Klausur Datenbanken
Wintersemester 2019/2020
Prof. Dr. Wolfgang May
27. Februar 2020, 14-16:30 Uhr
Bearbeitungszeit: 120 Minuten

Vorname:

Nachname:

Matrikelnummer:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner, etc.) erlaubt. Mobiltelefone müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller, etc.; Bleistift ist nicht erlaubt.
Zwecks besserer Lesbarkeit (insbesondere auch für Nicht-{Mut/d/Va}tersprachler*innen) wird in der Aufgabenstellung auf gegenderte Sprache verzichtet.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

(Unterschrift (DSGVO))

Meine Note soll mit meiner persönlichen Codezahl: 1234 so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ Meine Note soll nicht veröffentlicht werden; ich erfahre sie dann aus FlexNow oder beim zuständigen Prüfungsamt.

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (ER-Modell)	14	12
Aufgabe 2 (Transformation in das Relationale Modell)	18	14
Aufgabe 3 (SQL und Relationale Algebra)	45	16
Aufgabe 4 (Verschiedenes)	14	5
Summe	91	47

Note:

Themenstellung: Essenslieferdienst

Alle Klausuraufgaben basieren auf einem gemeinsamen "Auftrag": In der Klausur soll eine Datenbank für einen Lieferdienst für Speisen entworfen werden.

1. Der Lieferdienst umfasst mehrere *Anbieter*.

   Jeder Anbieter hat einen *Namen*, z.B., *Pizzawelt* oder *Kalle's Imbiss* und eine *Adresse*. Jede Adresse besteht aus *Straßenname+Hausnummer* und gehört zu einem Postleitzahl-*Gebiet*. Straßenname + Hausnummer sind innerhalb eines solchen Gebiets eindeutig.

   Es kann mehrere Anbieter mit demselben Namen an verschiedenen Adressen geben. So gibt es eine *Pizzawelt*-Filiale sowohl an der Adresse *Um's Eck 5* im Gebiet *30425* als auch mit der Adresse *Laange Straße 404* im Gebiet *30431*. Neben dieser, im Haus *Laange Straße 406* befindet sich *Kalle's Imbiss*.

   Optional können Anbieter einen *Mindestbestellwert* für Bestellungen bei ihnen angeben.

   Bei *Pizzawelt Um's Eck 5* in 30425 beträgt der Mindestbestellwert 10 EUR.

2. Jeder Anbieter bietet eine Reihe von *Gerichten* an.

   Jedes Gericht hat einen *Namen*, z.B. *Pizza Margarita*, *Pizza Salami* oder *Pommes Frites*, und gehört zu einer Kategorie (z.B. *Pizza* oder *Snack*).

   Jeder Anbieter bietet einige Gerichte zu einem bestimmten *Preis* an: Bei *Pizzawelt Um's Eck 5* im Gebiet 30425 gibt es *Pizza Margarita* für 4.00 Euro und *Pizza Salami* für 4.50 EUR, während die andere *Pizzawelt* an der *Laangen Straße 404* die *Pizza Margarita* für nur 3.80 EUR anbietet. *Pommes Frites* kosten bei *Kalle's Imbiss* 3 EUR.

   Die Preise und die angebotenen Gerichte eines Anbieters können sich zudem im Laufe der Zeit ändern.

3. Für den Lieferdienst arbeiten mehrere *Fahrer*. Sie werden über ihren eindeutigen internen *Nickname* identifiziert, außerdem ist ihre Handy-Nummer gespeichert. Jeder Fahrer fährt in einem oder mehreren Postleitzahl-*Gebieten*. Für jeden Fahrer ist gespeichert, ob er gerade Dienst hat, oder nicht (ja/nein).

   Der Fahrer *Timo*, Handynummer 0160-4711, fährt in den Gebieten 30423, 30424 und 30425 und hat gerade Dienst.

   Der Fahrer *Tilo*, Handynummer 0171-0815, fährt in den Gebieten 30425, 30426 und 30431 und hat gerade frei.

4. Im Mittelpunkt der Datenbank stehen natürlich die *Bestellungen*, die von Kunden (per Web-Frontend, aber das spielt hier keine Rolle) aufgegeben werden.

   Jeder Bestellung wird eine eindeutige *ID* zugewiesen und das aktuelle *Datum* und die *Uhrzeit* werden erfasst. Der Kunde gibt eine Liefer*adresse* (Straße+Hausnr., Postleitzahl, sowie den Namen, bei dem geklingelt werden soll) an, und wählt dann einen *Anbieter* aus.

   Jede Bestellung besteht aus einer Liste der bestellten *Gerichte* des ausgewählten Anbieters mit der bestellten *Anzahl* zu deren derzeitigem *Kaufpreis*.

Später wird der Bestellung dann ein *Fahrer* zugeordnet, der sowohl den entsprechenden Anbieter als auch die Lieferadresse des Kundens erreicht. Dabei wird der *Status* der Bestellung (Neu, Zugeordnet (zu einem Fahrer), Abgeholt, Ausgeliefert) verfolgt. Es wird auch gespeichert, wann die Bestellung ausgeliefert wurde.

