Kapitel 7

SQL

7.1 SQL-Server

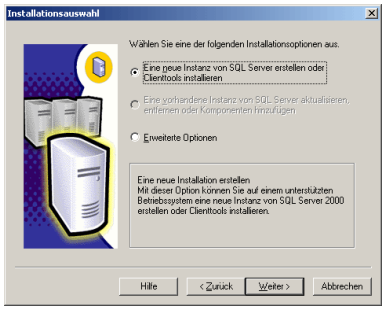
Stellvertretend für die zahlreichen am Markt befindlichen relationalen Datenbanksysteme wird in diesem Kapitel das System *Microsoft SQL-Server 2000* verwendet. Als Vorbereitungen zum Zugriff sind erforderlich:

Server :
Nach dem Installieren des Microsoft SQL-Servers muß vom DBA (Data Base Administrator) jeder User mit User-Name, Passwort und Zugriffsrechten eingerichtet werden.

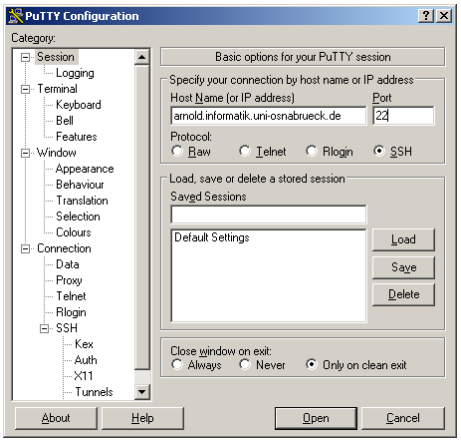
Client :
In jeder Windows-NT-Station wird die Klienten-Software *SQL Query Analyzer* installiert. Diese stellt (neben weiteren Funktionalitäten) eine ASCII-Schnittstelle zur Verfügung, auf der SQL-Statements abgesetzt werden können. Diese werden dann per TCP/IP an den SQL-Server übertragen, dort ausgeführt und das Ergebnis zurückgeschickt.

Es folgen einige Screenshots vom Installationsvorgang des Query-Analyzers und von der Konfiguration von Putty zum Tunneling zum Rechner arnold.

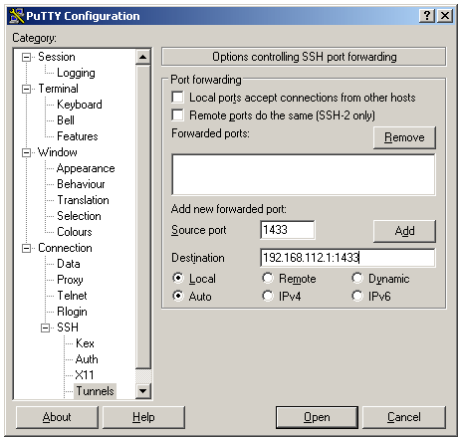
Datenbankserver installieren



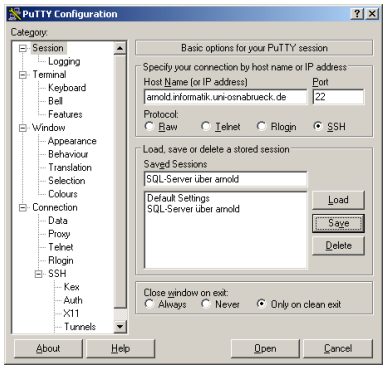
Server oder Clienttools installieren



arnold.informatik.uni-osnabrueck.de



Destination 192.168.112.1:1433



Einstellungen abspeichern

mit localhost verbinden

7.2 Sprachphilosophie

Die Relationale Algebra und der Relationenkalkül bilden die Grundlage für die Anfragesprache SQL. Zusätzlich zur Manipulation von Tabellen sind Möglichkeiten zur Definition des relationalen Schemas, zur Formulierung von Integritätsbedingungen, zur Vergabe von Zugriffsrechten und zur Transaktionskontrolle vorgesehen.

