Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

*Prof. H.-J. Schek*

*A. Feßler, U. Röhm, H. Schuldt, R. Weber*

# Informationssysteme IS-G SS 1997

## Übung 7 — Beispiellösung

### Aufgabe 1: Views

Gegeben sei die folgende View–Definition:

```
CREATE VIEW Lazy_Student AS
       SELECT *
       FROM   Student S
       WHERE  Semester ≥ 4
       AND    NOT EXISTS ( SELECT *
                           FROM   Prüfung P
                           WHERE  S.ETH_Nr = P.ETH_Nr)
```

a) Formulieren Sie die folgende Anfrage so um, dass Sie obige View nicht mehr verwenden.

```
SELECT S.ETH_Nr, S.S_Name, P.P_Name, P.Tel
FROM   Professor P, Abteilung A, Lazy_Student S
WHERE  S.Abt_Nr = A.Abt_Nr
AND    A.Vorsteher = P.P_Name
```

```
SELECT S.ETH_Nr, S.S_Name, P.P_Name, P.Tel
FROM   Professor P, Abteilung A, Student S
WHERE  S.Abt_Nr = A.Abt_Nr
AND    A.Vorsteher = P.P_Name
AND    Semester ≥ 4
AND    NOT EXISTS ( SELECT *
                    FROM   Prüfung Pr
                    WHERE  S.ETH_Nr = Pr.ETH_Nr)
```

b) Sie merken, dass der Name des Studenten *Hans Maier*, der als Student im 6. Semester noch immer keine Prüfung absolviert hat, mit y statt mit i geschrieben wurde. Formulieren Sie die Änderungsoperation in SQL auf der View. Kann das System diese Operation durchführen? Begründen Sie Ihre Antwort und geben Sie gegebenenfalls Alternativen an.

```
UPDATE Lazy_Student
SET S_Name = 'Hans Maier'
WHERE S_Name = 'Hans Mayer'
AND   Semester = 6
```

Das System kann diese Änderung durchführen, da es eine eindeutige Abbildung von der View auf die Originaltabelle gibt. Die Subquery wird dabei als Subquery, nicht als Join ausgewertet. Auch Oracle führt die Änderung korrekt aus.



c) Hans Mayer hat nun endlich seine erste Prüfung absolviert. Formulieren Sie die Änderungsoperation in SQL auf der View. Kann das System diese Operation durchführen? Begründen Sie Ihre Antwort und geben Sie gegebenenfalls Alternativen an. Probieren Sie die Änderungsoperation auf Oracle V7 aus. Verhält sich Oracle wie von Ihnen postuliert?

`INSERT INTO Prüfung VALUES ( ETH_Nr von Hans Mayer, 'Fach', 'Prüfer', Note );`

Danach ist Hans Mayer nicht mehr in der View enthalten, da er die Bedingung (noch keine Prüfung abgelegt) nicht mehr erfüllt. Ein anderer Ansatz könnte sein:

`DELETE FROM Lazy_Student WHERE S_Name = 'Hans Mayer'`

Das System kann jetzt aber nicht wissen, ob wir Hans Mayer aus der Tabelle `Student` gelöscht, oder einen neuen Eintrag in der Tabelle `Prüfung` eingefügt haben wollen (dann aber: mit welchen Werten?). Oracle V7 jedenfalls akzeptiert die `DELETE`-Anweisung und löscht den korrespondierenden Eintrag aus der Tabelle `Student` — was bezüglicher unserer Aufgabe semantisch falsch ist.

### Aufgabe 2: Zugriffsrechte

Betrachten Sie drei Benutzer $S_1$, $S_2$ und $S_3$, wobei $S_1$ ein Recht besitzt, das er oder sie weitergeben darf. Diskutieren Sie die folgenden Autorisierungsabläufe:

a)
1. $S_2$ erhält das Recht von $S_1$ und gibt es an $S_3$ weiter.
2. $S_3$ erhält das Recht von $S_1$.
3. $S_2$ entzieht $S_3$ das Recht.

$S_3$ besitzt weiterhin das Recht aufgrund der direkten Zuteilung des Rechts von $S_1$ in Schritt 2.

b)
1. $S_1$ gibt das Recht an $S_2$ und $S_3$ weiter.
2. $S_3$ erhält das Recht von $S_2$.
3. $S_2$ erhält das Recht von $S_3$.
4. $S_1$ entzieht $S_2$ und $S_3$ das Recht.

