AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

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

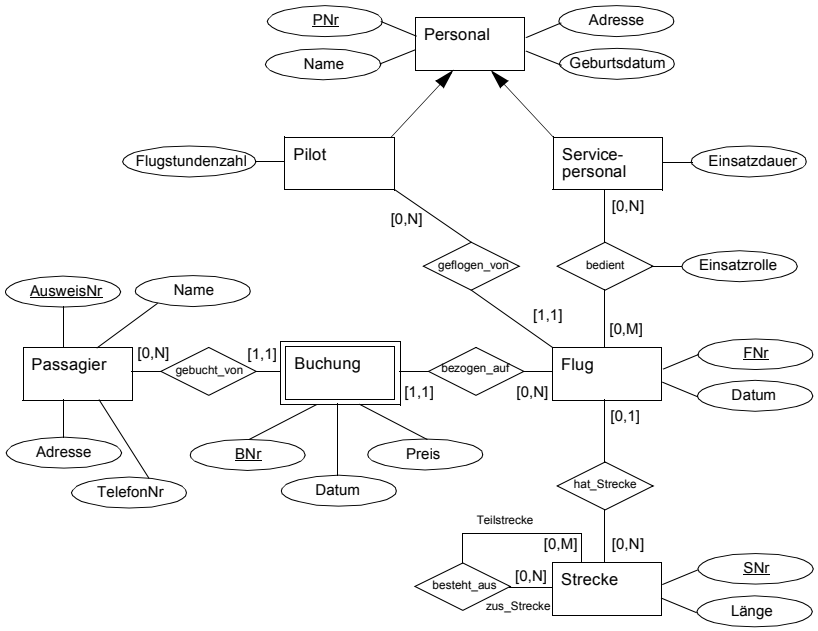
http://wwwdvs.informatik.uni-kl.de

1. Übungsblatt

Für die Übung am Donnerstag, **2. November 2006**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: E/R-Diagramm nach SQL-Schema am Beispiel „Flughafen“

Überführen Sie folgendes E/R-Diagramm in ein SQL-Schema. Modellieren Sie das Schema möglichst genau mit Hilfe von Integritätsbedingungen. Für die Abbildung der Generalisierungsbeziehungen setzen Sie jeweils das Hausklassenmodell und die vertikale Partitionierung ein.



Weiterhin formulieren Sie auf beiden DB-Schemata die folgende Anfrage in SQL:

- Liste die Namen des Personals auf, das am ’01.01.1970’ geboren wurde.

Lösung:

Für die Abbildung der Generalisierungsbeziehung soll entweder das Hausklassenmodell oder die vertikale Partitionierung eingesetzt werden. In beiden Fällen muss die Anwendung für die Zuordnung der Daten zu den entsprechenden Tabellen verantwortlich sein, damit es keine Redundanzen gibt. Weiterhin wird bei der vertikalen Partitionierung durch die Fremdschlüsselbeziehung zwischen den Primärschlüsseln einer Unter- und einer Ober-Entity-Menge ein eindeutiger Schlüssel für alle Tupel innerhalb der Generalisierungshierarchie gewährleistet.

Hausklassenmodell:

```
CREATE TABLE PERSONAL (
       PNR             INTEGER            PRIMARY KEY,
       NAME            VARCHAR(40)        NOT NULL,
       ADRESSE         VARCHAR(60)        NOT NULL,
       GEBURTSDATUM    DATE               NOT NULL);

CREATE TABLE PILOT (
       PNR             INTEGER            PRIMARY KEY,
       NAME            VARCHAR(40)        NOT NULL,
       ADRESSE         VARCHAR(60)        NOT NULL,
       GEBURTSDATUM    DATE               NOT NULL,
       FLUGSTUNDENZAHL INTEGER            NOT NULL);

CREATE TABLE SERVICEPERSONAL (
       PNR             INTEGER            PRIMARY KEY,
       NAME            VARCHAR(40)        NOT NULL,
       ADRESSE         VARCHAR(60)        NOT NULL,
       GEBURTSDATUM    DATE               NOT NULL,
       EINSATZDAUER    INTEGER            NOT NULL);
```

Vertikale Partitionierung:

```
CREATE TABLE PERSONAL (
       PNR             INTEGER            PRIMARY KEY,
       NAME            VARCHAR(40)        NOT NULL,
       ADRESSE         VARCHAR(60)        NOT NULL,
       GEBURTSDATUM    DATE               NOT NULL);

CREATE TABLE PILOT (
       PNR               INTEGER          PRIMARY KEY,
       FLUGSTUNDENZAHL   INTEGER          NOT NULL,
       FOREIGN KEY (PNR) REFERENCES PERSONAL (PNR)
                                          ON DELETE CASCADE
                                          ON UPDATE CASCADE);

CREATE TABLE SERVICEPERSONAL (
       PNR             INTEGER            PRIMARY KEY,
       EINSATZDAUER    INTEGER            NOT NULL,
       FOREIGN KEY (PNR) REFERENCES PERSONAL (PNR)
                                          ON DELETE CASCADE
                                          ON UPDATE CASCADE);
```

