Kapitel 7

Relationale Anfragesprachen

7.1 Oracle Datenbank

Stellvertretend für die zahlreichen am Markt befindlichen relationalen Datenbanksysteme wird in diesem Kapitel das System *Oracle, Version 8.05* verwendet. Als Vorbereitungen zum Zugriff sind erforderlich:

Server :
Nach dem Installieren des Oracle Servers muß vom DBA (Data Base Administrator) jeder User mit Passwort, Verbindungsrecht und Arbeitsbereich eingerichtet werden:

- Richte neuen User `erika` mit Passwort `mustermann` ein:

  ```
  create user erika identified by mustermann;
  ```

- Erteile erika das Recht, eine Oracle-Verbindung herzustellen:

  ```
  grant connect to erika;
  ```

- Erteile erika das Recht, in Oracle Objekte anzulegen:

  ```
  grant resource to erika;
  ```

- Weise dem User `erika` den temporären Arbeitsbereich `temp` zu:

  ```
  alter user erika temporary tablespace temp;
  ```

- Liste Angaben zu allen Benutzern:

  ```
  select * from all_users;
  ```

- Erteile `erika` das Leserecht für die Tabelle `Vorlesungen`:

  ```
  grant select on Vorlesungen to erika;
  ```

- Entziehe `erika` das Recht auf Verbindung:

  ```
  revoke connect from erika;
  ```

- Entferne `erika` aus der Datenbank:

  ```
  drop user erika;
  ```

Client :
In jeder Windows-NT-Station wird die Klienten-Software *Oracle SQL*PLUS* installiert. Diese stellt eine ASCII-Schnittstelle zur Verfügung, auf der SQL-Statements abgesetzt werden können.

Verbindung :
Mit dem Modul *Oracle Net8 Easy Config* wird pro Station ein sogenannter Dienst mit frei wählbarem Namen, z.B. `dbs99` eingerichtet, welcher über TCP/IP eine Verbindung zu einer im Server angelegten Datenbank, z. B. `ora1`, herstellt. Erforderlich ist die Angabe des Hostnamens (`dbserv.rz.uni-osnabrueck.de`) und der Portnummer (`1521`). Zum Testen ist der beim Server angelegte Benutzername (`erika`) erforderlich.

ODBC-Treiber :
In der Windows-NT-Systemsteuerung wird der von Oracle mitgelieferte ODBC (Open Data Base Connectivity) Treiber eingerichtet. Hierzu wird eine User Data Source hinzugefügt unter Verwendung des Oracle ODBC Treibers, die einen frei wählbaren Namen erhält, z. B. `dbs99.dsn` und Bezug nimmt auf den eingetragenen Dienst `dbs99`.

MS-Access :
MS-Access kann als Frontend zu Oracle verwendet werden. Hierzu wird in MS-Access nach dem Anlegen einer Datenbank eine Verknüpfung hergestellt zu dem Dateityp *ODBC Datenbanken*. Dort wird der eingetragene Dienst, z. B. `dbs99`, ausgewählt und nach dem Prompt Benutzername und Passwort eingegeben. Danach werden die gewünschten Tabellen ausgewählt.

7.2 SQL

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 SQL2 genannt. Er wird weitgehend vom relationalen Datenbanksystem *Oracle* unterstützt.

7.3 Datentypen in Oracle

Die von Oracle verwendeten Datentypen lauten:

Typ	Wertebereich
character(n)	String fester Länge mit n Zeichen
varchar2(n)	String variabler Länge mit bis zu n Zeichen
integer	ganze Zahl
number(n, m)	Festkommazahl mit n Stellen, davon m nach dem Komma
float	Gleitkommazahlen
date	Zeitangabe (für Daten zwischen 4712 v. Chr bis 4712 n. Chr.)
long	Zeichenkette bis zu 2 GByte
raw	Binärstrings bis zu 255 Bytes
long raw	Binärobjekte bis zu 2 GByte (Soundfiles, Videos, ...)

