Datenbanksysteme SS 2007

Frank Köster
(Oliver Vornberger)

Institut für Informatik
Universität Osnabrück

Kapitel 7b:
Structured Query Language (SQL)

(vgl. Lerneinheit 07a)

SQL: create (1/3)

```
CREATE TABLE Studenten
       (MatrNr      INTEGER PRIMARY KEY,
        Name        VARCHAR(20) NOT NULL,
        Semester    INTEGER,
        GebDatum    DATETIME);

CREATE TABLE Professoren
       (PersNr      INTEGER PRIMARY KEY,
        Name        VARCHAR(20) NOT NULL,
        Rang        CHAR(2) CHECK (Rang in ('C2','C3','C4')),
        Raum        INTEGER UNIQUE,
        Gebdatum    DATETIME);
```

(vgl. Lerneinheit 07a)

SQL: create (2/3)

```
CREATE TABLE Assistenten
       (PersNr       INTEGER PRIMARY KEY,
        Name         VARCHAR(20) NOT NULL,
        Fachgebiet   VARCHAR(20),
        Boss         INTEGER REFERENCES Professoren,
        GebDatum     DATETIME);

CREATE TABLE Vorlesungen
       (VorlNr       INTEGER PRIMARY KEY,
        Titel        VARCHAR(20),
        SWS          INTEGER,
        gelesenVon   INTEGER REFERENCES Professoren);
```

(vgl. Lerneinheit 07a)

SQL: create (3/3)

```
CREATE TABLE hoeren
       (MatrNr         INTEGER REFERENCES Studenten   ON UPDATE CASCADE
                                                      ON DELETE CASCADE,
        VorlNr         INTEGER REFERENCES Vorlesungen ON UPDATE CASCADE,
        PRIMARY KEY    (MatrNr, VorlNr));

CREATE TABLE voraussetzen
       (Vorgaenger     INTEGER REFERENCES Vorlesungen  ON UPDATE CASCADE,
        Nachfolger     INTEGER REFERENCES Vorlesungen,
        PRIMARY KEY    (Vorgaenger, Nachfolger));

CREATE TABLE pruefen
       (MatrNr         INTEGER REFERENCES Studenten   ON UPDATE CASCADE
                                                      ON DELETE CASCADE,
        VorlNr         INTEGER REFERENCES Vorlesungen ON UPDATE CASCADE,
        PersNr         INTEGER REFERENCES Professoren,
        Note           NUMERIC(3,1) CHECK (Note between 0.7 and 5.0),
        PRIMARY KEY    (MatrNr, VorlNr));
```

(vgl. Lerneinheit 07a)

SQL: insert

```
INSERT INTO Studenten(MatrNr, Name, Semester, GebDatum)
VALUES (24002, 'Xenokrates', 18, '1975-10-23');
…

INSERT INTO Professoren(PersNr, Name, Rang, Raum, GebDatum)
VALUES (2125, 'Sokrates', 'C4', 226,'1923-08-23');
…

INSERT INTO Assistenten(PersNr, Name, Fachgebiet, Boss, GebDatum)
VALUES (3002, 'Platon', 'Ideenlehre', 2125, '1966-08-14');
…

INSERT INTO Vorlesungen(VorlNr, Titel, SWS, gelesenVon)
VALUES (5001, 'Grundzüge', 4, 2137);
…

INSERT INTO hoeren(MatrNr, VorlNr)
VALUES (26120, 5001);
…

INSERT INTO voraussetzen(Vorgaenger, Nachfolger)
VALUES (5001, 5041);
…

INSERT INTO pruefen(MatrNr, VorlNr, PersNr, Note)
VALUES (28106, 5001, 2126, 1.0);
…
```

(vgl. Lerneinheit 07a)

SQL: like, upper, order, distinct

(Queries 1. bis 5. nur in Lerneinheit 07a)

6.) Liste alle Vorlesungen mit **`Ethik`** im Titel:

```
select * from Vorlesungen
where upper(Titel) like '%ETHIK'
```

7.) Liste Personalnummer, Name und Rang aller Professoren, absteigend sortiert nach Rang, innerhalb des Rangs aufsteigend sortiert nach Name:

```
select   PersNr, Name, Rang
from     Professoren
order by Rang desc, Name asc
```

8.) Liste alle verschiedenen Ränge der Relation Professoren:

```
select distinct Rang
from Professoren
```

(vgl. Lerneinheit 07a)

SQL: Verbund

9.) Liste den Dozenten der Vorlesung Logik:

```
select  Name, Titel
from    Professoren, Vorlesungen
where   PersNr = gelesenVon and Titel = 'Logik'
```