Relationale Datenbanksysteme realisieren keine Relationen im mathematischen Sinne, sondern Tabellen, die durchaus doppelte Einträge enthalten können. Bei Bedarf müssen die Duplikate explizit entfernt werden.

SQL geht zurück auf den von IBM Anfang der 70er Jahre entwickelten Prototyp *System R* mit der Anfragesprache *Sequel*. Der zur Zeit aktuelle Standard lautet SQL-92, auch SQL 2 genannt. Er wird weitgehend vom relationalen Datenbanksystem *SQL-Server* unterstützt.

7.3 Datentypen

Microsoft SQL-Server verwendet folgende Datentypen:

Typ	Bytes	Wertebereich
`bigint`	8	ganze Zahlen von -2^{63} bis $+2^{63}$
`int`	4	ganze Zahlen von -2^{31} bis $+2^{31}$
`smallint`	2	ganze Zahlen von -2^{15} bis $+2^{15}$
`tinyint`	1	ganze Zahlen von 0 bis 255
`bit`	1	ganze Zahlen von 0 bis 1
`decimal(n,m)`	n	numerische Daten mit fester Genauigkeit von -10^{38} bis $+10^{38}$
`numeric(n,m)`	n	entspricht decimal
`money`	8	Währungsdatenwerte von -2^{63} bis $+2^{63}$
`smallmoney`	2	Währungsdatenwerte von -2^{15} bis $+2^{15}$
`real`	4	Gleitkommazahlen von -10^{38} bis $+10^{38}$
`float`	8	Gleitkommazahlen von -10^{308} bis $+10^{308}$
`datetime`	8	Zeitangaben von 01.01.1753 bis 31.12.9999
`smalldatetime`	4	Zeitangaben von 01.01.1900 bis 06.06.2079
`char(n)`	n	String fester Länge mit maximal 8.000 Zeichen
`varchar(n)`		String variabler Länge mit maximal 8.000 Zeichen
`text`		String variabler Länge mit maximal 2^{31} Zeichen
`nchar(n)`	2n	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`	8	Datenbank-eindeutiger Wert
`uniqueidentifier`	16	weltweit eindeutiger Wert

`NULL` bezeichnet nicht besetzte Attributwerte

`default` bezeichnet vorbesetzte Attributwerte.

`numeric(n,m)`-Werte werden mit n Dezimalstellen angezeigt, davon m Nachkommastellen.

`int, smallint, tinyint` und `numeric` können durch den Zusatz `identity(i,j)` automatisch initialisert werden mit Startwert i und Schrittweite j.

Typ `money`-Werte werden mit 4 Nachkommastellen angezeigt.

`datetime`-Werte bestehen aus zwei 4-Byte-Worten: die Anzahl der Tage vor oder nach dem Basisdatum 01.01.1900 sowie die Anzahl der Millisekunden seit Mitternacht.

`smalldatetime`-Werte bestehen aus zwei 2-Byte-Werten: die Anzahl der Tage seit dem 01.01.1900 sowie die Anzahl der Minuten seit Mitternacht.

Spalten vom Typ `binary` oder `image` speichern umfangreiche Binärdaten innerhalb eines Zeilentupels und erfordern zusätzliche Zugriffstechniken, um die Daten einzufügen oder auszulesen.

Spalten vom Typ `rowversion` (früher: `timestamp`) sind innerhalb einer Datenbank eindeutig.

Sie werden automatisch für die betreffende Zeile bei einer INSERT- oder UPDATE-Operation zugewiesen.

Spalten vom Typ `uniqueidentifier` können durch die Funktion `newid()` einen (hoffentlich) weltweit eindeutigen Identifier erhalten.

