

Inhalt

Grundlagen Datenbanken	1
Intro	1
Datenbanken-Basis-Funktionalitäten nach Codd	1
DB, DBS, DBMS	1
5-Schichten-Architektur.....	1
ANSI/SPARC-Architektur	2
Datenbanken-Historie.....	3
Relationale Algebra.....	3
Relationenschema	3
Selektion	4
Projektion	4
Verbund	4
Rename	4
Beispiele	4
ER-Modell zu Relationalem Modell.....	6
Entwurfsprozess	6
ER-Modell	7
Theoretische Grundlagen der Relationalen Datenbanken	9
Funktionale Abhängigkeit.....	9
Armstrong-/ RAP-Axiome	10
Hüllen	10
Die Normalformen.....	11
Anomalien	11
1NF	11
2NF	11
3NF	11
Beispiel	12
Minimalitätseigenschaft	14
Transformationseigenschaften	14
Abhängigkeitstreue.....	14
Verbundtreue	15
Entwurfsverfahren.....	16
Dekomposition	16
Synthese	17
Boyce-Codd-Normalform (BCNF)	18

Multivalued Dependencies und 4NF	18
Transaktionen	19
ACID	19
Lost Update	19
Dirty read	19
Phantom / Non-repeatable read	19
Serialisierbarkeit	20
Sperrverfahren	20
Synchronisationsverfahren	21
SQL-Isolationsstufen	21
Interne Strukturen	22
Dateiorganisationsformen	22
ISAM	23
B-Baum	23
Hash-Verfahren	25
Sichten	25
Integrität	29
NoSQL / NewSQL	31
Grundlegende Konzepte und Motivation	31
Entstehung und Hintergrund	31
Das CAP-Theorem	31
NoSQL Datenmodelle	31
Key-Value Stores	31
Document Stores	31
Wide Column Stores	32
Graph Databases	32
Technische Konzepte	32
Konsistenzmodelle	32
Eventual Consistency	32
BASE vs ACID	33
Verteilungskonzepte	33
Sharding	33
Replikation	33
Architekturkonzepte	33
Verteilte Architekturen	33
Skalierung	33

Praktische Aspekte	33
Anwendungsfälle	33
Auswahl einer NoSQL-Datenbank	34
Vor- und Nachteile von NoSQL	34
Konsistenzsicherung und Datenzugriff	34
MVCC (Multiversion Concurrency Control)	34
Klassisches Locking (Sperrverfahren)	35
Konsistentes Hashing	35
Entity-Relationship-Modell	36
ANSI/SPARC	36
Datenmodellierung	36
Entitäten	37
Relationship	39
Beispiel für fertiges Modell	43
Datenbanksprache SQL	45
Datendefinition (Schema, Tabellen, Datentypen, Views, Indexe)	45
Datenabfrage (Suchen, Aggregieren, Schachtelungen, Verbund)	48
Datenmanipulation (Einfügen, ändern, löschen)	53
Anhang: SQL-Syntax	55

Grundlagen Datenbanken

Intro

Datenbanken-Basis-Funktionalitäten nach Codd

- Integration
- Operationen
- Katalog
- Benutzersichten
- Konsistenzüberwachung
- Zugriffskontrolle
- Transaktionen
- Synchronisation
- Datensicherung

DB, DBS, DBMS

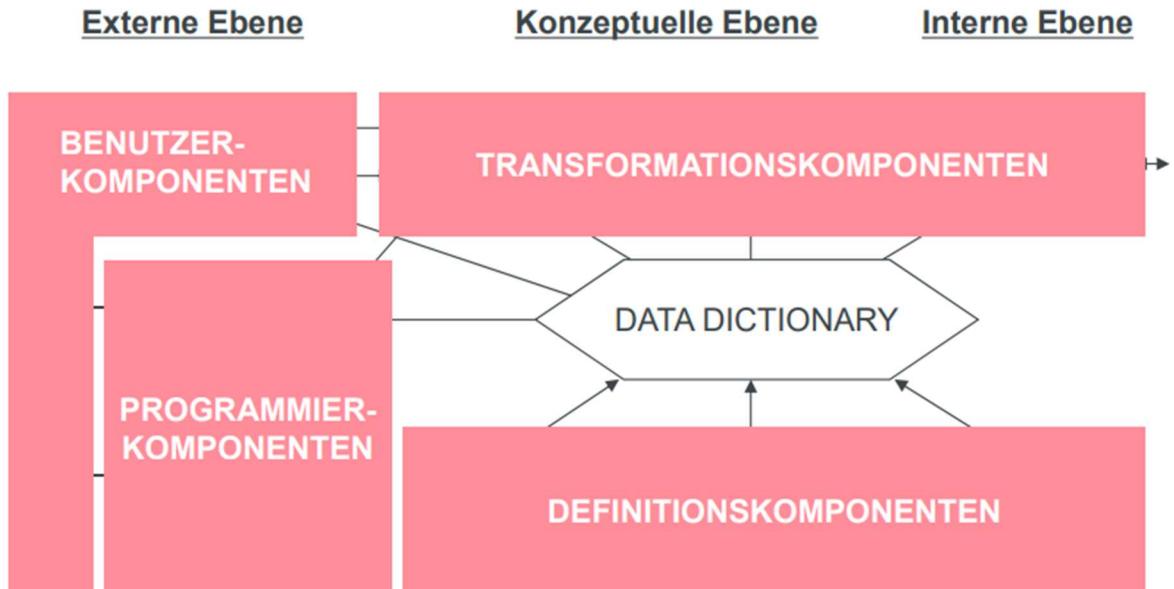
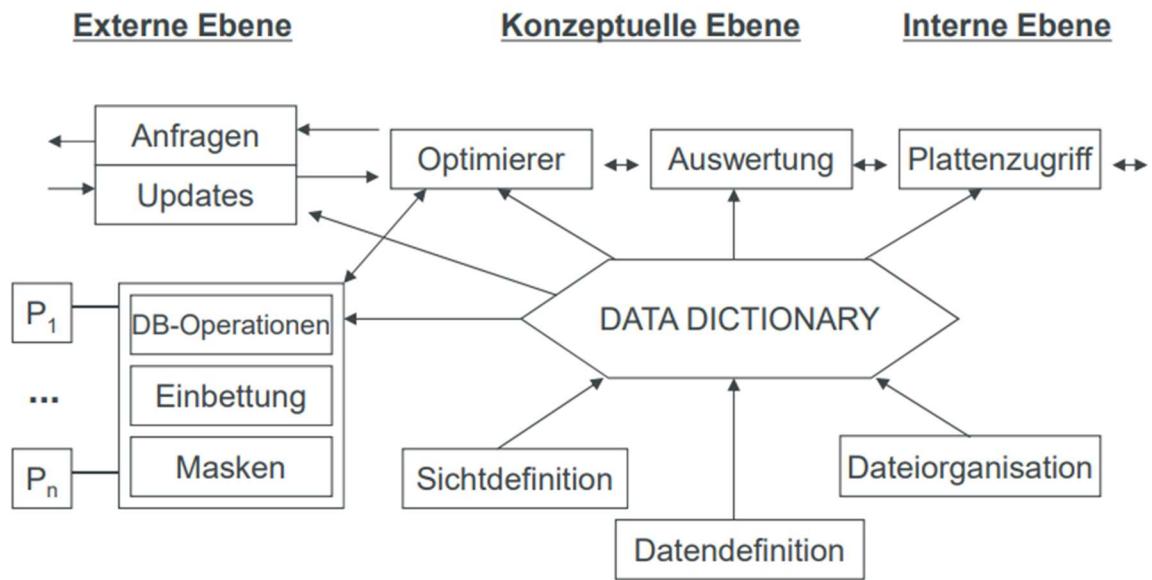
- DB: Strukturierter Datenbestand
- DBMS: Software zur Verwaltung einer DB
- DBS: DB + DBMS

5-Schichten-Architektur

5-Schichten-Architektur



ANSI/SPARC-Architektur

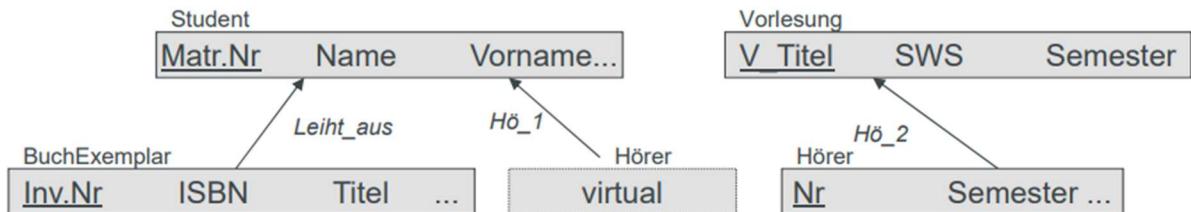


Das Hierarchische Modell

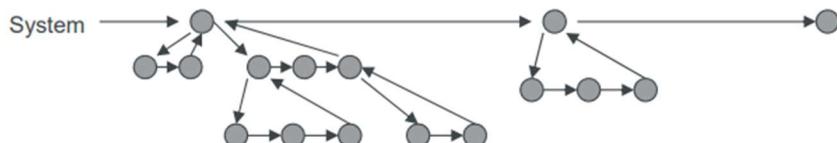
Hierarchie = Netzwerkschema, das ein *Wald* ist (Menge von Bäumen)

Kann man durch eine Hierarchie allgemeine Beziehungen darstellen?

Lösungshilfsmittel: virtual records



Speicherstruktur:

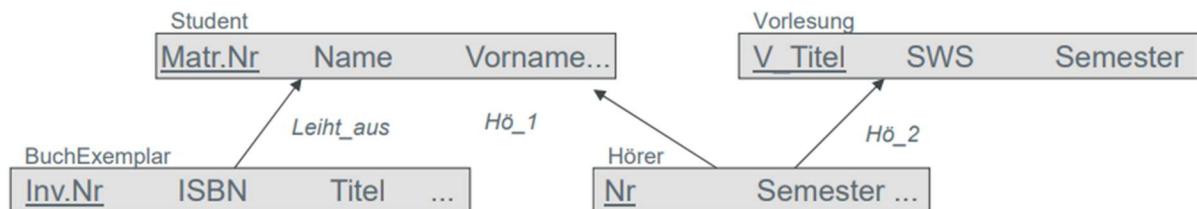


Das Netzwerkmodell

Entspricht „in etwa“ dem ER-Modell...

- nur
 - ... Relationships sind ausschließlich zweistellig und funktional
 - ... Relationships haben keine Attribute

ER-Modell	Relationenmodell	Netzwerkmodell
Entity	Tupel	logical record
Entity-Typ	Relationenschema	Record-Typ
Attribut	Attribut	Feld
binärer 1-n	Relation	Link oder Set-Typ
Beziehungstyp		



Relationale Algebra

Relationenschema

BUCH	INV_NR TITEL	ISBN AUTOR
------	----------------	--------------

Selektion

→ **Selektion** SEL [AUTOR = ,D.Reichardt'] (BUCH)

0001	Kommunikation-im-Netz	3-123	D.Reichardt
------	-----------------------	-------	-------------

Wählt alle Tupel der Relation BUCH, welche die Bedingung AUTOR = ,D.Reichardt' erfüllen aus.

Auswahl der Zeilen

Projektion

→ **Projektion** PROJ [AUTOR] (BUCH) →

Wählt aus der Relation (Tabelle) BUCH nur die Spalte AUTOR aus.

