JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

GK10.1 "Distributed Database"

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 0.1
Begonnen am 1.3.2018
Beendet am 8. März 2018

Note:
Betreuer: MICHT

Inhaltsverzeichnis

1 Einführung

Diese Übung soll helfen die Grundlagen und Funktionsweise von verteilten Datenbanken und deren Entwurf zu verstehen. Das Thema wird mit dem Entwurf von verteilten Datenbanken begonnen und wird weiters mit der Umsetzung einer verteilten Datenbank auf einem lokalen Rechner umgesetzt.

1.1 Ziele

Das Ziel dieser Übung ist eine bestehende lokale Datenbank und deren DB Modell zu analysieren. Danach soll ein Konzept erstellt werden, um diese lokale Datenbank in eine verteilte Datenbank zu transformieren. Im ersten Schritt soll die Verteilung nur lokal anhand von DB Schemata durchgeführt werden.

1.2 Voraussetzungen

- Grundlagen von verteilten Datenbanken
- Installation eines DBMS (Postgres, MySQL)
- SQL Kenntnisse (Laden einer Datenbank, SELECT)

1.3 Aufgabenstellung

Unter Verwendung der Sample Database "Dell DVD Storeßoll eine lokale Datenbank in eine verteilte Datenbank transferiert werden. Mit dieser Aufgabe soll die Fragmentierung einer Datenbank durchgeführt werden. Basierend auf einer Tabelle/View des DVD Stores sollen folgende Fragmentierungsarten umgesetzt werden:

- horizontale Fragmentierung nach mindestens 2 Kriterien
- vertikale Fragmentierung
- kombinierte Fragmentierung

Die Fragmente sollen jeweils in einem Schema mit der Bezeichnung

- Schema horizontal
- Schema vertical
- Schema combination

in Tabellen mit sinnvollen Namen gespeichert werden. Die Fragmentierung soll sinnvoll unter selbst definierten Annahmen spezifiziert werden.

Dokumentieren Sie Arbeitsschritte und die Definition Deiner Fragmente in einem Abgabeprotokoll.

Im Anschluss soll zu jeder Fragmentierungsart ein SELECT Statement zum Sammeln aller Daten entworfen werden. Dabei soll gezeigt werden, dass durch die Verteilung keine Datensätze verloren gegangen sind. Verwenden Sie dazu SSELECT count(*) FROM ...."

- Anzahl der Datensätze vor der Fragmentierung
- Anzahl der Datensätze der einzelnen Fragmente
- Anzahl der Datensätze aus dem SELECT statement, das alle Daten wieder zusammenfügt

2 Ergebnisse

2.1 Datenbank

Es handelt sich hierbei um Datenbank welche alle möglichen Daten in einem DVD-Store speichert. Es ist eine sehr große Datenbank, mit sehr vielen Daten, wobei sensitive Daten (Name, Email, Telefonnummer) mit einem Hash zensiert wurden.

2.1.1 Einspielen

Der erste Schritt war es einer Docker-Instanz anzulegen, auf welcher die Datenbank dann eingespielt wird. Es wurde eine simple postgres-Docker Instanz erstellt mit:

```
docker run --name psql -v pgdata -d postgres
```

Wobei mit `-v` eine docker Volume ausgewählt wird, welche mit:

```
docker volume create pgdata
```

erstellt wurde. Dies garantiert eine persistente Datenbank um Änderungen gespeichert zu halten.

Nun wurde ein Nutzer namens **ds2** mit dem Passwort **ds2** angelegt, damit sich das Script, welches später ausgeführt wird, an die Datenbank anbinden kann und den **ds2** User verwendet um die Datenbank einzuladen.

Anschließend wurde folgendes Script ausgeführt, in welchem bereits die Anbindung für die Docker-Instanz angegeben wurde:

```
REM pgsqlds2_create_all.sh
REM start in ./ds2/pgsqlds2
SET CONNSTR=-h localhost -p 5432
REM If using vFabrid Data Director vPostgres then connection string will look like this
REM CONNSTR="-h {cc25670-1854-4476-9764-c384759f93d}.10.10.10.10 -p 5432"
SET DBNAME=ds2
SET SYSDBA=ds2
SET PGPASSWORD=ds2
SET createlang plpgsql ds2
cd build
REM Assumes DB and SYSDBA are already created
REM If building on vFabric Data Director vPostgres then you will need to comment out
REM     pgsqlds2_create_db.sql line becuase the DB is already created
psql %CONNSTR% -U %SYSDBA% -d postgres < pgsqlds2_create_db.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_delete_all.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_create_tbl.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_create_sp.sql
cd ../load/cust
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_cust.sql
cd ../orders
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_orders.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_orderlines.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_cust_hist.sql
cd ../prod
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_prod.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_load_inv.sql
cd ../../build
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_create_ind.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_create_trig.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_reset_seq.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% < pgsqlds2_create_user.sql
psql %CONNSTR% -U %SYSDBA% -d %DBNAME% -c "ANALYZE;"
```


