

Datenbanken 1 Lernskript

Dozent:
Pro. Dr. Stefan Kramer

L^AT_EX von:
Sven Bamberger

Zuletzt Aktualisiert:
28. Februar 2015



JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Abstraktionsebenen:	1
1.3	Datenbankschema	1
1.4	Datenbankausprägung	1
2	Datenbankentwurf	2
2.1	Abstraktionsebenen	2
2.2	Datenbankentwurfsschritte:	2
2.3	Generalisierung & Aggregation	2
2.4	Notation	3
2.4.1	min,max	3
2.4.2	Funktionalität	3
2.4.3	UML	3
2.4.4	Überblick	4
3	Das relationale Modell	5
3.1	relationales Schema	5
3.2	Relationale Algebra	5
3.2.1	Selektion σ	5
3.2.2	Projektion Π	5
3.2.3	Kartesisches Produkt \times	6
3.2.4	Umbenennen ρ	6
3.2.5	weitere Möglichkeiten bei gleichen Schema	6
3.2.6	Joins	7
3.3	Tupelkalkül	7
3.4	Domänenkalkül	8
3.5	Quantoren Umwandlung	8
4	SQL	9
4.1	Datentypen	9
4.2	Schemadefinition CREATE TABLE	9
4.3	Schemaveränderungen ALTER	10
4.4	Einfügen INSERT INTO	10
4.5	Datenmanipulation DELETE UPDATE	10
4.6	Anfragen SELECT	10
4.6.1	Joins	11
4.6.2	Gruppierung und Aggregation	11
4.6.3	Geschachtelte Anfragen	11
4.6.4	Schachtelung im From-Teil (Bilden einer neuen Relation):	12

4.6.5	Modularisierung von Anfragen with	13
4.6.6	Mengenoperationen union, intersect, except	13
4.6.7	Quantifizierte Anfrage	13
4.7	Null-Werte	14
4.8	Spezielle Sprachkonstrukte	14
4.9	Rekursionen	15
4.10	Sichten	15
5	Datenintegrität	16
5.1	Allgemein	16
5.2	Trigger	16
6	Relationale Entwurfstheorie	17
6.1	Funktionale Abhängigkeiten	17
6.2	Schlüssel	17
6.2.1	Superschlüssel	17
6.2.2	Kandidatenschlüssel	17
6.2.3	Primärschlüssel	18
6.3	Bestimmung Funktionaler Abhängigkeiten	18
6.3.1	kanonische Überdeckung	18
6.4	„Schlechte“ Relationschemata	19
6.4.1	Updateanomalie	19
6.4.2	Einfügeanomalien	19
6.4.3	Löschanomalien	19
6.5	Zerlegung von Relationen	19
6.6	Erste Normalform	19
6.7	Zweite Normalform	19
6.8	Dritte Normalform	20
6.9	Boyce-Codd Normalform	20
6.10	Mehrwertige Abhängigkeiten	21
6.11	Vierte Normalform	21
6.12	Zusammenfassung	21
7	Physische Datenorganisation	22
7.1	ISAM	22
7.2	B-Baum	22
7.3	B ⁺ Baum	23
7.4	Hashing	23
7.5	Erweiterbares Hashing	23
8	Anfragebearbeitung	24
8.1	Äquivalenzen in der relationalen Algebra	24
8.2	Anwendung der Transformationsregeln	24
9	Transaktionsverwaltung	26
9.1	Operationen auf Transaktionsebene	26
9.2	Eigenschaften von Transaktionen	26
9.3	Transaktionsverwaltung in SQL	27
9.4	Zustandsübergänge einer Transaktion	27

Kapitel 1

Einleitung

1.1 Motivation

Motivations zum Einsatz eines DBMS

- Redundanz und Inkonsistenzen verhindern
- Beschränkte Zugriffsmöglichkeiten
- Mehrbenutzerbetrieb
- Verlust von Daten vorbeugen dank Recovery Eigenschaften
- Integritätsverletzungen verhindern

1.2 Abstraktionsebenen:

Physische Ebene: wie werden die Daten gespeichert.

Logische Ebene: welche Daten werden abgespeichert.

Sichten: Teilmengen des Schemas für Benutzergruppen.

1.3 Datenbankschema

Ist die Struktur der abgespeicherten Daten

1.4 Datenbankausprägung

Momentan gültiger Zustand der Datenbasis, gehorcht dem Datenbankschema und muss stets konsistenten Zustand aufweisen.

Kapitel 2

Datenbankentwurf

2.1 Abstraktionsebenen

konzeptuelle Ebene: Entwicklung des konzeptuellen Entwurfs als Informationsstrukturbeschreibung, z.B. mit ERM


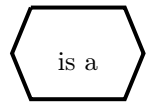

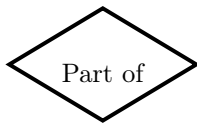
Implementationsebene: Datenmodell des zum Einsatz kommenden Datenbanksystems (z. B. relationales Modell)

Physische Ebene: Hardware, OS, Datenbanksystem, Zeiger und Indexstrukturen für Effizienz

2.2 Datenbankentwurfsschritte:

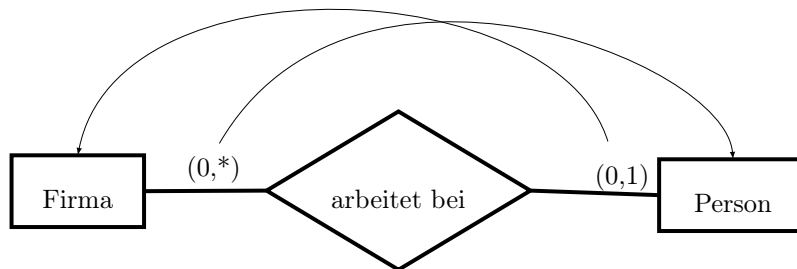
- Anforderungsanalyse (Pflichtenheft)
- konzeptueller Entwurf (ER-Schema), unabhängig von Datenmodell
- Implementationsentwurf (Datenbankschema – z. B. relationales Modell)
- physischer Entwurf