Auswahl der Spalten

D.Reichardt
S.Davis
D.Flanagan
...
A.Heuer

Verbund

Verbund (*Join / Natural Join*) Kombination aller Tupel in denen gleichnamige Attribute gleiche Werte haben.

R JOIN[A₁,...A_n] S

Wählt alle Tupel der Relation R und S aus, die in den Attributen A₁, ..., A_n übereinstimmen und fügt diese in jeweils einer neuen Relationenzeile zusammen.

Rename

REN [B:= A] (R) - *Rename* = Umbenennen von Attributnamen

X - Kreuzprodukt (Spezialfall des Verbunds R JOIN[] S)

Mengenoperationen:

U ∩ \

Beispiele

SEL [VON = Stuttgart AND NACH = Palma AND DATUM = <heute>] (FLUG)

PROJ [FLUGNR, ABFLUG]
 (SEL [VON = Stuttgart AND NACH = Palma AND DATUM = <heute>] (FLUG))

SEL [VON = Stuttgart AND NACH = Palma AND DATUM = <heute> AND LAND = D]
 (FLUG JOIN FLUGGESELLSCHAFT)

→ Wie stellt man einen "Allquantor" dar?

→ „DIVISION“

R sei dabei die Menge der Attribute, die in A enthalten sind
 ohne die in B enthaltenen Attribute.

$$A \div B = \text{PROJ}_R(A) - \text{PROJ}_R((\text{PROJ}_R A) \text{ JOIN } B) - A$$

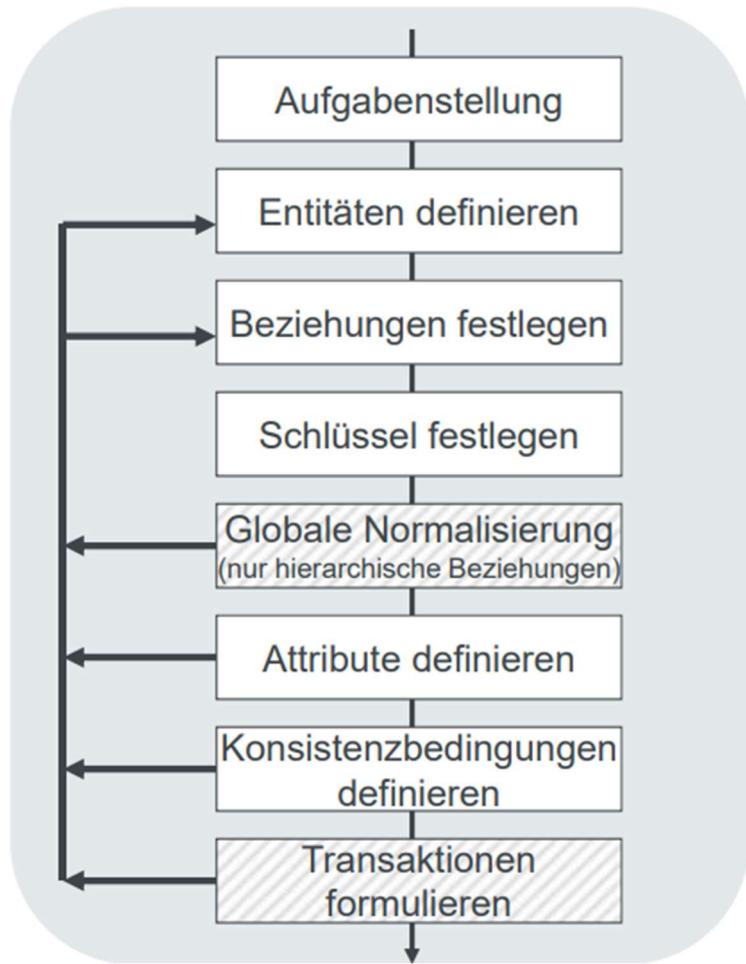
1	PILOT	FLUGZEUG	r2	FLUGZEUG	„Piloten, die alle Flugzeuge aus r2 fliegen können“
	Snoopy	707		707	
	Snoopy	727		727	
	Snoopy	747		747	
	Meyer	707			
	Meyer	727			
	Müller	707	r3	FLUGZEUG	„Piloten, die alle Flugzeuge aus r3 fliegen können“
	Müller	727		707	
	Müller	747			
	Müller	777			
	Lüdenscheid	727			

$$r1 \div r2$$

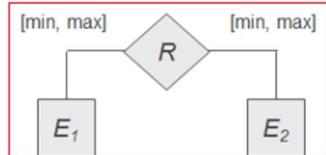
$$r1 \div r3$$

ER-Modell zu Relationalem Modell

Entwurfsprozess

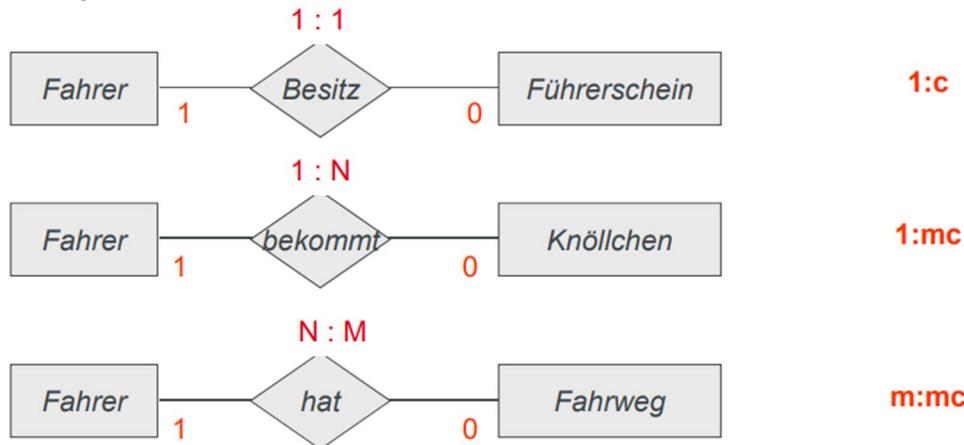


ER-Modell



Abk.	Assoziationstyp	Anzahl Tupel in der referenzierten Tabelle
1	einfach	genau ein Tupel
c	konditionell	kein oder ein Tupel
m	multipel	mindestens ein Tupel
mc	multipel konditionell	beliebig viele Tupel

Beispiele

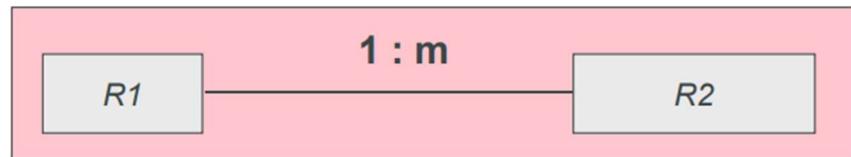


Beziehungstypen

R1 \ R2	1	c	m	mc	
1	1-1	c-1	m-1	mc-1	Hierarchische Beziehungen
c	1-c	c-c	m-c	mc-c	Konditionelle Beziehungen
m	1-m	c-m	m-m	mc-m	
mc	1-mc	c-mc	m-mc	mc-mc	Netzwerkförmige Beziehungen

Es können im Relationenmodell **nur hierarchische Beziehungen** direkt aus dem ER-Modell abgebildet werden. Andere sind nicht erlaubt.

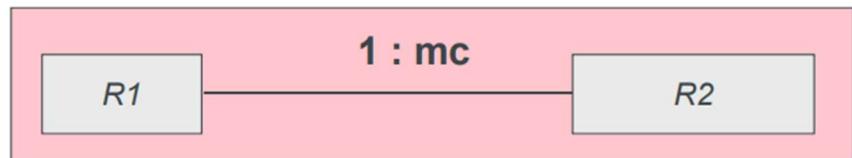
1-m Beziehungen



In der Tabelle R2 wird ein Fremdschlüssel eingefügt, der auf R1 referenziert.

Beispiel: Person (R1) kann ein oder mehrere Autos (R2) besitzen. Die Tabelle „Autos“ ist daher mindestens so gross wie „Person“.

1-mc Beziehungen

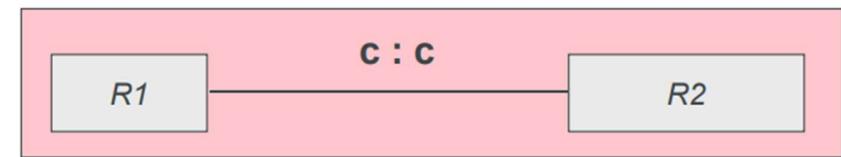


c-c Beziehungen

Analog für:

c-m

c-mc



Mögliche Lösung:

In R1 und R2 werden Fremdschlüssel eingeführt, die auf die jeweils andere Relation verweisen.

→ Nullwerte wären ermöglicht!

Besser:

Transformation in eine Darstellung mit einer *Beziehungsrelation*:

R1 #key1|A1|A2 ...

R2 #key2|A1|A2 ...

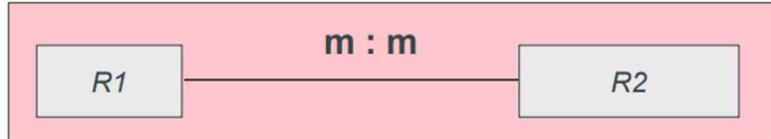
BR #key1|#key2

m-m Beziehungen

Analog für:

mc-m

mc-mc



Beispiel:

Eine Person kann eines oder mehrere Autos besitzen und ein Auto kann einen oder mehrere Besitzer haben.

Personen

PNr	Name
1	Peter
2	Karl
3	Gabi
4	Monika

Autos

ANr	Name	PNr
1	Manta	1
2	C-Klasse	3
3	Uno	2
4	Golf	1
4	Golf	3
5	XJ12	4

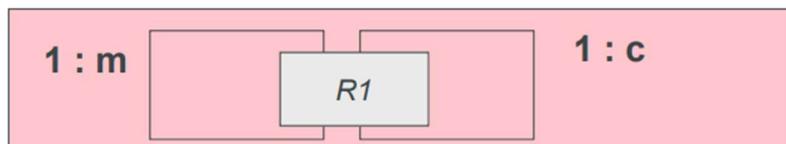
Redundanzen!

Kein Schlüssel mehr!

Besser: Umformung mit Beziehungsrelation wie c-c

...

Rekursive
Beziehungen



Beispiel:

Sei R1 die Gruppe der Musiker und die 1:c Beziehung stellt dar, dass ein Musiker Dirigent sein kann. Die 1:m Beziehung stellt dar, dass ein Dirigent mehrere Musiker dirigieren kann.

Musiker

MNr	DNr	Name
1	2	Schmid
2	3	Karajan
3	2	Bernstein
4	2	Müller
5	3	Meier

Korrekt?

Probleme:

Schlüsselbenennung!

Wie findet man heraus ob ein Musiker Dirigent ist?

→ Suche

Theoretische Grundlagen der Relationalen Datenbanken

Funktionale Abhängigkeit

Funktionale Abhängigkeit:
Ein Attribut (bzw. eine Attributkombination) B ist von einem Attribut (oder Attributkombination) A funktional abhängig, wenn zu einem bestimmten Attributwert von A genau ein Attributwert von B gehört.

Voll funktionale Abhängigkeit:

B ist von A abhängig, nicht jedoch schon von einem Teil der Attributkombination A.

Transitive Abhängigkeit:

C ist von B funktional abhängig, B wiederum von A, dann ist C von A transitiv abhängig.

Armstrong-/ RAP-Axiome

Armstrong Axiome:

Ist eine Menge von funktionalen Abhängigkeiten gegeben, so lassen sich weitere daraus ableiten – wie findet man diese?