10.) Liste die Namen der Studenten mit ihren Vorlesungstiteln:

```
select  Name, Titel
from    Studenten, hoeren, Vorlesungen
where   Studenten.MatrNr = hoeren.MatrNr
and     hoeren.VorlNr    = Vorlesungen.VorlNr
```

alternativ:

```
select  s.Name, v.Titel
from    Studenten s, hoeren h, Vorlesungen
where   s.MatrNr = h.MatrNr
and     h.VorlNr = v.VorlNr
```

(vgl. Lerneinheit 07a)

SQL: Self Join

11.) Liste die Namen der Assistenten, die für denselben Professor arbeiten, für den Aristoteles arbeitet:

```
select  a2.Name
from    Assistenten a1, Assistenten a2
where   a2.boss  =  a1.boss
and     a1.name  = 'Aristoteles'
and     a2.name != 'Aristoteles'
```

(vgl. Lerneinheit 07a)

SQL: avg, group

12.) Liste die durchschnittliche Semesterzahl:

```
select  avg(Semester)
from    Studenten
```

13.) Liste Geburtstage der Gehaltsklassenältesten (ohne Namen!):

```
select   Rang, min(GebDatum) as Ältester
from     Professoren
group by Rang
```

14.) Liste Summe der SWS pro Professor:

```
select   gelesenVon as PersNr, sum(SWS) as
Lehrbelastung
from     Vorlesungen
group by gelesenVon
```

(vgl. Lerneinheit 07a)

SQL: having

15.) Liste Summe der SWS pro Professor, sofern seine Durchschnitts-SWS größer als 3 ist:

```
select gelesenVon as PersNr, sum(SWS) as Lehrbelastung
from   Vorlesungen
group by gelesenVon
having   avg(SWS) > 3
```

(vgl. Lerneinheit 07a)

SQL: where & having

16.) Liste Summe der SWS pro C4-Professor, sofern seine Durchschnitts-SWS größer als 3 ist:

```
select   Name, sum(SWS)
from     Vorlesungen, Professoren
where    gelesenVon = PersNr and Rang='C4,
group by gelesenVon, Name
having   avg(SWS) > 3.0
```

(vgl. Lerneinheit 07a)

SQL: Sub-Query

17.) Liste alle Prüfungen, die als Ergebnis die schlechteste Note haben:

```
select *
from   pruefen
where  Note = (select max(Note) from pruefen)
```

18.) Liste alle Professoren zusammen mit ihrer Lehrbelastung:

```
select PersNr,
       Name,
      (select sum(SWS)
       from Vorlesungen
       where gelesenVon = PersNr) as Lehrbelastung
from Professoren
```

(vgl. Lerneinheit 07a)

SQL: exists

19.) Liste alle Studenten, die älter sind als der jüngste Professor:

```
select s.*
from   Studenten s
where  exists (select p.*
               from Professoren p
               where p.GebDatum > s.GebDatum)
```

alternativ:

```
select s.*
from   Studenten s
where  s.GebDatum < (select max(p.GebDatum)
                     from Professoren p)
```

(vgl. Lerneinheit 07a)

SQL: Verbund mit <

20.) Liste alle Assistenten, die für einen jüngeren Professor arbeiten:

```
select a.*
from   Assistenten a, Professoren p
where  a.Boss = p.PersNr
and    a.GebDatum < p.GebDatum
```

(vgl. Lerneinheit 07a)

SQL: geschachtelt

21.) Liste alle Studenten mit der Zahl ihrer Vorlesungen, sofern diese Zahl größer als 2 ist:

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr,s.Name,count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr = h.MatrNr
      group by s.MatrNr, s.Name) tmp
where tmp.VorlAnzahl > 2
```

(vgl. Lerneinheit 07a)

SQL: geschachtelt

22.) Liste die Namen und Geburtstage der Gehaltsklassen-ältesten:

```
select p.Rang, p.Name, p.GebDatum
from   Professoren p,
       (select Rang, min(GebDatum) as maximum
        from Professoren
        group by Rang) tmp
where  p.Rang = tmp.Rang
       and p.GebDatum = tmp.maximum
```

(vgl. Lerneinheit 07a)

SQL: union

23.) Liste die Vereinigung von Professoren- und Assistenten-Namen:

```
(select Name from Assistenten)
union
(select Name from Professoren)
```

--.) Liste die Differenz von Professoren- und Assistenten-Namen (nur SQL-92):

```
(select Name from Assistenten)
minus
(select Name from Professoren)
```

--.) Liste den Durchschnitt von Professoren- und Assistenten-Namen (nur SQL-92):