2.3 Generalisierung & Aggregation

	UML	ERM
Generalisierung		
Aggregation		

2.4 Notation

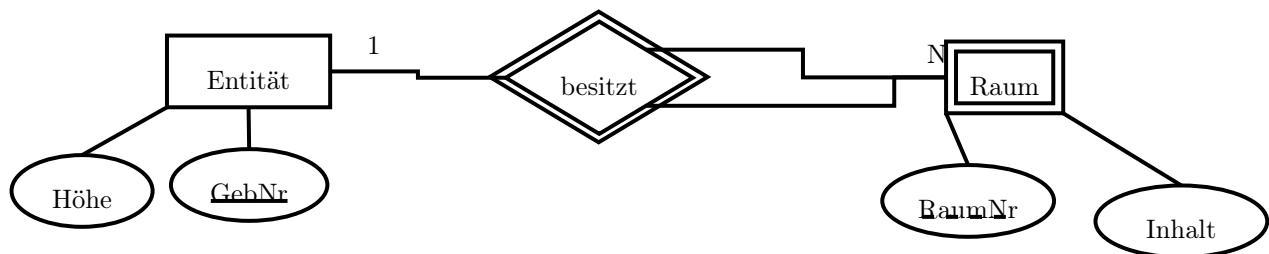
2.4.1 min,max



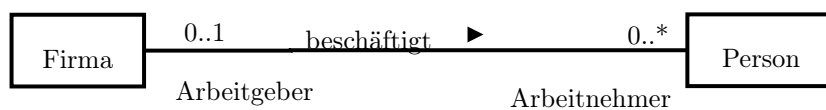
2.4.2 Funktionalität



mit schwachen Entitäten



2.4.3 UML



2.4.4 Überblick

$F_1 : F_2$	$(\min_1, \max_1) (\min_2, \max_2)$	UML
1 : 1	(0,1) (0,1)	0..1 0..1
1: N	(0,*) (0,1)	0..1 0.. *
N : 1	(0,1) (0,*)	0..* 0..1
N : M	(0,*) (0,*)	0..* 0..*

Kapitel 3

Das relationale Modell

3.1 relationales Schema

Darstellung von Entitytypen:

Student: {[MatNr: integer, Name:string, Semester: integer]}

Allgemein:

Entityname (Tabellenname) : {[x:y | x = Attributbeschreibung; y = Datentyp]}

Bei dem Initialen Entwurf ist jedwede Relation und Entität in diesem Schema vorhanden. Im nächsten Schritt werden alle Relationen mit dem gleichen Schlüssel zusammengefasst, dabei gilt es zu beachten, das 1:1; 1:N; N:1 Beziehungen mit Fremdschlüssel zusammen gefasst werden können. Auch sollte nach Möglichkeit Nullwerte vermieden werden.

3.2 Relationale Algebra

3.2.1 Selektion σ

Bei der *Selektion* werden diejenigen Tupel einer Relation ausgewählt, die das sogenannte *Selektionsprädikat* erfüllen.

$\sigma_{\text{Semester} > 10}(\text{Studenten})$

```
SELECT *  
FROM Student  
WHERE Semester > 10;
```

3.2.2 Projektion Π

Extraktion von Attributen von einer Menge

$\Pi_{\text{Rang}}(\text{Professoren})$

```
SELECT Rang  
FROM Professoren
```


3.2.3 Kartesisches Produkt \times

$R \times S$ enthält alle möglichen Paare von Tupel aus R und S .

```
SELECT *
FROM Professoren , Assistenten
```

3.2.4 Umbenennen ρ

ρ wird benötigt, sofern eine Relation mehrfach verwendet wird.

$\Pi_{V1.Vorgänger}(\sigma_{V2.Nachfolger = 5216 \wedge V1.Nachfolger = V2.Vorgänger}(\rho_{V1}(\text{vorraussetzen}) \times \rho_{V2}(\text{vorraussetzen})))$
indirekter Vorgänger von Vorlesung 5216

Attributumbenennung:

$\rho_{\text{Vorraussetzung} \leftarrow \text{Vorgänger}}(\text{vorraussetzen})$

```
SELECT Vorgaenger AS Vorraussetzung
FROM vorraussetzen;
```

3.2.5 weitere Möglichkeiten bei gleichen Schema

Vereinigung \cup UNION

Vereinigt 2 Mengen, welche projiziert wurden. Union hat automatisch Duplikateliminierung! Mit union all vermeidbar. Nur Zeichenketten mit Zeichenketten vereinen.

```
(SELECT Name
 FROM Assistenten)
UNION
(SELECT Name
 FROM Professoren);
```

Mengendifferenz – EXCEPT

Es wird Tupelmenge aus S von Tupelmenge aus R abgezogen, z. B. Matrikelnummer der Studenten subtrahiert um die Matrikelnummer der Studenten, die Prüfungen geschrieben haben -> wer hat noch keine Prüfung abgelegt?

Schnittmenge \cap INTERSECT

Schnittmenge der Tupelmenge zweier Relationen mit gleichem Schema *rightarrow* Selektion der Tupel, die beide Relationen enthalten.

$\Pi_{\text{PersNr}}(\rho_{\text{PersNr} \leftarrow \text{gelesenVon}}(\text{Vorlesungen})) \cap \Pi_{\text{PersNr}}(\sigma_{\text{Rang}=\text{C4}}(\text{Professoren}))$

Division \div

$R \div S$

Es bleiben alle Elemente von R übrig die vollständig gejoint werden können mit S .

Gruppierung und Aggregation γ

$\gamma_{\text{Semester}; \text{COUNT}(*)(\text{Studenten})}$

Zählt die Studenten basierend auf deren Semesterwert. Also, wie viele Studenten gibt es in den entsprechenden Semestern.