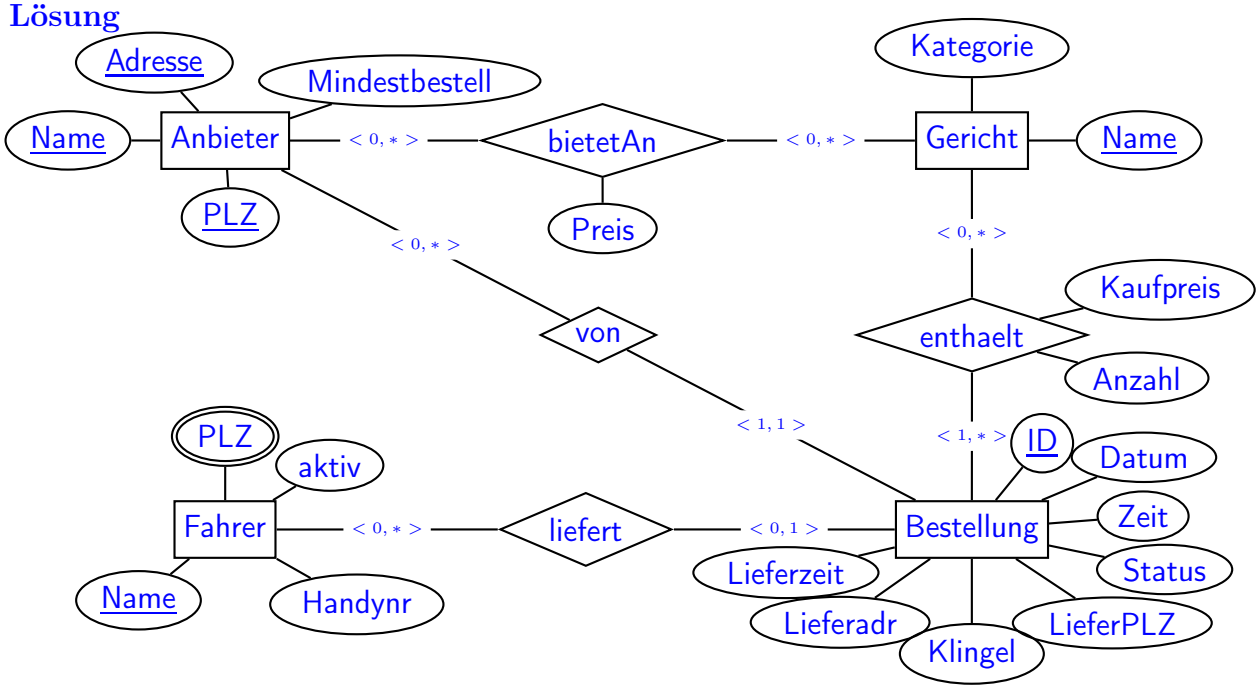
Alice gab soeben (27.02.2020, 14:05 Uhr) eine *Bestellung* auf, die die ID 777 bekommen hat. Sie bestellte von *Pizzawelt* an der Adresse *Um's Eck 5* im Gebiet 30425:

- 1 x *Pizza Margarita* für 4.00 Euro
- 2 x *Pizza Salami* für jeweils 4.50 Euro

Als Lieferadresse gab sie *Schmidt*, *Hinter dem Mond 15* im Gebiet 30423 an. Bisher wurde die Bestellung noch keinem Fahrer zugeteilt.

Aufgabe 1 (ER-Modell [14 Punkte])
Entwickeln Sie ein ER-Modell für das Szenario. Geben Sie darin die Schlüsselattribute sowie die Beziehungskardinalitäten an.

Lösung

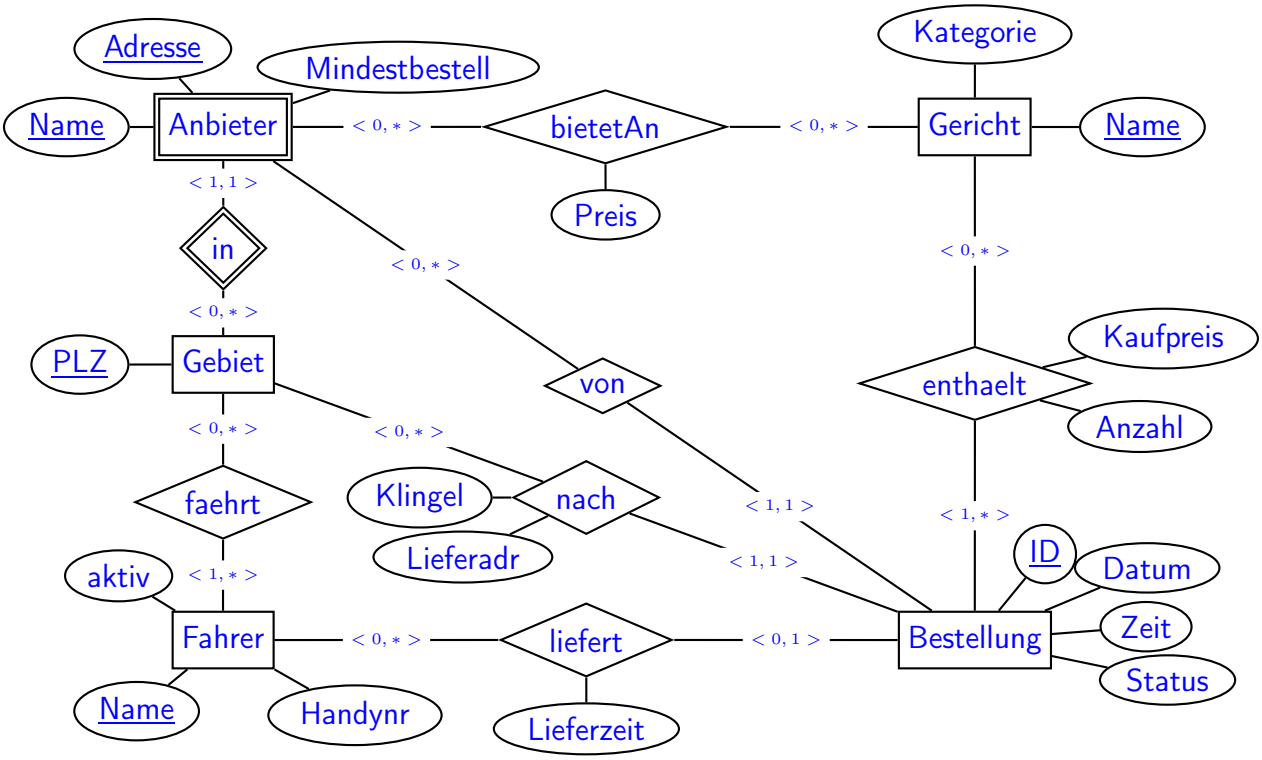


Alternativen und Kommentare:

- Der Primary Key von Anbieter muss (Name, Adresse, PLZ) sein, da es an einer Adresse mehrere Anbieter (z.B. eine Pizzeria und eine Pommesbude nebeneinander im selben Haus) geben kann. Adresse ist notwendig, da es durchaus mehrere "Pizzawelt" im selben PLZ-Bereich geben kann, und PLZ ist notwendig, da in zwei verschiedenen Orten die "Pizzawelt" in der "Hauptstraße 10" sein kann.
- Da sich Angebot und Preise von Anbietern ändern können, kann <enthaelt> nur mit [Gericht] verbunden sein. Die [Gerichte]-Tabelle kann man nicht weglassen, da daran die Kategorie hängt.
- Man kann anstatt <enthaelt> auch einen (schwachen) Entitätstyp [BestellItem] mit entsprechenden <1,1>-Beziehungen zu [Bestellung] und [Gericht] modellieren.
- **F**ehlentwurf: Einige Lösungen enthielten eine Aggregation oder einen schwachen Entitätstyp [[Angebot (Anbieter, Gericht, Preis)]], der dann wiederum als [Bestellung]-<enthält(Anzahl)>-[Angebot] verwendet wurde. Dies ist hier nicht sinnvoll, da der Angebotspreis sich verändern kann, und aber der Kaufpreis fest gespeichert werden muss. Weiterhin kann ein Gericht aus dem angebot eines Anbieters wegfallen, dennoch muss die Kombination (Gericht, Anbieter) in älteren Bestellungen erhalten bleiben; ein Foreign Key auf [Angebot] ist also nicht sinnvoll. Ausserdem darf eine Bestellung nur Items *eines* einzelnen Anbieters enthalten.

