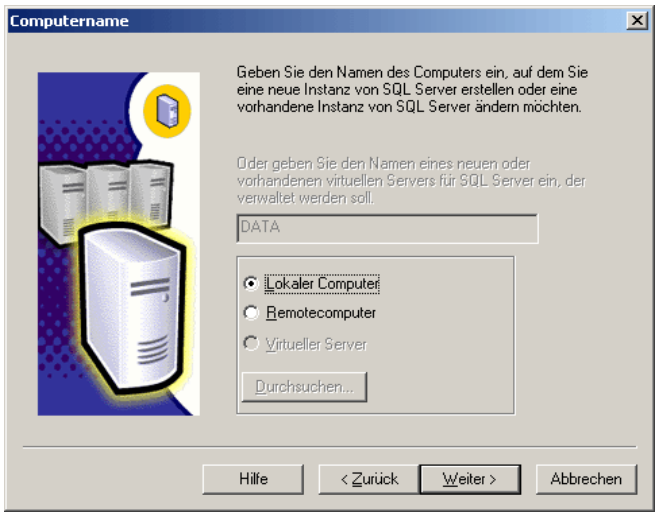
Kapitel 7:
SQL

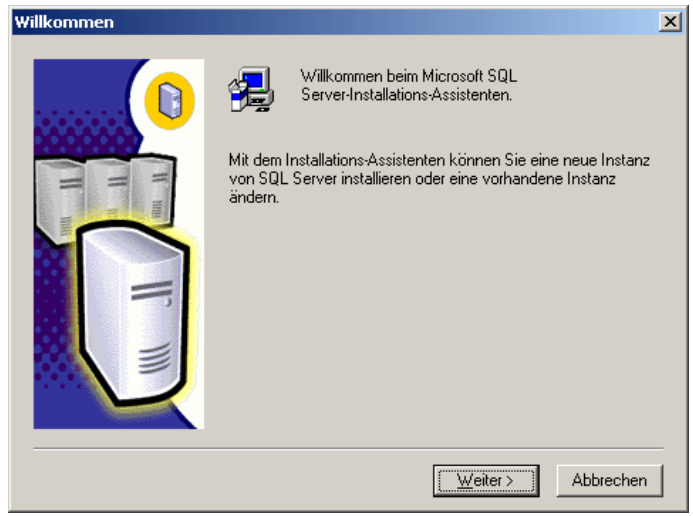
Microsoft SQL-Server 2000

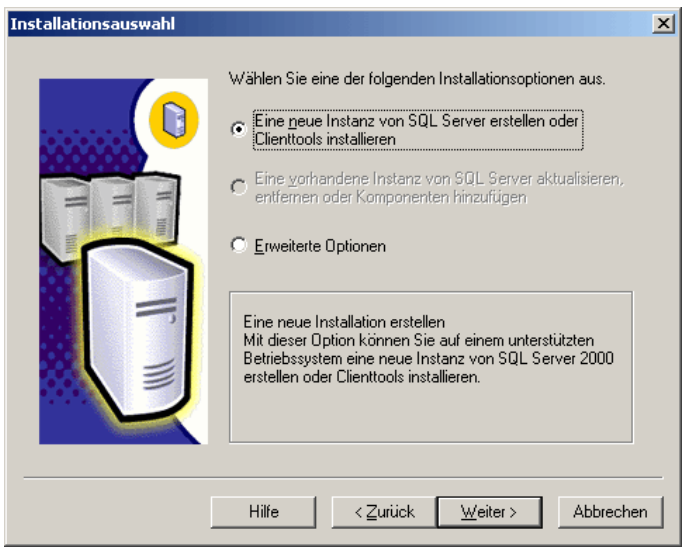
Server: Enterprise-Manager

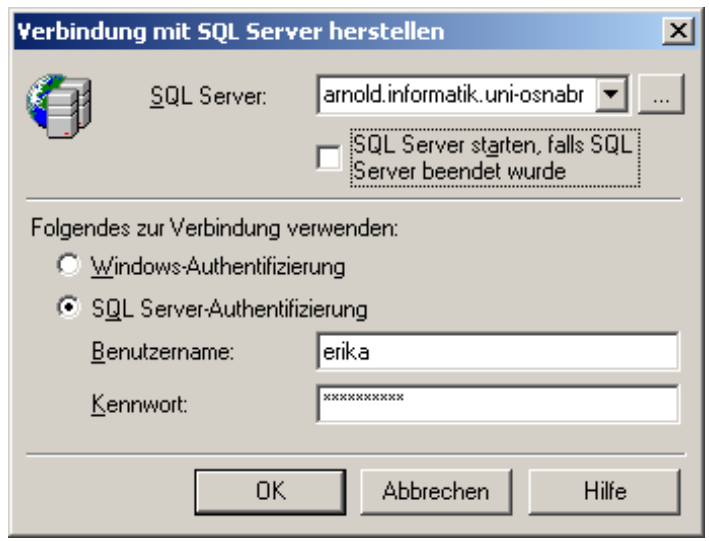
Client: Query-Analyzer

Installation Query Analyzer

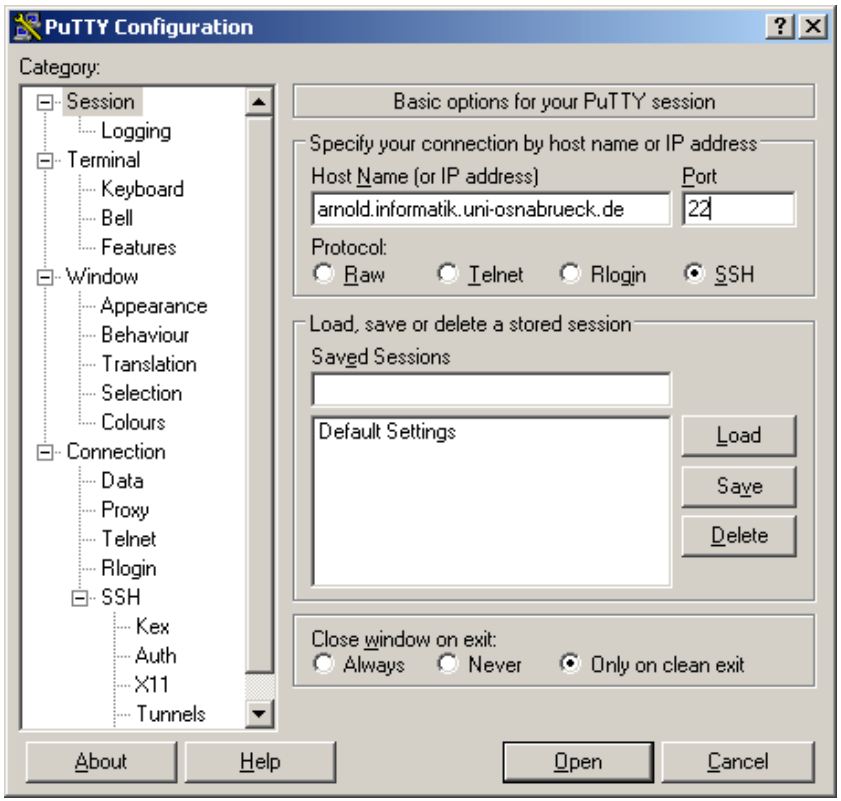


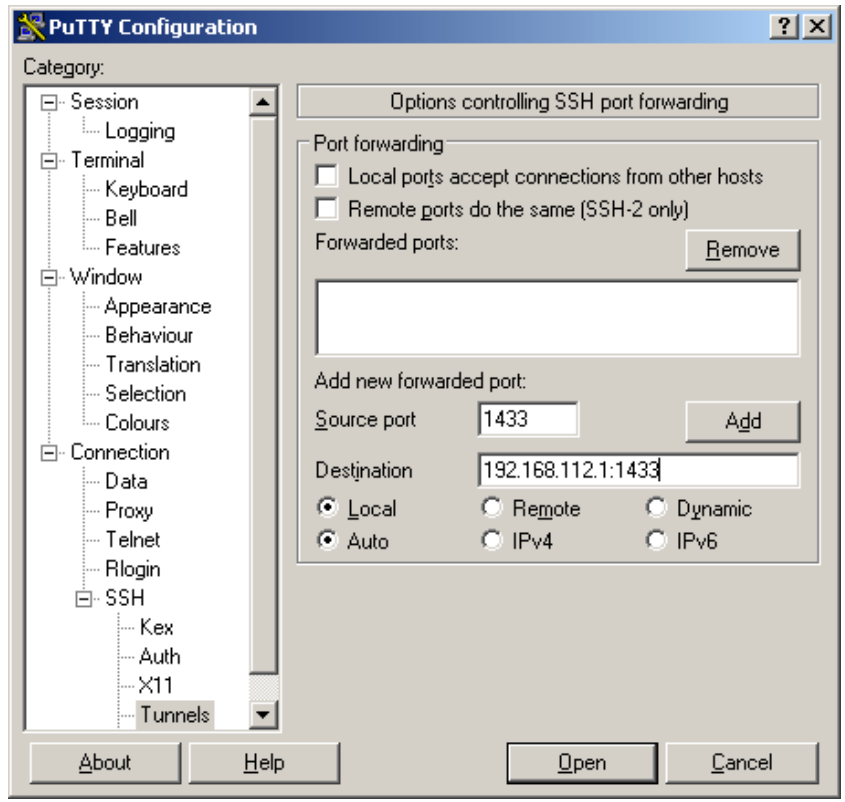




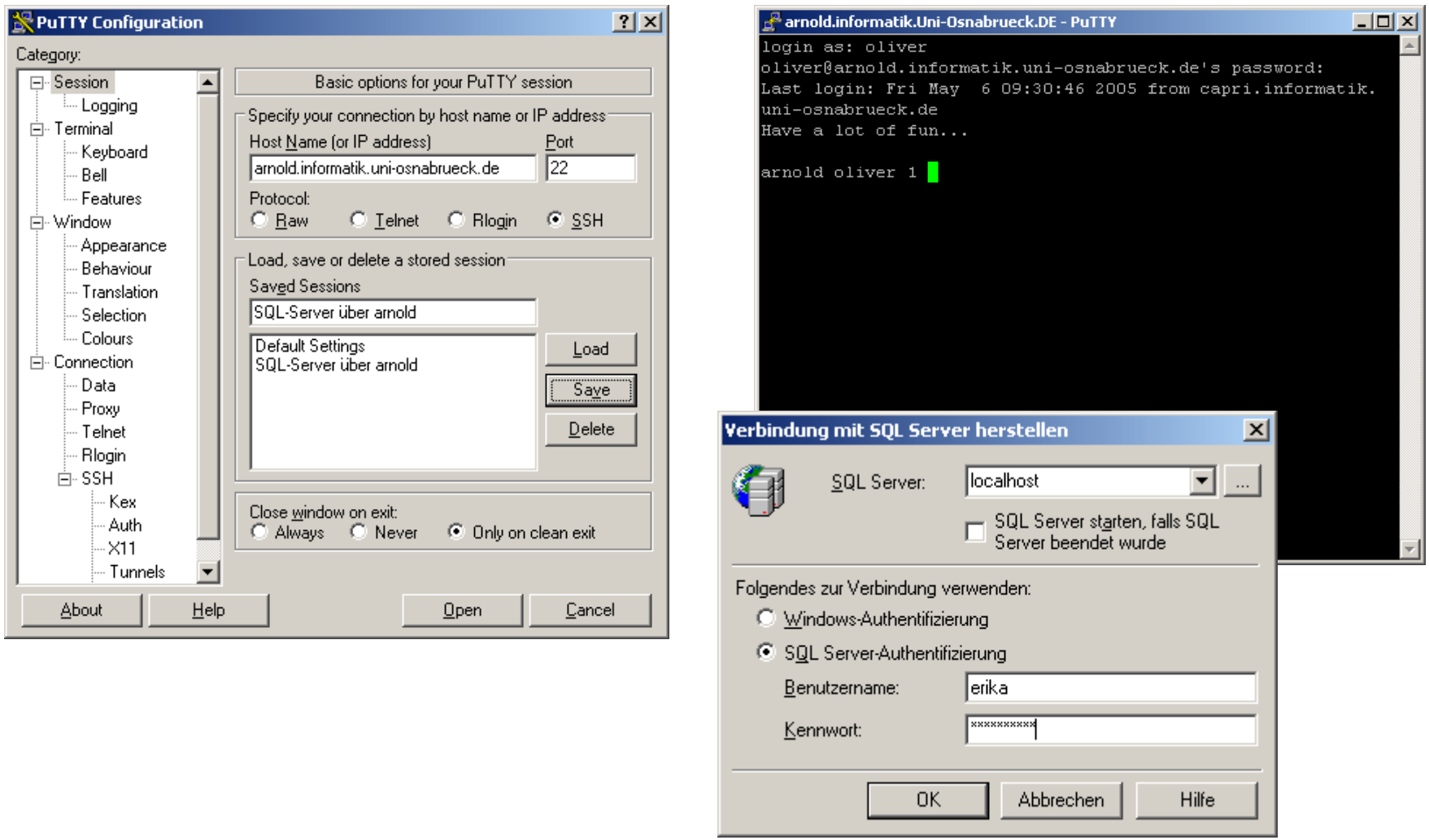


Tunnel mit Putty





Tunnel mit Putty



SQL: Geschichte

1970: System R mit Abfragesprache Sequel
1992: SQL-92 = SQL 2
1999: SQL:1999 = SQL 3 (objektorientiert)

SQL: Datentypen

8	bigint	ganze Zahlen von -2^{63} bis $+2^{63}$