7.4 SQL-Statements zur Schemadefinition

SQL-Statements zum Anlegen, Erweitern, Verkürzen und Entfernen einer Tabelle:

1. Tabelle anlegen:

```
create table Person (
  PersNr     int identity(100000,1),   -- ganze Zahl, automatisch vergeben
  Name       varchar(15) not null,     -- max. 15 Zeichen langer Wert
  Geschlecht bit default 0,            -- boolscher Wert, vorbesetzt mit 0
  Note       numeric (5,2),            -- 5 Gesamt-, 2 Nachkommastellen
  Groesse    real,                     -- einfache Genauigket
  Gewicht    float,                    -- doppelte Genauigkeit
  Gehalt     money,                    -- Waehrungswert
  GebDatum   datetime,                 -- Uhrzeit- und Datumsangabe
  Bemerkung  text,                     -- laengerer Text
  Photo      image,                    -- Binaerdaten
  Termin     rowversion,               -- Zeitstempel fuer Zeilenzugriff
  Kennung    uniqueidentifier          -- global eindeutiger Wert
             default newid(),          -- vorbesetzt durch newid()
)
```

2. Tabelle um eine Spalte erweitern:

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

3. Tabellenspalte ändern:

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

4. Tabelle um eine Spalte verkürzen:

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

5. Tabelle entfernen:

```
drop table Person
```

7.5 Aufbau einer SQL-Query zum Anfragen

Eine SQL-Query zum Abfragen von Relationen hat den folgenden generischen Aufbau:

`SELECT`	Spalten-1
`FROM`	Tabellen
`WHERE`	Bedingung-1
`GROUP BY`	Spalten-2
`HAVING`	Bedingung-2
`ORDER BY`	Spalten-3

Nur die Klauseln `SELECT` und `FROM` sind erforderlich, der Rest ist optional.

Es bedeuten ...

Spalten-1	Bezeichnungen der Spalten, die ausgegeben werden
Tabellen	Bezeichnungen der verwendeten Tabellen
Bedingung-1	Auswahlbedingung für die auszugebenden Zeilen; verwendet werden `AND OR NOT = > < != <= >= IS NULL BETWEEN IN LIKE`
Spalten-2	Bezeichnungen der Spalten, die eine Gruppe definieren. Eine Gruppe bzgl. Spalte x sind diejenigen Zeilen, die bzgl. x identische Werte haben.
Bedingung-2	Bedingung zur Auswahl einer Gruppe
Spalten-3	Ordnungsreihenfolge für `<Spalten-1>`

Vor `<Spalten-1>` kann das Schlüsselwort `DISTINCT` stehen, welches identische Ausgabezeilen unterdrückt.

Sogenannte *Aggregate Functions* fassen die Werte einer Spalte oder Gruppe zusammen.

Es liefert ...

`COUNT (*)`	Anzahl der Zeilen
`COUNT (DISTINCT x)`	Anzahl der verschiedenen Werte in Spalte x
`SUM (x)`	Summe der Werte in Spalte x
`SUM (DISTINCT x)`	Summe der verschiedenen Werte in Spalte x
`AVG (x)`	Durchschnitt der Werte in Spalte x
`AVG (DISTINCT x)`	Durchschnitt der verschiedenen Werte in Spalte x
`MAX (x)`	Maximum der Werte in Spalte x
`MIN (x)`	Minimum der Werte in Spalte x

jeweils bezogen auf solche Zeilen, welche die `WHERE`-Bedingung erfüllen. Null-Einträge werden bei `AVG`, `MIN`, `MAX` und `SUM` ignoriert.

Spalten der Ergebnisrelation können umbenannt werden mit Hilfe der `AS`-Klausel.

7.6 SQL-Queries zum Anfragen

Folgende Beispiele beziehen sich auf die Universitätsdatenbank, wobei die Relationen *Professoren*, *Assistenten* und *Studenten* jeweils um ein Attribut *GebDatum* vom Typ *Datetime* erweitert worden sind.