```
(select Name from Assistenten)
intersect
(select Name from Professoren)
```

(vgl. Lerneinheit 07a)

SQL: not in / exists

24.) Liste alle Professoren, die keine Vorlesung halten:

```
select Name
from   Professoren
where  PersNr not in ( select gelesenVon
                       from Vorlesungen )
```

alternativ:

```
select Name
from   Professoren
where  not exists ( select *
                    from Vorlesungen
                    where gelesenVon = PersNr )
```

(vgl. Lerneinheit 07a)

SQL: all, some

25.) Liste Studenten mit größter Semesterzahl:

```
select Name
from   Studenten
where  Semester >= all ( select Semester
                         from Studenten )
```

26.) Liste Studenten, die nicht die größte Semesterzahl haben:

```
select Name
from   Studenten
where  Semester < some ( select Semester
                         from Studenten )
```

(vgl. Lerneinheit 07a)

SQL: not, exists

27.) Liste Studenten, die alle 4-stündigen Vorlesungen hören:

```
select s.*
from   Studenten s
where  not exists
       (select *
        from Vorlesungen v
        where v.SWS = 4 and not exists
             (select *
              from hoeren h
              where h.VorlNr = v.VorlNr
                    and h.MatrNr = s.MatrNr
             )
        )
```

(vgl. Lerneinheit 07a)

SQL: transitive Hülle

--.) Transitive Hülle einer rekursiven Relation (nur in Oracle): Liste alle Voraussetzungen für Vorlesung „Der Wiener Kreis“:

```
select Titel
from   Vorlesungen
where  VorlNr in (
        select Vorgaenger
        from voraussetzen
        connect by Nachfolger = prior Vorgaenger
        start with Nachfolger = (
                select VorlNr
                from Vorlesungen
                where Titel = 'Der Wiener Kreis'
        )
)
```

SQL: insert

1.) Füge neue Vorlesung mit einigen Angaben ein:

```
insert into Vorlesungen (VorlNr, Titel, gelesenVon)
values (4711, 'Selber Atmen', 2125)
```

2.) Schicke alle Studenten in die Vorlesung *Selber Atmen*:

```
insert  into hoeren
select  MatrNr, VorlNr
from    Studenten, Vorlesungen
where   Titel = 'Selber Atmen'
```

SQL: update

3.) Erweitere die neue Vorlesung um ihre Semesterwochenstundenzahl:

```
update  vorlesungen
set     SWS=6
where   Titel='Selber Atmen'
```

SQL: delete

4.) Entferne alle Studenten aus der Vorlesung *Selber Atmen*:

```
delete from hoeren
where vorlnr=
      (select VorlNr from Vorlesungen
       where Titel = 'Selber Atmen')
```

5.) Entferne die Vorlesung *Selber Atmen*:

```
delete from Vorlesungen
where titel = 'Selber Atmen'
```

SQL: Sichten

1.) Lege Sicht an für Prüfungen ohne Note:

```
create view pruefenSicht as
select MatrNr, VorlNr, PersNr
from   pruefen
```

2.) Lege Sicht an für Studenten mit ihren Professoren:

```
create  view StudProf (Sname, Semester, Titel, Pname) as
select  s.Name, s.Semester, v.Titel, p.Name
from    Studenten s, hoeren h, Vorlesungen v, Professoren p
where   s.MatrNr       = h.MatrNr
and     h.VorlNr       = v.VorlNr
and     v.gelesenVon   = p.PersNr
```

SQL: Sichten

3.) Lege Sicht an mit Professoren und ihren Durchschnittsnoten:

```
create   view ProfNote (PersNr, Durchschnittsnote) as
select   PersNr, avg (Note)
from     pruefen
group by PersNr
```

SQL: Generalisierung durch Verbund

4.) Lege Untertyp als Verbund von Obertyp und Erweiterung an:

```
create table Angestellte   (PersNr      integer not null,
                            Name       varchar(30) not null);
create table ProfDaten     (PersNr      integer not null,
                            Rang       character(2),
                            Raum       integer);
create table AssiDaten     (PersNr      integer not null,
                            Fachgebiet varchar(30),
                            Boss       integer);


create view Profs as
   select a.persnr, a.name, d.rang, d.raum
   from   Angestellte a, ProfDaten d
   where  a.PersNr = d.PersNr


create view Assis as
   select a.persnr, a.name, d.fachgebiet, d.boss
   from   Angestellte a, AssiDaten d
   where  a.PersNr = d.PersNr
```

SQL: Tabellen und Sichten entfernen

Entferne die Tabellen und Sichten wieder:

```
drop  table Angestellte
drop  table AssiDaten
drop  table ProfDaten
drop  view  Profs
drop  view  Assis
```

SQL: Generalisierung durch Vereinigung

5.) Lege Obertyp als Vereinigung von Untertypen an (zwei der drei Untertypen sind schon vorhanden):

```
create table AndereAngestellte ( PersNr integer not null,
                                 Name   varchar(30) not null )


create view Angestellte as
   (select PersNr, Name from Professoren) union
   (select PersNr, Name from Assistenten) union
   (select PersNr, Name from AndereAngestellte)
```

Entferne die Tabelle und die Sichten wieder:

```
drop table andereAngestellte
drop view Angestellte
```

SQL: Stored Procedures

1.) Erstelle Prozeduren, die das „Archivieren“ von Nachrichten in einer Backup-Relation unterstützen:

```
create table Messages
   ( MID      integer auto_increment not null,
     Message  varchar(32),
     primary key (MID) );

create table MessagesBackup
   ( MBID      integer auto_increment not null,
     MOldID    integer,
     MessageB  varchar(32),
     primary key (MBID) );
```

SQL: Stored Procedures

2.) Extension aufbauen:

```
insert into Messages (Message) values ('Nachricht--01');
insert into Messages (Message) values ('Nachricht--02');
insert into Messages (Message) values ('Nachricht--03');
insert into Messages (Message) values ('Nachricht--04');
insert into Messages (Message) values ('Nachricht--05');
insert into Messages (Message) values ('Nachricht--06');
```

SQL: Stored Procedures

3.) Copy-Procedure anlegen:

```
-- DELIMITER $$
drop procedure if exists spCopyMessage $$
create procedure spCopyMessage( id integer )
begin
    insert into MessagesBackup (MOldID, MessageB)
    select MID, Message from Messages where MID = id;
end $$
-- DELIMITER ;

call spCopyMessage(1);
call spCopyMessage(3);

select * from Messages;
select * from MessagesBackup;
```

SQL: Stored Procedures

4.) Insert_OR_Update-Procedure anlegen:

```
-- DELIMITER $$
drop procedure if exists spInsert_OR_Update $$
create procedure spInsert_OR_Update( id integer, Msg varchar(32) )
begin
    if ( id = 0 ) then
        set id = null;
    end if;

    if ( id is not null ) and
       ( exists ( select * from Messages where MID = id ) )
    then
        call spCopyMessage(id);
        update Messages set Message = Msg where MID = id;
    else
        insert into Messages (MID, Message) values (id, Msg);
    end if;
end $$
-- DELIMITER ;
```

SQL: Stored Procedures

5.) Insert_OR_Update-Procedure nutzen:

```
call spInsert_OR_Update( 2, 'Test' );

select * from Messages;
select * from MessagesBackup;

call spInsert_OR_Update( 11, 'Frank' );

select * from Messages;
select * from MessagesBackup;
```

SQL: Stored Procedures

6.) Tabellen und Prozeduren entfernen:

```
drop procedure spCopyMessage;
drop procedure spInsert_OR_Update;

drop table Messages;
drop table MessagesBackup;
```

SQL: Stored Functions

1.) Prozeduren mit Rückgabewerten:

```
-- DELIMITER $$
create function Vorlesungsanzahl() returns integer
deterministic
begin
    declare tval integer;
    select count(*) into tval from Vorlesungen;
    return tval;
end $$
-- DELIMITER ;

select Vorlesungsanzahl();
```

SQL: Trigger

1.) Automatischen Log von Nachrichten:

```
create table News
    ( NID integer auto_increment not null,
      Msgs varchar(210),
      primary key (NID) );

create table NewsLog
    ( NLID    integer auto_increment not null,
      NOldID  integer,
      LMsg    varchar(210),
      primary key (NLID) );
```

SQL: Trigger

2.) Trigger auf der Tabelle News:

```
-- DELIMITER $$
drop trigger if exists trLogNews $$
create trigger trLogNews
    after insert on News
    for each row begin
        insert into NewsLog (NOldID, LMsg) select NID, Msgs from
        News where NID=(select max(NID) from News);
    end $$
-- DELIMITER ;
```

SQL: Trigger

3.) Effekt des Triggers beobachten:

```
insert into News(Msgs) values ('News-1');
insert into News(Msgs) values ('News-2');
insert into News(Msgs) values ('News-3');

select * from News;
select * from NewsLog;
```

SQL: Trigger

4.) Tabellen und Trigger entfernen:

```
drop trigger trLogNews;

drop table News;
drop table NewsLog;
```

Datenbanksysteme SS 2007

Ende von Kapitel 7b: SQL