4	int	ganze Zahlen von -2^{31} bis $+2^{31}$
2	smallint	ganze Zahlen von -2^{15} bis $+2^{15}$
1	tinyint	ganze Zahlen von 0 bis 255
1	bit	ganze Zahlen von 0 bis 1
n	decimal(n,k)	numerische Daten, feste Genauigkeit von -10^{38} bis $+10^{38}$ Anzeige mit n Stellen, davon k nach dem Komma
n	numeric(n,k)	entspricht decimal
8	money	Währungsdatenwerte von -2^{63} bis $+2^{63}$ Anzeige: 4 Nachkommastellen
4	smallmoney	Währungsdatenwerte von -2^{15} bis $+2^{15}$
4	real	Gleitkommazahlen von -10^{38} bis $+10^{38}$
8	float	Gleitkommazahlen von -10^{308} bis $+10^{308}$

SQL: Datentypen

datetime	Zeitangaben von 01.01.1753 bis 31.12.9999 4 Byte für Zahl der Tage vor/nach dem 1.1.1900 4 Byte für Zeit nach Mitternacht Genauigkeit 3/100 Sekunden
smalldatetime	Zeitangaben von 01.01.1900 bis 06.06.2079 2 Byte für Zahl der Tage nach dem 1.1.1900 2 Byte für Zahl der Minuten nach Mitternacht
char(n)	String fester Länge mit maximal 8.000 Zeichen
varchar(n)	String variabler Länge mit maximal 8.000 Zeichen

SQL: Datentypen

text	String variabler Länge mit maximal 2^{31} Zeichen
nchar(n)	Unicode-Daten fester Länge mit maximal 4.000 Zeichen
nvarchar(n)	Unicode-Daten variabler Länge mit maximal 4.000 Zeichen
ntext	Unicode-Daten variabler Länge mit maximal 2^{30} Zeichen
binary	Binärdaten fester Länge mit maximal 8.000 Bytes
nbinary	Binärdaten variabler Länge mit maximal 8.000 Bytes
image	Binärdaten variabler Länge mit maximal 2^{31} Bytes
rowversion	(früher: timestamp) eindeutig pro Datenbank wird hochgezählt bei INSERT + UPDATE
uniqueidentifier	weltweit eindeutiger Bezeichner (16 Byte lang) vergeben durch newid()

SQL: create

```
create table Person (
  PersNr      int identity(100000,1),
  Name        varchar(15) not null,
  Geschlecht  bit default 0,
  Note        numeric (5,2),
  Gewicht     float,
  Gehalt      money,
  GebDatum    datetime,
  Bemerkung   text,
  Photo       image,
  Termin      rowversion,
  Kennung     uniqueidentifier default newid(),
)
```