Nun konnte mit dem Befehl `docker exec -ti psql bash` auf den Container zugegriffen werden. In diesem musste mit `su postgres` auf den **postgres** Benutzer gewechselt, da auf die postgresql Datenbank innerhalb vom Container nicht mit **root** zugegriffen werden kann.

Anschließend wurde mit dem Befehl `psql` postgresql gestartet, mit `\c ds2` in die Datenbank gewechselt und mit `\dt` kann überprüft werden, ob die Einspielung funktioniert hat:

```
ds2=# \dt
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | categories | table | ds2
 public | cust_hist  | table | ds2
 public | customers  | table | ds2
 public | inventory  | table | ds2
 public | orderlines | table | ds2
 public | orders     | table | ds2
 public | products   | table | ds2
 public | prooftable | table | postgres
 public | reorder    | table | ds2
(9 rows)
```

Abbildung 1: Datenbank wurde in das System eingespielt

2.1.2 Analysieren

Um die Datenbank zu analysieren wurde **pgAdmin** lokal installiert, die Informationen der Datenbank am Docker Container angegeben und schließlich die Tabellen angeschaut. Die existierenden Tabellen können unter `Servers/ds2/Databases/ds2/Schemas/public` gefunden werden:

```
ds2=# \dt
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | categories | table | ds2
 public | cust_hist  | table | ds2
 public | customers  | table | ds2
 public | inventory  | table | ds2
 public | orderlines | table | ds2
 public | orders     | table | ds2
 public | products   | table | ds2
 public | prooftable | table | postgres
 public | reorder    | table | ds2
(9 rows)
```

Abbildung 2: Tabellen können in pgAdmin eingesehen werden

Um nun das ERD (Entitiy Relational Diagram) zu sehen, wurde das Programm `dbis` verwendet, um besagtes Diagramm zu erstellen:

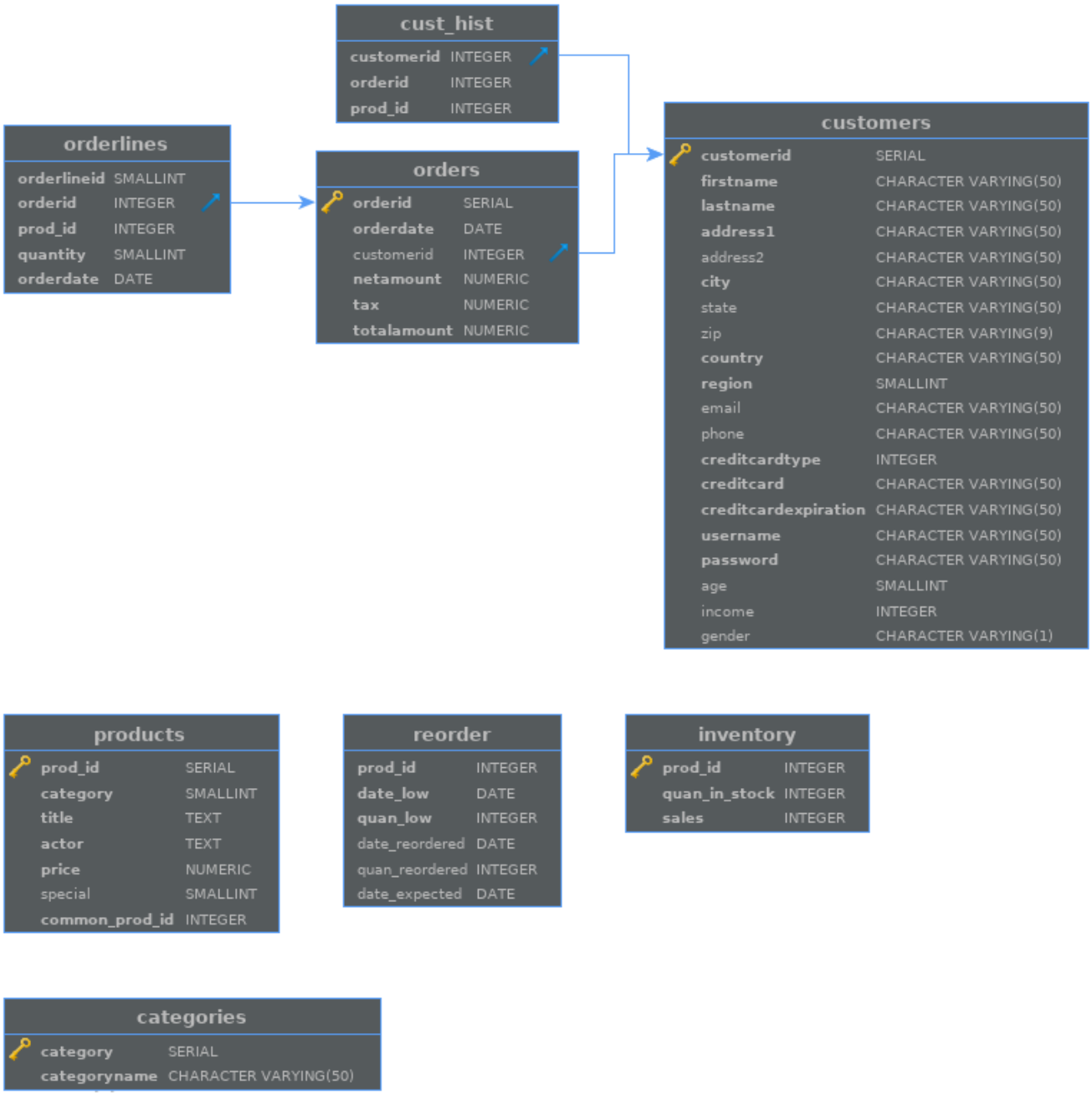

Abbildung 3: Tabellen wurden visualisiert und in Relation gestellt

Durch dieses Diagramm kann analysiert werden wie die Fragmentierung gesetzt wird.

2.2 Horizontale Fragmentierung

Die horizontale Fragmentierung wurde so gesetzt, dass von den Bestellungen nur jene angezeigt werden, welche im ersten Halbjahr 2009 bestellt wurden und einen Nettobetrag über 300 haben. Der Sinn dahinter könnte sein, dass eine interne Analyse im Geschäft gemacht werden soll, bei welcher die analysiert wird wer im ersten Halbjahr große Einkäufe getätigt hat.

Der erste schritt ist ein horizontales Schema zu erstellen:

```
DROP SCHEMA IF EXISTS horizontal;

CREATE SCHEMA horizontal;
```

Danach wird in dem Schema ein Fragment als Tabelle erstellt. Eine Fragmentstabelle kann am angenehmsten mit folgender Syntax erstellt werden: `CREATE TABLE schema.fragment AS (...)`, wobei nach `AS` eine SELECT-Abfrage steht welche anschließend das Fragment darstellt. Für den vorhin beschriebenen Fall sieht es so aus:

```
CREATE TABLE horizontal.firstHalfYearBigger300 AS (SELECT * FROM orders WHERE netamount > 300 AND
    orderdate < '2009-07-01');
```

```
ds2=# SELECT * FROM horizontal.firsthalfyearBigger300
;
 orderid | orderdate  | customerid | netamount |  tax  | totalamount
---------+------------+------------+-----------+-------+-------------
       1 | 2009-01-27 |       7888 |    313.24 | 25.84 |      339.08
       6 | 2009-01-11 |      13734 |    382.59 | 31.56 |      414.15
      11 | 2009-01-31 |       1082 |    348.22 | 28.73 |      376.95
      18 | 2009-01-23 |       4045 |    368.37 | 30.39 |      398.76
      20 | 2009-01-06 |       8730 |    383.46 | 31.64 |      415.10
      21 | 2009-01-06 |       5479 |    345.84 | 28.53 |      374.37
      23 | 2009-01-11 |      18638 |    363.40 | 29.98 |      393.38
      26 | 2009-01-31 |       6765 |    359.03 | 29.62 |      388.65
      27 | 2009-01-10 |       2294 |    346.22 | 28.56 |      374.78
      30 | 2009-01-15 |       4510 |    330.71 | 27.28 |      357.99
      38 | 2009-01-15 |       3340 |    318.20 | 26.25 |      344.45
```

Abbildung 4: SELECT * FROM horizontal.firstHalfYearBigger300;