Axiom 1:	$B \subseteq A \Rightarrow A \rightarrow B$	Reflexivität Triviale FDs
Axiom 2:	$A \rightarrow B \Rightarrow AC \rightarrow BC$	Augmentation / Erweiterung
Axiom 3:	$A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$	Transitivität

RAP-Axiome (nach Saake/Heuer)

Reflexivität:	$A \rightarrow A$
Akkumulation:	$A \rightarrow BC, C \rightarrow DE \Rightarrow A \rightarrow BCE$
Projektivität:	$A \rightarrow BC \Rightarrow A \rightarrow B$

Hüllen

$$FD = \{ A \rightarrow B, B \rightarrow C, AB \rightarrow DC, C \rightarrow E \}$$

Was ist die **Attributhülle** von A?

D.h. Welche Attributwerte sind eindeutig bestimmt, wenn der Wert von A bekannt ist?

```
AH(A, FD) = {A}                                // Startwert
              {A} ∪ {B}                            // B aufgrund von A → B
              {A} ∪ {B} ∪ {C, D}                      // da AB → CD
              {A} ∪ {B} ∪ {C, D} ∪ {E}                // da C → E
= {A, B, C, D, E}
```

Anwendung:

Prüfen ob A Schlüssel von R(A,B,C,D,E) sein kann.

Die Normalformen

Anomalien

Anomalien entstehen aus der Struktur der Relationen

Prüfungsgeschehen		Prüfer	Student MATNR	Name	Geb	Adr	Fachbereich	Dekan	Note
PNR	Fach								
3	SP	Hofmann	9516570	Huber	010177	Bachstr. 1	Informatik	Heinze	1
			9564342	Meier	030276	Uferweg 2	Informatik	Heinze	2
4	BS	Kallte	9516570	Huber	010177	Bachstr. 1	Informatik	Heinze	2
			9564342	Meier	030276	Uferweg 2	Informatik	Heinze	2
5	RO	Paulus	9516570	Huber	010177	Bachstr. 1	Informatik	Heinze	3
			9111232	Bauer	120472	Obertorstr, 2	BWL	Wirre	2

Einfüge-Anomalie: (Definiertheit des Vaters / Übergeordnetes Segment)

Bsp: Informationen über einen Studenten, der noch an keiner Prüfung teilgenommen hat, können nicht gespeichert werden.

Löschen-Anomalie: Wird Student Bauer gelöscht, verschwindet auch die Information, dass Wirre der Dekan des Bereichs BWL ist.

Änderungs-Anomalie: zieht Student Meier um, so muss sein Adresswechsel in mehreren Tupeln nachgezogen werden.

1NF

Ein Relationenschema heißt normalisiert, oder ist in 1.Normalform, wenn seine Wertebereiche elementar sind, d.h. keine Relationen als Elemente besitzen.

2NF

Ein Relationenschema ist in 2.Normalform, wenn jedes Nicht-Primärattribut voll funktional von jedem Schlüsselkandidaten der Relation abhängt.

3NF

Ein Relationenschema ist in 3.Normalform, wenn es sich in 2. Normalform befindet und jedes Nicht-Primärattribut nicht transitiv abhängig ist von jedem Schlüsselkandidaten des Schemas.

Beispiel

Beispiel

CD_Lieder		
CD_ID	Album	Titelliste
4811	Anastacia - Not That Kind	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd - Wish You Were Here	{1. Shine On You Crazy Diamond}

Zusammengesetztes Attribut
bestehend aus Interpret und Albumtitel

Mehrwertiges Attribut „Titelliste“

aufspalten

aufteilen

2. Normalisierte Daten : 1.Normalform ist hergestellt

CD_Lieder				
CD_ID	Albumtitel	Interpret	Track	Titel
4811	Not That Kind	Anastacia	1	Not That Kind
4811		Anastacia	2	I'm Outta Love
4811		Anastacia	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1	Shine On You Crazy Diamond

Primärschlüssel

2.Normalform?

→ Nein, denn:
Albumtitel und Interpret hängen
nur von CD_ID ab !

3. Herstellen der 2.Normalform

Aufteilen in zwei Relationen, so dass eine Relation diejenigen Attribute enthält, die nur von **einem** der Attribute abhängen (CD_ID) und die zweite diejenigen, die von **beiden** (CD_ID, Track) abhängen.

CD

CD_ID	Album	Interpret
4811	Not That Kind	Anastacia
4712	Wish You Were Here	Pink Floyd

Lieder

CD_ID	Track	Titel
4811	1	Not That Kind
4811	2	I'm Outta Love
4811	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond

3.Normalform?

→ Ja, keine transitiven Abhängigkeiten vorhanden

Fremdschlüssel

4. Kleine, gemeine Erweiterung des Schemas ...

CD_ID	Album	Interpret
4811	Not That Kind	A
4712	Wish You Were Here	P

CD

CD_ID	Album	Interpret	Gründungsjahr
4811	Not That Kind	Anastacia	1999
4713	Freak of Nature	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965

3.Normalform?

→ Nein, transitive Abhängigkeit
 $CD_ID \rightarrow Interpret \rightarrow Gründungsjahr$

Gründungsjahr ist von Interpret abhängig

5. Herstellen der 3.Normalform

CD		CD_Künstler		Künstler		
CD_ID	Album	CD_ID	I_ID	I_ID	Interpret	Gründungsjahr
4811	Not That Kind	4811	2423	2423	Anastacia	1999
4713	Freak of Nature	4713	2423	3433	Pink Floyd	1965
4712	Wish You Were Here	4712	3433			

→ Wikipedia – Lösung mit Voraussetzung: evtl. mehrere Interpreten für gleiche CD ...

Lösung 2:			Künstler		
CD_ID	Album	I_ID	I_ID	Interpret	Gründungsjahr
4811	Not That Kind	2423	2423	Anastacia	1999
4713	Freak of Nature	2423	3433	Pink Floyd	1965
4712	Wish You Were Here	3433			

Minimalitätseigenschaft

Beispiel: $R = (\{A,B,C\}, \{A \rightarrow B, B \rightarrow C\})$

Daraus lassen sich folgende Datenbankschemata bilden:

$$D_1 = \{(\{A,B\}, \{A \rightarrow AB\}), (\{C,B\}, \{B \rightarrow BC\})\}$$

... sowie:

$$D_2 = \{(\{A,B\}, \{A \rightarrow AB\}), (\{C,B\}, \{B \rightarrow BC\}), (\{C,A\}, \{A \rightarrow AC\})\}$$

Beide Datenbankschemata sind in 3NF D_1 ist jedoch zudem **minimal**.

Transformationseigenschaften

Abhängigkeitstreue

(a) Abhängigkeitstreue

Ist die Menge der funktionalen Abhängigkeiten einer Ausgangsrelation äquivalent mit den Schlüsselabhängigkeiten des abgeleiteten Datenbankschemas, so ist **Abhängigkeitstreue** gegeben.

Verbundtreue

Verbundtreue

Verbundtreue ist eine Qualitätseigenschaft einer Relationalen Datenbank. Diese ist gegeben, wenn die Ursprungsrelation mit Hilfe von *Natural Joins* wieder herstellbar ist.

Beispiel:

Nehmen wir an, wir haben eine Attributmenge A,B,C und funktionale Abhängigkeiten A->B und C->B. Aus diesen setzt sich nun die *Ursprungsrelation R* zusammen. Nun wird diese Relation zerlegt in die beiden neuen Relationen R1 = (A,B) und R2 = (B,C). Diesen können die funktionalen Abhängigkeiten zugeordnet werden. Somit ist die Aufteilung *abhängigkeitstreu*. Es ergeben sich die erweiterten Relationenschemata R1 = (AB, {A->B}) und R2 = (BC, {C->B}).

In diesem Fall ist eine Verbundtreue nicht gegeben, da die Ursprungsrelation durch einen Natural Join nicht wieder hergestellt werden kann. Grund dafür ist, dass das Attribut B, über welches der Join läuft, kein Schlüsselattribut ist.

Um dies zu veranschaulichen, können wir die Relationen mit fiktiven Daten füllen, welche die Integritätsbedingungen (Abhängigkeiten) erfüllen (bitte prüfen Sie dies nach!).

R	A	B	C
	1	2	1
	2	2	2
	3	4	3
	4	4	4
	3	4	5

Nehmen wir an, R sei mit den nebenstehenden 5 Tupeln gefüllt. Wie man erkennen kann, erfüllt die Relation die Bedingung A->B und C->B. Diese Relation wird nun durch die Aufspaltung des Relationenschemas auf zwei verschiedene Relationen verteilt.

Das Ergebnis sieht wie folgt aus:

R1	A	B	R2	B	C
	1	2		2	1
	2	2		2	2
	3	4		4	3
	4	4		4	4
				4	5

Wenn nun diese beiden Relationen per Natural Join verbunden werden, so werden jeweils zeilenweise die Tupel zusammengefasst, die den gleichen Wert in B haben. Dadurch wird das Tupel (1,2) aus Relation R1 mit den Tupeln (2,1) und (2,2) verbunden. Es entstehen entsprechend zwei Tupel in der Ergebnisrelation R' mit (1,2,1) und (1,2,2). Führt man den Join nun Zeile für Zeile durch, so entsteht eine neue Relation mit 10 Einträgen. Dies sind deutlich mehr als in der Ursprungsrelation. Somit ist die Verbundtreue nicht gegeben.

R1 JOIN R2	A	B	C
(R')	1	2	1
	1	2	2
	2	2	1
	2	2	2
	3	4	3
	3	4	4
	3	4	5
	4	4	3
	4	4	4
	4	4	5

Dekompositionsverfahren

- (1) Gehe von einer **Gesamtattributmenge** U und einer Menge funktionaler Abhangigkeiten F aus
- (2) Finde alle **Schlessel** K_i fur diese Attributmenge U
- (3) Finde **transitive Abhangigkeiten** in diesem initialen Relationenschema, die von einem K_i ausgehen.
- (4) Fur jede transitive Abhangigkeit K→X X→Y (wenn nicht X→K gilt) **teile** das erweiterte Relationenschema auf in:
 - a) R₁ = R \ Y mit den bestehenden Schlesseln
 - b) R₂ = XY mit Schlusseleigenschaft X→Y
- (5) Wiederhole (3) und (4) bis keine Aufteilung mehr moglich ist.

~~abhangigkeitstreu 3NF verbundtreu minimal~~

DER SYNTHESEALGORITHMUS – Grundlagen in Kurzfassung

Der so genannte Synthesealgorithmus erzeugt ein Relationenschema, welches eine verlustlose und abhängigkeitsbewahrende Zerlegung eines gegebenen Relationenschemas ist, in welchem sich alle Relationen in 3. Normalform befinden. Dies geschieht in vier Schritten:

1. Zu den gegebenen funktionalen Abhängigkeiten wird die kanonische Überdeckung F' bestimmt (die kleinstmögliche Menge an funktionalen Abhängigkeiten die zu F äquivalent ist).
2. Für jede in F' enthaltene Abhängigkeit wird ein Relationenschema erzeugt, welches sowohl alle darin enthaltenen Attribute als auch alle unter diesen Attributen geltenden Abhängigkeiten umfasst.
3. Enthält keines der erzeugten Schemata einen Gesamtrelations-Schlüsselkandidaten, so muss eine weitere Relation erzeugt werden, welche genau aus den zu einem solchen Kandidaten gehörenden Attributen besteht.
4. Relationenschemata, welche in anderen enthalten sind, werden gelöscht (die darin enthaltene funktionale Abhängigkeit wird der enthaltenden Relation zugefügt).