3.2.6 Joins

natürlicher Join \bowtie

$$R \bowtie S$$

Kreuzprodukt über Gleichnamige Attributspalten

Theta-Join \bowtie_{Θ}

$$R \bowtie_{\Theta} S$$

Θ wird ersetzt mit einer Selektion, Projektion oder Umbenennung. Damit die Tabellen entsprechend vor dem Join angepasst und verkleinert werden.

linker äußerer Join

$$R \Joinleft S$$

Alles aus R bleibt erhalten und Fehlende Kombinationen werden mit Null aufgefüllt.

rechter äußerer Join

$$R \Joinright S$$

Alles aus S bleibt erhalten und Fehlende Kombinationen werden mit Null aufgefüllt.

äußerer Join

$$R \Join S$$

Alles aus R und S bleibt erhalten und Fehlende Kombinationen werden mit Null aufgefüllt.

Semi-Join

$$R \ltimes S$$

Passende Elemente aus R bleiben erhalten und nur aus R

$$R \rtimes S$$

Passende Elemente aus S bleiben erhalten und nur aus S

Anti-Semi Join

$$R \rhd S$$

NICHT passende Elemente aus R bleiben erhalten und nur die aus R .

$$R \lhd S$$

NICHT passende Elemente aus S bleiben erhalten und nur die aus S .

3.3 Tupelkalkül

generische Form: $\{t|P(t)\}$

Um sichere Anfragen gestalten zu können benötigt man eine Domäne. Dies schränkt die Menge auf eine endliche Menge ein.

Anfrage auf alle C4 Professoren: $\{p \mid p \in \text{Professoren} \wedge \underbrace{p.\text{Rang} = \text{'C4'}}_{\text{Domäne}}\}$

```

SELECT *
FROM Professoren
WHERE Rang = 'C4'

```

1. Das Tupel p muss in der Relation *Professoren* enthalten sein.
2. Das Tupel p muss für das Attribut *Rang* den Wert 'C4' besitzen.

Anfragen können auch mit Quantifizierungen gestellt werden.

$\exists t \in R(Q(t))$ bzw. $\forall t \in R(Q(t))$
 Studenten, welche mindestens eine Vorlesung bei 'Curie' gehört haben.
 $\{s \mid s \in \text{Studenten}$
 $\quad \wedge \exists h \in \text{hören}(s.\text{MatrNr} = h.\text{MatrNr}$
 $\quad \wedge \exists v \in \text{Vorlesungen}(h.\text{VorlNr} = v.\text{VorlNr}$
 $\quad \wedge \exists p \in \text{Professoren}(v.\text{PersNr} = v.\text{gelesenVon}$
 $\quad \wedge p.\text{Name} = \text{'Curie'})\})\}$

```

SELECT s.Name
FROM Studenten s, hoeren h, Vorlesungen v, Professoren p
WHERE s.MatrNr = h.MatrNr AND h.VorlNr = v.VorlNr
AND p.PersNr = v.gelesenVon AND p.Name = 'Curie';

```

3.4 Domänenkalkül

Struktur: $\{[v_1, v_2, \dots, v_n] \mid P(v_1, \dots, v_n)\}$

v_1, v_2, \dots, v_n sind die Attribute welche man erhalten will. $P(v_1, \dots, v_n)$ ist die genaue Anfragenkonstruktion, hier müssen alle Attribute eine eindeutige Variable bekommen.

Implizite Angabe der Joins über die gleiche Variablenbelegung. explizit durch Gleichsetzung der Impliziten Variablen.

$\{[n] \mid \exists s([m, n, s] \in \text{Studenten}$
 $\quad \wedge \exists v([m, v] \in \text{ hoeren}$
 $\quad \wedge \exists p([v, t, sw, p] \in \text{Vorlesungen}$
 $\quad \wedge \exists a([p, a, r, b] \in \text{Professoren} \wedge a = \text{'Curie'}))\})\}$

3.5 Quantoren Umwandlung

$\forall t \in R(P(t)) = \neg(\exists t \in R(\neg P(t)))$
 $\exists t \in R(P(t)) = \neg(\forall t \in R(\neg P(t)))$

Kapitel 4

SQL

4.1 Datentypen

```
date                // Datum
char(x)             // Zeichenkette wird mit Leerzeichen aufgefuellt
varchar(x)          // Zeichenkette wird NICHT aufgefuellt
numeric(p,s)        // Zahl p = totale laenge s = # nachkommastellen
integer / int
float
blob / raw          // fuer binaer Dateien
xml                 // fuer XML Dateien
```

4.2 Schemadefinition CREATE TABLE

```
CREATE TABLE Professoren(
  PersNr integer primary key,
  Name varchar(30) NOT NULL,
  Rang char(2) check (Rang in ('C2', 'C3', 'C4')),
  Raum integer unique,
);
CREATE TABLE pruefen(
  MatrNr integer references Studenten on delete cascade,
  VorlNr integer references Vorlesungen,
  PersNr integer,
  Note numer(2,1) check (Note between 0.7 and 5.0),
  primary key (MatrNr, VorlNr),
  foreign key (PersNr) references Professoren on delete set null
  constraint Vorher hoeren
    check(exists(select *
                  from hoeren h
                  where h.VorlNr = pruefen.VorlNr and
                        h.MatrNr = pruefen.MatrNr))
);
```

4.3 Schemaveränderungen ALTER

```
ALTER TABLE tabellenname
    add (Attribut Datentyp);
    add column Attribut Datentyp;
    modify (Attribut Datentyp); // Bedingungen wie NOT NULL bleiben erhalten.
    alter column Attribut Datentyp;
```

4.4 Einfügen INSERT INTO

```
INSERT INTO Studenten (MatrNr, Name)
    values (28121, 'Archimedes');
```

```
INSERT INTO hoeren
    SELECT MatrNr, VorlNr
    FROM Studenten, Vorlesungen
    WHERE Titel = 'Logik';
```

4.5 Datenmanipulation DELETE UPDATE

```
DELETE FROM Studenten
    WHERE Semester > 13;
```

```
DELETE FROM voraussetzen
    WHERE Vorgaenger in (SELECT Nachfolger
                        FROM voraussetzen);
```

```
UPDATE Studenten
    SET Semester = Semester +1;
```

4.6 Anfragen SELECT

```
SELECT DISTINCT Rang // zeige elemente der select Menge
    FROM Professoren;
```

```
SELECT PersNr, Name
    FROM Professoren
    WHERE Rang = 'C4'
    order by Rang desc, Name asc
```