  Dieser Fehlentwurf hat jedoch keine schlimmen Auswirkungen auf Aufgaben 2 und 3 – alle Anfragen sind unter dieser Bedingung ganz ähnlich lösbar.
- Bei der Zuordnung von [Bestellung]-<liefert>-[Fahrer] ist <0,1> wichtig, da bei einer neuen Bestellung Fahrer=NULL ist (siehe Beispieltupel in Aufgabe 2).

- Anstatt Fahrer.PLZ als mehrwertiges Attribut zu machen, kann man auch eine Entität [Arbeitsgebiet] mit Attribut (PLZ) und Relation <faehrtAn> zu [Fahrer], sowie Beziehungen zu [[Anbieter]] (was dann ein schwacher Entitätstyp wird) und [Bestellung] (dort kann man dann auch gleich die Lieferadresse und den Klingelname mit in die Beziehung rausziehen) modellieren:
  (hier auch "Lieferzeit" als Attribut von "liefert" modelliert)



- Für Aufgabe 2 würde das den Unterschied machen, dass man eine Tabelle Gebiet mit nur einer Spalte "PLZ" hätte, die durch Foreign Keys referenziert würde. Diese Tabelle könnte man auch weglassen. Ihr Vorteil wäre, dass niemand versehentlich eine PLZ angeben könnte, die es nicht gibt.

- Weitergehend könnte man [[Adresse (Strasse,Hausnummer)]] auch als eigenen schwachen Entitätstyp (von PLZ abhängend) *modellieren*. Dann wäre es aber endgültig so weit, dass man die entsprechende Tabelle, die *alle* Adressen (die es gibt) enthalten müsste, in der Datenbank nicht haben will.

- **F**ehlentwurf: Einige Lösungen enthielten eine Beziehung [Fahrer] – <zugeordnetZu> – [Anbieter] (zum Teil auch noch mit Kardinalität <1,1>). Diese ist im Text nicht vorgesehen. Es wäre auch im Kontext eines allgemeinen Lieferdienstes sinnlos, die Fahrer den Lieferanten fest zuzuordnen. Die Beziehung [Fahrer] – <fährtIn > – [Gebiet] wird sowieso benötigt, um zu beschreiben, welche Kundenadressen ein Fahrer erreichen kann.

  Auch dieser Fehlentwurf hat keine schlimmen Auswirkungen auf Aufgaben 2 und 3.

Aufgabe 2 (Transformation in das Relationale Modell [18 Punkte])

a) Geben Sie an, welche Tabellen (mit Attributen, Schlüsseln etc.) Ihre Datenbank enthält (keine SQL `CREATE TABLE`-Statements, sondern einfach grafisch). (12 P)

Markieren Sie dabei auch Schlüssel (durch unterstreichen) und Fremdschlüssel (durch überstreichen).

Geben Sie die Tabellen mit jeweils mindestens zwei Beispieltupeln (z.B. denen, die sich aus dem Aufgabentext ergeben, und/oder weiteren erfundenen) an.

Lösung

Anbieter

Name	Adresse	PLZ	Mindestb.
Pizzawelt	Um's Eck 5	30425	10
Pizzawelt	Laange Strasse 404	30431	null
Kalle's Imbiss	Laange Strasse 406	30431	null
:	:	:	:

Gericht

Name	Kategorie
Pizza Margarita	Pizza
Pizza Salami	Pizza
Pommes Frites	Snack
:	:

bietetAn

Anbieter	Adresse	PLZ	Gericht	Preis
Pizzawelt	Um's Eck 5	30425	Pizza Margarita	4.00
Pizzawelt	Um's Eck 5	30425	Pizza Salami	4.50
Pizzawelt	Laange Straße 404	30431	Pizza Margarita	3.80
Kalle's Imbiss	Laange Straße 406	30431	Pommes Frites	3.00
:	:	:	:	:

faehrt

Fahrer	PLZ
Timo	30423
Timo	30424
Timo	30425
Tilo	30425
Tilo	30426
Tilo	30431
:	:

Bestellung

ID	Klingel	Lieferadr	LieferPLZ	Anbieter	AnbAdr	AnbPLZ	Fahrer	Status	Datum	Zeit	LZeit
777	Schmidt	H.d.M.15	30423	Pizzawelt	Um's Eck 5	30245	null	neu	27.2.2020	14:05	null
:	:	:	:	:	:	:	:	:	:	:	:

Fahrer

Name	Handynr	aktiv
Timo	0160-4711	1
Tilo	0171-0815	0
:	:	:

enthaelt

Bestellung	Gericht	Anzahl	Kaufpreis
777	Pizza Margarita	1	4.00
777	Pizza Salami	2	4.50
:	:	:	:

- Die Überprüfung, dass ein Gericht in der Bestellung wirklich von dem Anbieter verkauft wird, muss vom System passieren. Eine Absicherung über referentielle Integrität kann nicht geschehen, da die Bestellungen in der Datenbank gespeichert bleiben, und sich das Angebot aber ändern kann (und damit Gerichte bei einem Anbieter wegfallen können).
- Der dauerhaft gespeicherte Kaufpreis in [enthaelt] kann von dem aktuellen Preis in [bietetAn] abweichen. Daher kann die Referenz nur zu [Gericht] gehen.

b) Geben Sie das `CREATE TABLE`-Statement für diejenige Tabelle, in der das Angebot der Anbieter gespeichert wird, so vollständig wie möglich (d.h. mit allen notwendigen Constraints) an (6 P).