Die Vorgehensweise wird durch die Bestimmung einer kanonischen Überdeckung definiert. Um diese zu erhalten ist zunächst eine Linksreduktion durchzuführen.

Linksreduktion:

Für jedes $\alpha \rightarrow \beta \in F$ wird geprüft, ob ein Element x der Attributmenge α zu streichen ist, ohne dass dies die Hülle von F verändert. Konkret oder in Formeln ausgedrückt:

Gilt $\beta \subseteq AH(F, \alpha \setminus \{x\})$, dann ist dies gegeben und die funktionale Abhängigkeit kann reduziert werden.

Nach der Linksreduktion folgt eine Rechtsreduktion der funktionalen Abhängigkeiten.

Rechtsreduktion:

Es wird für jedes $\alpha \rightarrow \beta \in F$ geprüft, ob ein Attribut x auf der rechten Seite einer Regel ($x \in \beta$) überflüssig ist. Geprüft wird das dadurch, dass man es aus der Regel entfernt und schaut ob es dennoch in der Attributhülle der linken Seite der Regel ist. Konkret oder in Formeln ausgedrückt:

Gilt $x \in AH((F \setminus \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta \setminus \{x\})\}, \alpha)$, dann ist dies gegeben und die funktionale Abhängigkeit kann reduziert werden.

Abschließend werden funktionale Abhängigkeiten mit gleicher linker Seite zusammengefasst und solche, bei denen eine Seite leer ist, werden gelöscht.

Boyce-Codd-Normalform (BCNF)

1

Ein Relationenschema heißt normalisiert, oder ist in 1.Normalform, wenn seine Wertebereiche elementar sind, d.h. keine Relationen als Elemente besitzen.

2

Ein Relationenschema ist in 2.Normalform, wenn jedes Nicht-Primärattribut voll funktional von jedem Schlüsselkandidaten der Relation abhängt.

3

Ein Relationenschema ist in 3.Normalform, wenn es sich in 2. Normalform befindet und jedes Nicht-Primärattribut nicht transitiv abhängig ist von jedem Schlüsselkandidaten des Schemas.

**BC
NF**

Ein Relationenschema ist in Boyce Codd Normalform, wenn jeder Determinant ein Schlüsselkandidat des Schemas ist.

4

Eine Relation R ist in 4. Normalform wenn sie in BCNF-Normalform ist und nicht zwei oder mehr unzusammenhängende mehrwertige Abhängigkeiten enthält.

Multivalued Dependencies und 4NF

Multivalued Dependency (MVD)

In einer Relation wird einem Attributwert **eine Menge von Werten** eines weiteren Attributs zugeordnet.

Im Gegensatz zur **funktionalen Abhängigkeit** bei welcher **genau ein Wert** zugeordnet wird.

Beispiel:

Einem durch eine **ISBN** identifizierten Buch sind mehrere **Autoren** zugeordnet.

ISBN → → Autor

Gibt es nun mehrwertige Abhängigkeiten, so ist eine weitere Normalform zu betrachten ...

Eine Relation R ist in 4. Normalform wenn sie in BCNF-Normalform ist und nicht zwei oder mehr unzusammenhängende mehrwertige Abhängigkeiten enthält.

Transaktionen

ACID

Atomicity

Eine Transaktion ist die kleinste, nicht mehr weiter zerlegbare Einheit.

Consistency

Nach Durchführung der Transaktion ist die Datenbank wieder in einem konsistenten Zustand, ansonsten wird sie zurückgesetzt.

Isolation

Nebenläufige Transaktionen beeinflussen sich nicht gegenseitig. Jede wird „logisch“ ausgeführt als gäbe es keine andere.

Durability

Dauerhaftigkeit (Persistenz) durchgeföhrter Aktionen muss gewährleistet sein. Alle späteren Fehler betreffen die hier durchgeföhrten Änderungen nicht.

DBS übernimmt die Transaktionsverwaltung und gewährleistet ACID.

Aber: „logische“ Konsistenz kann nicht automatisch geprüft werden
- explizite Festlegung durch **Integritätsbedingungen**

Lost Update

Tritt auf, wenn zwei Transaktionen parallel auf denselben Daten arbeiten. Beispiel: Zwei Banktransaktionen greifen gleichzeitig auf einen Kontostand zu:

- T1 liest Kontostand
- T2 liest Kontostand
- T1 ändert und schreibt neuen Wert
- T2 ändert und schreibt neuen Wert

Dirty read

Eine Transaktion liest Daten, die von einer anderen noch nicht abgeschlossenen Transaktion geändert wurden. Wenn die ändernde Transaktion zurückgerollt wird, hat die lesende Transaktion mit "schmutzigen" (ungültigen) Daten gearbeitet. Beispiel:

- T1 ändert Daten
- T2 liest die geänderten Daten
- T1 wird zurückgerollt
- T2 arbeitet mit ungültigen Daten weiter

Phantom / Non-repeatable read

Tritt auf wenn eine Transaktion zweimal die gleichen Daten liest, aber unterschiedliche Ergebnisse erhält. In der Zwischenzeit hat eine andere Transaktion die Daten geändert oder neue hinzugefügt. Beispiel:

- T1 liest Datensätze
- T2 fügt neue Datensätze hinzu oder ändert bestehende
- T1 liest nochmal und erhält andere Ergebnisse

Serialisierbarkeit

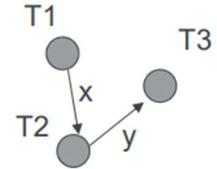
Wie kann man diese Anomalien ausschließen?

Serialisierbarkeit Ist die parallele Ausführung der Transaktionen äquivalent zu einer seriellen Ausführungsfolge, dann ist dies serialisierbar.

Schedule überführen in seriellen Schedule

Konflikte RW-Abhängigkeit
WW-Abhängigkeit
WR-Abhängigkeit

} Abhängigkeitsgraph



Ein Schedule ist **serialisierbar** gdw. der Abhängigkeitsgraph zyklusfrei ist

→ Implementierung / Nicht praktikabel (Nachträglichkeit!)

Sperrverfahren

Two-Phase-Locking



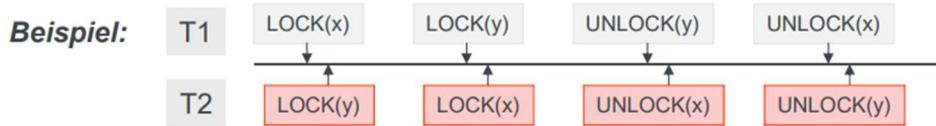
Jedes referenzierte Objekt wird zu Anfang geblockt
Fremde Sperren werden beachtet
Keine bereits besessene Sperre wird angefordert
Sperren werden zweiphasig angefordert und freigegeben
Alle Sperren sind spätestens bei Transaktionsende zurückzugeben

RX-Sperrverfahren

Zwei Sperrmodi: Read Lock und Exclusive Lock

Synchronisationsverfahren

Verklemmungen



Deadlocks sind *inhärentes* Problem aller sperrbasierten Synchronisationsverfahren

Umgang mit Verklemmungen (einige Möglichkeiten)

- A) **Timeout:** Abbruch wenn T länger als t_{max} auf eine Sperre wartet
- B) **Wartegraphen:** Kanten: T_1 wartet auf $T_2 \dots$ Ein Zyklus im Graphen entspricht einer Verklemmung - Löschen einer Transaktion auf dem Graphen!
- C) **Preclaiming:** Alle Sperren zu Beginn anfordern!
Problem: Obermengen/Grosse Sperrbereiche
- D) **Wait Depth Limit:** z.B. Warten nur auf laufende Transaktionen
- E) **Zeitmarken:** Wait/Die- und Wound-Verfahren

SQL-Isolationsstufen

Die Isolationsstufen bieten folgende Schutzmechanismen:

READ COMMITTED:

- Verhindert Dirty Reads
- Sperrt geänderte Daten bis zum Commit
- Standard in den meisten Datenbanken

REPEATABLE READ:

- Verhindert zusätzlich non-repeatable reads
- Sperrt gelesene Daten für die gesamte Transaktion
- Phantome sind aber noch möglich

SERIALIZABLE:

- Höchste Isolationsstufe
- Verhindert alle Anomalien
- Transaktionen laufen praktisch sequentiell ab
- Hat den höchsten Performance-Overhead

Die Wahl der Isolationsstufe ist also immer ein Trade-off zwischen Konsistenz und Performance.

Interne Strukturen

Stell dir vor, du hast eine Bibliothek mit Büchern und willst sie organisieren:

ISAM ist wie ein Bibliothekskatalog mit festen Regalen:

- Die Bücher stehen sortiert in den Regalen
- Du hast eine Liste (Index), die dir sagt "Fantasy-Bücher stehen in Regal 1-3"
- Problem: Wenn ein neues Buch kommt und das Regal ist voll, musst du alle Bücher umräumen

B-Baum ist wie ein flexibles Regalsystem:

- Du hast mehrere Etagen von Wegweisern:
 - Oberste Etage: "A-M links, N-Z rechts"
 - Mittlere Etage: genauere Unterteilung
 - Unterste Etage: die eigentlichen Bücher
- Wenn ein neues Buch kommt:
 - Der B-Baum "macht automatisch Platz"
 - Er verteilt die Bücher so um, dass wieder alles schön sortiert ist
 - Dabei bleiben alle Regale gleichmäßig gefüllt

Der große Vorteil des B-Baums:

- Du findest jedes Buch sehr schnell (wenige "Wegweiser" folgen)
- Neue Bücher einzufügen ist kein Problem
- Das System bleibt immer gut organisiert

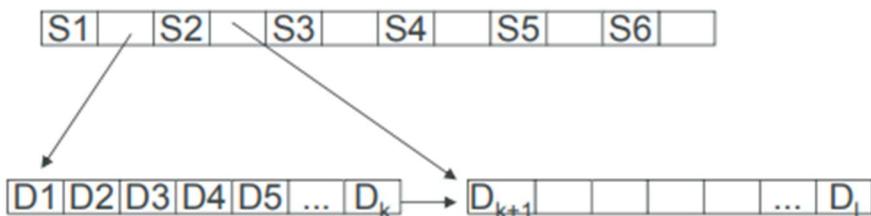
ISAM ist wie ein starres Bibliothekssystem, B-Baum wie ein flexibles, selbstorganisierendes System. Deshalb wird der B-Baum heute auch viel häufiger in Datenbanken eingesetzt.

Dateiorganisationsformen

- ⇒ *Heap - Organisation*
Tupel werden unsortiert in Dateien/Blöcken abgelegt
- ⇒ *Sequentielle Organisation*
Die Tupel werden in einer festgelegten Reihenfolge sortiert gespeichert
- ⇒ *Hash - Organisation*
Die Tupel werden „gestreut“ gespeichert
- ⇒ *Mehrdimensionale Organisation*
Ausschlaggebend ist nicht nur ein, sondern mehrere Schlüssel

ISAM

- ⇒ Geordnetes Speichern der Datensätze und Schlüssel - Indexseiten sequentiell im Speicher abgelegt



$$S_2 \geq D_1, \dots, D_k > S_1$$

- ⇒ **Suchen:** Binärsuche im Index - lineares Durchsuchen der Seite und ihrer Nachfolgeseiten bis zu einem „zu großen“ Schlüssel
- ⇒ **Einfügen:** Aufwändig ! Bei voller Datenseite Ausgleich mit Nachbarseite versuchen, ansonsten neue Seite anlegen und Index verschieben

B-Baum

B-Bäume

- ⇒ Im Hauptspeicher können Binäräbäume als Speicherstruktur verwendet werden - diese sind jedoch nicht effizient auf die Seitenstruktur des Hintergrundspeichers abbildbar.
- ⇒ B-Bäume (Bayer-Bäume) begrenzen die Zahl der Seitenzugriffe

Definition: B-Baum vom Grad k

Jeder Knoten hat mindestens $\lceil k/2 \rceil$ und höchstens k Schlüsseleinträge.