Restliche Tabellen:

```
CREATE TABLE STRECKE (
       SNR             INTEGER             PRIMARY KEY,
       LAENGE          INTEGER             NOT NULL);

CREATE TABLE BESTEHT_AUS (
       ZUS_SNR         INTEGER,
       TEIL_SNR        INTEGER,
       PRIMARY KEY (ZUS_SNR, TEIL_SNR),
       FOREIGN KEY (ZUS_SNR) REFERENCES STRECKE (SNR),
       FOREIGN KEY (TEIL_SNR) REFERENCES SRTECKE (SNR));

CREATE TABLE FLUG (
       FNR             INTEGER             PRIMARY KEY,
       DATUM           DATE                NOT NULL,
       GEFLOGEN_VON    INTEGER             NOT NULL,
       HAT_STRECKE     INTEGER,
       FOREIGN KEY (GEFLOGEN_VON) REFERENCES PILOT (PNR),
       FOREIGN KEY (HAT_STRECKE) REFERENCES STRECKE (SNR));

CREATE TABLE BEDIENT (
       PNR             INTEGER,
       FNR             INTEGER,
       EINSATZROLLE    VARCHAR(20)         NOT NULL,
       PRIMARY KEY (PNR, FNR),
       FOREIGN KEY (PNR) REFERENCES SERVICEPERSONAL (PNR),
       FOREIGN KEY (FNR) REFERENCES FLUG (FNR));

CREATE TABLE PASSAGIER (
       AUSWEISNR       INTEGER             PRIMARY KEY,
       NAME            VARCHAR(40)         NOT NULL,
       ADRESSE         VARCHAR(60)         NOT NULL,
       TELEFONNR       VARCHAR(20)         NOT NULL);


CREATE TABLE BUCHUNG (
       BNR             INTEGER             PRIMARY KEY,
       DATUM           DATE                NOT NULL,
       PREIS           DECIMAL(6,2)        NOT NULL,
       GEBUCHT_VON     INTEGER             NOT NULL,
       BEZOGEN_AUF     INTEGER             NOT NULL,
       FOREIGN KEY (GEBUCHT_VON) REFERENCES PASSAGIER (AUSWEISNR)
                       ON DELETE CASCADE ON UPDATE CASCADE,
       FOREIGN KEY (BEZOGEN_AUF) REFERENCES FLUG (FNR)
                       ON DELETE CASCADE ON UPDATE CASCADE);
```

Anfrage: Liste die Namen des Personals, das am 01.01.1970 geboren wurde.

Hausklassen-Modell:

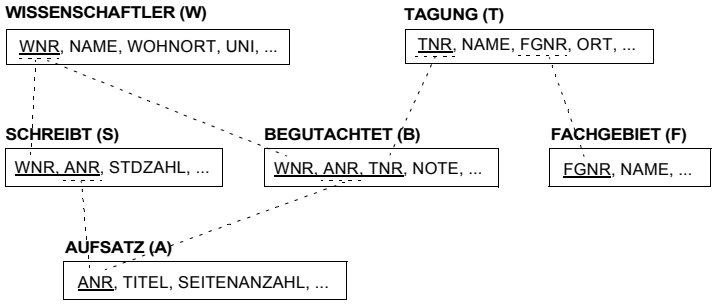
```
SELECT NAME FROM PERSONAL WHERE GEBURTSDATUM = "01.01.1970"
UNION
SELECT NAME FROM PILOT WHERE GEBURTSDATUM = "01.01.1970"
UNION
SELECT NAME FROM SERVICEPERSONAL WHERE GEBURTSDATUM = "01.01.1970"
```

Vertikale Partitionierung:

```
SELECT NAME FROM PERSONAL WHERE GEBURTSDATUM = "01.01.1970"
```


Aufgabe 2: Anfragen in SQL und Relationenalgebra am Beispiel „Wissenschaftlicher Aufsatz“

Gegeben sei folgendes DB-Schema, das mit den relevanten Attributen sowie referentiellen Beziehungen graphisch dargestellt ist.



