DBS SoSe 2013, Di. 8-10

Projektdokumentation: Iteratation No. 3

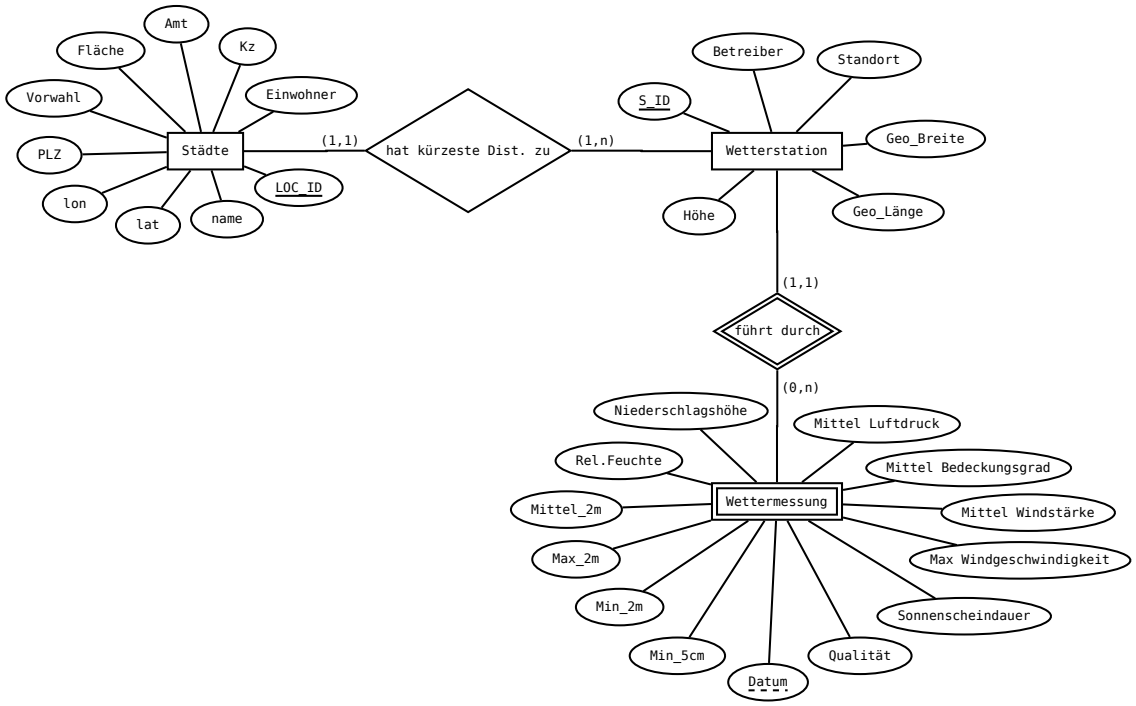
Christoph van Heteren-Frese (Matr.-Nr.: 4465677),
Sven Wildermann (Matr.-Nr.: 4567553)

1 Rahmenbedingungen

- DBS: PostgreSQL
- JavaServer Pages (JSP): Apache Tomcat 7

2 Modellierung

Wir modellieren unser Datenbankschema wie folgt:
Da wir davon ausgehen, dass nicht regelmäßig neue Wetterstationen entstehen oder abgebaut werden, haben wir die Relation "hat kürzeste Distanz zu" gebildet. Diese muss aktualisiert werden, sobald sich die Wetterstationen ändern. Wenn eine neue Stadt (bzw. "location") hinzu kommt oder entfernt wird, reicht es entsprechend einen Eintrag hinzuzufügen bzw. zu entfernen.
Da die Wetterdaten über einen Schlüssel eindeutig mit den Wetterstationen verknüpft sind, haben wir hier eine Relation "führt durch". Im Endeffekt wird hier jede Verknüpfung zwischen einer Wettermessung und einem Standort hergestellt. Aktualisieren sich die Wettermessungen, so muss auch diese Relation aktualisiert werden.



3 Herkunft der Daten

Die Daten der Städte stammen aus der frei zugänglichen Datenbank GeoDB. Die Wetterdaten werden von der Uni Bayreuth zur Verfügung gestellt.

4 Erzeugen der Relationen/Tabellen

Die erzeugten Relationen weichen teilweise von den in [1] dokumentierten Schemata ab.

Beispiel: geodb_intdata

```
1  CREATE TABLE geodb_intdata (
2      loc_id integer,
3      int_type integer,
4      int_val integer,
5      valid_since date,
```

```
    date_type_since integer,
    valid_until date,
    date_type_until integer
);
```

Beispiel: staedte

```
CREATE TABLE staedte (
  id serial NOT NULL PRIMARY KEY,
  plz VARCHAR ( 10 ) NOT NULL ,
  name VARCHAR ( 255 ) NOT NULL ,
  vorwahl text,
  einwohner integer,
  flaeche float,
  amt text,
  kz varchar (3),
  lat float,
  lon float
);
```

5 Datenimport

Zeilen, die der JOIN-Bedingung nicht genügen, werden durch 'OUTER LEFT JOIN' nur dann in die Zieltabelle eingefügt, wenn keine weiteren Bedingungen dies verhindern. Letzteres ist genau genau dann der Fall, wenn der gewünschte Texttyp im WHERE-Statement 'selektiert' wird. Daher muss dies bereits in der Joinbedingung geschehen.

