AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

14. Übungsblatt

Für die Übung am Donnerstag, **15. Februar 2007**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Rekursion in SQL:1999 am Beispiel "Flugstrecke"

Eine Fluggesellschaft verwaltet die von ihr angebotenen Flugstrecken in einer Datenbank, aus der wir folgenden (hier relevanten) Ausschnitt der Tabelle „Flüge“ zur Verfügung gestellt bekommen haben:

FlugNr	Start	Ziel	Preis
HY120	DFW	JFK	255
HY130	DFW	LAX	200
HY140	DFW	ORD	100
HY150	DFW	SFO	300
HY210	JFK	DFW	225
HY240	JFK	ORD	250
HY310	LAX	DFW	200
HY350	LAX	SFO	50
HY410	ORD	DFW	100
HY420	ORD	JFK	250
HY450	ORD	SFO	275
HY510	SFO	DFW	300
HY530	SFO	LAX	50
HY540	SF0	ORD	275

Entwickeln Sie in SQL:1999 Anfragen für folgende Fragestellungen:

a) Finde den billigsten Flug von San Fransisco (SFO) nach New York (JFK).

b) Finde den Flug von San Fransisco nach New York, bei dem man am wenigsten umsteigen muß.

Lösung:

a)

1. Versuch

```
WITH RECURSIVE trips (ziel, route, gesamtpreis) AS
( (   SELECT ziel, ziel, preis
      FROM fluege
      WHERE start = 'SFO')

  UNION ALL
  (   SELECT f.ziel, t.route ||','||f.ziel, t.gesamtpreis+f.preis
      FROM trips t, fluege f
      WHERE t.ziel = f.start))
SELECT route, gesamtpreis
FROM trips
WHERE ziel = 'JFK';
```

Dieser Versuch geht aus mehreren Gründen schief:

1. Grund: Die route-Spalte der initialen Unteranfrage sind von Typ CHAR(3), der rekursive Anfrageanteil erweitert aber diese Spalte immer weiter.

Lösung: CAST auf VARCHAR(30) in der initialen Anfrage

2. Grund: Wann soll die Rekursion abbrechen? Da es (außer den Systemressourcen) keine Abbruchbedingung gibt, werden beliebig viele Schleifen ausgewertet und die Rekursion bricht „niemals“ ab.
Lösung: Finde geeignete Abbruchbedingungen

Als geeignet können angesehen werden:

- Kein Flug mit Ziel SFO wird beachtet (Schleife zum Ausgangspunkt)

- Kein Flug mit Start JFK wird beachtet (Schleife von Zielpunkt aus)

- Begrenze Flug auf maximal zwei Umstiege (künstliche Abbruchbedingung)
Daraus ergibt sich folgende Lösung:

```
WITH RECURSIVE trips (ziel, route, anzseg, gesamtpreis) AS

  ((      SELECT ziel, CAST(ziel AS VARCHAR(30)), 1, preis
          FROM fluege
          WHERE start = 'SFO'=
  UNION ALL
  (       SELECT       f.ziel, CAST(t.route||','||f.ziel AS VAR-
CHAR(30)),
                       t.anzsegm+1, t.gesamtpreis+f.preis
          FROM trips t, fluege f
          WHERE        t.ziel = f.start

              AND      f.ziel <> 'SFO'
              AND      f.start <> 'JFK'
              AND t.anzsegm < 3))
SELEFT route, gesamtpreis
FROM trips
WHERE ziel = 'JFK'
  AND      gesamtpreis = (SELECT min(gesamtpreis) FROM trips WHERE
ziel = 'JFK')
```

Noch besser ist die folgende Lösung, da in ihr nicht die Start- und Zielflughäfen fest kodiert werden müssen. Die Abbruchbedingung wird dadurch formuliert, daß ein Flughafen nur einmal in einer Route vorkommen darf. Der Vergleich wird durch das "LIKE" realisiert.