Lösung

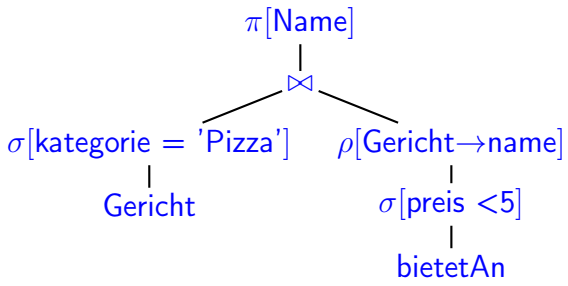
```
CREATE TABLE bietetAn                                          2.5P Basis
( Anbieter VARCHAR2(255),
  Adresse VARCHAR2(255),
  PLZ NUMBER,
  Gericht VARCHAR2(255) REFERENCES Gericht(Name),                1/2P
  Preis NUMBER NOT NULL CHECK (Preis >= 0) ,                   1/2P +  1/2P
  FOREIGN KEY (Anbieter, Adresse, PLZ)
            REFERENCES Anbieter(Name, Adresse, PLZ),              1P
  PRIMARY KEY (Anbieter, Adresse, PLZ, Gericht)                   1P
);
```

Aufgabe 3 (SQL und Relationale Algebra [45 Punkte])
Verwenden Sie für diese Aufgabe die von Ihnen entworfene relationale Datenbasis. Keine der Antworten soll Duplikate enthalten.

a) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die die Namen aller Pizzas angeben, die zum aktuellen Zeitpunkt bei mindestens einem Anbieter für weniger als 5 EUR zu bekommen sind. (2+2 P)

Lösung

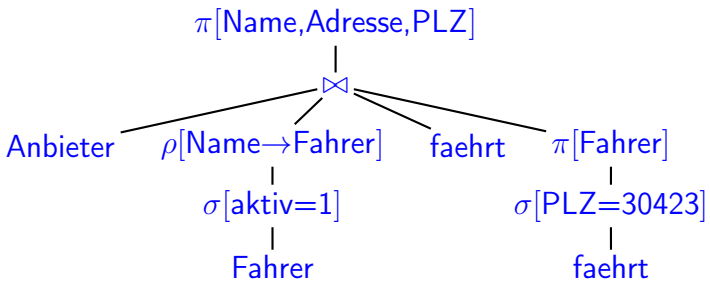
```
SELECT g.name
FROM Gericht g, bietetAn b
WHERE g.name = b.Gericht
   AND g.kategorie = 'Pizza'
   AND b.preis < 5
```



b) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die für die Lieferadresse *Hinter dem Mond 15, 30423* alle Anbieter ausgeben bei denen bestellt werden kann (d.h., für die es einen Fahrer gibt, der die Lieferadresse und den Anbieter erreichen kann, und gerade Dienst hat). (3+3 P)

Lösung

```
SELECT DISTINCT Anbieter.Name, Anbieter.Adresse, Anbieter.PLZ
FROM Anbieter, faehrt f1, faehrt f2, Fahrer f
WHERE Anbieter.PLZ = f1.PLZ
   AND f1.Fahrer = f2.Fahrer
   AND f1.Fahrer = f.Name
   AND f.aktiv = 1
   AND f2.PLZ = 30423

SELECT DISTINCT Anbieter.Name, Anbieter.Adresse, Anbieter.PLZ
FROM Anbieter, faehrt, Fahrer f
WHERE Anbieter.PLZ = f1.PLZ
   AND f1.Fahrer = f2.Fahrer
   AND f1.Fahrer = f.Name
   AND f.aktiv = 1
   AND f.name in (SELECT Fahrer
                  FROM faehrt
                  WHERE PLZ = 30423)
```



Wenn hier *faehrt* nicht *zweimal* verwendet wird, wird *Anbieter* direkt mit PLZ=30423 gejoint und es werden nur Anbieter ausgegeben, die selber auch im Gebiet 30423 liegen.

c) Geben Sie eine SQL-Anfrage an, die für jeden Anbieter berechnet, wieviel Umsatz er im 2. Halbjahr 2019 über den Lieferservice gemacht hat. (3 P)

Lösung

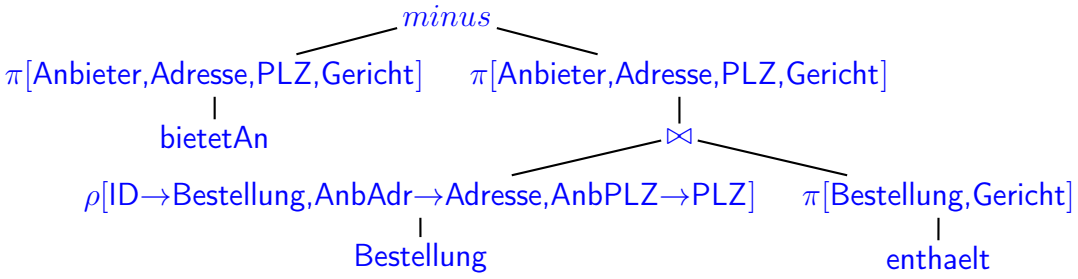
```
SELECT Anbieter, AnbAdr, AnbPLZ, SUM(Anzahl * Preis)
FROM Bestellung b, enthaelt e
WHERE b.ID = e.Bestellung
  AND Datum BETWEEN 1.7.2019 AND 31.12.2019
GROUP BY Anbieter, AnbAdr, AnbPLZ
```

d) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die für jeden Anbieter alle von ihm aktuell angebotenen Gerichte auflisten, die bei ihm noch *nie* bestellt wurden. (3+3 P)

Lösung

```
SELECT Anbieter, Adresse, PLZ, Gericht
FROM bietetAn a
WHERE NOT EXISTS (
    SELECT *
    FROM Bestellung b, enthaelt e
    WHERE a.Anbieter = b.Anbieter
    AND a.Adresse = b.AnbAdr
    AND a.PLZ = b.AnbPLZ
    AND b.ID = e.Bestellung
    AND e.Gericht = a.Gericht )

SELECT Anbieter, Adresse, PLZ, Gericht
FROM bietetAn a
WHERE (Anbieter, Adresse, PLZ, Gericht)
  NOT IN ( SELECT Anbieter, AnbAdr, AnbPLZ, Gericht
           FROM Bestellung b, enthaelt e
           WHERE b.ID = e.Bestellung )
```

```
SELECT Anbieter, Adresse, PLZ, Gericht
FROM bietetAn a
WHERE Gericht
  NOT IN ( SELECT Gericht
           FROM Bestellung b, enthaelt e
           WHERE b.ID = e.Bestellung
             AND b.Anbieter = a.Name)
             AND b.AnbAdr = a.Adresse)
             AND b.PLZ = a.PLZ)

( SELECT Anbieter, Adresse, PLZ, Gericht
  FROM bietetAn )
MINUS
( SELECT Anbieter, AnbAdr, AnbPLZ, Gericht
  FROM Bestellung b, enthaelt e
  WHERE b.ID = e.Bestellung )
```



e) Geben Sie eine SQL-Anfrage an, die für jede Lieferadresse deren "Lieblingsanbieter" ausgibt, d.h., denjenigen Anbieter (bzw. diejenigen Anbieter, falls mehrere gleich beliebt sind), bei dem/denen (zu dieser Lieferadresse) am *häufigsten* bestellt wurde. (5 P)