1. Liste alle Studenten:

   ```
   select * from studenten
   ```

2. Liste Personalnummer und Name der C4-Professoren:

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

3. Zähle alle Studenten:

   ```
   select count(*)
   from   Studenten
   ```

4. Liste Namen und Studiendauer in Jahren von allen Studenten,

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

5. Liste alle 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)
   ```

6. Liste alle Vorlesungen, die im Titel den String `Ethik` enthalten, klein oder groß geschrieben:

   ```
   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 Einträge in der Spalte Rang der Relation Professoren:

```
select distinct Rang
from            Professoren
```

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
```

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
```

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
```

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'
```

16. Liste die durchschnittliche Semesterzahl:

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

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

```
select   Rang, min(GebDatum) as Aeltester
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
```

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
```

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
```

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
```

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 )
```

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

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

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
```

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
```

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)
```

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 )
```

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 )
```

34. Liste solche 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
    )
  )
```

35. Berechnung der transitiven Hülle einer rekursiven Relation (nur in Oracle):
Liste alle Voraussetzungen für die 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'
        )
)
```

7.7 SQL-Statements zum Einfügen, Modifizieren und Löschen

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'
```

3. Erweitere die neue Vorlesung um ihre Semesterwochenstundenzahl:

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

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'
```

7.8 SQL-Statements zum Anlegen von Sichten

Die mangelnden Modellierungsmöglichkeiten des relationalen Modells in Bezug auf Generalisierung und Spezialisierung können teilweise kompensiert werden durch die Verwendung von Sichten. Nicht alle Sichten sind *update-fähig*, da sich eine Änderung ihrer Daten nicht immer auf die Originaltabellen zurückpropagieren läßt

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
```

3. Lege Sicht an mit Professoren und ihren Durchschnittsnoten:

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

4. Entferne die Sichten wieder:

```
drop view PruefenSicht
drop view StudProf
drop view ProfNote
```

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
```

Entferne die Tabellen und Sichten wieder:

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

6. 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)
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
```

7.9 SQL-Statements zum Anlegen von Indizes

1. Lege einen Index an für die aufsteigend sortierten Titel der Tabelle Vorlesung:

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

2. Lege einen eindeutigen Index an für die Personalnummern der Tabelle Vorlesung:

```
create unique index hilfepersnr
on professoren(persnr)
```

3. Entferne die Indizes titel und persnr:

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

7.10 Text

Einen Wert vom Typ `text` schreiben:

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

Einen Teil eines Textes ändern (3 Zeichen ab Position 5):

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

Einen Teil eines Textes lesen (3 Zeichen ab Position 5):

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

7.11 Image

Einen Wert vom Typ `image` schreiben (hexadezimal kodiert, je 2 Buchstaben bezeichnen ein Byte):

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

Einen Teil eines Image ändern (3 Bytes ab Position 5):

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

Einen Teil eines Image lesen (3 Bytes ab Position 5):

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

7.12 Bulk insert

Gegeben sei eine tabellenartig strukturierte Textdatei auf dem Rechner des Datenbankservers:

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

Durch den Befehl `bulkinsert` kann der Inhalt einer Datei komplett in eine SQL-Server-Tabelle eingefügt werden, wobei das Trennzeichen zwischen Feldern und Zeilen angegeben werden muß:

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

7.13 SQL-Scripte

Microsoft SQL-Server 2000 bietet eine prozedurale Erweiterung von SQL an, genannt *SQL-Scripte* oder auch *Stored Procedures*. Hiermit können SQL-Statements zu namenlosen Blöcken, Prozeduren oder Funktionen zusammengefasst und ihr Ablauf durch Kontrollstrukturen gesteuert werden.

Sei eine Tabelle *konto* mit Kontonummern und Kontoständen angelegt durch

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

Listing 7.1 zeigt eine while-Schleife, die 50 Konten mit durchlaufender Numerierung einrichtet und alle Kontostände bis auf das Konto 42 mit 0.0 initialisiert.

```
declare @i int                                      -- lokale Variable @i
set @i = 1                                          -- setze @i auf 1
while @i < 50                                       -- solange @i < 50
begin
  if (@i=42) insert into konto values (@i,200)      -- fuege 200 Euro ein
  else       insert into konto values (@i,  0)      -- fuege   0 Euro ein
  set @i = @i+1                                     -- erhoehe @i
end
```