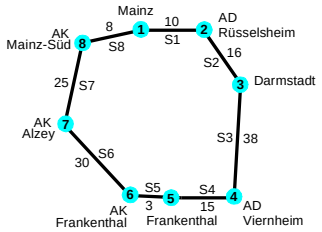
```
WITH RECURSIVE trips (ziel, route, gesamtpreis) AS
  ((       SELECT ziel, CAST(start || ',' || ziel AS VARCHAR(30)),
preis
           FROM fluege
  )
  UNION ALL
  (        SELECT       f.ziel, CAST(t.route||','||f.ziel AS VAR-
CHAR(30)),
                        t.gesamtpreis+f.preis
           FROM trips t, fluege f
           WHERE        t.ziel = f.start AND
                        NOT (t.route LIKE '%'||f.ziel'||'%')
  ))
SELEFT route, gesamtpreis
FROM trips
WHERE ziel = 'JFK' AND t.route LIKE 'SFO,%'
  AND     gesamtpreis = (SELECT min(gesamtpreis) FROM trips WHERE
ziel = 'JFK'
                                        AND t.route LIKE 'SFO,%')
```

b)

```
SELECT route, gesamtpreis
FROM trips
WHERE ziel = 'JFK' AND anzsegm = (SELECT min(anzsegm) FROM trips WHERE
ziel = 'JFK')
```


Aufgabe 0.0.1: Rekursion in SQL:1999 am Beispiel "Autobahnkarte"

In der Karte rechts wird ein Ausschnitt aus dem Autobahnnetz dargestellt. Die einzelnen Autobahnteilstücke sind durch die Strecken S1 bis S8 mit entsprechenden Längen in Kilometern gegeben. Die Strecken verbinden jeweils zwei Navigationspunkte miteinander. Ein Navigationspunkt ist durch eine eindeutige ID und dessen Name gegeben, bspw. (1,Mainz).



Die Navigationspunkte sollen in der Tabelle

navpkt (ID, Name)

verwaltet werden, die Strecken in der Tabelle

navstr (anfang, ende, name, laenge),

wobei *anfang* und *ende* jeweils die ID eines Navigationspunktes referenzieren und somit zwei Navigationspunkte miteinander verbinden.

a) Geben Sie die SQL-DDL-Anweisungen an, mit denen die Tabellen *navpkt* und *navstr* angelegt werden, und fügen sie anschließend mit entsprechenden SQL-DML-Anweisungen die notwendigen Tupel in die Tabellen ein, um die gegebene Karte zu speichern. Beachten Sie dabei, dass eine Strecke zwischen zwei Navigationspunkten für beiden Richtungen gespeichert werden muss.

b) Berechnen Sie mit einer rekursionsfreien SQL-Anweisung die Entfernung zwischen Frankenthal und Darmstadt über das Autobahndreieck Viernheim.

c) Berechnen Sie mit einer rekursiven SQL-Anweisung die kürzeste Verbindung von Frankenthal nach Mainz. Geben Sie dazu die berechnete Route und deren Länge aus.

Lösung:

a) SQL-DDL-Anweisungen

```
create table navpkt
(
  id int primary key not null,
  name varchar(100)
)
create table navstr
(
  anfang int not null references navpkt(id),
  ende int not null references navpkt(id),
  name varchar(200) not null,
  laenge int not null,
  primary key (anfang, ende)
)
```

SQL-DML-Anweisungen

```
insert into navpkt values (1,'Mainz')
insert into navpkt values (2,'AD Rüsselsheim')
insert into navpkt values (3,'Darmstadt')
insert into navpkt values (4,'AD Viernheim')
insert into navpkt values (5,'Frankenthal')
insert into navpkt values (6,'AK Frankenthal')
insert into navpkt values (7,'AK Alzey')
insert into navpkt values (8,'AK Mainz-Süd')
```

```
insert into navstr values (1,2,'S1',10)
insert into navstr values (2,1,'S1',10)
insert into navstr values (2,3,'S2',16)
insert into navstr values (3,2,'S2',16)
insert into navstr values (3,4,'S3',38)
insert into navstr values (4,3,'S3',38)
insert into navstr values (4,5,'S4',15)
insert into navstr values (5,4,'S4',15)
insert into navstr values (5,6,'S5',3)
insert into navstr values (6,5,'S5',3)
insert into navstr values (6,7,'S6',30)
insert into navstr values (7,6,'S6',30)
insert into navstr values (7,8,'S7',25)
insert into navstr values (8,7,'S7',25)
insert into navstr values (8,1,'S8',8)
insert into navstr values (1,8,'S8',8)
```