Der Wurzelknoten hat zwischen 1 und k Schlüsseleinträge.

Die Schlüsseleinträge in den Knoten sind sortiert.

Jeder Knoten außer den Blattknoten, der s Schlüsseleinträge besitzt $s+1$ Kindknoten

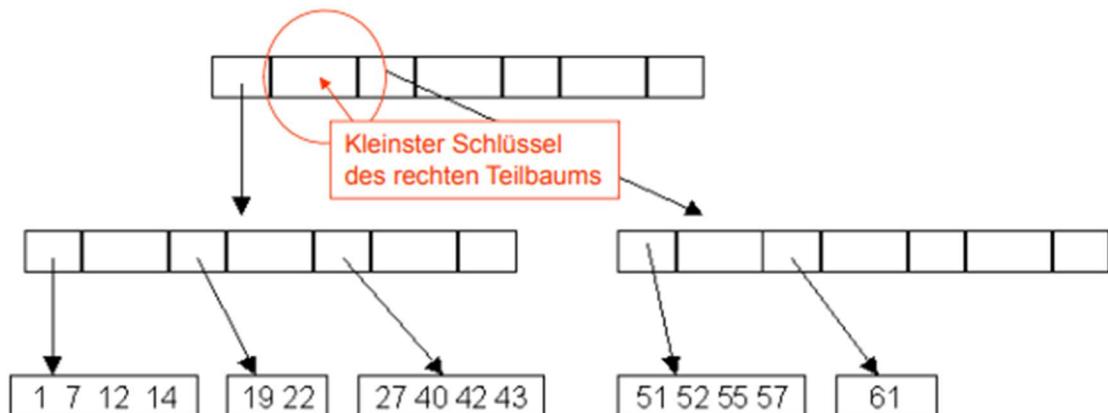
Seien $S_1 \dots S_n$ die Schlüsseleinträge eines Knotens und T_0, \dots, T_n die Verweise auf Kindknoten, dann gilt:

T_0 verweist auf Kindknoten, dessen Schlüsseleinträge kleiner als S_1 sind

T_i verweist auf Kindknoten, dessen Schlüsseleinträge $\geq S_i$ und kleiner als S_{i+1} sind
Blattknoten besitzen keine weiteren Verweise

B-Bäume

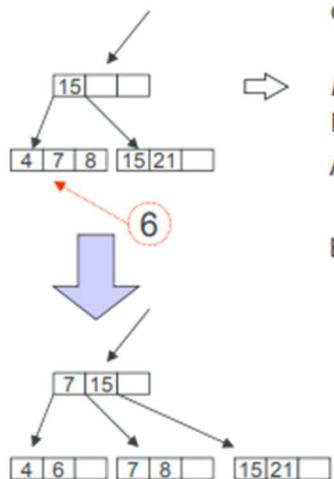
Beispielbaum der Ordnung 3



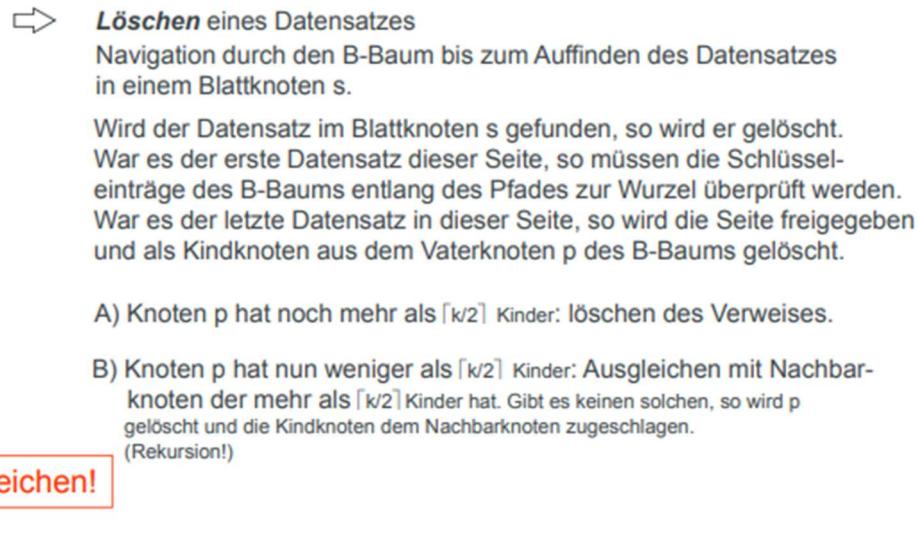
- ⇒ Realität: einige **Größenordnungen** mehr Einträge pro Knoten!
- ⇒ Bei Verwendung als Primärindex stehen im Blatt die gesamten Datensätze, im Sekundärindex nur die Schlüsse sowie Verweise auf den realen Datensatz.

B-Bäume

- ⇒ **Auffinden** eines Datensatzes
Navigation durch den B-Baum. Anzahl der Seitenaufrufe durch Höhe des Baumes begrenzt. (logarithmische Komplexität!)
- ⇒ **Eintragen** eines Datensatzes
Navigation durch den B-Baum bis zur entsprechenden Seite s.
 - A) Seite s ist noch nicht voll : eintragen des Datensatzes und anpassen der Schlüsseleinträge in den Knoten bis zur Wurzel.
 - B) Seite s ist voll: Es wird eine neue Seite r angefordert und die Elemente von s, sowie das neue Element gleichmäßig auf s und r verteilt. Anschließend wird die Seite r als neues Element in den Vaterknoten von s eingefügt.



B-Bäume

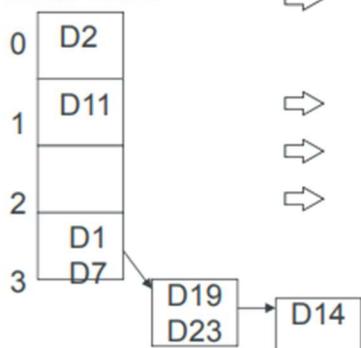


Hash-Verfahren

Hash-Verfahren

- Idee: direkte Berechnung der Speicheradresse aus dem Schlüssel eines Datensatzes.
- Gestreute Speicherungsstrukturen. Geeignet für gezielte Anfragen, weniger für Bereichsanfragen.
- Hashfunktion : Abbildung des Schlüssels auf einen Behälter (Bucket) (ggf. eine Speicherseite) $h : S \rightarrow B$ $B = [0..n]$
 In der Regel gilt $|S| >> |B|$!
- Realisierung : $h(x) = x \bmod n+1$
- **Offenes vs. Geschlossenes Hashing**
- Offenes Hashing : *Überlaufliste*
Achtung: Struktur kann entarten !

Hash-Table



Sichten

Definition und Motivation

Sicht = „virtuelle“ oder „berechnete“ Datenbank, keine „tatsächliche“ !

Wird aus einer tatsächlichen durch eine feste Berechnungsvorschrift abgeleitet

Zweck: Daten „ausblenden“

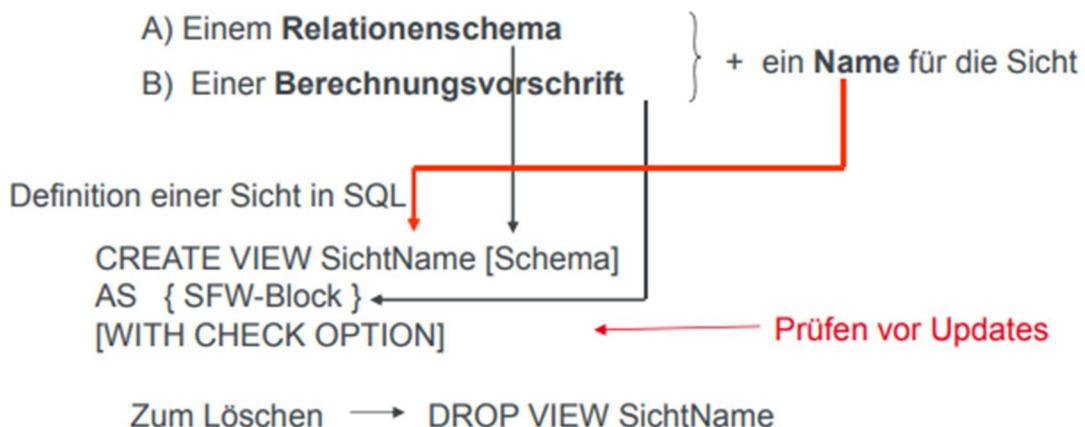
Daten in anderer Form darstellen

Sicht = gesamte „virtuelle“ Datenbank oder Element einer abgeleiteten Datenbank

Manche Nomenklaturen: nur eine virtuelle Relation

Sichten : Definition

Eine Sicht besteht aus:



Vereinfachung von Anfragen



Datenbankschemata können groß, komplex und für den Einzelanwender undurchsichtig werden.

→ *Benutzerbezogene Strukturierung*



Stabilität der Anwenderschnittstelle bei Datenbankstrukturänderungen

→ *Logische Datenunabhängigkeit*



Zugriffskontrolle

Probleme mit Sichten

Umsetzung der Sicht wird vom DBMS automatisch vorgenommen

→ Anfragen und Änderungen werden automatisch von der Sicht auf die „echte“ Datenbank umgesetzt



SQL ist nicht „orthogonal“, d.h. es kann nicht an jeder Stelle statt einer Basisrelation auch eine Anfrage stehen.

→ Anfragetransformation wird **komplexer!** (z.T. unmöglich)



- Änderungen auf Sichten können nicht immer auf sinnvolle Änderungen der „echten“ Datenbank übertragen werden.

1. Projektionssichten (Ausblenden von Informationen)

sql

Copy

```
CREATE VIEW MA AS  
SELECT Mitarbeiter, Abteilung FROM MGA
```

Probleme:

- Beim Einfügen: Was passiert mit ausgeblendeten Attributen? (Nullwerte/Default-Werte nötig)
- Mengeneigenschaftsproblem: Ein Eintrag könnte mehreren DB-Einträgen entsprechen

2. Selektionssichten (Ausfiltern von Zeilen)

sql

Copy

```
CREATE VIEW MG AS  
SELECT Mitarbeiter, Gehalt FROM MGA  
WHERE Gehalt > 20
```

Probleme:

- Tupelmigration: Durch Updates könnten Datensätze aus der Sicht "verschwinden"
- Lösbar durch WITH CHECK OPTION

3. Verbundssichten (Zusammenstellen einer neuen virtuellen Relation) Probleme:

- Beim Einfügen: Welche Basistabelle wird aktualisiert?
- Beim Löschen: Aus welcher Basistabelle soll gelöscht werden?
- SQL92 untersagt generell Änderungen auf Verbundssichten

4. Aggregierungssichten (Gruppierung/Aggregatfunktionen) Probleme:

- Keine sinnvolle Übersetzung von Änderungen möglich
- Beispiel: Wie soll eine Erhöhung der Gesamtsumme auf einzelne Datensätze verteilt werden?