Sowohl $S_2$, als auch $S_3$ haben nun das Recht entzogen bekommen. Die gegenseitige, mittelbare Zuteilung ist mit dem Entzug seitens $S_1$ hinfällig geworden, da dieser transitiv wirkt.

Geben Sie Algorithmen für die Rechtevergabe an, die die obigen Probleme berücksichtigen.

**GRANT *privilege* TO *user*:** Füge Knoten für den aktuellen Benutzer und *user* in einen Autorisationsgraphen ein, sofern sie dort noch nicht vorhanden sind. Verbinde die beiden Knoten mit einer Kante für das Recht *privilege*.

**REVOKE *privilege* FROM *user*:** Wurde das Recht an *user* direkt vergeben? Wenn ja, entferne die entsprechende Kante im Graphen. Ist *user* im Autorisationsgraphen von der Wurzel (= DB-Administrator) aus bzgl. *privilege* noch erreichbar? Wenn nein, dann führe denselben Algorithmus rekursiv fuer *user* und alle anderen Benutzer aus, denen *user* das Recht weitergegeben hatte. Ansonsten Ende.

Alternativ kann man bei der Weitergabe von Rechten die Kanten mit Nummern versehen, und zwar wird bei jeder Verzweigung bei jedem Knoten der ersten ausgehenden Kante die von der eingehenden Kante „ererbte“ Nummer gegeben und jeder weiteren Kante eine neue Nummer. Wenn man eine neue Kante einfügt und damit auf einen Knoten stößt, auf den schon eine andere Kante zeigt, so müssen alle von diesem Knoten ausgehenden Kanten mit der Nummer der neu eingefügten Kante zusätzlich rekursiv markiert werden. Beim Löschen eines Rechts, d.h. einer Kante, lösch man nun stets alle von der Kante weiterführenden Kanten mit derselben Nummer rekursiv.

Als andere Alternative kann man bei der Weitergabe von Rechten auf Zyklen prüfen und solche verbieten, was den Algorithmus erheblich vereinfacht.

Oracle benutzt den ersten Algorithmus. Sich selbst kann man keine Rechte geben oder entziehen.

**lookup *privilege* for *user*:** Finde den Knoten für *user* im Autorisationsgraphen und prüfe, ob es eine eingehende Kante für *privilege* gibt. Wenn ja, fahre fort, sonst melde Fehler.

### Aufgabe 3: Zugriffsstrukturen

Wir betrachten einen Teil unserer Musikdatenbank mit der folgenden Schemadefinition:

```
CREATE TABLE Disk
  (DiskID        NUMBER PRIMARY KEY NOT NULL,
   DiskTitel     VARCHAR(255));
CREATE TABLE Musikstueck
  (DiskID     NUMBER,
   StueckID   NUMBER,
   Titel      VARCHAR(255),
   PRIMARY KEY (DiskID, StueckID));
CREATE TABLE Person
  (PID           NUMBER PRIMARY KEY NOT NULL,
   Name          VARCHAR(40) NOT NULL,
   Nationalitaet CHAR(5));
CREATE TABLE Musikstueck_Autor
  (PID         NUMBER NOT NULL REFERENCES Person,
   DiskID      NUMBER NOT NULL,
   StueckID    NUMBER NOT NULL,
   Taetigkeit  VARCHAR(20) NOT NULL,
   FOREIGN KEY (DiskID, StueckID) REFERENCES Musikstueck);
```

Als Kardinalitäten und mittlere Tupellängen der verschiedenen Relationen nehmen wir an:

| | | | |
|---|---|---|---|
| Disk | 1000 | Tupel à | 100 Bytes |
| Musikstueck | 10000 | Tupel à | 50 Bytes |
| Person | 6000 | Tupel à | 40 Bytes |
| Musikstueck_Autor | 30000 | Tupel à | 40 Bytes |

Zur Effizienzsteigerung von Anfragen sind unter anderem die folgenden mittelbaren Indexe angelegt, die jeweils als B*-Baum implementiert sind:

```
CREATE UNIQUE INDEX DiskID ON Disk(DiskID);
CREATE INDEX DiskTitel ON Disk(DiskTitel);
CREATE INDEX AutorStueck ON Musikstueck_Autor (PID);
```