2.3 Vertikale Fragmentierung

Für die vertikale Fragmentierung war die Idee Regalprodukte zu erstellen, welche lediglich Titel, Preis und natürlich die ID beinhalten. Nach dem Erstellen des Schemas namens `vertical` wurde folgendes Fragment erstellt:

```
CREATE TABLE vertical.shelfProducts AS (SELECT prod\_id, title, price FROM products);
```

```
ds2=# SELECT * FROM vertical.shelfproducts;
 prod_id |          title           | price
---------+--------------------------+-------
       1 | ACADEMY ACADEMY          | 25.99
       2 | ACADEMY ACE              | 20.99
       3 | ACADEMY ADAPTATION       | 28.99
       4 | ACADEMY AFFAIR           | 14.99
       5 | ACADEMY AFRICAN          | 11.99
       6 | ACADEMY AGENT            | 15.99
       7 | ACADEMY AIRPLANE         | 25.99
       8 | ACADEMY AIRPORT          | 16.99
```

Abbildung 5: SELECT * FROM vertical.shelProducts;

2.4 Kombinierte Fragmentierung

Für die kombinierte Fragmentierung wurde ein Konzept überlegt, bei welchem wieder von den Bestellungen horizontal geteilt wird indem dass erste Halbjahr genommen wird und Bestellungen welche einen Nettobetrag über 300 haben. Vertikal wurde nun nach dem Steuergeldbetrag und Gesamtbetrag, und natürlich ID gefiltert, um eine staatliche Analyse der großen Einkaufe zu bewerkstelligen.

Es wurde wieder ein Schema erstellt, namens `combined`:

```
1 CREATE TABLE combined.firstHalfYearBigger300Tax AS (SELECT orderid, tax, totalamount FROM orders WHERE
      netamount > 300 AND orderdate < '2009-07-01');
```

```
ds2=# SELECT * FROM combined.firstHalfYearBigger300Tax;
 orderid |  tax  | totalamount
---------+-------+-------------
       1 | 25.84 |      339.08
       6 | 31.56 |      414.15
      11 | 28.73 |      376.95
      18 | 30.39 |      398.76
      20 | 31.64 |      415.10
      21 | 28.53 |      374.37
      23 | 29.98 |      393.38
      26 | 29.62 |      388.65
      27 | 28.56 |      374.78
      30 | 27.28 |      357.99
      38 | 26.25 |      344.45
      39 | 26.87 |      352.62
      42 | 25.13 |      329.78
      52 | 31.65 |      415.24
```

Abbildung 6: SELECT * FROM combined.firstHalfYearBigger300Tax;

2.5 Testung

Um zu testen, mussten zuerst alle Gegenfragmente der jeweiligen Tabellen erstellt werden, um wieder die gesamte Tabelle zusammenfügen zu können

2.5.1 Horizontal

```
1 CREATE TABLE horizontal.secondHalfYearBigger300 AS (SELECT * FROM orders WHERE netamount > 300 AND
      orderdate >= '2009-07-01');

3 CREATE TABLE horizontal.firstHalfYearSmaller300 AS (SELECT * FROM orders WHERE netamount <= 300 AND
      orderdate < '2009-07-01');

5 CREATE TABLE horizontal.secondHalfYearSmaller300 AS (SELECT * FROM orders WHERE netamount <= 300 AND
      orderdate >= '2009-07-01');
```

Bei der horizontalen Fragmentierung müssen alle Fragmente mit `UNION` zusammengefügt werden und anschließend werden diese gezählt und vergleicht mit der originalen Anzahl der orders:

```
SELECT COUNT(*) FROM (SELECT * FROM horizontal.firstHalfYearBigger300 UNION SELECT * FROM horizontal.
    secondHalfYearBigger300 UNION SELECT * FROM horizontal.firstHalfYearSmaller300 UNION SELECT * FROM
    horizontal.secondHalfYearSmaller300) AS "sub";

SELECT COUNT(*) FROM orders;
```

2.5.2 Vertical

```
CREATE TABLE vertical.otherProducts AS (SELECT prod_id, category, actor, special, common_prod_id FROM
    products);
```

Bei der vertikalen kann geprüft werden, indem die Attribute der Tabelle verglichen werden. Zusammengefügt wird nicht mit `UNION` sondern mit `NATURAL JOIN`:

```
CREATE Table vertical.proofTable AS (SELECT * FROM vertical.shelfProducts NATURAL JOIN vertical.
    otherProducts);

-- Originale Tabelle
\d products
-- Beweis tabelle
\d proofTable
```

