(lon/2) INTO a;
        SELECT (2*atan2(sqrt(a),sqrt(1-a))) INTO c;
RETURN (6371*c);
END;
' LANGUAGE plpgsql;

SELECT * FROM distance('Dresden','Sachsen','D','Berlin','Berlin','D');
-> 158km
```

Aufgabe 4 (Trigger) *8 Punkte*

Schreiben Sie Trigger für die Mondial Datenbank, die die folgenden Aufgaben erfüllen. Beachten Sie, dass Sie auf Grund der Rechteverwaltung die Änderungen, die die Trigger auslösen auf *esel* nicht durchführen können. Sie können daher ihre Trigger nur auf lokal installierten Datenbanken testen.

a) Schreiben Sie einen Trigger, der bei Änderung der Länderkennung (*country.code*) ausgeführt wird und die Tabelle *province* entsprechend aktualisiert.

```
CREATE OR REPLACE FUNCTION Fkt_laenderkennung() RETURNS trigger AS '
  BEGIN
  UPDATE Province p SET country = NEW.code WHERE p.country = OLD.code;
  RETURN NULL;
  END;'
  LANGUAGE 'plpgsql' VOLATILE;

--DROP TRIGGER Laenderkennung_Province ON Country ;
CREATE TRIGGER Laenderkennung_Province
  AFTER UPDATE ON Country
  FOR EACH ROW
  EXECUTE PROCEDURE Fkt_laenderkennung();

Test:
UPDATE Country SET code = 'GER3' WHERE Name = 'Germany';
SELECT * FROM Province WHERE Name = 'Bayern';
```

b) Schreiben Sie einen Trigger, der vor dem Einfügen eines Tupels (*country1, country2, length*) in die Tabelle *borders* prüft, ob ein Tupel (*country2, country1, x*) bereits existiert. Ist dies der Fall, soll wird das Einfügen des Tupels verhindert. Ansonsten soll das Tupel als (*country1, country2, length*) eingefügt werden, wenn *country1 < country2*; ansonsten wird das Tupel (*country2, country1, length*) eingefügt.

```
CREATE OR REPLACE FUNCTION border_trigger_func() RETURNS trigger AS'
DECLARE
 counter integer;
BEGIN
 SELECT COUNT(*) INTO counter
 FROM borders
 WHERE country1=NEW.country2
  AND country2=NEW.country1;

 IF counter>0 THEN
  RAISE EXCEPTION ''Grenze schon eingetragen'';

 ELSE
  IF NEW.country1>NEW.country2 THEN
        INSERT INTO borders VALUES (NEW.country2, NEW.country1, NEW.length);
        RETURN NULL;
  END IF;
END IF;
RETURN NEW;
END;
' LANGUAGE plpgsql;

CREATE TRIGGER border_trigger
BEFORE INSERT ON borders
FOR EACH ROW EXECUTE PROCEDURE border_trigger_func();
END;
```

c) Schreiben Sie einen Trigger, der beim Einfügen/Ändern von Tupeln in die/der Relation *EthnicGroup* prüft, ob die Summe des Attributs *percentage* pro Land unter 100% bleibt. Wenn die Summe mit dem geänderten Tupel bzw. dem neu eingefügten Tupel größer als 100% ist, soll die Änderung bzw. das Einfügen abgelehnt werden und eine entsprechende Meldung ausgegeben werden.

```
CREATE OR REPLACE FUNCTION ethnic_func() RETURNS trigger AS'
DECLARE
summe double precision;
BEGIN
SELECT SUM(percentage) INTO summe FROM EthnicGroup WHERE country=NEW.country
AND name!=NEW.name;
IF (summe+NEW.percentage)>100 THEN
RAISE EXCEPTION ''zuviel!'';
END IF;
RETURN NEW;
END;
' LANGUAGE plpgsql;

CREATE TRIGGER ethnic
BEFORE INSERT OR UPDATE ON EthnicGroup
FOR EACH ROW EXECUTE PROCEDURE ethnic_func();
END;
```