Formulieren Sie folgende Anfragen in SQL und – soweit möglich – mit relationaler Algebra:

a) Finde die Wissenschaftler an der ’TU KL’, die nicht in ’Kaiserslautern’ wohnen.

b) Finde alle Tagungen, die zum Fachgebiet ’Datenbanken’ oder ’Informationssysteme’ gehören.

c) Finde die Wissenschaftler, die Aufsätze mit Note 1 für in ihrem Wohnort stattfindende Tagungen im Fachgebiet ’Datenbanken’ begutachtet haben.

d) Liste alle Tagungen (TNR), ihre zugehörigen Aufsätze (ANR) sowie die Durchschnittsnote der Aufsätze.

e) Liste die Nummern und Titel der Aufsätze, an denen alle ihre Autoren zusammengenommen mehr als 100 Stunden geschrieben haben.

f) Liste die Namen der Wissenschaftler, die (mindestens) einen ihrer Aufsätze selbst begutachtet haben, die Titel der Aufsätze sowie die Begutachtungsnoten. Die Ausgabeliste soll aufsteigend nach Namen, aufsteigend nach Aufsatztitel und absteigend nach Note sortiert werden.

g) Liste die Aufsätze (ANR), die in mehreren Tagungen begutachtet wurden und deren Gesamtdurchschnittsnote besser als 4 ist.

h) Finde die Wissenschaftler, die noch keinen einzigen Aufsatz geschrieben haben aber mindestens einmal als Gutachter tätig waren.

i) Finde alle Aufsätze, die in allen Tagungen begutachtet wurden.

Lösung:

a) Finde die Wissenschaftler an der 'TU KL', die nicht in 'Kaiserslautern' wohnen.

```
SELECT *
FROM   WISSENSCHAFTLER
WHERE  UNI = 'TU KL' AND WOHNORT <> 'Kaiserslautern'
```

$\sigma_{\text{UNI='TU KL'} \wedge \text{WOHNORT<>'Kaiserslautern'}} W$

b) Finde alle Tagungen, die zum Fachgebiet 'Datenbanken' oder 'Informationssysteme' gehören.

```
SELECT T.*
FROM   TAGUNG T, FACHGEBIET F
WHERE  T.FGNR = F.FGNR
AND    (F.NAME = 'Datenbanken' OR F.NAME = 'Informationssysteme')
```

ODER:

```
SELECT *
FROM   TAGUNG
WHERE  FGNR IN
       (SELECT        FGNR
        FROM          FACHGEBIET
        WHERE         NAME IN ('Datenbanken', 'Informationssysteme')
       )
```

$\pi_{\text{TNR, T.NAME, ...}} (\sigma_{\text{F.NAME='Datenbanken'} \vee \text{F.NAME='Informationssysteme'}} F) \underset{F.FGNR=T.FGNR}{\bowtie} T$

c) Finde die Wissenschaftler, die Aufsätze mit Note 1 für in ihrem Wohnort stattfindende Tagungen im Fachgebiet 'Datenbanken' begutachtet haben.

```
SELECT DISTINCT W.*
FROM   WISSENSCHAFTLER W, BEGUTACHTET B, TAGUNG T, FACHGEBIET F
WHERE  W.WNR = B.WNR
AND    B.TNR = T.TNR
AND    T.FGNR = F.FGNR
AND    F.NAME = 'Datenbanken'
AND    B.NOTE = 1
AND    W.WOHNORT = T.ORT
```

$\pi_{\text{WNR, W.NAME, ...}} (\sigma_{\text{W.WOHNORT=T.ORT}} (W \bowtie (\sigma_{\text{NOTE=1}} B) \underset{B.TNR=T.TNR}{\bowtie} T) \underset{T.FGNR=F.FGNR}{\bowtie} (\sigma_{\text{NAME='Datenbanken'}} F))$

d) Liste alle Tagungen (TNR) , ihre zugehörigen Aufsätze (ANR) sowie die Durchschnittsnote der Aufsätze.

```
SELECT TNR, ANR, AVG(NOTE)
FROM   BEGUTACHTET
GROUP BY TNR, ANR
```

Mit Relationenalgebra nicht möglich, da **keine Aggregationsfunktionen unterstützt** werden.

e) Liste die Nummern und Titel der Aufsätze, an denen alle ihre Autoren zusammengenommen mehr als 100 Stunden geschrieben haben.

```
SELECT A.ANR, A.TITEL
FROM   SCHREIBT S, AUFSATZ A
WHERE  S.ANR = A.ANR
GROUP BY A.ANR, A.TITEL
HAVING SUM(STDZAHL) >100
AND    S.STDZAHL > 100
```

Mit Relationenalgebra nicht möglich, da **keine Aggregationsfunktionen unterstützt** werden.