b)

```
select sum(laenge)
from navstr
where (anfang=(select id from navpkt where name='Frankenthal')
       and ende=(select id from navpkt where name='AD Viernheim'))
      or
      (anfang=(select id from navpkt where name='AD Viernheim')
      and ende=(select id from navpkt where name='Darmstadt'))
```

c)

```
with recursive route (ende, beschreibung, gesamtlaenge)
as
((select ende, name, laenge
  from navstr
  where anfang=(select id from navpkt where name='Frankenthal')
 union all
 (select str.ende, r.beschreibung || ',' || str.name,
         r.gesamtlaenge + str.laenge
  from route r, navstr str
  where r.ende=str.anfang and r.gesamtlaenge<100))
)
select gesamtlaenge, beschreibung
from route
where ende=(select id from navpkt where name='Mainz')
  and gesamtlaenge=(select min(gesamtlaenge)
                    from route
                    where ende=(select id
                                from navpkt
                               where name='Mainz'))
```


Aufgabe 2: Lokatoren und Dateireferenzen

Gegeben sei eine DB mit einer Relation „bewerber“, die Stipendienbewerbungen enthält und deren Schema durch folgende SQL/DDL-Anweisung erzeugt wurde:

```
CREATE TABLE bewerber (
name          VARCHAR(30) PRIMARY KEY,
eingangsjahr  INTEGER,
status        VARCHAR(6),
bewerbung     CLOB(100K))
```

Jede Bewerbung besteht aus mehreren Abschnitten. Ein Abschnitt enthält eine Abschrift aus dem Studienbuch, die in einem einheitlichen 1KB-Format erfaßt ist, das mit den Zeichenfolgen ’*LEISTUNG*’ beginnt. In einem späteren Abschnitt der Bewerbung befindet sich ein Empfehlungsschreiben, das bis zum Ende der Bewerbung geht und mit ’*EMPFEHLUNG*’ anfängt .

Skizierren Sie ein C-Programm mit eingebetteten SQL-Anweisungen, das nach allen im Jahr 2000 eingegangenen Bewerbungen mit dem Status ’OK’ sucht. Von diesen finden Sie diejenige Bewerbung mit der besten Studienleistung heraus. Schreiben Sie die Studienleistung und das zugehörige Empfehlungsschreiben in die Ausgabedatei „gewinner.txt“. Weiterhin sei eine Funktion gegeben, die zwei Studienleistungen vergleichen und entscheiden kann, welche von den beiden die „bessere“ ist.

Benutzen Sie folgende Funktionen zur Lösung der Aufgabe:

- `int vergleicheStudienleistung (char *sl1, char *sl2)`
  Liefert 1, falls die erste Studienleistung `sl1` besser als die zweite `sl2` ist und sonst 2.
- `char *strcpy (char *s1, const char *s2)`
  Kopiert die Zeichenfolge `s2` in die Zeichenfolge `s1`. Der Funktionswert ist der Zeiger `s1`.
- `void *memcpy (void *s1, const void *s2, size_t n)`
  Kopiert n Zeichen von dem Speicherbereich, auf den `s2` zeigt, nach dem Speicherbereich, auf den `s1` zeigt. Der Funktionswert ist der Zeiger `s1`.
- CLOB-/BLOB-/Dateireferenz-Datentyp und -Funktionen, die in der Vorlesung vorgestellt wurden.