Nehmen Sie an, dass jede Relation und jeder Index jeweils einen eigenen Tablespace belegt, wobei die Seitengrösse 4 KBytes sei und der Seiten-Header die Länge 96 Bytes habe. Datenseiten seien zu 90 Prozent gefüllt, Indexseiten zu 70 Prozent. Der Datentyp NUMBER belege 10 Bytes, der Disktitel sei Schlüsselkandidat der Relation `Disk` und habe eine durchschnittliche Länge von 90 Bytes, ein TID oder Zeiger im B*-Baum habe die Länge 6 Bytes. Nehmen Sie zusätzlich an, dass alle Werte eines Fremdschlüsselattributes jeweils gleichhäufig vorkommen (Gleichverteilungsannahme).



Analysieren Sie die Speicherplatz- und Indexzugriffskosten dieser Datenbank. Dabei können Sie den Platzbedarf für die Slotlisten und für die Längenfelder des Datentyps VARCHAR vernachlässigen.

**Abgabetermin dieser Aufgabe verlängert bis 3.6.1997!**

### Aufgabe 4: Pufferverwaltung

Überlegen Sie sich, welche Informationen ein Pufferverwalter für das Bereitstellen (*fix*) und Freigeben (*unfix*) von Seiten benötigt. Unter der Annahme, dass der Datenbankpuffer beliebig gross sei, wollen wir das folgende Szenario betrachten:

Eine Datenbank habe sechs Seiten ($a \ldots f$). Drei parallele Prozesse arbeiten auf diesen Seiten und rufen den Pufferverwalter mit einer Bereitstellungsoperation ($B_a$ = Bereitstellen der Seite $a$) oder einer Freigabeoperation ($F_a$ = Freigeben der Seite $a$) auf. Die Folge der Operationen ist in zeitlicher Reihenfolge gegeben durch:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_1$: | $B_a$ | | $B_b$ | | $F_a$ | $F_b$ | | | $B_e$ | | | |
| $P_2$: | | $B_b$ | | | | | | $B_e$ | | $B_b$ | † | |
| $P_3$: | | | | $B_c$ | | | $B_f$ | | | | | $B_e$ |

| Zeitpunkt: | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | $T_{10}$ | $T_{11}$ | $T_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

a) Welche Seiten befinden sich zum Zeitpunkt $T_7$ im Puffer, ohne dass sie aktuell für einen Prozess bereitgestellt sind? Welche Information wird mindestens benötigt, um dies zu jedem Zeitpunkt feststellen zu können?

Ein Zähler („reference count“) je Pufferseite hilft, nicht mehr benötigte Seiten festzustellen. Zum Zeitpunkt $T_7$ ist der Puffer somit in folgendem Zustand:

| Zeitpunkt | Aktion | Pufferinhalt (LRU-Kette) mit Zählern |
|---|---|---|
| $T_1$ | $B_a$ | $a(1)$ |
| $T_2$ | $B_b$ | $a(1)$ $b(1)$ |
| $T_3$ | $B_b$ | $a(1)$ $b(2)$ |
| $T_4$ | $B_c$ | $a(1)$ $b(2)$ $c(1)$ |
| $T_5$ | $F_a$ | $b(2)$ $c(1)$ $a(0)$ |
| $T_6$ | $F_b$ | $c(1)$ $a(0)$ $b(1)$ |
| $T_7$ | | $c(1)$ $a(0)$ $b(1)$ |

Am Zählerstand 0 der Seite $a$ sieht man, dass diese nun kein Prozess mehr benötigt.

b) Zum Zeitpunkt $T_{12}$ bricht der Prozess $P_2$ ab, ohne „seine“ Seiten freigegeben zu haben. Welche Informationen werden benötigt, um den Puffer wieder in einen „ordentlichen“ Zustand zu überführen?

Der Zähler je Pufferseite gibt nur an, wie häufig eine Seite irgendeinem Prozess bereitgestellt wurde. Zusätzlich müsste noch zu jedem Prozess eine Liste der Seiten geführt werden, die von ihm angefordert wurden. Anhand dieser Liste könnte man dann die Seiten freigeben.