SELECT = Π FROM = \times WHERE = σ

4.6.1 Joins

Implizite:

```
SELECT s.Name
FROM Studenten s, hoeren h, Vorlesungen v, Professoren p
WHERE s.MatrNr = h.MatrNr AND h.VorlNr = v.VorlNr
AND p.PersNr = v.gelesenVon AND p.Name = 'Curie';
```

Explizit:

```
SELECT p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from Professoren p join ( pruefen f join Studenten s
on f.MatrNr = s.MatrNr)
on p.PersNr = f.PersNr;
```

4.6.2 Gruppierung und Aggregation

Gruppierung von Werten mit einem Wert durch avg, max, min, sum, count

```
SELECT gelesenVon, Name, sum(sws)
FROM Vorlesungen
WHERE gelesenVon = PersNr and Rang = 'C4'
Group by gelesenVon, Name
Having avg(SWS) >= 3;
```

In der Select-Klausel dürfen nur Aggregatfunktionen oder Attribute, nach denen gruppiert wurde verwendet werden. Somit muss Name auch in die Group by - Klausel (weil nur ein Tupel pro Gruppe vorhanden ist)

4.6.3 Geschachtelte Anfragen

Zu unterscheiden sind dabei Anfragen, die nur ein Tupel zurückliefern und Anfragen, die mehrere Tupel liefern.

Wird nur ein Tupel geliefert kann in Select-klausel oder where-Klausel Schachtelung verwendet werden, bei denen ein skalarer Wert gefordert wird.

```
Select *
From pruefen
Where Note = ( select avg(Note)
From pruefen);
```

```
Select PersNr, Name, ( select sum(SWS) as Lehrbelastung
From Vorlesungen
Where gelesenVon = PersNr)
From Professoren
```

Erste Anfrage: ist eigenständig, verwendet nur ihre eigenen Attribute, sie wird nur einmal ausgewertet

Zweite Anfrage: ist korreliert, sie verwendet Attribute aus ihrer äußeren Hülle und muss so für jedes Tupel neu durchgeführt werden (für jedes zu überprüfende Tupel, wenn Unteranfrage in where-Bedingung steht und jedes auszugebende Tupel, wenn sie in select-klausel steht)

Noch ein Beispiel:

```

Select s.*                                // korreliert
From Studenten
Where exists ( select p.*
                From Professoren
                Where p.GebDatum < s.GebDatum );

```

```

Select s.*                                // unkorreliert
From Studenten
Where s.GebDatum < ( select max(p.Gebdatum)
                    From Professoren p );

```

Noch ein Beispiel (nur durch Join kann Anfrage unkorreliert werden):

```

Select a.*
From Assistenten
Where exists ( select p.*
                From Professoren
                Where p.PersNr = a.Boss and p.GebDatum > a.GebDatum );

```

```

Select a.*
From Assistenten a, Professoren p
Where a.GebDatum < p.GebDatum and a.Boss = p.PersNr;

```

4.6.4 Schachtelung im From-Teil (Bilden einer neuen Relation):

```

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;

```

```

Select h.VorlNr, h.StudProVorl, g.GesStud, h.StudProVorl /
g.GesStud as Marktanteil
From ( select VorlNr, count(*) as StudProVorl
        From hoeren h
        Group by VorlNr; ) h,
      ( select count(*) as GesStud
        From Studenten s ) g;

```

4.6.5 Modularisierung von Anfragen with

Das Keyword with generiert temporäre Sichten für die Dauer der Abfragebearbeitung. Dadurch können Anfragen übersichtlicher werden.

```
with h as (select VorlNr, count(*) as StudProVorl
           From hoeren h
           Group by VorlNr )
      g as (select count(*) as GesStud
           From Studenten s )

Select h.VorlNr, h.StudProVorl, g.GesStud, h.StudProVorl /
g.GesStud as Marktanteil
      From h, g;
```

4.6.6 Mengenoperationen union, intersect, except

```
(SELECT Name
 FROM Assistenten)
UNION
(SELECT Name
 FROM Professoren);

SELECT Name
FROM Professoren
WHERE PersNr not in (SELECT gelesenVon
                    from Vorlesungen);
```

4.6.7 Quantifizierte Anfrage

- in ist äquivalent zu any
- any und all sind quantifizierende Bedingungen mit Vergleichsoperatoren
- any testet, ob es mindestens ein Element im Ergebnis der Unteranfrage gibt, für das der Vergleich mit linkem Argument erfüllt ist
- all prüft ob alle Elemente die Bedingung erfüllen -> allerdings ist all nicht der Allquantor aus den beschreibenden Sprachen

```
Select Name
from Studenten
where Semester >= all (
    select Semester
    from Studenten );           // effizienter waere hier der max-Operator
```

Existenzquantifizierer

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


Allquantifizierung

Da es leider keinen Allquantor in SQL gibt, müssen solche Anfragen über einen Existenzquantor ausgedrückt werden. Nachfolgend ein Beispiel

$$\{s \mid s \in \text{Studenten} \wedge \forall v \in \text{Vorlesungen}(v.SWS = 4 \Rightarrow \exists h \in \text{ hoeren}(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Umformen mit Hilfe von:

$$\begin{aligned} \forall t \in R(P(t)) &= \neg(\exists t \in R(\neg P(t))) \\ R \Rightarrow T &= \neg R \vee T \end{aligned}$$

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen}(\neg(\neg(v.SWS = 4) \vee \neg(\exists h \in \text{ hoeren}(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))))))\}$$

Mit der Regel von De-Morgan die Negation nach innen ziehen:

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen}(v.SWS = 4 \wedge \neg(\exists h \in \text{ hoeren}(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))))))\}$$

4.7 Null-Werte

Null wird zu unknown ausgewertet und hat teilweise ein seltsames Verhalten.