Lösung

```
SELECT Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
FROM Bestellung b1
GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
HAVING count(*) =
  ( SELECT max(count(*))   ### so in Oracle zulaessig, in postgres nicht
    FROM Bestellung b2
    WHERE b2.Lieferadr = b1.LieferAdr
      AND b2.LieferPLZ = b1.LieferPLZ
      AND b2.Klingel = b1.Klingel
    GROUP BY Anbieter, AnbAdr, AnbPLZ )

SELECT Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
FROM Bestellung b1
GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
```

```
HAVING count(*) >= ALL
  ( SELECT count(*)      ### so auch in postgres zulaessig
    FROM Bestellung b2
    WHERE b2.Lieferadr = b1.LieferAdr
      AND b2.LieferPLZ = b1.LieferPLZ
      AND b2.Klingel = b1.Klingel
    GROUP BY Anbieter, AnbAdr, AnbPLZ )

Dies ist ein Fall, wo >= ALL sinnvoll ist.

### oder mit geschachteltem SFW im inneren FROM:
SELECT Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
FROM Bestellung b1
GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
HAVING count(*) =
  ( SELECT MAX(bla.anzahl)  ### so auch in postgres zulaessig
    FROM (SELECT COUNT(*) as anzahl
          FROM Bestellung b2
          WHERE b2.Lieferadr = b1.LieferAdr
          AND b2.LieferPLZ = b1.LieferPLZ
          AND b2.Klingel = b1.Klingel
          GROUP BY Anbieter, AnbAdr, AnbPLZ ) bla)

### oder mit geschachteltem (...) IN (...) im HAVING:
SELECT Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
FROM Bestellung
GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
HAVING (Lieferadr, LieferPLZ, Klingel, count(*))
  IN
  ( SELECT Lieferadr, LieferPLZ, Klingel, max(anzahl)
    FROM (SELECT Lieferadr, LieferPLZ, Klingel,
                 Anbieter, AnbAdr, AnbPLZ, count(*) as anzahl
          FROM Bestellung
          GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ)
    GROUP BY Lieferadr, LieferPLZ, Klingel )

### oder mit Zurechtlegen zweier Zwischenergebnisse:
SELECT Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ
FROM
( SELECT Lieferadr, LieferPLZ, Klingel, max(anzahl)
  FROM (SELECT Lieferadr, LieferPLZ, Klingel,
               Anbieter, AnbAdr, AnbPLZ, count(*) as anzahl
        FROM Bestellung
        GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ)
  GROUP BY Lieferadr, LieferPLZ, Klingel ) maxx,
( SELECT Lieferadr, LieferPLZ, Klingel,
         Anbieter, AnbAdr, AnbPLZ, count(*) as anzahl
  FROM Bestellung
  GROUP BY Lieferadr, LieferPLZ, Klingel, Anbieter, AnbAdr, AnbPLZ ) alle
```

```
WHERE maxx.Lieferadr = alle.Lieferadr
  AND maxx.LieferPLZ = alle.PLZ
  AND maxx.Klingel = alle.Klingel
  AND maxx.anzahl = alle.anzahl
```

f) Alice macht gleich nach der in der Themenstellung beschriebenen Bestellung noch weitere Bestellungen bei anderen Anbietern *an dieselbe Lieferadresse (und denselben Empfängername)* (z.B. bestellt sie noch 2 Portionen *Pommes Frites* bei *Kalle's Imbiss*). Aus diesem Grund wartet das System nach Eingang einer Bestellung immer einige Minuten, ob noch weitere Bestellungen mit demselben Ziel eingehen.

Wenn dann eine solche –noch offene– Bestellung einem Fahrer zugeordnet werden soll, wird im optimalen Fall ein einziger Fahrer für alle *noch nicht bearbeiteten* Bestellungen, die an dieselbe Lieferadresse ausgeliefert werden sollen, zugeordnet.

Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die Namen derjenigen zur Zeit diensthabenden Fahrer ausgibt, die *jede* der neuen, noch nicht zugeordneten Bestellungen an *Schmidt*, *Hinter dem Mond 15, 30423*, abholen und ausliefern können. (6+6 P)