Nicht besetzte Attributwerte werden durch das Schlüsselwort `NULL` gekennzeichnet.

7.4 Schemadefinition

SQL-Statements zum Anlegen, Ändern und Entfernen einer Tabelle:

1. Tabelle anlegen:

```
create table Professoren (
  PersNr  integer       not null,
  Name    varchar2(10)  not null,
  Rang    character(2) );
```

2. Tabelle erweitern:

```
alter table Professoren
add (Raum integer);
```

3. Tabelle ändern:

```
alter table Professoren
modify (Name varchar2(30));
```

4. Tabelle verkürzen (nicht in Oracle):

```
alter table Professoren
drop column Gebdatum;
```

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 *Date* 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, die eine Semesterangabe haben:

```
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 to_char(GebDatum,'month DD, YYYY') AS Geburtstag
from studenten;
```

10. Liste das Alter der Studenten in Jahren:

```
select (sysdate - GebDatum) / 365 as Alter_in_Jahren
from studenten;
```

11. Liste die Wochentage der Geburtsdaten der Studenten:

```
select to_char(GebDatum,'day')
from studenten;
```

12. Liste die Uhrzeiten der Geburtsdaten der Studenten:

```
select to_char(GebDatum,'hh:mi:ss')
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, s.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, max(GebDatum)
from Professoren
group by rang;
```

18. Liste Summe der SWS pro Professor:

```
select gelesenVon, sum(SWS)
from Vorlesungen
group by gelesenVon;
```

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

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

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

```
select gelesenVon, Name, sum(SWS)
from Vorlesungen, Professoren
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon, Name
      having avg(SWS) > 3;
```

21. Liste alle Prüfungen, die als Ergebnis die Durchschnittsnote haben:

```
select *
from pruefen
where Note = (select avg(Note)
              from pruefen);
```

22. Liste alle Professoren zusammen mit ihrer Lehrbelastung (nicht Oracle):

```
select PersNr, Name, (select sum(SWS) as Lehrbelastung
                      from Vorlesungen
                      where gelesenVon = PersNr)
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, tmp.maximum
from  Professoren p,
      (select Rang, max(GebDatum) as maximum
       from Professoren
       group by Rang) tmp
where p.Rang = tmp.Rang and p.GebDat = tmp.maximum;
```

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

```
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,
       h.AnzProVorl/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:

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

30. Liste den Durchschnitt von Professoren- und Assistenten-Namen:

```
( 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. Natürlicher Verbund (nur in SQL-92):

```
select *
from Studenten
natural join hoeren;
```

36. 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-Queries zum Einfügen, Modifizieren und Löschen

1. Füge Student mit Matrikelnummer und Name ein:

```
insert into Studenten (MatrNr, Name)
            values (28121, 'Archimedes');
```

2. Alle Studenten sollen die Vorlesung 'Selber Atmen' hören:

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

3. Alle Studenten um 10 Tage älter machen:

```
update studenten
set GebDatum = GebDatum + 10;
```

4. Alle Studenten mit Semesterzahlen größer als 13 löschen:

```
delete from Studenten
   where Semester > 13;
```

5. Niemand soll mehr die Vorlesung 'Selber Atmen' hören:

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

7.8 SQL-Queries 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. Lege Untertyp als Verbund von Obertyp und Erweiterung an:

```
create table Angestellte   (PersNr      integer not null,
                            Name        varchar2(30) not null);
create table ProfDaten     (PersNr      integer not null,
                            Rang        character(2),
                            Raum        integer);
create table AssiDaten      (PersNr     integer not null,
                            Fachgebiet  varchar2(30),
                            Boss        integer);

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

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

5. Lege Obertyp als Vereinigung von Untertypen an:

```
create table Professoren       (PersNr      integer not null,
                                Name        varchar2(30) not null,
                                Rang        character(2),
                                Raum        integer);

create table Assistenten       (PersNr      integer not null,
                                Name        varchar2(30) not null,
                                Fachgebiet  varchar2(30),
                                Boss        integer);

create table AndereAngestellte (PersNr      integer not null,
                                Name        varchar2(30) not null);

create view Angestellte as

  (select PersNr, Name from Professoren) union
  (select PersNr, Name from Assistenten) union
  (select PersNr, Name from AndereAngestellte);
```

7.9 Query by Example

Query-by-Example (QBE) beruht auf dem relationalen Domänenkalkül und erwartet vom Benutzer das beispielhafte Ausfüllen eines Tabellenskeletts.

Liste alle Vorlesungen von Sokrates mit mehr als 3 SWS:

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
		p._t	> 3	`Sokrates`

Die Spalten eines Formulars enthalten Variablen, Konstanten, Bedingungen und Kommandos. Variablen beginnen mit einem Unterstrich (_), Konstanten haben keinen Präfix. Der Druckbefehl **p._t** veranlaßt die Ausgabe von _t.

Im Domänenkalkül lautet diese Anfrage

$$\{[t] | \exists v, s, r([v, t, s, r] \in \ Vorlesungen \wedge s > 3)\}$$

Ein Join wird durch die Bindung einer Variablen an mehrere Spalten möglich:

Liste alle Professoren, die Logik lesen:

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
		`Logik`		_x

Professoren	PersNr	Name	Rang	Raum
	_x	**p.**_n		

Über eine *condition box* wird das Einhalten von Bedingungen erzwungen:

Liste alle Studenten, die in einem höheren Semester sind als Feuerbach:

Studenten	MatrNr	Name	Semester
		p._s	_a
		`Feuerbach`	_b

conditions
_a > _b

Das Kommando zur Gruppierung lautet **g. ,** die Aggregatfunktionen heißen wie gewohnt **sum.**, **avg.**, **min.**, **max.** und **cnt.**. Die Duplikateliminierung wird durch **all.** erreicht:

Liste die Summe der SWS der Professoren, die überwiegend lange Vorlesungen halten:

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
			p.sum.all._x	**p.g.**

conditions
avg.all._x>2

Einfügen, Ändern und Löschen geschieht mit den Kommandos **i.**, **u.**, **d.**.

Füge neuen Studenten ein:

Studenten	MatrNr	Name	Semester
i.	4711	`Wacker`	5

Setze die Semesterzahlen von Feuerbach auf 3:

Studenten	MatrNr	Name	Semester