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	unknown	false	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

4.8 Spezielle Sprachkonstrukte

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

```

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

select *
  from Studenten
 where Name like 'T%eophrastos';

select Matr, ( case when Note < 1.5 then 'sehr_gut'
                   when Note < 2.5 then 'gut'
                   when Note < 3.5 then 'befriedigend'
                   when Note <= 4.0 then 'ausreichend'
                   else 'nicht_bestanden' end)
  from pruefen;

```

4.9 Rekursionen

Rekursionen sind im aktuellen SQL 92 Standard nicht integriert und müssen daher immer explizit komplett ausgeschreiben werden. wie z.B.

```

SELECT v1.Vorgaenger
  FROM voraussetzen v1, voraussetzen v2, Vorlesungen v
 WHERE v1.Nachfolger = v2.Vorgaenger and
       v2.Nachfolger = v.VorlNr and
       v.Titel = 'Der_Wiener_Kreis';

```

4.10 Sichten

Sichten sind sehr nützlich, um Standardabfragen, welche über mehrere Tabellen gehen, welche sich nur sehr selten ändern zu verschnellern. Beispiel:

```

CREATE VIEW pruefenSicht as
  SELECT MatrNr, VorlNr, PersNr
  FROM pruefen;

```

Kapitel 5

Datenintegrität

5.1 Allgemein

Datenintegrität meint semantische Integritätsbedingungen, die sich aus der zu modellierenden Miniwelt ableiten → Modell soll Konsistenz der Daten gewährleisten

Es gibt statische (Wertebereiche mit check zu realisieren) und dynamische (Update-Prüfung mit Trigger zu realisieren) Integritätsbedingungen

Schlüssel haben Unique-Charakter, Kardinalitäten legen Beziehungen fest z. B. anhand von Fremdschlüsseln, jedes Attribut hat Domäne z. B. 5 Ziffern für Matrikelnummer

Referentielle Integrität = ein Fremdschlüssel von R enthält entweder Null-Werte oder einen in S vorkommenden Primärschlüssel (auch mehrere Attribute als Schlüssel denkbar)

- in SQL durch unique, primary key → not null, foreign key → unique foreign key bezeichnet 1:1-Beziehung
- cascade → Änderungen Primärschlüssel bedingen Änderungen Fremdschlüssel, on delete cascade → zahlreiche Löschoperationen, on update/delete set null → keine kaskadierenden Löschoperationen

5.2 Trigger

Diese ermöglichen das eine Datenbank bei Änderungen de Inhalts oder Schemas selbst aktiv wird und macht somit eine Datendank zu einer aktiven Datenbank.

```
create trigger keineDegradierung before update on Professoren
for each row
when (old.Rang is not null)
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
```

Kapitel 6

Relationale Entwurfstheorie

6.1 Funktionale Abhängigkeiten

Eine funktionale Abhängigkeit stellt eine Bedingung an die möglichen gültigen Ausprägungen des Datenbankschemas dar.

Darstellung: $\alpha \rightarrow \beta$ Dies bedeutet, dass α β funktional bestimmt. Also man kann von α direkt auf β schließen. Somit ist α die Determinante von β . Dabei ist zu beachten, dass α und β Mengen von Attributen repräsentieren.

Funktionale Abhängigkeiten stellen Konsistenzbedingungen dar, die zu allen Zeiten in jedem (gültigen) Datenbankzustand eingehalten werden müssen.

6.2 Schlüssel

6.2.1 Superschlüssel

$\alpha \subseteq R$ ist ein Superschlüssel, falls gilt

$$\alpha \rightarrow R$$

Also α bestimmt alle anderen Attributwerte. Dabei gilt zu beachten, dass

$$R \rightarrow R$$

Also bildet die Menge aller Attribute einer Relation einen Superschlüssel. Das heißt, Superschlüssel sind nicht minimal und jede funktionale Abhängigkeit, von der aus man alle anderen Attribute erreichen kann, sind Superschlüssel.

6.2.2 Kandidatenschlüssel

Kandidatenschlüssel sind Superschlüssel mit folgenden Eigenschaften.

1. $\alpha \rightarrow \beta$ d.h. β ist funktional abhängig von α
2. α kann nicht mehr „verkleinert“ werden. Dies gilt nach Anwendung der Kanonischen Überdeckung.

6.2.3 Primärschlüssel

Ein Primärschlüssel ist ein ausgewählter Kandidatenschlüssel. Denn alle Kandidatenschlüssel können als Primärschlüssel herhalten.

6.3 Bestimmung Funktionaler Abhängigkeiten

Durch Überlegen und scharfes hinsehen können FD's erarbeitet werden aus einem vorherigen Relationalen Schema. Oder durch ermitteln der Attributhülle.

Dann ermittelt man die Attributhüllen der einzelnen Funktionalen Abhängigkeiten. Beispiel:

Man gehe von $F' = \{AB \rightarrow C, AB \rightarrow D, A \rightarrow E, E \rightarrow D, AD \rightarrow A, AD \rightarrow F\}$ aus.

Erstelle Attributhülle von A:

1. A ist auf alle Fälle immer erreichbar, wenn A gegeben ist, daher ist A in der Attributhülle.
2. Der nächste nur durch A erreichbare Buchstabe ist E ($A \rightarrow E$). E wird zur Attributhülle gegeben
3. Nachdem E jetzt vorhanden ist, kann ich auch D erreichen ($E \rightarrow D$). D zur Attributhülle hinzufügen.
4. Da A und D vorhanden sind, kann nun auch F hinzugefügt werden ($AD \rightarrow F$).

Also Attributhülle von A ist somit AEDF

Erstelle Attributhülle von B:

1. B ist auf alle Fälle immer erreichbar, wenn B gegeben ist, daher ist B in der Attributhülle.
2. Sonst ist nichts mehr erreichbar.