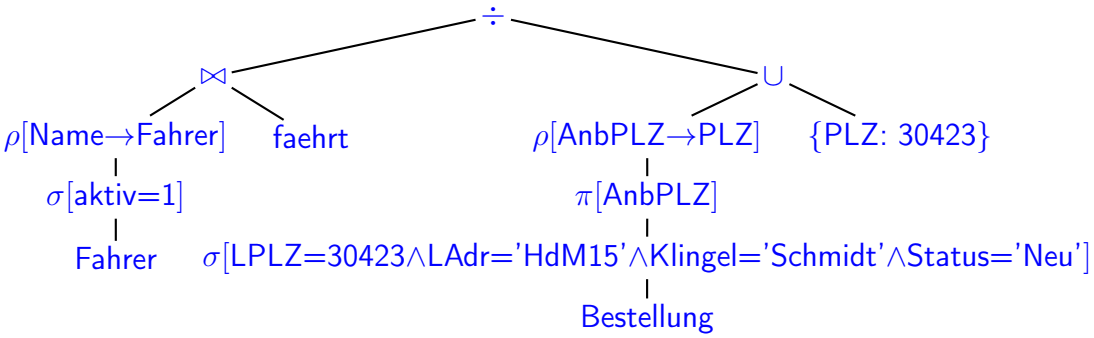
Lösung

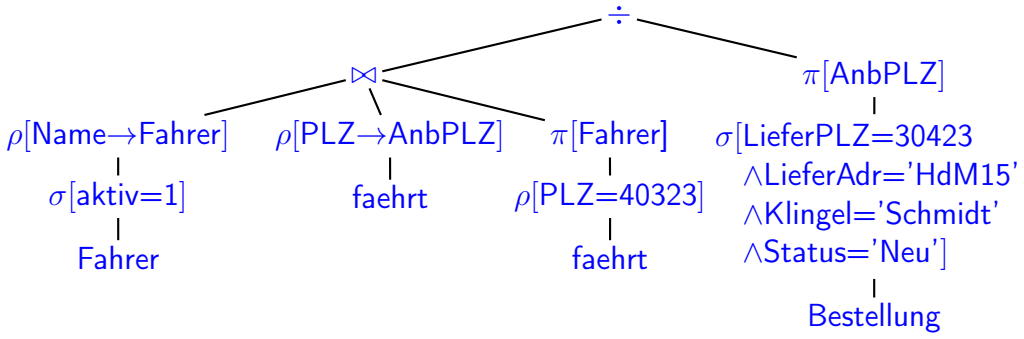
```
SELECT Name
FROM Fahrer f
WHERE aktiv = 1
  AND (f.name, '30423') in (select * from faehrt)
  AND NOT EXISTS
     ( SELECT *
       FROM Bestellung b
       WHERE LieferAdr = 'Hinter dem Mond 15'
         AND LieferPLZ = '30423'
         AND Klingel = 'Schmidt'
         AND Status = 'Neu'                <<<<<<<<< !!!
         AND NOT EXISTS (
             SELECT *
             FROM faehrt f2
             WHERE f2.Fahrer = f.Name
               AND f2.PLZ = b.AnbPLZ   ))
```

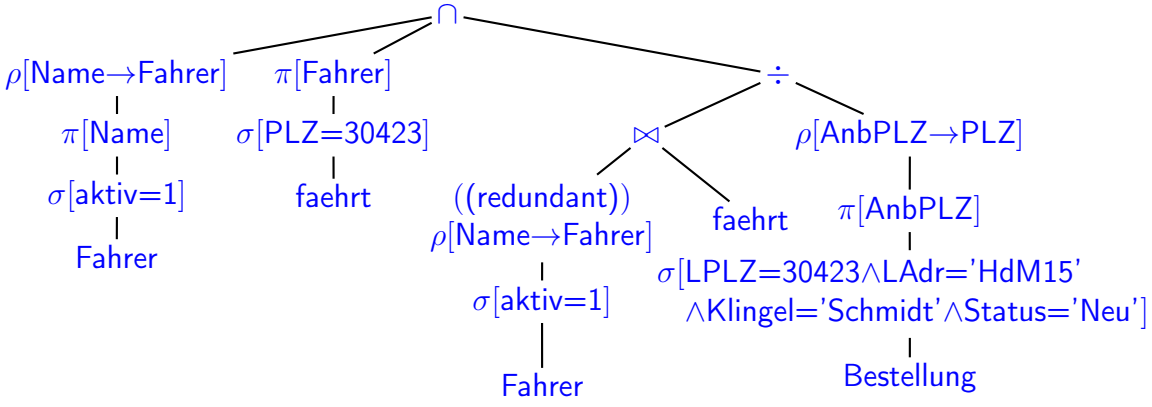
```
SELECT Name
FROM Fahrer f
WHERE aktiv = 1
  AND (f.name, '30423') in (select * from faehrt)
  AND NOT EXISTS
     ( SELECT *
       FROM Bestellung b
       WHERE LieferAdr = 'Hinter dem Mond 15'
         AND LieferPLZ = '30423'
         AND Klingel = 'Schmidt'
         AND Status = 'Neu'                <<<<<<<<< !!!
```

```
        AND NOT (f.name, b.AnbPLZ) IN
                (SELECT fahrer, plz FROM faehrt))

SELECT Name
FROM Fahrer f
WHERE aktiv = 1
  AND NOT EXISTS (
   ( SELECT *
     FROM ((SELECT AnbieterPLZ as PLZ
            FROM Bestellung b
            WHERE LieferAdr = 'Hinter dem Mond 15'
               AND LieferPLZ = '30423'
               AND Klingel = 'Schmidt'
               AND Status = 'Neu')             <<<<<<<<< !!!
            UNION
           (SELECT '30423' as PLZ FROM DUAL)) plzs
     WHERE NOT EXISTS (
            SELECT *
            FROM faehrt f2
            WHERE f2.Fahrer = f.Name
              AND f2.PLZ = plzs.plz)))

SELECT Name
FROM Fahrer f
WHERE aktiv = 1
  AND NOT EXISTS
  ( SELECT *
     FROM Bestellung b
     WHERE LieferAdr = 'Hinter dem Mond 15'
       AND LieferPLZ = '30423'
       AND Klingel = 'Schmidt'
       AND Status = 'Neu'
       AND NOT EXISTS
         (SELECT *
          FROM faehrt f1, faehrt f2
          WHERE f1.PLZ = b.LieferPLZ AND f2.PLZ = b.AnbPLZ
            AND f1.Fahrer = f.Fahrer AND f2.Fahrer = f.Fahrer))

### Hier mal eine Loesung mit MINUS:
( SELECT Name
  FROM Fahrer f
  WHERE aktiv = 1)
MINUS
( SELECT nickname
  FROM (( SELECT f2.nickname, b.AnbPLZ, b.LieferPLZ ### was er fahren soll
          FROM Fahrer f2, Bestellung    ### kart Produkt
          WHERE LieferAdr = 'Hinter dem Mond 15'
            AND LieferPLZ = '30423'
            AND Klingel = 'Schmidt'
```

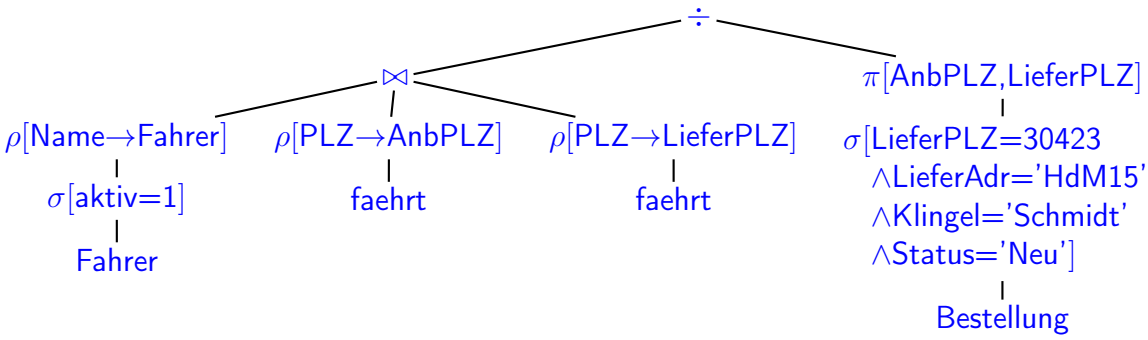
```
      AND Status = 'Neu' )
MINUS
(SELECT f3.Fahrer, f3.PLZ as AnbPLZ, f4.PLZ as LieferPLZ
  FROM faehrt f3, faehrt f4              ### was er fahren kann
  WHERE f3.Fahrer = f4.Fahrer)))
```