u.		`Feuerbach`	**u.3**

Entferne Sokrates und alle seine Vorlesungen:

Professoren	PersNr	Name	Rang	Raum
d.	_x	`Sokrates`		

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
d.	_y			_x

hören	VorlNr	MatrNr
d.	_y	

Kapitel 8

Datenintegrität

8.1 Grundlagen

In diesem Kapitel werden *semantische Integritätsbedingungen* behandelt, also solche, die sich aus den Eigenschaften der modellierten Welt ableiten lassen. Wir unterscheiden statische und dynamische Integritätsbedingungen. Eine statische Bedingung muß von jedem Zustand der Datenbank erfüllt werden (z. B. Professoren haben entweder den Rang C2, C3 oder C4). Eine dynamische Bedingung betrifft eine Zustandsänderung (z. B. Professoren dürfen nur befördert, aber nicht degradiert werden).

Einige Integritätsbedingungen wurden schon behandelt:

- Die Definition des Schlüssels verhindert, daß zwei Studenten die gleiche Matrikelnummer haben.
- Die Modellierung der Beziehung *lesen* durch eine 1:N-Beziehung verhindert, daß eine Vorlesung von mehreren Dozenten gehalten wird.
- Durch Angabe einer Domäne für ein Attribut kann z. B. verlangt werden, daß eine Matrikelnummer aus maximal 5 Ziffern besteht (allerdings wird nicht verhindert, daß Matrikelnummern mit Vorlesungsnummern verglichen werden).

8.2 Referentielle Integrität

Seien R und S zwei Relationen mit den Schemata $\mathcal{R}$ und

$\mathcal{S}$. Sei κ Primärschlüssel von $\mathcal{R}$.

Dann ist $\alpha \subset \mathcal{S}$ ein Fremdschlüssel, wenn für alle Tupel $s \in S$ gilt:

1. s.α enthält entweder nur Nullwerte oder nur Werte ungleich Null
2. Enthält $s.\alpha$ keine Nullwerte, existiert ein Tupel $r \in R$ mit $s.\alpha = r.\kappa$

Die Erfüllung dieser Eigenschaft heißt *referentielle Integrität*. Die Attribute von Primär- und Fremdschlüssel haben jeweils dieselbe Bedeutung und oft auch dieselbe Bezeichnung (falls

möglich). Ohne Überprüfung der referentiellen Integrität kann man leicht einen inkonsistenten Zustand der Datenbasis erzeugen, indem z. B. eine Vorlesung mit nichtexistentem Dozenten eingefügt wird.

Zur Gewährleistung der referentiellen Integrität muß also beim Einfügen, Löschen und Ändern immer sichergestellt sein, daß gilt

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

Erlaubte Änderungen sind daher:

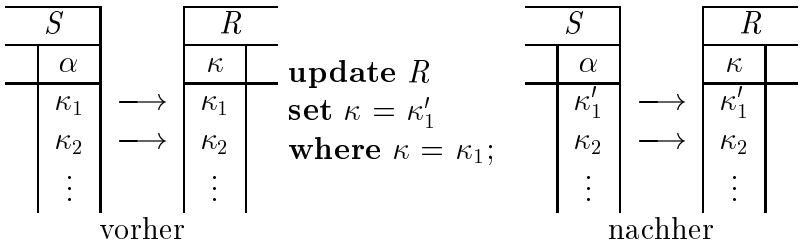
- Einfügen eines Tupels in S verlangt, daß der Fremdschlüssel auf ein existierendes Tupel in R verweist.
- Ändern eines Tupels in S verlangt, daß der neue Fremdschlüssel auf ein existierendes Tupel in R verweist.