SQL: alter, modify, drop

Tabelle um eine Spalte erweitern:

```
alter table Person
add   Vorname varchar(15)
```

Tabellenspalte ändern:

```
alter table Person
modify Vorname varchar(20)
```

Tabelle um eine Spalte verkürzen:

```
alter table Person
drop  column Vorname
```

Tabelle entfernen:

```
drop table Person
```

SQL: Schlüsselworte

select
from
where
order by
asc
desc
as
like
upper
lower

distinct
count
sum
avg
max
min
group by
having

is
not
null
exists
all
some

SQL: select, from, where

1.) Liste alle Studenten:

```
select * from studenten
```

2.) Liste Personalnummer und Name der C4-Professoren:

```
select PersNr, Name
from   Professoren
where  Rang='C4'
```

SQL: count, as, is not, null

3.) Zähle alle Studenten

```
select count (*) from studenten
```

4.) Liste Name und Studiendauer in Jahren von allen Studenten:

```
select Name, Semester/2 as Studienjahr
from   Studenten
where  Semester is not null
```

SQL: between, in

5.) Liste aller Studenten mit Semesterzahlen zwischen 1 und 4:

```
select *
from   Studenten
where  Semester >= 1 and Semester <= 4
```

alternativ

```
select *
from   Studenten
where  Semester between 1 and 4
```

alternativ

```
select *
from   Studenten
where  Semester in (1,2,3,4)
```

SQL: like, upper, order, distinct

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
```

SQL: datename, datediff

9.) Liste alle Geburtstage mit ausgeschriebenem Monatsnamen:

```
select  Name,
        Datename(Day,   Gebdatum) as Tag,
        Datename(Month, GebDatum) as Monat,
        Datename(Year,  GebDatum) as Jahr
from  studenten
```

10.) Liste das Alter der Studenten in Jahren:

```
select Name,
     datediff(year,GebDatum, getdate()) as Jahre
from studenten
```

SQL: datename

11.) Liste die Wochentage der Geburtsdaten der Studenten:

```
select Name,
datename(weekday,GebDatum) as Wochentag
from   studenten
```

12.) Liste die Kalenderwochen der Geburtsdaten der Studenten:

```
select Name,
datename(week,GebDatum) as Kalenderwoche
from   studenten
```

SQL: Verbund

13.) Liste den Dozenten der Vorlesung Logik:

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

14.) 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 v
where   s.MatrNr = h.MatrNr
and     h.VorlNr = v.VorlNr
```

SQL: Self Join

15.) 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'
```

SQL: avg, group

16.) Liste die durchschnittliche Semesterzahl:

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

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

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

18.) Liste Summe der SWS pro Professor:

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

SQL: having

19.) 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
```

alternativ unter Verwendung von Gleichkommadurchschnitt:

```
select gelesenVon as PersNr, sum (SWS) as Lehrbelastung
from     Vorlesungen
group by gelesenVon
having   avg(cast(SWS as float)) > 3.0
```

SQL: where & having

20.) 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(cast(SWS as float)) > 3.0
```

SQL: Sub-Query

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

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

22.) Liste alle Professoren zusammen mit ihrer Lehrbelastung:

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

SQL: exists

23.) 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 )
```

SQL: Verbund mit <

24.) 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
```

SQL: geschachtelt

25.) 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
```

alternativ:

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

SQL: geschachtelt

26.) Liste die Namen und Geburtstage der Gehaltsklassenältesten:

```
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
```

27.) Liste Vorlesungen zusammen mit Marktanteil, definiert als = Hörerzahl/Gesamtzahl:

```
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,
cast(h.AnzProVorl as float)/g.GesamtAnz as Marktanteil
from (select VorlNr, count(*) as AnzProVorl
       from hoeren group by VorlNr) h,
       (select count(*) as GesamtAnz
       from Studenten) g
```

SQL: union

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

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

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

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

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

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

SQL: not, in, exists

31.) 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 )
```

SQL: all, some

32.) Liste Studenten mit größter Semesterzahl:

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

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

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

SQL: not, exists

34.) 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
        )
     )
```

SQL: transitive Hülle

35.) 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'
```

Befördere den Dozenten von 'Der Wiener Kreis' nach C4