Attributhülle von B ist somit B

Erstelle Attributhülle von D:

1. D ist auf alle Fälle immer erreichbar, wenn D gegeben ist, daher ist D in der Attributhülle.

Attributhülle von D ist somit D

6.3.1 kanonische Überdeckung

1. Linksreduktion: Überprüfe in jeder funktionalen Abhängigkeit $\alpha \rightarrow \beta \in F$ ob $A \in \alpha$ überflüssig ist
Wenn $\beta \subseteq \text{AttrHuelle}(F, \alpha - A)$, dann ist A überflüssig. Ersetze $\alpha \rightarrow \beta$ durch $(\alpha - A) \rightarrow \beta$
2. Rechtsreduktion: Überprüfe in jeder funktionalen Abhängigkeit $\alpha \rightarrow \beta \in F'$ ob $B \in \beta$ überflüssig ist
Wenn $B \in \text{AttrHuelle}(F' - (\alpha \rightarrow \beta) \cup (\alpha \rightarrow (\beta - B)), \alpha)$, dann ist B überflüssig. Ersetze $\alpha \rightarrow \beta$ durch $\alpha \rightarrow (\beta - B)$
3. Entferne alle $\alpha \rightarrow$ für beliebige α
4. Vereinige alle $\alpha \rightarrow \beta_i$ zu $\alpha \rightarrow (\beta_1 \cup \dots \cup \beta_n)$

Umgangssprachlich formuliert:

- Komme ich, falls ich etwas streiche, direkt auf meine Attributhülle? Wenn ja, streiche es. Falls nein lasse es drinnen. Führe diesen Schritt für alle links und rechts aus, bis alles minimal ist.
- Streiche leere FD's
- Füge gleiche Linke Seiten zusammen und deren rechten Seiten.

6.4 „Schlechte“ Relationschemata

Schlecht entworfene Relationenschemata können zu sogenannten *Anomalien* führen.

6.4.1 Updateanomalie

Lediglich ein kleiner Teil der Daten werden auf den neusten Stand gebracht, da das Schemata schlecht entworfen wurde. Folgen:

1. Erhöhter Speicherbedarf wegen der redundant zu speichernden Informationen
2. Leistungseinbußen bei Änderungen, da mehrere Einträge abgeändert werden müssen.

6.4.2 Einfügeanomalien

Es kann zu unnötigen NULL Werten oder zum zwangsweise einfügen von unnötig Redundaten Daten kommen.

6.4.3 Löschanomalien

Nicht alle zu löschenden Daten werden oder können gelöscht werden, da sonst starke Inkonsistenzen entstehen, teilweise müssen die Daten mit NULL ersetzt werden.

6.5 Zerlegung von Relationen

Um diese schlechten Schemata zu vermeiden sollten Relationen Zerlegt werden um diese in die passende Normalform zu bringen. Dabei gilt es folgendes zu beachten.

1. *Verlustlosigkeit* Die in der ursprünglichen Relation enthaltenen Informationen müssen rekonstruierbar sein.
2. *Abhängigkeitserhaltend* Die geltenden funktionalen Abhängigkeiten müssen auf das neue Schemata übertragbar sein. (hüllentreue Dekomposition.)

6.6 Erste Normalform

1. alle Attribute haben atomare Wertebereiche.

Die Attribute enthalten keine Mengen, jedoch sind Verschachtelungen erlaubt.

6.7 Zweite Normalform

Eine Relation mit zugehörigen FD's ist in zweiter Normalform, falls jedes Nichtschlüssel Attribut $A \in R$ voll funktional abhängig ist von jedem Kandidatenschlüssel. Jedoch können dadurch jede Kombination von Attributen, welche voll funktional abhängig sind in einer Tabelle gespeichert werden, dies führt noch immer zu Anomalien. (Sieht aus wie ein \times von zwei Tabellen *würgs*)

6.8 Dritte Normalform

Ein Relationenschema R ist in *dritter Normalform*, wenn für jede für R geltende funktionale Abhängigkeit der Form $\alpha \rightarrow B$ mit $\alpha \subseteq R$ und $B \in R$ mindestens eine von drei Bedingungen gilt:

1. $B \in \alpha$ d.h. die D ist trivial
2. Das Attribut B ist in einem Kandidatenschlüssel von R enthalten
3. α ist Superschlüssel von R

Wie bekommt man nun die dritte Normalform hin?

- Ziel = Synthesealgorithmus sorgt dafür, dass
 - R_1, \dots, R_n verlustlose Zerlegung von R ist
 - Alle Abhängigkeiten bewahrt werden
 - Alle R_i in dritter Normalform sind

Synthesealgorithmus

1. Reduktion von F , d.h. die Bestimmung der kanonischen Überdeckung
2. Erzeugen der neuen Relationenschemata aus der kanonischen Überdeckung
3. ggf. die Hinzunahme einer Relation, die nur den Ursprungsschlüssel enthält
4. Elimination der Schemata, die in einem anderen Schema enthalten sind

6.9 Boyce-Codd Normalform

Ein Relationenschema ist in BCNF, wenn es in der 3NF ist und jede Determinante (Attributmenge, von der andere Attribute funktional abhängen) ein Superschlüssel ist (oder die Abhängigkeit ist trivial). Die Überführung in die BCNF ist zwar immer verlustfrei möglich, aber nicht immer abhängigkeiterhaltend. Die Boyce-Codd-Normalform war ursprünglich als Vereinfachung der 3NF gedacht, führte aber zu einer neuen Normalform, die diese verschärft: Eine Relation ist automatisch frei von transitiven Abhängigkeiten, wenn alle Determinanten Schlüsselkandidaten sind.