Beispiel: Städte-Relation

```
INSERT INTO staedte (plz, name, vorwahl, einwohner, flaeche, amt, kz, lat, lon)
  SELECT plz.text_val1 as plz, name.text_val1 as name, vorwahl.text_val1 as vorwahl,
    einw.int_val as einwohner, flaeche.float_val as flaeche, amt.text_val1 as amt,
    kz.text_val1 as kz, coord.lat,coord.lon
  FROM geodb_textdata plz
  LEFT OUTER JOIN geodb_textdata name
    ON plz.loc_id = name.loc_id
  LEFT OUTER JOIN geodb_textdata vorwahl
    ON (plz.loc_id = vorwahl.loc_id AND vorwahl.text_type = 500400000)
  LEFT OUTER JOIN geodb_floatdata flaeche
    ON (plz.loc_id = flaeche.loc_id AND flaeche.float_type = 610000000)
  LEFT OUTER JOIN geodb_intdata einw
    ON (plz.loc_id = einw.loc_id AND einw.int_type = 600700000)
  LEFT OUTER JOIN geodb_textdata amt
    ON (plz.loc_id = amt.loc_id AND amt.text_type = 500700000)
  LEFT OUTER JOIN geodb_textdata kz
    ON (plz.loc_id = kz.loc_id AND kz.text_type = 500500000)
  LEFT OUTER JOIN geodb_locations gl
    ON (plz.loc_id = gl.loc_id AND gl.loc_type IN (100600000 ,100700000))
  LEFT OUTER JOIN geodb_coordinates coord
    ON plz.loc_id = coord.loc_id
  WHERE plz.text_type = 500300000
    AND name.text_type = 500100000 ORDER by name, plz;
```

6 Berechnung der Relation "hat kürzeste Distanz zu"

Für die Berechnung der Relation "hat kürzteste Distanz zu", die zwischen den Städten und den Wetterstationen vorliegt, wurden eigene SQL-Funktionen implementiert.

Die erste Funktion berechnet den Abstand zwischen zwei Koordinaten in km auf einer Kugel, so dass nicht nur die Distanz innerhalb Deutschlands korrekt berchnet wird, sondern auch die Entfernungen zwischen zwei weiter von einander entfernten Punkten.

```
CREATE OR REPLACE FUNCTION distance (Lat1 IN double precision,
                                     Lon1 IN double precision,
                                     Lat2 IN double precision,
                                     Lon2 IN double precision,
                                     Radius IN double precision DEFAULT 6387.7)
                    RETURNS double precision AS '

DECLARE
    DEGTORAD double precision := 57.29577951;
BEGIN
    RETURN(COALESCE(Radius,0) * ACOS((sin(COALESCE(Lat1,0) / DEGTORAD) *
    SIN(COALESCE(Lat2,0) / DEGTORAD)) +
        (COS(COALESCE(Lat1,0) / DEGTORAD) * COS(COALESCE(Lat2,0) / DEGTORAD) *
         COS(COALESCE(Lon2,0) / DEGTORAD - COALESCE(Lon1,0)/ DEGTORAD))));
END;'
LANGUAGE 'plpgsql';
```

Die zweite Funktion iteriert die Funktion distance auf allen Städten zu allen Wetterstationen, nimmt nur den kürzesten Eintrag einer Stadt zu einer Wetterstation und fügt diesen Eintrag in die Tabelle shortestdistance hinzu. Wenn diese Berechnung für alle Datensätze erneut durchgeführt werden muss, muss zuerst geprüft werden welchen Index der letzte Datensatz in der Relation städte hat (um diesen in der Funktion bei Abweichung zu ersetzen) und die Relation shortestdistance muss vorher geleert werden.

```
CREATE OR REPLACE FUNCTION shortest_distance() RETURNS VOID AS '

DECLARE
BEGIN

  FOR i in 1..61832 LOOP
  -- for all the data
    INSERT INTO shortest_distance (select wetterstation.s_id, wetterstation.standort,
    locations.id,locations.name, "distance"(locations.lat,locations.lon,
    wetterstation.geo_breite,wetterstation.geo_laenge)
    as Distanz from locations, wetterstation where
    locations.id=i ORDER BY Distanz LIMIT 1);

  END LOOP;


END;'
LANGUAGE 'plpgsql';
```

Literatur

[1] OpenGeoDb – Dokumentation. URL `http://opengeodb.org/wiki/Datenbank\_der\_OpengeoDB`.