```
update professoren
set    rang='C4'
from   professoren p, vorlesungen v
where  p.persnr = v.gelesenvon
and    v.titel='Der Wiener Kreis'
```

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: Generalisierung durch Verbund

5.) 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)
GO
create view Profs as
   select a.persnr, a.name, d.rang, d.raum
   from   Angestellte a, ProfDaten d
   where  a.PersNr = d.PersNr
GO
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

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

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

Index

```
create index hilfetitel
on vorlesungen(titel asc)

create unique index hilfepersnr
on professoren(persnr)

drop index vorlesungen.titel
drop index professoren.persnr
```

Große Datenmengen

Werte vom Typ

- Text
- Image

müssen ggf. in „Portionen“ transferiert werden.

Text schreiben, ändern, lesen

```
DECLARE @ptrval binary(16)
SELECT  @ptrval = TEXTPTR(Bemerkung)
FROM    Person WHERE Name='Erika'
WRITETEXT Person.Bemerkung @ptrval 'Dies ist ein Satz'

DECLARE @ptrval binary(16)
SELECT  @ptrval = TEXTPTR(Bemerkung)
FROM    Person
WHERE   Name='Erika'
UPDATETEXT Person.Bemerkung @ptrval 5 3 'war'

DECLARE  @ptrval varbinary(16)
SELECT   @ptrval = TEXTPTR(Bemerkung)
FROM     Person where Name='Erika'
READTEXT Person.Bemerkung  @ptrval 5 3
```

Image schreiben, ändern, lesen

```
DECLARE @ptrval binary(16)
SELECT  @ptrval = TEXTPTR(Photo)
FROM    Person WHERE Name='Erika'
WRITETEXT Person.Photo @ptrval 0x0123456789ABCDEF

DECLARE @ptrval binary(16)
SELECT  @ptrval = TEXTPTR(Photo)
FROM    Person WHERE Name='Erika'
UPDATETEXT Person.Bemerkung @ptrval 5 3 0xFFFFFF

DECLARE  @ptrval varbinary(16)
SELECT   @ptrval = TEXTPTR(Photo)
FROM     Person where Name='Erika'
READTEXT Person.Photo  @ptrval 5 3
```

Bulkinsert

```
4711;Willi;C4;339;24.03.1951
4712;Erika;C3;222;18.09.1962
```

```
BULK INSERT Professoren
   FROM 'K:\DatenbankSysteme\professoren.txt'
   WITH
      (
         FIELDTERMINATOR = ';',
         ROWTERMINATOR = '\n'
      )
```

Schleife

```
create table konto (nr int, stand money)


declare @i int
set @i = 1
while @i < 50
begin
  if (@i=42) insert into konto values (@i,200)
  else       insert into konto values (@i,  0)
  set @i = @i+1
end
```

Stored Procedure

```
create procedure ueberweisung
  (@x int,
  @y int,
  @betrag money)
as
  declare @s money
  SELECT @s = stand FROM konto
  WHERE nr = @x
  IF @s < @betrag BEGIN
    INSERT INTO abgelehnt
    VALUES (getdate(), @x, @y, @betrag)
  END ELSE
  BEGIN
    UPDATE konto
      SET stand = stand-@betrag
      WHERE nr = @x
    UPDATE konto
      SET stand = stand+@betrag
      WHERE nr = @y
    INSERT INTO gebucht
    VALUES (getdate(), @x, @y, @betrag)
  END
```

Funktion

```
create function f2c
  (@fahrenheit int)
  returns int
as
begin
  declare @celsius int
  set @celsius=(@fahrenheit-32)/9.0 * 5.0
  return @celsius
end
```

Cursor

```
declare @name varchar(20)
declare @titel varchar(20)
declare prof_cursor cursor for
  select p.name, v.titel
  from professoren p, vorlesungen v
  where p.persnr=v.gelesenvon
open prof_cursor
fetch next from prof_cursor into @name, @titel
while @@fetch_status = 0
begin
   print @name + ' ' + @titel
   fetch next from prof_cursor into @name, @titel
end
close prof_cursor
deallocate prof_cursor
```