Um dies hinzu bekommen, benötigt man den Dekompensitionsalgorithmus

1. Gegeben ist ein relationales Schema $R = (\overline{R}, \mathcal{F})$, mit der Menge aller Attribute \overline{R} und der Menge der funktionalen Abhängigkeiten \mathcal{F} über diesen Attributen.
2. Die Ergebnismenge Dekomposition, bestehend aus den zerlegten Schemata, wird mit R initialisiert.
3. Solange es ein Schema S in der Menge Dekomposition gibt, das nicht in der BCNF ist, führe folgende Zerlegung
4. Sei $\overline{XY} \subseteq \overline{S}$ eine Attributmenge für die eine funktionale Abhängigkeit $\overline{X} \rightarrow \overline{Y}$ definiert ist, welche der BCNF widerspricht.
5. Ersetze S in der Ergebnismenge Dekomposition durch zwei neue Schemata $S_1 = (\overline{XY}, \mathcal{F}_1)$, ein Schema bestehend nur aus den Attributen der Abhängigkeit, welche die BCNF ursprünglich verletzt hat; und $S_2 = ((\overline{S} - \overline{Y}) \cup \overline{X}, \mathcal{F}_2)$, ein Schema mit allen Attributen, außer denen die nur in der abhängigen Menge \overline{Y} und nicht in der Determinante \overline{X} enthalten sind. Die Menge der funktionalen Abhängigkeiten \mathcal{F}_1 enthält nur noch die Abhängigkeiten, welche lediglich Attribute aus \overline{XY} enthalten, entsprechendes gilt für \mathcal{F}_2 . Damit fallen alle Abhängigkeiten weg, welche Attribute aus beiden Schemata benötigen.
6. Ergebnis: Dekomposition – eine Menge von relationalen Schemata, welche in der BCNF sind.

6.10 Mehrwertige Abhängigkeiten

Mehrwertige Abhängigkeiten sind eine Verallgemeinerung funktionaler Abhängigkeiten, d.h. jede FD ist auch eine MVD, aber nicht umgekehrt. Beispiel:

PersNr	Sprache	Programmiersprache
3002	griechisch	C
3002	lateinisch	Pascal
3002	griechisch	Pascal
3002	lateinisch	C
3005	deutsch	Ada

Hier sind eindeutig Redundanzen zu erkennen welche eliminiert werden müssen. Denn die Personalnummer und die natürliche Sprache ermitteln zusammen die Programmiersprache aber auch die Personalnummer in Verbindung mit der Programmiersprache ergibt die natürliche Sprache. Daher definiert die Personalnummer zum einen die natürliche Sprache als auch die Programmiersprache. jedoch erst in Kombination kann es funktionieren. Diese MVD's zu beseitigen zerlegt man diese Tabelle in 2 Tabellen, wo nur noch die Personalnummer entscheidend ist.

6.11 Vierte Normalform

Relation R mit Menge D von FD's und MVD's ist in 4NF, wenn für jede MVD $\alpha \twoheadrightarrow \beta$ der Hülle D^+ gilt MVD ist trivial ODER Alpha ist Superschlüssel von R \rightarrow gleiche Bedingungen wie BCNF! Nur allgemeiner durch MVD (Überdeckung)

Vorgehen:

1. Zerlege die Schemata in die passenden Abhängigkeiten
achte zunächst noch nicht auf MVD und bilde diese zusammen
2. Zerlege als nächstes die MVD's in eigene Schemata.

6.12 Zusammenfassung

- $1\text{ NF} < 2\text{ NF} < 3\text{ NF} < \text{BCNF} < 4\text{ NF}$
- Die Verlustlosigkeit ist überall gewährleistet
- Die Abhängigkeitserhaltung kann nur bei den Zerlegungen bis zur dritten Normalform garantiert werden.

Kapitel 7

Physische Datenorganisation

7.1 ISAM

Index-Sequentical Access Method

Funktioniert wie SkipList nur ohne den Zufall und ist daher beim Einfügen recht aufwendig.

Suchen: Man macht eine Binärsuche auf den Indexseiten bis man die Stelle gefunden hat, die entweder der Wert ist oder wo der Wert dazwischen liegt.

Einfügen: Leider sehr aufwendig. Wenn Datenseite füllt ist, muss ein Teil von dieser in die Indexseiten verschoben werden und eventuell eine neue Seite gebaut werden. wo die Daten drin stehen.

Löschen: Löschen ist solange einfach, bis die Datenseite leer ist, dann muss eventuell ein Indexseiten shift gemacht werden, um zu große Sprünge zu vermeiden, oder falls der Wert aus der Indexseite gelöscht wird, muss dieser Ersetzt werde.

7.2 B-Baum

- Die Knoten des Baumes sind meist auf eine Seite abgestimmt -> feste Grenzen für die Anzahl und Auslastung von Seitenzugriffen
- Grad eines Baumes bezeichnet die Anzahl der Einträge pro Knoten -> Grad k = minimal k Einträge, maximal $2k$ Einträge bis zur Verlagerung von Daten
- Hat ein Knoten n Einträge, hat er $n+1$ Kinder (außer Kinder)

Einfügealgorithmus

1. . Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.
2. Füge den Schlüssel dort ein.
3. Ist der Knoten überfüllt, teile ihn:

Erzeuge einen neuen Knoten und belege ihn mit den Einträgen des überfüllten Knotens, deren Schlüssel größer ist als der des mittleren Eintrags.

Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.

Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten.

4. Ist der Vaterknoten jetzt überfüllt?

Handelt es sich um die Wurzel, so lege eine neue Wurzel an.

Wiederhole Schritt 3 mit dem Vaterknoten.

Löschen

- Im Blatt einfach Löschen
- Wert im Vaterknoten? Suche nächstkleineren / -größeren Wert und verschiebe diesen hoch.
- verschmelze Kinder falls notwendig

7.3 B⁺ Baum

- Die Daten werden nur noch in den Blättern gespeichert – somit keine Verzweigung und Zugriff erst im Blattknoten (auch: hohler Baum)
- Zusätzlich sind Blattknoten mit vor- und nachgelagerten Blättern verzweigt
- B+-Baum vom Typ (k, k^*) hat also gleiche Wegelängen zum Blatt mit Datenzugriff
- Jeder Knoten hat $k - 2k$ Einträge \rightarrow Blätter haben $k^* - 2k^*$ Einträge
- Jeder Knoten mit n Einträgen außer Blättern hat $n+1$ Kinder
- Wartung ist einfacher, da nur Referenzen umgebogen werden müssen bei Zusammenlegungen, Löschungen in Knoten (Daten sind nur in den Blättern)
- AUCH: Präfix-B+-Baum = da nur Referenzen verwendet werden, können Synonyme bzw. kleine Schlüssel verwendet werden, z. B. Präfixe wie Chars für ganze String-Daten