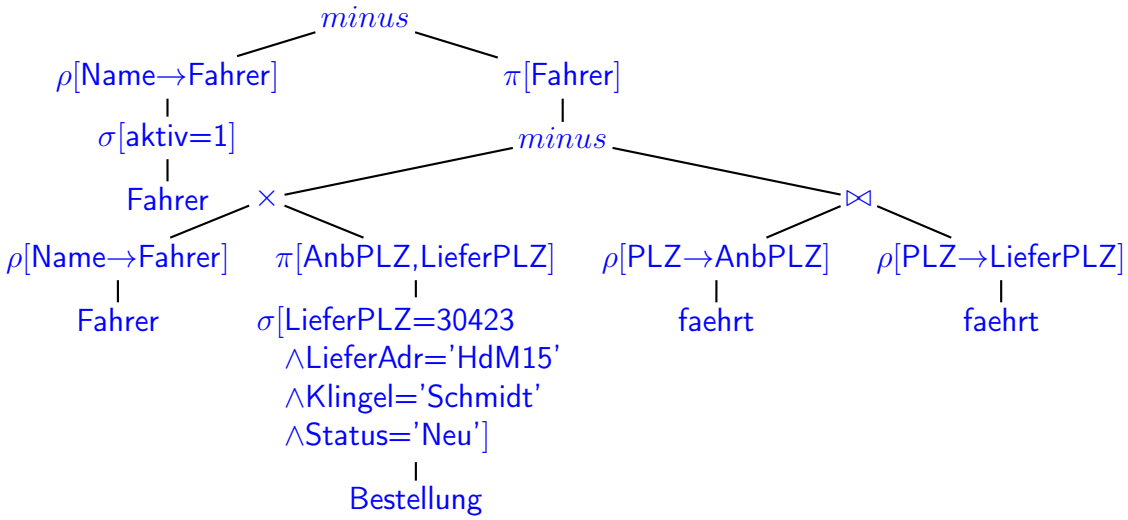




Schön ist hier auch eine Lösung, wo auf der rechten Seite der Division mal eine zweistellige Relation steht: der Fahrer muss alle Paare (AnbPLZ,LieferPLZ=40323) bedienen (diese Lösung wäre auch für allgemeinere Mengen von Fahrten geeignet):



Und dasselbe mit *minus* ausgedrückt:



g) Ein bisschen Theorie. (10 P)

Seien $R(\bar{A}, B)$ eine Relation, und Q eine Anfrage mit Format $[B]$ mit nicht-leerem Ergebnis.

Betrachten Sie die Ausdrücke

(1) $\pi[\bar{A}](R \ltimes Q)$ (Hinweis: $\ltimes$ ist ein linkes Semijoin[1])

(2) $R \div Q$

Hinweis: Q könnte z.B. die Anfrage aus (3a) sein (die die Namen aller einfachen Pizzen ergibt), und R eine geeignete Projektion der Relation, in der gespeichert ist, welche Anbieter welche Gerichte anbieten.

– Zeigen sie, dass (1) = (2) nicht immer gilt. (2 P)

Lösung

R	
A	B
Pizzawelt1	Pizza Margarita
Pizzawelt1	Pizza Salami
Pizzawelt2	Pizza Margarita

Q
B
Pizza Margarita
Pizza Salami

[1] von rechten Semijoins distanzieren wir uns hiermit und lehnen jede Zusammenarbeit mit ihnen ab.

(1) ist dann {Pizzawelt1, Pizzawelt2} – jeder Anbieter, der *irgendeine* Pizza aus Q anbietet, und (2) ist nur { Pizzawelt1}, bei der *alle* Pizzen aus Q zu kaufen sind.

– Ist wenigstens eine der beiden folgenden Aussagen allgemeingültig? (4 P)

ja	nein		
☐	☒	(1) $\subseteq$ (2)	(1P)
☒	☐	(1) $\supseteq$ (2)	(3P)

(mit Beweis oder fundierter Begründung)

Lösung Die erste Aussage ergibt sich aus obigem Gegenbeispiel. Die zweite aus dem Text oben: (1) sind alle $\pi[A](R)$, die mit *irgendeinem* $q \in Q$ zusammen in R vorkommen, und (2) sind alle $\pi[A](R)$, die mit *jedem* $q \in Q$ zusammen in R vorkommen. Die Bedingung (1) ("exists") ist also schwächer als (2) ("forall") (da $Q \neq \emptyset$ ist).
Formal:

$$\begin{aligned}
(1) &= \{\mu \in \mathsf{Tup}(\bar{A}) \mid \exists \beta \in \mathsf{Tup}(B) : \mu\beta \in R \text{ und } \beta = \pi[B](\mu\beta) \in Q\} \\
&= \{\mu \in \mathsf{Tup}(\bar{A}) \mid \exists \beta \in Q : \mu\beta \in R\} \\
&\quad \text{da } Q \neq \emptyset\text{:} \\
&\supseteq \{\mu \in \mathsf{Tup}(\bar{A})| \mid \forall \beta \in Q : \mu\beta \in R\} \\
&= \{\mu \in \mathsf{Tup}(\bar{A}) \mid \{\mu\} \times Q \subseteq R\} = R \div Q = (2)
\end{aligned}$$

– Was bedeutet es, für den Zusammenhang der Relation R mit einer festen Ergebnisrelation B_0 von Q (die o.E.d.A. mehr als ein Element enthält), wenn die Gleichheit (1) = (2) im aktuellen Datenbankzustand gilt. (3 P)

Lösung Abstrakt: Wenn ein $a \in \pi[\bar{A}](R)$ mit *einem* der $b \in B_0$ in R auftritt (1), dann tritt dieses a mit *jedem* $b \in B_0$ in R auf (2).
Bzw.
Für jedes Tupel $\mu \in \pi[\bar{A}](R)$ gilt: wenn es ein Tupel $\beta \in B_0$ gibt, so dass das Tupel $\mu\beta \in R$ ist, dann ist für jedes $\beta' \in B_0$ auch das Tupel $\mu\beta' \in R$.
Bzw.
Formal: Für jedes Tupel $\mu \in \pi[\bar{A}](R)$ gilt: Entweder $\{\mu\beta|\beta \in B_0\} \cap R = \emptyset$ oder $\{\mu\beta \mid \beta \in B_0\} \subseteq R$
Mit Pizzas (Q als die Anfrage aus Aufgabe 3a) ausgedrückt:
Wenn diese Gleichheit gilt, hat jeder Anbieter, der mindestens *eine* der billigen Pizzavarianten (B_0) anbietet, *alle* billigen Pizzavarianten im Angebot (wobei sie bei ihm durchaus auch mehr als 5 EUR kosten können).