Listing 7.1: namenloses SQL-Script

Listing 7.2 zeigt eine benannte Stored Procedure, welche versucht, innerhalb der Tabelle *konto* eine Überweisung durchzuführen und danach das Ergebnis in zwei Tabellen festhält:

```
create table gebucht   (datum DATE, nr_1 int, nr_2 int, betrag money)
create table abgelehnt (datum DATE, nr_1 int, nr_2 int, betrag money)
```

```
create procedure ueberweisen                -- lege Prozedur an
 (@x int,                                   -- Konto-Nr. zum Belasten
  @y int,                                   -- Konto-Nr. fuer Gutschrift
  @betrag money)                            -- Ueberweisungsbetrag
as
  declare @s money                          -- lokale Variable
  SELECT @s = stand FROM konto              -- hole Kontostand nach s
  WHERE nr = @x                             -- von Konto-Nr. x
  IF @s < @betrag BEGIN                     -- falls Konto ueberzogen
    INSERT INTO abgelehnt                   -- notiere den Fehlschlag
    VALUES (getdate(), @x, @y, @betrag)     -- in der Tabelle abgelehnt
  END ELSE
  BEGIN
    UPDATE konto                            -- setze in der Tabelle konto
      SET stand = stand-@betrag             -- neuen Betrag ein
      WHERE nr = @x                         -- fuer Kontonr @x
    UPDATE konto                            -- setze in der Tabelle konto
      SET stand = stand+@betrag             -- neuen Betrag ein
      WHERE nr = @y                         -- fuer Kontonr @y
    INSERT INTO gebucht                     -- notiere die Ueberweisung
    VALUES (getdate(), @x, @y, @betrag)     -- in der Tabelle gebucht
  END
```

Listing 7.2: stored procedure ueberweisung

Im Gegensatz zu einem konventionellen Benutzerprogramm wird eine *stored procedure* in der Datenbank gespeichert. Sie wird aufgerufen und (später) wieder entfernt durch

```
execute ueberweisung 42,37,50
drop procedure ueberweisung
```

In Listing 7.3 wird eine Funktion `f2c` definiert, die eine übergebene Zahl als Temperatur in Fahrenheit auffaßt und den Wert nach Celsius umrechnet.

```
create function f2c                     -- definiere eine Funktion f2c
  (@fahrenheit int)                     -- Eingangsparameter vom Typ int
  returns int                           -- Ausgangsparameter vom Typ int
as begin
  declare @celsius int                  -- lokale Variable
  set @celsius=(@fahrenheit-32)/9.0*5.0 -- Umrechnung nach Celsius
  return @celsius;                      -- Rueckgabe des Funktionswertes
end                                     -- Ende der Funktion
```

Listing 7.3: stored function f2c

Der Aufruf der Funktion erfolgt innerhalb einer SQL-Abfrage unter Angabe des Besitzers (hier: dbo):

```
select temperatur, dbo.f2c(temperatur) from daten
```

Oft besteht das Ergebnis eines Select-Statements aus einer variablen Anzahl von Tupeln. Diese können nacheinander verarbeitet werden mit Hilfe eines sogenannten *Cursor*. Listing 7.4 zeigt den typischen Einsatz.

Zunächst wird der Cursor durch `declare` mit einer SQL-Query assoziiert, welche die Professorennamen mit ihren Vorlesungstiteln ermittelt. Dann wird er mit `open` für die Abarbeitung geöffnet. Mittels `fetch` wird das nächste Tupel aus der Trefferliste geholt und mit `into` in lokalen Variablen abgespeichert. Falls das geklappt hat (`fetch_status = 0`) kann durch `print` der Inhalt der Variablen ausgegeben werden. Zum Abschluss wird der Cursor geschlossen und deallokiert.

```
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
print @name + ' ' + @titel

while @@fetch-status = 0
begin
   fetch next from prof_cursor
   into @name, @titel
   print @name + ' ' + @titel
end

close prof_cursor
deallocate prof_cursor
```

Listing 7.4: Umgang mit einem Cursor