7.4 Hashing

- Durch Hashfunktion $h : S \rightarrow B$ wird Schlüssel auf Behälter abgebildet, der die Daten enthält.
- Da nicht für den gesamten Wertebereich des Schlüssels Platz vorhanden ist, werden Datensätze oft an gleichen Stellen gespeichert (Kollisionsbehandlung) $\rightarrow h$ ist also nicht injektiv
- Oft verwendet / statisch!: Schlüsselwert mod Tabellengröße $\rightarrow h(x) = x \bmod 3$, wenn 3 Seiten verwendet werden \rightarrow Ergebnis ist 0, 1 oder 2
- Bei Überlauf einen Überlaufbehälter einfügen mit Verweisen auf diesen

7.5 Erweiterbares Hashing

- Hashfunktion verweist auf deutlich größeren Bereich und verweist nicht unbedingt auf einen tatsächlich vorhanden Behälter
- Nur ein Bruchteil des binären Funktionswertes wird als Index verwendet, z. B. die ersten 2 Bit, somit Abbildung auf 4 Seiten möglich
- $h(x) = dp$, wobei d die globale Tiefe ist in 2^d gemessen, p ist der unbenutzte Teil, falls durch Überlauf neue Bereiche zugänglich gemacht werden müssen, dann z. B. Erweiterung zu 2^3 mit 8 Seiten

Kapitel 8

Anfragebearbeitung

8.1 Äquivalenzen in der relationalen Algebra

1. Join, Vereinigung, Schnitt und Kreuzprodukt sind kommutativ und assoziativ
2. Selektionen sind untereinander vertauschbar
3. \wedge in einer Selektionsbedingung können in mehrere Selektionen aufgebrochen bzw. nacheinander ausgeführte Selektionen können durch \wedge zusammengefügt werden.
4. Geschachtelte Projektionen können auf die äußere beschränkt werden.

Dabei gilt es zu beachten, dass die Projektion und alle vorherigen eine Submenge des Schemas sein müssen.

5. Eine Selektion kann an einer Projektion “vorbeigeschoben,” werden, falls die Projektion keine Attribute aus der Selektionsbedingung entfernt
6. Eine Selektion kann an einer Joinoperation (oder Kreuzprodukt) vorbeigeschoben werden, falls sie nur Attribute eines der beiden Join-Argumente verwendet.
- 7.
8. Selektion auf eine fertige Mengenoperation (Vereinigung, Schnitt und Differenz) = Selektion auf Teile und dann Mengenoperation
9. Bei Projektionen nur bei Vereinigung möglich
10. Selektion + Kreuzprodukt = Join mit Selektionsbedingung.
11. DeMorgans Gesetz:

$$\neg(p_1 \vee p_2) = \neg p_1 \wedge \neg p_2$$

$$\neg(p_1 \wedge p_2) = \neg p_1 \vee \neg p_2$$

8.2 Anwendung der Transformationsregeln

1. Aufbrechen von Selektionen
2. Verschieben der Selektionen soweit wie möglich nach unten im Operatorbaum
3. Zusammenfassen von Selektionen und Kreuzprodukten zu Joins

4. Bestimmung der Reihenfolge der Joins in der Form, dass möglichst kleine Zwischenergebnisse entstehen.
5. unter Umständen Einfügen von Projektionen
6. Verschieben der Projektionen soweit wie möglich nach unten im Operatorbaum

Kapitel 9

Transaktionsverwaltung

Grundlegende Anforderungen an Transaktionen:

1. Recovery, d.h. die Behebung von eingetretenen, oft unvermeidbaren Fehlersituationen
2. Synchronisation von mehreren gleichzeitig auf der Datenbank ablaufenden Transaktionen.

9.1 Operationen auf Transaktionsebene

begin of transaction (BOT) Mit diesem Befehl wird der Beginn einer Transaktion dargestellt.

commit Hierdurch wird die Beendigung der Transaktion eingeleitet. Alle Änderungen der Datenbasis werden durch diesen Befehl **festgeschrieben**.

abort Dieser Befehl führt zu einem Selbstabbruch der Transaktion. Das Datenbanksystem stellt sicher, dass der ursprüngliche Zustand wieder erreicht wird.

define savepoint An diesem Punkt in einer Transaktion wird ein Sicherungspunkt angelegt, an dem später wieder eingestiegen werden kann. Die Änderungen werden aber noch nicht in die Datenbank geschrieben.

backup transaction Dieser Befehl dient dazu, die noch aktive Transaktion auf den jüngsten savepoint zurück zu setzen und dort weiter zu arbeiten.

9.2 Eigenschaften von Transaktionen

Atomicity (Atomarität) Alles oder nichts Prinzip. Die Transaktion ist nicht weiter zerlegbar und wird komplett oder gar nicht ausgeführt.

Consistency Eine Transaktion einen Konsistenten Datenbasiszustand, ansonsten wird diese zurückgesetzt. Zwischenzustand darf inkonsistent sein, sonst nie.

Isolation Keine Beeinflussung von parallel Laufenden Transaktionen. Alle anderen parallel ausgeführten Transaktionen bzw. deren Effekte dürfen nicht sichtbar sein.

Durability (Dauerhaftigkeit) Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten. Nur eine weitere Transaktion kann diese rückgängig machen.

9.3 Transaktionsverwaltung in SQL

commit work Änderungen werden committed keine Rückmeldung

rollback work Alle Änderungen werden zurück gesetzt. Rückmeldung vorgeschrieben.

9.4 Zustandsübergänge einer Transaktion

potentiell Codiert und wartet auf die Ausführung

aktiv wird aktuell ausgeführt.

wartend Bei Überlast in wartenden Zustand verdrängt.

abgeschlossen Durch commit beendete aktive Transaktion.

persistent Die Änderungen wurden in die Datenbank geschrieben

gescheitert Abbruch führt in diesen Zustand.

wiederholbar Falls Transaktion wiederholbar ist, versuche dies erneut nach zurücksetzen der Datenbasis.

aufgegeben eine gescheiterte Transaktion ist hoffnungslos und alle Änderungen werden zurückgesetzt. Danach wird in den Zustand aufgegeben gewechselt.