- Ändern eines Primärschlüssels in R verlangt, daß kein Tupel aus S auf ihn verwiesen hat.
- Löschen eines Tupels in R verlangt, daß kein Tupel aus S auf ihn verwiesen hat.

8.3 Referentielle Integrität in SQL

SQL bietet folgende Sprachkonstrukte zur Gewährleistung der referentiellen Integrität:

- Ein Schlüsselkandidat wird durch die Angabe von `unique` gekennzeichnet.
- Der Primärschlüssel wird mit `primary key` markiert. Seine Attribute sind automatisch `not null`.
- Ein Fremdschlüssel heißt `foreign key`. Seine Attribute können auch `null` sein, falls nicht explizit `not null` verlangt wird.
- Ein `unique foreign key` modelliert eine 1:1 Beziehung.
- Innerhalb der Tabellendefinition von S legt die Klausel α `integer references R` fest, daß der Fremdschlüssel α (hier vom Typ Integer) sich auf den Primärschlüssel von Tabelle R bezieht. Ein Löschen von Tupeln aus R wird also zurückgewiesen, solange noch Verweise aus S bestehen.
- Durch die Klausel `on update cascade` werden Veränderungen des Primärschlüssels auf den Fremdschlüssel propagiert (Abbildung 8.1a).
- Durch die Klausel `on delete cascade` zieht das Löschen eines Tupels in R das Entfernen des auf ihn verweisenden Tupels in S nach sich (Abbildung 8.1b).
- Durch die Klauseln `on update set null` und `on delete set null` erhalten beim Ändern bzw. Löschen eines Tupels in R die entsprechenden Tupel in S einen Nulleintrag (Abbildung 8.2).

(a) **create table** S $(\ldots, \alpha$ **integer references** R **on update cascade**);



(b) **create table** S $(\ldots, \alpha$ **integer references** R **on delete cascade**);

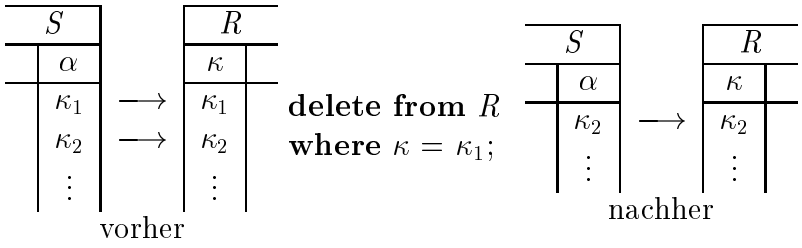
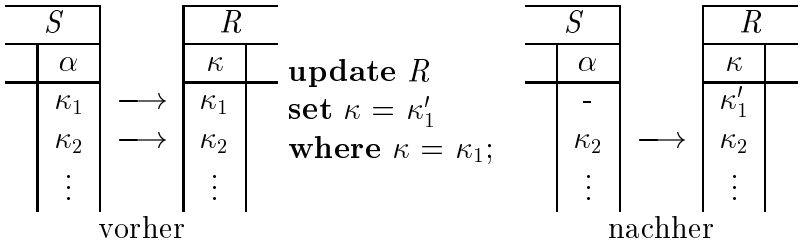


Abbildung 8.1: Referentielle Integrität durch Kaskadieren

a) **create table** S $(\ldots, \alpha$ **integer references** R **on update set null**);



(b) **create table** S $(\ldots, \alpha$ **integer references** R **on delete set null**);

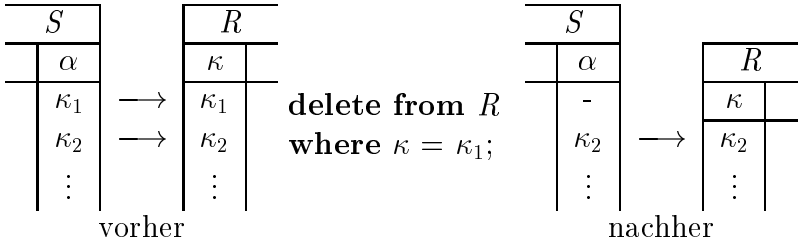
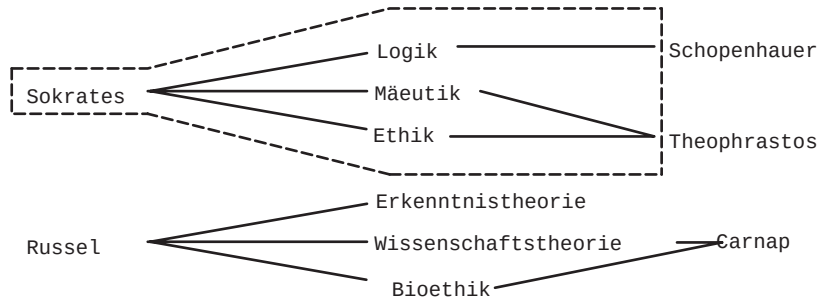


Abbildung 8.2: Referentielle Integrität durch Nullsetzen

Kaskadierendes Löschen kann ggf. eine Kettenreaktion nach sich ziehen. In Abbildung 8.3 wird durch das Löschen von `Sokrates` der gestrichelte Bereich mit drei Vorlesungen und drei *hören*-Beziehungen entfernt, weil der Fremdschlüssel *gelesenVon* die Tupel in *Professoren* mit `on delete cascade` referenziert und der Fremdschlüssel *VorlNr* in *hören* die Tupel in *Vorlesungen* mit `on delete cascade` referenziert.