Allgemein: Probleme bei der Verwaltung von Sichten

Integritätsverletzungen Beispiel: Einfügen von Nullwerten, die nicht zulässig sind

Zugriffsschutz Seiteneffekte auf nicht-sichtbaren Bereich

Auswahlproblem Mehrere Transformationsmöglichkeiten

Sinnhaftigkeit In manchen Fällen ist keine sinnvolle Transformation in Änderungen im zugrundeliegenden Modell möglich

1:1 Beziehung zwischen Sichttupeln und Tupeln der Datenbank

Dies ist bei „Herausprojizieren“ eines Schlüsselattributs nicht gegeben.

Rechtevergabe

Freigeben von Ausschnitten der Datenbank für bestimmte Nutzer/Nutzergruppen.
Freigeben von bestimmten *Operationen* auf die Datenbanktabellen.

SQL-92

<i>Gegenstück</i> REVOKE <Rechte> ON <Tabelle> FROM <Nutzer> [RESTRICT / CASCADE]	GRANT <die ausführbaren Operationen> ON <Sicht/Tabelle> TO <Benutzer/Benutzerliste> [WITH GRANT OPTION]	Insert / Select / Update / Delete
---	--	--

Beispiel: „public“ - View

```
CREATE VIEW MeineVorlesung AS  
SELECT * FROM Vorlesung WHERE Prof = user;  
GRANT select, insert ON MeineVorlesung TO public;
```

Jeder Dozent kann seine Vorlesungen einsehen

Einschränkungen in SQL 92

Änderungen sind nur erlaubt, wenn:

- die Anfrage eine reine Selektionsanfrage ist
- die Anfrage zur Sichterzeugung kein DISTINCT enthält
- keine Arithmetik und Aggregatfunktionen im SELECT-Teil verwendet werden
- nur eine Basisrelation, d.h. eine Referenz im FROM-Teil verwendet wird
- keine Unteranfragen mit Selbstbezug vorhanden sind
- keine Gruppierungen (GROUP BY / HAVING) vorhanden sind

Solche Sichten werden in SQL 92 als „**updatable**“ geführt

Integrität

Integrität / Integritätsbedingung

Integritätsbedingungen legen fest, wann Zulässigkeit gegeben ist für



Integrität / Integritätsbedingung ... in SQL

Innere Integritätsbedingungen

Typintegrität (Festlegung von Wertebereichen von Attributen)

Schlüsselintegrität

Referentielle Integrität

Spezielle DDL-Befehle in SQL 92

NOT NULL
CHECK ← Primär für 1-Tupel-Prüfungen
PRIMARY KEY
FOREIGN KEY <...> REFERENCES ... <...>
ASSERTION ← Relationenübergreifende Bedingung

Integrität / Integritätsbedingung

... in SQL

ASSERTION

```
CREATE ASSERTION Gesamtwert CHECK  
((SELECT sum(Preis) FROM Buch) < 10000)
```

beliebige boole'sche Bedingung

Behandlung von Verletzungen der Integrität

Auslösendes Ereignis → on update / on delete → Konsistenzhaltende Handlung

Cascade / Set NULL / set default / no action

Transaktionen: deferred / immediate

Wird am Ende der Transaktion geprüft

Muss sofort gelten

Integritätsbedingungen... in SQL

- ① Studenten werden durch Name, Vorname und Geburtsdatum identifiziert. → Durch Festlegung eines Schlüsse
- ② Der Kontostand eines Girokontos darf nicht unter die Kreditlinie von 5000 EURO fallen. → Kontostand numeric(20,2)
CHECK (Kontostand > -5000)
- ③ Der Kontostand von Herrn X darf nicht unter 0 fallen. → Kontostand numeric(20,2)
CHECK (NOT (Name = „X“) OR Kontostand > 0)
- ④ Der Minimalpreis eines antialkoholischen Getränks muss unter dem Minimalpreis eines alkoholischen Getränks liegen. → CREATE ASSERTION PubLaw CHECK
((SELECT min(Preis) FROM Getränk WHERE alk > 0.0) > (SELECT min(Preis) FROM Getränk WHERE alk = 0.0))
- ⑤ Es dürfen nur Waren bestellt werden, für die es mindestens einen Lieferanten gibt.
- ⑥ Der Mietzins darf dieses Jahr nicht erhöht werden.
- ⑦ Kunden dürfen nur gelöscht werden, wenn sie keine Waren mehr bestellt haben.
- ⑧ Kunden müssen gelöscht werden, wenn sie keine Waren mehr bestellt haben. → Dynamisch !?
Trigger Konzept
- ⑨ Der Mietzins darf innerhalb von 3 Jahren um nicht mehr als 10 % erhöht werden. → Nicht unterstützbar!

NoSQL / NewSQL

Grundlegende Konzepte und Motivation

Entstehung und Hintergrund

NoSQL ("Not Only SQL") entstand aus der Notwendigkeit, mit den neuen Anforderungen durch Big Data umzugehen:

- Wachsende Datenvolumen
- Weniger strukturierte Daten
- Schnellere/zeitnähere Verarbeitung
- Bessere Skalierbarkeit
- Neue Datenquellen (Logs, Blogs, Sensoren etc.)

Das CAP-Theorem

Das CAP-Theorem besagt, dass in verteilten Systemen nur zwei der folgenden drei Eigenschaften gleichzeitig garantiert werden können:

- Consistency (Konsistenz): Alle Knoten sehen die gleichen Daten zur gleichen Zeit
- Availability (Verfügbarkeit): Jede Anfrage erhält eine Antwort
- Partition Tolerance (Partitionstoleranz): System funktioniert trotz Netzwerkausfällen

Daraus ergeben sich drei mögliche Kombinationen:

1. CA: Konsistenz + Verfügbarkeit (klassische RDBMS)
2. CP: Konsistenz + Partitionstoleranz
3. AP: Verfügbarkeit + Partitionstoleranz (viele NoSQL-Systeme)

NoSQL Datenmodelle

Key-Value Stores

- Einfachste Form der NoSQL-Datenbanken
- Speichern Schlüssel-Wert-Paare
- Sehr schneller Zugriff über Schlüssel
- Werte können beliebige Daten sein (BLOB)
- Beispiele: Redis, DynamoDB

Vorteile:

- Sehr schnell bei einfachen Operationen
- Gut skalierbar
- Einfach zu verstehen

Nachteile:

- Keine komplexen Abfragen möglich
- Keine Beziehungen zwischen Daten
- Keine Schemavalidierung

Document Stores

- Speichern strukturierte Dokumente (meist JSON)
- Dokumente können unterschiedliche Strukturen haben
- Unterstützen komplexere Abfragen
- Beispiele: MongoDB, CouchDB

Vorteile:

- Flexibles Schema
- Gut für verschachtelte Daten
- Bessere Abfragemöglichkeiten als Key-Value

Nachteile:

- Komplexere Konsistenzerhaltung
- Weniger performant als Key-Value
- Schwierigere Datenmigration

Wide Column Stores

- Spaltenorientierte Speicherung
- Flexible Anzahl von Spalten pro Zeile
- Gut für analytische Abfragen
- Beispiele: Cassandra, HBase

Vorteile:

- Gut für große Datenmengen
- Effizient bei Spaltenoperationen
- Gute Komprimierbarkeit

Nachteile:

- Komplexes Datenmodell
- Aufwändige Schreiboperationen
- Schwierige Schemaänderungen

Graph Databases

- Speichern Knoten und Kanten
- Gut für stark vernetzte Daten
- Effizient bei Traversierungen
- Beispiele: Neo4j, ArangoDB

Vorteile:

- Natürliche Abbildung von Beziehungen
- Effiziente Graphalgorithmen
- Flexibel bei Schemaänderungen

Nachteile:

- Schwieriger zu skalieren
- Komplexe Abfragesprachen
- Aufwändige Datenmigration

Technische Konzepte

Konsistenzmodelle

Eventual Consistency

- Schwächeres Konsistenzmodell als ACID
- System wird "irgendwann" konsistent

- Höhere Verfügbarkeit und bessere Performance
- Basis für viele NoSQL-Systeme

BASE vs ACID

BASE (Basic Availability, Soft state, Eventual consistency):

- Grundsätzlich verfügbar
- Weicher Zustand (temporäre Inkonsistenzen)
- Letztendliche Konsistenz

ACID (Atomicity, Consistency, Isolation, Durability):

- Traditionelles Transaktionsmodell
- Strenge Konsistenz
- Schlechtere Skalierbarkeit

Verteilungskonzepte

Sharding

- Horizontale Partitionierung der Daten
- Verteilung auf mehrere Server
- Verbessert Skalierbarkeit
- Verschiedene Strategien möglich:
 - Range-based
 - Hash-based
 - Directory-based

Replikation

- Speicherung von Kopien der Daten
- Verbessert Verfügbarkeit und Leseleistung
- Verschiedene Modelle:
 - Master-Slave
 - Multi-Master
 - Peer-to-Peer

Architekturkonzepte

Verteilte Architekturen

- Shared Nothing: Jeder Knoten arbeitet unabhängig
- Horizontale Skalierung durch Hinzufügen von Knoten
- Lastverteilung über mehrere Server

Skalierung

- Scale-Out (horizontal): Hinzufügen von Servern
- Scale-Up (vertikal): Aufrüstung bestehender Server
- NoSQL optimiert für Scale-Out

Praktische Aspekte

Anwendungsfälle

Gut geeignet für:

- Große Datenmengen
- Hohe Schreib-/Leseraten

- Flexible Schemas
- Verteilte Systeme

Weniger geeignet für:

- Komplexe Transaktionen
- Strenge Konsistenzanforderungen
- Relationale Daten
- Ad-hoc Queries

Auswahl einer NoSQL-Datenbank

Kriterien:

- Datenmodell und Anwendungsfall
- Konsistenzanforderungen
- Skalierungsanforderungen
- Performance-Anforderungen
- Betriebskosten

Vor- und Nachteile von NoSQL

Vorteile:

- Bessere Skalierbarkeit
- Flexibleres Datenmodell
- Höhere Verfügbarkeit
- Bessere Performance bei bestimmten Anwendungsfällen
- Einfachere Entwicklung bei passenden Use-Cases

Nachteile:

- Schwächere Konsistenzgarantien
- Komplexere Administration
- Weniger standardisierte Schnittstellen
- Eingeschränkte Abfragemöglichkeiten
- Steilere Lernkurve

Konsistenzsicherung und Datenzugriff

MVCC (Multiversion Concurrency Control)

- Synchronisationsverfahren ohne klassische Sperren
- Erzeugt verschiedene Versionen eines Datenobjekts
- Bei Schreibzugriff wird neue Version erstellt
- Versionen werden mit Zeitstempeln/Transaktionsnummern versehen

Funktionsweise:

- Transaktionen sehen verschiedene Versionen der Daten
- Keine blockierenden Sperren notwendig
- Konfliktüberprüfung am Transaktionsende

Vorteile:

- Höherer Durchsatz

- Leser blockieren Schreiber nicht und umgekehrt

Nachteile:

- Verwaltungsaufwand für Versionen
- Mögliches Rücksetzen von Transaktionen
- Garbage Collection nötig

Klassisches Locking (Sperrverfahren)

RX-Sperrverfahren:

- Read Lock (shared lock)
- Exclusive Lock (exclusive lock)
- Mehrere Reader möglich
- Nur ein Writer zur Zeit

Two-Phase-Locking (2PL):

- Wachstumsphase: Sperren anfordern
- Schrumpfungsphase: Sperren freigeben

Striktes 2PL: Freigabe erst am Transaktionsende

Probleme:

- Deadlocks möglich
- Wartezeiten
- Geringerer Durchsatz

Konsistentes Hashing

Grundprinzip:

- Verteilung von K Objekten auf n Knoten
- Hash-Ring-Konzept statt linearer Hash-Tabelle
- Objekte und Server werden auf Ring abgebildet

Vorteile:

- Effiziente Skalierung
- Bei Server-Ausfall/Hinzufügung nur K/N Objekte neu zuordnen
- Gut für verteilte Systeme

Funktionsweise:

- Hash-Funktion bildet Server und Daten auf Ring ab
- Daten werden dem nächsten Server im Uhrzeigersinn zugeordnet
- Bei Serverausfall übernimmt nächster Server im Ring
- Bei neuem Server werden nur direkt betroffene Daten verschoben

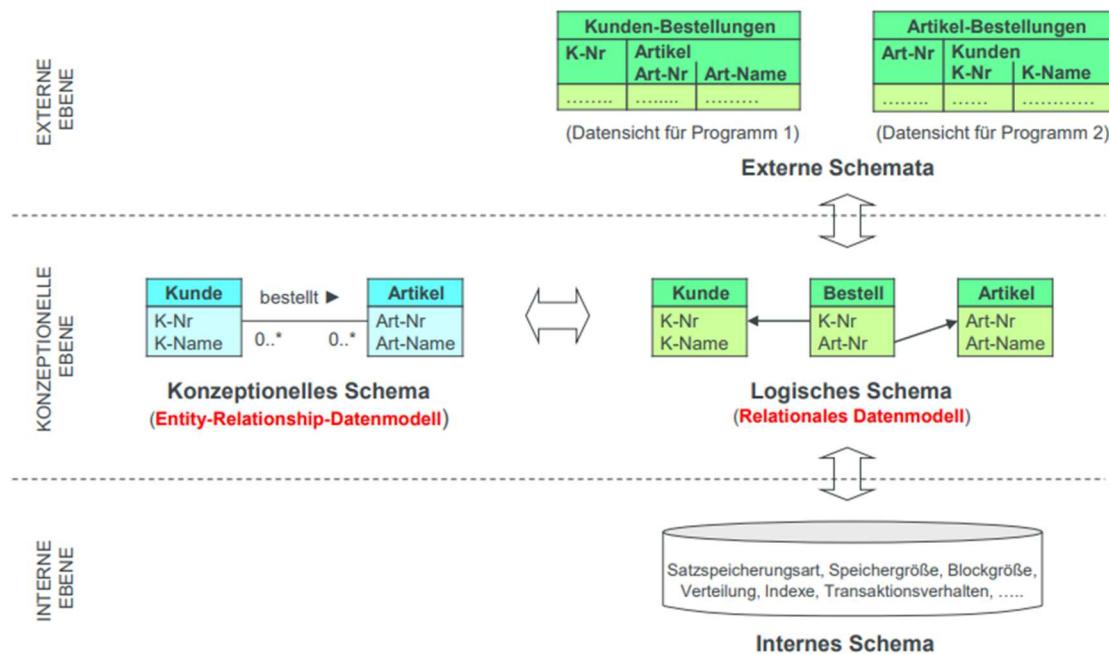
Optimierung:

- Virtuelle Knoten (mehrere Hash-Werte pro Server)
- Bessere Lastverteilung
- Flexiblere Gewichtung der Server

Entity-Relationship-Modell

ANSI/SPARC

3-Ebenen-Architektur (nach ANSI/SPARC):



Datenmodellierung

Ziel der Datenmodellierung

- ☞ Analyse: Welche Daten(felder) sollen in der Datenbank verwaltet werden?
- ☞ deren Zusammengehörigkeiten und Beziehungen aufzeigen
- ☞ Strukturierung mittels **Entity-Relationship-Datenmodellierung** als spezielle Form der Klassenmodellierung (**UML**)
- ☞ einfache und redundanzfreie Darstellung in einem **ER-Diagramm**

Entitäten

Entitätstyp

- grundlegendes Strukturierungselement (entsprechend Entity-Klasse aus UML)
- Beschreibung mittels
 - **Attribut,**
 - **Wertebereich,**
 - **Primärschlüssel,**
 - **Weak-Entitätstyp und**
 - **Integritätsbedingung.**



1. Allgemeines

- strukturierte Beschreibung gleichartiger **Informationsobjekte** (Entitäten, Instanzen)
- Substantive als Bezeichner
- im ER-Diagramm: Rechtecke

2. Attribut

- Beschreibung eines Entitätstyps mittels Attributen (**Eigenschaften**, Merkmalen)
 - **atomar**
 - **zusammengesetzt**
- **abgeleitete Attribute nicht** erlaubt

3. Wertebereich

- Menge der **erlaubten Werte** für Attribute
- **Standard-Wertebereiche**

Wertebereich	Bedeutung
ZAHL	ganze Zahl
ZAHL (X)	ganze Zahl mit X Stellen (inkl. führende Nullen)
ZAHL (X, Y)	Dezimalzahl mit x Stellen insgesamt (mit führenden Nullen) und Y Nachkommastellen
ZEICHEN (X)	Zeichenkette mit x Stellen (feste Länge)
VARZEICHEN (X)	variabel lange Zeichenkette mit max. X Stellen
DATUM	Datum (interne Verwaltung: JJJJ.MM.TT)
ZEIT	Zeit (interne Verwaltung: HH:MM:SS)
WAHRHEIT	{wahr falsch} (alternativ: {0 1} oder {ja nein})

- explizite **Aufzählung** der erlaubten Werte (mit Mengenklammer)
 - Werte getrennt durch Komma
 - Werte getrennt durch XOR („ | “)
- einwertige und **mehrwertige** Attribute

4. Primärschlüssel

- minimale Attributkombination zur eindeutigen **Identifikation** von Informationsobjekten
- evtl. mehrere **Schlüsselkandidaten**
- Primärschlüssel nicht mit gleicher Semantik wie in UML (dort Objekt-Id)

5. Weak-Entitätstyp

- **von anderen Entitätstypen abhängig**
- **kein eigener Primärschlüssel**

6. Integritätsbedingungen

- garantieren ordnungsgemäßen Zustand / Integrität einer Datenbank
- Angaben für Attribute:
 - Wertebereich und Primärschlüssel
 - „**EINDEUTIG**“ bzw. „**EINDEUTIG (<Attributkombination>)**“
 - „**[min..max]**“-Angabe („*“ für „mehrere“; Standard-Wert: „[0..1]“)
 - „**STANDARD = <Wert>**“
 - „**CHECK (<Bedingung>)**“

Relationship

Beziehungstyp

- Entitätstypen mit **Verbindungen** zu anderen Entitätstypen
- zweites grundlegendes Strukturierungselement (entsprechend Beziehungstyp aus UML)
- Beschreibung mittels
 - **Grad**,
 - **Beziehungstypen (Assoziation, Abhängigkeit, Aggregation, Generalisierung)**,
 - **Beziehungsentitätstyp**,
 - **Multiplizität**,
 - **Rolle**,
 - **Attribut**,
 - **Fremdschlüssel und**
 - **Primärschlüssel**.



1. Allgemeiner Beziehungstyp

- **Assoziation** zwischen Entitätstypen
- **Grad** = Anzahl der beteiligten Entitätstypen
 - meistens **binärer** Beziehungstyp mit Grad 2 (auch bei Reflexion)
- **Verben** als Bezeichner
- im ER-Diagramm: *Verbindungslien* zwischen den Entitätstypen
 - bei Grad > 2 Verbindungslien zentral in einer *Raute* sammeln

2. Spezielle Beziehungstypen

- fest vorgegebene Bedeutung => kein Bezeichner zu vergeben
 - Abhangigkeit**
 - Aggregation**
 - Generalisierung**

Abhangigkeit

- zwischen normalem Entitatstyp und *Existenz-abhangigem Weak-Entitatstyp*

Aggregation

- Verbindung zwischen einem **Ganzen** (Aggregat) und dessen **Bestandteilen** (Komponenten)
- „**ist Teil von**“-Beziehungstyp
 - normale Aggregation:**
Komponente-IO gehort zu keinem, einem oder mehreren Ganzes-IO
 - starke Aggregation (Komposition):**
Komponente-IO existiert nur mit genau einem Ganzes-IO
- im ER-Diagramm:
 - starke Aggregation mit **gefillter Raute**, normale Aggregation mit **leerer Raute**
 - Komponenten-Beziehungstypen entweder einzeln oder uber gemeinsame Raute



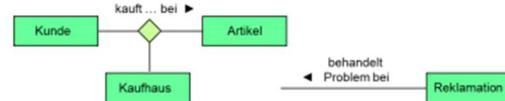
Generalisierung

- „**ist ein**“-Beziehungstyp (engl.: „**is-a**“)
- von Eltern-Entitatstyp (**Super-Entitatstyp**) an Kind-Entitatstyp (**Sub-Entitatstyp**):
Vererbung gemeinsamer Attribute und Beziehungstypen
 - insbesondere auch Primarschlssel
- im ER-Diagramm: Generalisierung entweder einzeln oder uber gemeinsamen Pfeil

3. Beziehungsentitätstyp

- Beziehungstyp zwischen Entitätstyp und anderem Beziehungstyp

Beispiel 13a: Für den „*kauft...bei*“-Beziehungstyp aus Abb. 7 ist zu beachten, dass es bei bestimmten Käufen zu Reklamationen kommen kann. Die **Reklamation** selbst wird dabei in einem eigenen Entitätstyp mit *Reklamationsnummer*, *Datum*, *Problemtext* und *Maßnahme* verwaltet.



Zuordnung der **Reklamation** zum entsprechenden **Kauf**:

- Verbindung vom Entitätstyp **Reklamation** zu einem der drei am Kauf beteiligten Entitätstypen **Kunde**, **Artikel** oder **Kaufhaus** ist NICHT MÖGLICH
- Zuordnung bezieht sich nicht auf einen einzelnen Entitätstyp, sondern auf die Kombination aus allen drei am Kauf beteiligten Entitätstypen
=> auf Beziehungstyp **kauft...bei**
- Verbindung von **Reklamation** über einen Beziehungstyp **behandelt Problem bei** zum Beziehungstyp **kauft...bei**
- Interpretation eines **Beziehungstyps als Entitätstyp**
- als (Hilfs-)Entitätstyp dem betreffenden Beziehungstyp zugeordnet

4. Multiplizität

- Wie viele konkrete Beziehungen von IO des einen Typs zu IOen des anderen Typs?
- Kardinalität / Komplexität
- bei Beziehungstyp für jeden beteiligten Entitätstyp eine Multiplizitätsangabe
 - **Unter- und Obergrenze** für konkrete Beziehungen eines IO

allerdings unklar: **Aufteilung** der IOe des Super-Entitätstyps auf die Sub-Entitätstypen

- **total**: alle IOe des Super-Entitätstyps auch als IO eines Sub-Entitätstyps
- **partiell**: nicht alle IOe des Super-Entitätstyps auch als IO eines Sub-Entitätstyps
- **disjunkt**: alle IOe des Super-Entitätstyps als IO höchstens eines Sub-Entitätstyps
- **nicht disjunkt**: IOe des Super-Entitätstyps als IO mehrerer Sub-Entitätstypen

Generalisierung/Spezialisierung

- einerseits *total oder partiell*
- und andererseits *disjunkt oder nicht disjunkt*

im ER-Diagramm: **t** (total), **p** (partiell), **d** (disjunkt), **n** (nicht disjunkt)

5. Rolle

- bei Beziehungstypen für beteiligte Entitätstypen
 - insbesondere bei **reflexiven** Beziehungstypen sehr hilfreich