Lösung:

```
#include <stdio.h>
#include <string.h>
#include <sglenv.h>

int vergleicheStudienleistung (char *sl1, char *sl2);
EXEC SQL INCLUDE SQLCA;
int main ()
{
  EXEC SQL BEGIN DECLARE SECTION
       char dbName[]="testDB";
       char kandidat[30], besterKandidat[30];
       long leistungPos, empfehlungPos;
       char leistung[1024], besteLeistung[1024];
       char fehlerMeldung[500];
       int leistungLen = 1024;

       SQL TYPE IS CLOB_LOCATOR loc, besterLoc;
       SQL TYPE IS CLOB_FILE gewinnerDatei;
  EXEC SQL END DECLARE SECTION;
  int geradeBegonnen = 1;
  EXEC SQL DECLARE c1 CURSOR FOR
       SELECT name, bewerbung FROM bewerber
       WHERE eingangsjahr = 2000 AND status = 'OK';
  EXEC SQL WHENEVER SQLERROR GOTO fehlerAusgang;
  EXEC SQL CONNECT TO :dbName;
  EXEC SQL OPEN c1;

  while (1)
  {
       /* Hole nächsten Kandidat */
       EXEC SQL FETCH c1 INTO :kandidat, :loc;
       if (SQLCODE == 100) break; /* Keine weitere Kandidaten */

       /* Hole die Studienleistung dieses Kandidats */
       EXEC SQL VALUES POSITION ('*LEISTUNG*' IN :loc) INTO :lei-
stungPos;
       EXEC SQL
            VALUES SUBSTRING (:loc FROM 1 FOR :leistungLen)
                INTO :leistung;

       if (geradeBegonnen == 1)
       {
            /* Erster Kandidat ist immer der beste */
            strcpy(besterKandidat, kandidat);
            memcpy(besteLeistung, leistung, leistungLen);
            besterLoc = loc;
            geradeBegonnen = 0;
            }
       else
       {
            if (vergleicheStudienleistung(besteLeistung, leistung) ==
2)
            {
                 /* Aktueller Kandidat ist besser als der bisher beste
Kandidat */
                 memcpy(besteLeistung, leistung, leistungLen);
                 EXEC SQL FREE LOCATOR :besterLoc;
                 besterLoc = loc;
                 strcpy(besterKandidat, kandidat);
                 }
            else
            {
                 /* loc wird nicht mehr benötigt */
                 EXEC SQL FREE LOCATOR :loc;
                 }
            }
       } /* Ende von while-Schleife */

  EXEC SQL CLOSE c1;

  if (geradeBegonnen == 1)
  {
       printf("Keine qualifizierten Bewerbungen gefunden.\n");
       }
  else
  {
       /* Finde das Empfehlungsschreiben des besten Kandidats */
       EXEC SQL
            VALUES POSITION ('*EMPFEHLUNG*' IN :besterLoc) INTO :emp-
fehlungPos;

       /* Dateireferenz für die Ausgabedatei vorbereiten */
       strcpy(gewinnerDatei.name, "gewinner.txt");
       gewinnerDatei.name_length = strlen(gewinnerDatei.name);
       gewinnerDatei.file_options = SQL_FILE_OVERWRITE;

       /* Kopieren von Studienleistung und Empfehlungsschreiben
       ** des besten Kandidats in die Ausgabedatei.
       */
       EXEC SQL
            VALUES :besteLeistung || SUBSTRING (:besterLoc FROM :emp-
fehlungPos)
            INTO :gewinnerDatei;

       printf("Der beste Kandidat lautet %s\n", besterKandidat);
       }

  EXEC SQL COMMIT;
  return 0;

  fehlerAusgang:
       printf("Unerwarteter DB2 Return Code.\n");
       sqlaintp(fehlerMeldung, 500, 70, &sqlca);
       printf("Fehler Meldung: %s\n", fehlerMeldung)
       EXEC SQL ROLLBACK;
       return -1;
} /* Ende von main */
```


Datenbankanwendung WS 2006/2007 – Lösungsvorschläge zu Übungsblatt 14