2.5.3 Combined

```
CREATE TABLE combined.firstHalfYearBigger300Tax AS (SELECT orderid, tax, totalamount FROM orders WHERE
    netamount > 300 AND orderdate < '2009-07-01');

CREATE TABLE combined.secondHalfYearBigger300Tax AS (SELECT orderid, tax, totalamount FROM orders WHERE
    netamount > 300 AND orderdate >= '2009-07-01');

CREATE TABLE combined.firstHalfYearSmaller300Tax AS (SELECT orderid, tax, totalamount FROM orders WHERE
    netamount <= 300 AND orderdate < '2009-07-01');

CREATE TABLE combined.secondHalfYearSmaller300Tax AS (SELECT orderid, tax, totalamount FROM orders WHERE
     netamount <= 300 AND orderdate >= '2009-07-01');

CREATE TABLE combined.otherOrders AS (SELECT orderid, orderdate, customerid, netamount FROM orders);
```

Bei combined müssen zuerst die Zeilen mit `UNION` zusammengefügt werden, und anschließend mit `NATURAL JOIN`:

```
CREATE Table combined.taxOrders AS (SELECT * FROM combined.firstHalfYearBigger300Tax UNION SELECT * FROM
     combined.secondHalfYearBigger300Tax UNION SELECT * FROM combined.firstHalfYearSmaller300Tax UNION
    SELECT * FROM combined.secondHalfYearSmaller300Tax);

DROP TABLE IF EXISTS combined.proofTable;
CREATE TABLE combined.proofTable AS (SELECT * FROM combined.taxOrders NATURAL JOIN combined.otherOrders)
    ;

-- Sollte das gleiche Ergebnis wie von SELECT COUNT(*) FROM orders; sein
SELECT COUNT(*) FROM combined.proofTable;
```

3 Fragestellungen

3.1 Was versteht man unter dem Begriff Allokation beim Entwurf einer verteilten Datenbank?

Es wird zwischen Fragmentierung und Allokation unterschieden. Anstatt wie bei Fragmentierung, enthalten bei Allokation die Daten verschiedenes Zugriffsverhalten indem diese verschiedenen Stationen zugeordnet werden.

3.2 Beschreiben Sie die Korrektheitsanforderungen bei der Fragementierung von verteilten Datenbanken.

- **Rekonstruierbarkeit**: die Ursprungsrelation lässt sich aus den Fragmenten wiederherstellen
- **Vollständigkeit**: jedes Datum ist einem Fragment zugeordnet
- **Disjunktheit**: Fragmente überlappen sich nicht, d.h. ein Datum ist nicht mehreren Fragmenten zugeordnet

3.3 Wie geht man bei einer horizontalen DB-Fragemtierung vor? Beantworten Sie diese Frage anhand eines Beispiels.

Zerlegung einer Relation in disjunkte Tupelmengen, Zerlegung durch Selektionen

Dabei ist gemeint, die Menge in Fragmente zu teilen, um diese auf verschiedene Standorte zu verteilen oder andere Gründe aufzuteilen. Beispiel könnte eine Tabelle sein, welche Preise enthält, könnte horizontal geteilt werden, wenn der Preis größer als 500 ist.

3.4 Die Transparenz von verteilten Datenbanken ist in mehrere Stufen gegliedert. Beschreiben Sie die Lokale-Schema-Transparenz.

- **Fragmentierungstransparenz**
- **Allokationstransparenz**
- **Lokale Schema-Transparenz**: Benutzer muss Fragment als auch Standort kennen

3.5 Was versteht man unter dem Begriff Fragmentierung beim Entwurf einer verteilten Datenbank? Wie sieht eine vertikale Fragmentierung aus? Erklären Sie die Begriffe anhand von einem Beispiel.

Unter Fragmentierung versteht man das Aufteilen der Datenbank.

Das Problem bei der vertikalen Fragmentierung ist der Verstoß gegen die Rekonstruierbarkeit. Man lässt es zu, solange folgende Kriterien gelten:

- Jedes Fragment enthält Primärschlüssel
- Jedem Tupel der Originalrelation wird künstlicher Surrogatschlüssel zugewiesen, der in Fragment übernommen wird

Ein Beispiel wäre, eine Tabelle mit sehr vielen Attributen zu haben, aus welcher man nur einige bestimmte Attribute in einer anderen Tabelle haben will. Beispielsweise könnte man aus Kontaktetabelle nur die E-Mails und Namen haben wollen, um an diese Briefe zu senden.

Literatur

Abbildungsverzeichnis