f) Liste die Namen der Wissenschaftler, die (mindestens) einen ihrer Aufsätze selbst begutachtet haben, die Titel der Aufsätze sowie die Begutachtungsnoten. Die Ausgabeliste soll aufsteigend nach Namen, aufsteigend nach Aufsatztitel und absteigend nach Note sortiert werden.

```
SELECT DISTINCT W.NAME, A.TITEL, B.NOTE
FROM   WISSENSCHAFTLER W, BEGUTACHTET B, AUFSATZ A, SCHREIBT S
WHERE  W.WNR = S.WNR
AND    S.ANR = A.ANR
AND    W.WNR = B.WNR
AND    B.ANR = A.ANR
ORDER BY W.NAME ASC, A.TITEL ASC, B.NOTE DESC
```

Mit Relationenalgebra nicht möglich, da **keine Sortierung unterstützt** wird. Falls Sortierung nicht notwendig ist, sieht die Anfrage so aus:

$\pi_{\text{NAME, TITEL, NOTE}} (W \bowtie B \bowtie A \bowtie S)$

g) Liste die Aufsätze (ANR), die in mehreren Tagungen begutachtet wurden und deren Gesamtdurchschnittsnote besser als 4 ist.

```
SELECT B.ANR
FROM   BEGUTACHTET B
WHERE  1 <
       (SELECT  COUNT(DISTINCT B2.TNR)
        FROM    BEGUTACHTET B2
        WHERE   B.ANR = B2.ANR
       )
GROUP BY B.ANR
HAVING AVG (B.NOTE) < 4

SELECT ANR
FROM BEGUTACHTET
GROUP BY ANR
HAVING AVG(NOTE)<4 AND COUNT(DISTINCT TNR)>1
```

Mit Relationenalgebra nicht möglich, da **keine Aggregationsfunktionen unterstützt** werden.

h) Finde die Wissenschaftler, die noch keinen einzigen Aufsatz geschrieben haben aber mindestens einmal als Gutachter tätig waren.

```
SELECT  *
FROM    WISSENSCHAFTLER W
WHERE   NOT EXISTS
        (SELECT  *
         FROM    SCHREIBT S
         WHERE   S.WNR = W.WNR
        )
AND     EXISTS
        (SELECT  *
         FROM    BEGUTACHTET B
         WHERE   B.WNR = W.WNR
        )
```

$\pi_{WNR, NAME, ...}(((\pi_{WNR} W) - (\pi_{WNR} S)) \bowtie W \bowtie B)$

i) Finde alle Aufsätze, die in allen Tagungen begutachtet wurden.

```
SELECT  *
FROM    AUFSATZ
WHERE   NOT EXISTS
        (SELECT  *
         FROM    TAGUNG T
         WHERE   NOT EXISTS
                 (SELECT  *
                  FROM    BEGUTACHTET B
                  WHERE   A.ANR = B.ANR
                  AND     T.TNR = B.TNR
                 )
        )
```

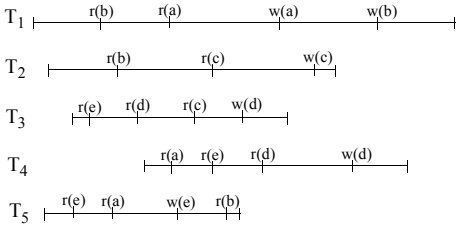
"Finde **alle**..., die **alle**..." führt (meistens) zu **zweifach geschachteltem** NOT EXISTS !

$A \bowtie ((\pi_{ANR, TNR} B) \div (\pi_{TNR} T))$

Ergänzung: SELECT ohne DISTINCT von Nicht-Schlüsselattributen kann nicht auf die RA abgebildet werden, da die Projektion der RA immer Duplikate entfernt!

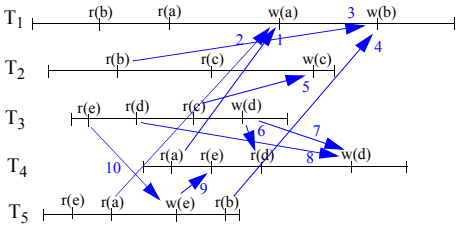
Aufgabe 3: Serialisierbarkeit

Gegeben sei folgender nebenläufiger Ablauf der Transaktionen T1-T5:

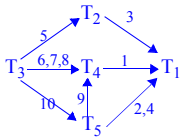


Ermitteln Sie die Konfliktoperationen und geben Sie den Serialisierbarkeitsgraphen an. Ist der Ablauf der Transaktionen serialisierbar (Begründung!)? Falls ja, geben Sie alle möglichen seriellen Ablauffolgen an.

Lösung:



Serialisierbarkeitsgraph SG(H)



Der Ablaufplan ist serialisierbar, da der Serialisierbarkeitsgraph keine Zyklen enthält.
Durch topologische Sortierung erhält man als mögliche serielle Ablaufpläne die Reihenfolgen:
T3, T5, T4, T2, T1
T3, T5, T2, T4, T1
T3, T2, T5, T4, T1