```
create table Vorlesung
(...,
gelesenVon integer
            references Professoren
            on delete cascade);
```

```
create table hoeren
   (...,
    VorlNr integer
             references   Vorlesungen
           on delete cascade);
```

Abbildung 8.3: Kaskadierende Löschoperationen

8.4 Statische Integrität in SQL

Durch die *check-Klausel* können einem Attribut Bereichseinschränkungen auferlegt werden.

Zum Beispiel erzwingen

```
... check Semester between 1 and 13 ...
... check Rang in ('C2', 'C3', 'C4') ...
```

gewisse Vorgaben für die Semesterzahl bzw. den Professorenrang.

Listing 8.1 zeigt die Formulierung der Uni-Datenbank mit den Klauseln zur Überwachung von statischer und referentieller Integrität.

```
CREATE TABLE Studenten
        (MatrNr         INTEGER PRIMARY KEY,
         Name           VARCHAR2(20) NOT NULL,
         Semester       INTEGER,
         GebDatum       DATE);

CREATE TABLE Professoren
        (PersNr         INTEGER PRIMARY KEY,
         Name           VARCHAR2(20) NOT NULL,
         Rang           CHAR(2) CHECK (Rang in ('C2', 'C3', 'C4')),
         Raum           INTEGER UNIQUE,
         Gebdatum       DATE);

CREATE TABLE Assistenten
        (PersNr         INTEGER PRIMARY KEY,
         Name           VARCHAR2(20) NOT NULL,
         Fachgebiet     VARCHAR2(20),
         Boss           INTEGER,
         FOREIGN KEY    (Boss) REFERENCES Professoren,
         GebDatum       DATE);

CREATE TABLE Vorlesungen
        (VorlNr         INTEGER PRIMARY KEY,
         Titel          VARCHAR2(20),
         SWS            INTEGER,
         gelesenVon     INTEGER REFERENCES Professoren);

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

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

CREATE TABLE pruefen
        (MatrNr         INTEGER REFERENCES Studenten ON DELETE CASCADE,
         VorlNr         INTEGER REFERENCES Vorlesungen,
         PersNr         INTEGER REFERENCES Professoren,
         Note           NUMERIC(2,1) CHECK (Note between 0.7 and 5.0),
         PRIMARY KEY    (MatrNr, VorlNr));
```

Listing 8.1: Universitätsschema mit Integritätsbedingungen

8.5 Trigger

Die allgemeinste Konsistenzsicherung geschieht durch einen *Trigger*. Dies ist eine benutzerdefinierte Prozedur, die automatisch bei Erfüllung einer bestimmten Bedingung vom DBMS gestartet wird. SQL-92 sieht diesen Mechanismus noch nicht vor, wohl aber Oracle.

Listing 8.2 zeigt einen Trigger für die Tabelle Professoren, der vor jedem Update die Tupel mit nichtleerem Rang daraufhin untersucht, ob sie eine Degradierung verursachen würden.

```
create or replace trigger keineDegradierung
before update on Professoren
for each row
when (old.Rang is not null)
declare
begin
  if :old.Rang = 'C3' and :new.Rang ='C2' then :new.Rang := 'C3'; end if;
  if :old.Rang = 'C4'                     then :new.Rang := 'C4'; end if;
  if :new.Rang is null                    then :new.Rang := :old.Rang;
end if;
end;
```

Listing 8.2: Trigger zur Verhinderung einer Degradierung

Listing 8.3 zeigt einen Trigger, der immer nach dem Einfügen eines Tupels in die Tabelle *hoeren* einen Professor sucht, der jetzt mehr als 10 Hörer hat und ihn dann nach C4 befördert.

```
create or replace trigger befoerderung
after update on hoeren
declare
begin
UPDATE professoren SET rang = 'C4'
WHERE persnr in
  (SELECT persnr
   FROM professoren, vorlesungen v, hoeren h
   WHERE persnr = v.gelesenvon
   and v.vorlnr = h.vorlnr
   GROUP BY persnr
   having count(*) > 10);
end;
```

Listing 8.3: Trigger zum Auslösen einer Beförderung

Das Schlüsselwort `drop` entfernt einen Trigger:

```
drop trigger befoerderung;
```