Aufgabe 4 (Verschiedenes [14 Punkte])

a) Fahrer Timo bekommt einen Studienplatz und kündigt.

a1) Welche Daten von ihm müssen in der Datenbank erhalten bleiben, und warum? (2P)

Lösung

* inhaltlich: es muss erhalten bleiben, wann wer was zu wem ausgeliefert hat. Man sollte damit auch vermeiden, dass derselbe Nickname später wieder an einen anderen Fahrer vergeben wird.
* technisch: Fahrer.Nickname muss erhalten bleiben, weil es von einem Foreign Key referenziert wird.
* datenbanktechnisch möglich wäre auch, alle entsprechenden Einträge in Bestellung.Fahrer auf NULL zu setzen. Der Foreign key würde damit nicht verletzt.

a2) Welche Daten über ihn werden sinnvollerweise gelöscht?
Geben Sie an, mit welchem SQL-Statement(s) dies auf dem Datenbankserver getan werden kann. (3P)

Lösung

Die Handynr sollte in der (operativen) Datenbank gelöscht werden (die administrative Datenbank der Lohnbuchhaltung, in der seine "echten" schützenswerten und n Jahre zu speichernden Daten gespeichert sind, hat hiermit nichts zu tun). In welchen Gebieten er fährt (bzw. fuhr), kann auch gelöscht werden.

(Einwand: "man muss ja eine Nachfolge für ihn organisieren, die diese Gebiete versorgt" – es lässt sich auch einfach aus den ausgelieferten Bestellungen herausfinden, in welchen Gebieten mehr oder weniger Bedarf bestand).

```
UPDATE Fahrer SET handynr = null WHERE name='Timo';
  -- 'aktiv' wird nach der Schicht sowieso auf 0 gesetzt, und er
  -- wird nie wieder eingeteilt. Man koennte aber auch das auf null
  -- setzen.
DELETE FROM faehrt WHERE name='Timo';
```

a3) Die diese Statement(s) ausführende Person verschreibt sich beim Tippen des SQL-Statements und hat statt "Timo" "Tilo" geschrieben, den es auch gibt. Er merkt es sofort, als die Datenbank ausgibt

```
sql> ... delete or update-statement ...
successfully deleted/updated ... rows
sql>
```

Was ist jetzt das Problem? (1P, kurz)

Lösung Tilo ist "weg". Seinen Namen sieht der Eingebende noch, aber niemand weiss mehr, welche Handynr Tilo hat, und in welchen Gebieten Tilo fährt (und man kann ihn auch nicht schnell fragen, weil man ja auch die Handynr nicht mehr hat). (Die Gebiete könnte man aus den ausgelieferten Bestellungen rekonstruieren).

– Wie kann man am besten die Auswirkungen beheben? (2P)

Lösung

* die beste: ROLLBACK und dann (a2) richtig ausführen.
* technisch richtig, aber in der Situation nicht notwendig: letztes Backup einspielen, und alle committeten Transaktionen anhand des logs nochmal ausführen lassen.
* Nichtbeste Lösung: (a2) richtig ausführen, und warten, bis Tilo mal wieder ins Büro kommt oder anruft, weil er keine Aufträge mehr bekommt.

b) Bei der Auswertung der Anfrage in Aufgabe 3f (der letzten Anfrage) müssen alle Bestellungen, auch die bereits schon lange erledigten, geprüft werden, ob sie die Bedingungen erfüllen.

Das sollte man effizienter lösen.

b1) Warum ist die Antwort "das sind doch sowieso die neuesten, die sind ganz schnell zu finden, meistens sogar noch im Cache" falsch? (2P)

Lösung Die Datenbank weiss nicht, dass alle relevanten Ergebnisse so schnell zu finden sind, sondern sucht (erfolglos) trotzdem in den alten (abgeschlossenen) Bestellungen auch. Denken kann sie nämlich nicht.

b2) Beschreiben Sie eine Möglichkeit (es gibt einige, ganz verschiedene), wie man das System in diesem Aspekt effizienter gestalten kann? (3P)

Lösung

Indexe in der Datenbank:

* einen Index auf Bestellung.Status (ein schönes Beispiel, wo ein Bitlisten-Index ideal ist, aber jeder andere Index tut es auch). Dieser wird automatisch bei der Anfrageauswertung verwendet.

oder

* man nimmt noch `Datum=today` (in Oracle heisst das `SYSDATE`) in die Anfrage dazu, ggf. auch noch eine Zeitbeschränkung. Auch das bringt aber nur etwas, wenn man auf Datum (und ggf. Zeit) auch einen (Baum-)Index anlegt. Sonst werden doch wieder alle Daten durchsucht.

Vertikale Partitionierung der Tabelle: Man teilt die Tabelle "Bestellung" in "ErledigteBestellungen" und "AktiveBestellungen" auf, und stellt die Anfrage nur an "AktiveBestellungen" (das beschleunigt auch alle Updates bei den weiteren Statuswechseln). Erst wenn die Bestellung ausgeliefert wird, wird sie in "ErledigteBestellungen" verlagert (dort kann dann auf die Status-Spalte verzichtet werden).
Analog kann man "HeutigeBestellungen" und "AlteBestellungen" aufteilen und nach Feierabend wegkopieren.

Im Anwendungsprogramm: Aktive Bestellungen werden garnicht in der Datenbank verwaltet, sondern in einer globalen Datenstruktur des Anwendungsprogrammes [Vorsicht bei Web-Services: selbst wenn eine Instanz des Servlets eine globale Variable besitzt, kann ein anderer Aufruf eine andere Instanz des

Servlets verwenden – man muss also separaten eigenen Prozess für die Datenstruktur haben]. Erst wenn sie erledigt sind, werden sie in die DB kopiert. Risiko dabei ist, dass man dann bei einem kurzen Stromausfall alle aktiven Bestellungen verlieren kann.