6. Attribut und Fremdschlüssel

- **Attribute auch bei Beziehungstyp**
(Integritätsbedingungen wie bei Entitätstyp)
 - normale **eigene Attribute**
 - **fremde Attribute** bzw. **Fremdschlüsselattribute**
 - Primärschlüssel der am Beziehungstyp beteiligten Entitätstypen werden ausgeliehen
 - Fremdschlüsselattribut besitzt immer einen Wert
=> Attribut-Komplexität immer
 - im ER-Diagramm:
Fremdschlüsselattribut mit Zusatz



7. Primärschlüssel

- *minimale* Attributkombination zur eindeutigen **Identifikation** von konkreten Beziehungen
 - künstliches Attribut
 - Bestimmung **aus der Menge der Fremdschlüsselattribute**
 - ✓ Primärschlüssel ergibt sich aus Multiplizitäten des Beziehungstyps

8. Beziehungstypen mit einem Grad > 2

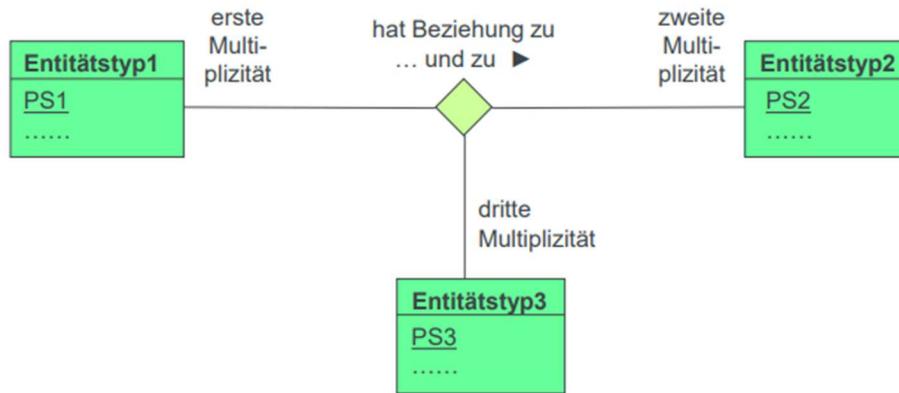
- vorwiegend **binäre Beziehungstypen** (Grad = 2)
- aber auch **Beziehungstypen mit höherem Grad**
 - Handhabung wesentlich schwieriger

Multiplizitäten

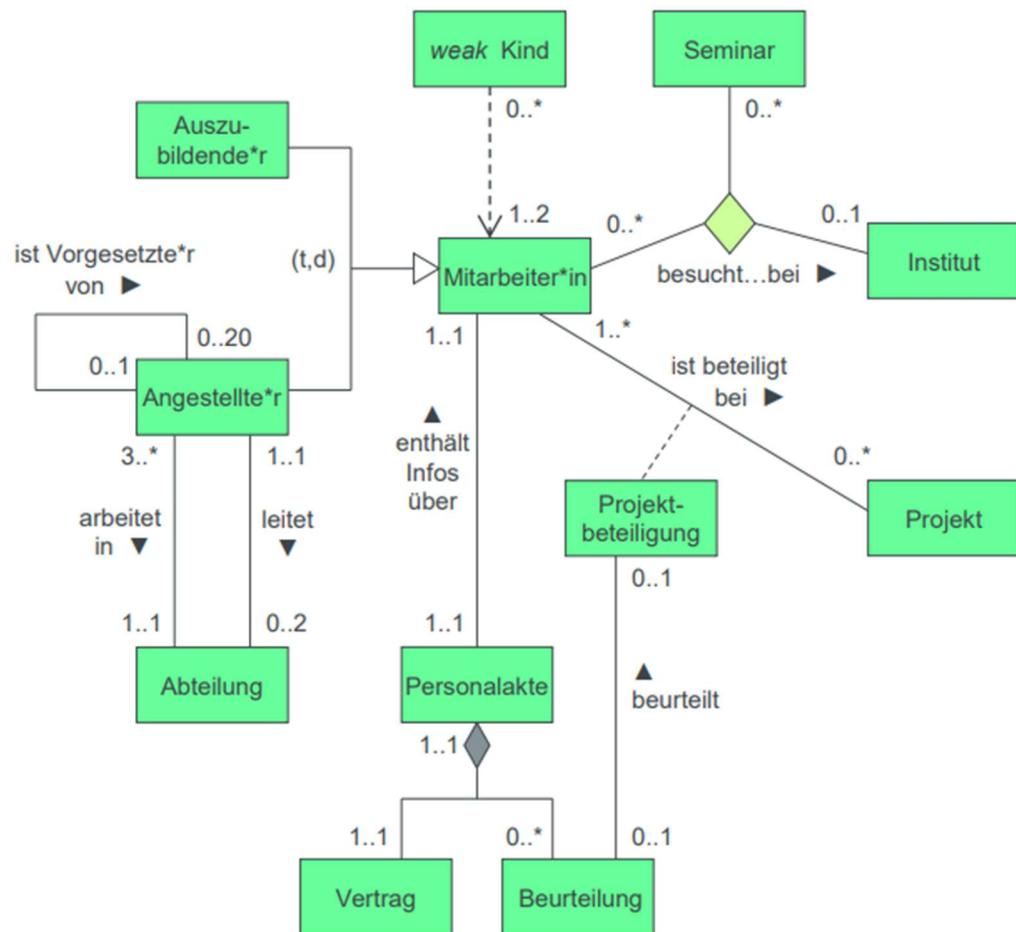
- Multiplizitätsangabe bei einem Entitätstyp gilt für **Kombination** der restlichen am Beziehungstyp beteiligten Entitätstypen
- bei Grad > 2 kann nicht mehr ausgedrückt werden, ob einzelnes IO ohne konkrete Beziehung auftreten kann (s. Bsp. 22)
 - nur bei binären Beziehungstypen möglich

Primärschlüssel

- Regeln zur Bestimmung des Primärschlüssels aus den Fremdschlüssen beim Beziehungstyp aus Abb. 23



Beispiel für fertiges Modell



Mitarbeiter*in		
<u>Personalnummer:</u> ZEICHEN (10)	EINDEUTIG	[1..1]
Name:		[1..1]
Vorname: VARZEICHEN (20)		[1..1]
Nachname: VARZEICHEN (20)		[1..1]
Anschrift:		[1..*]
Straße: VARZEICHEN (30)		[1..1]
PLZ: ZAHL (5)	CHECK (> 0)	[1..1]
Ort: VARZEICHEN (30)		[1..1]
Abteilung		
<u>Kürzel:</u> ZEICHEN (5)	EINDEUTIG	[1..1]
Name: VARZEICHEN (25)		[1..1]
Seminar		
<u>Seminarnummer:</u> ZEICHEN (10)	EINDEUTIG	[1..1]
Titel: VARZEICHEN (30)		[1..1]
Beschreibung: VARZEICHEN (1000)		[1..1]
Institut		
<u>Name:</u> VARZEICHEN (30)	EINDEUTIG	[1..1]
Anschrift:		[1..1]
Straße: VARZEICHEN (30)		[1..1]
PLZ: ZAHL (5)	CHECK (> 0)	[1..1]
Ort: VARZEICHEN (30)		[1..1]
Personalakte		
<u>Aktennummer:</u> ZEICHEN (10)	EINDEUTIG	[1..1]
Vertrag		
<u>Vertragsnummer:</u> ZEICHEN (10)	EINDEUTIG	[1..1]
Vertragstyp: {Arbeit Ausbildung}	STANDARD = Arbeit	[1..1]
Abschluss: DATUM		[1..1]
Text: VARZEICHEN (500.000)		[1..1]
Beurteilung		
<u>Beurteilungsnummer:</u> ZEICHEN (10)	EINDEUTIG	[1..1]
Datum: DATUM		[1..1]
Fachkompetenz: VARZEICHEN (10.000)		[1..1]
Sozialkompetenz: VARZEICHEN (10.000)		[1..1]
Projekt		
<u>Projektnname:</u> VARZEICHEN (30)	EINDEUTIG	[1..1]
Start: DATUM		[1..1]
Ende: DATUM		[0..1]
Budget: ZAHL (9,2)	CHECK (> 0) und (< 5.000.000))	[0..1]
Strategisch: WAHRHEIT	STANDARD = falsch	[1..1]
Projektbeteiligung		
<u>Personalnummer:</u> ZEICHEN (10)	FS	[1..1]
<u>Projektnname:</u> VARZEICHEN (30)	FS	[1..1]
Rolle: VARZEICHEN (20)		[1..*]
Weak Kind		
Vorname: VARZEICHEN (20)		[1..1]
Geburtsdatum: DATUM		[1..1]
Auszubildende*		
Beruf: VARZEICHEN (20)		[1..1]
Angestellte*r		
Gehalt: ZAHL (8,2)	CHECK (> 0)	[1..1]

Datenbanksprache SQL

Datendefinition (Schema, Tabellen, Datentypen, Views, Indexe)

1.1. Schema und Tabellen

- alle Definitionen für relationale Datenbank werden im relationalen Schema gesammelt

☞ zunächst **leeres Schema definieren:**

```
CREATE SCHEMA <Schema-Name>
```

☞ Datenbankschema löschen:

```
DROP SCHEMA <Schema-Name> CASCADE | RESTRICT
```

☞ **Tabelle (Relation) definieren:**

```
CREATE TABLE <Tabellen-Name>
  ( <Attribut-Definitionen>,
    [<Tabellen-Integritätsbedingungen>] )
```

☞ **Attribut definieren:**

```
<Attribut-Name> <Datentyp> [<Attribut-Integritätsbedingungen>]
```

Basis-Datentypen für Attribute:

- **INT | INTEGER:** ganze Zahl zwischen -2^{31} und $2^{31}-1$
- **SMALLINT:** ganze Zahl zwischen -2^{15} und $2^{15}-1$
- **DEC | DECIMAL | DECIMAL(p, [q]):** Dezimalzahl (mit insgesamt p Stellen, davon q Dezimalstellen)
- **NUM | NUMERIC | NUMERIC(p, [q]):** (wie DECIMAL)
- **FLOAT | FLOAT(p):** Zahl in (4 Byte-)Fließkomma-Darstellung mit Vorzeichen
- **REAL | DOUBLE PRECISION:** reelle Zahl (double precision ist genauer)

- **CHAR(n):** alphanumerische Zeichenkette mit fester Länge n
- **VARCHAR(n):** alphanumerische Zeichenkette mit variabler Länge (maximal n)
- **CLOB(m [K|M|G]):** Text mit max. m Zeichen (angebbar in Kilo-, Mega- oder Giga-Byte)
- **BOOLEAN:** Wahrheitswerte true, false, unknown
- **DATE:** Datum - als Zeichenkette in der Form JJJJ-MM-TT
- **TIME:** (Uhr-)Zeit - als Zeichenkette in der Form HH:MM:SS
- **TIMESTAMP:** Zeitstempel - als Zeichenkette in der Form JJJJ-MM-TT HH:MM:SS

☞ **Integritätsbedingungen** für Attribute:

- **UNIQUE**
- **NOT NULL**
- **DEFAULT** <Default-Wert>
- **CHECK** (<>Attribut-Name> | VALUE> <Prüfbedingung 1>
[AND|OR ... AND|OR <>Attribut-Name> | VALUE> <Prüfbedingung n>])
- **PRIMARY KEY**
- **REFERENCES** <Tabellen-Name> [<Attribut-Name>]
[ON DELETE CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION]
[ON UPDATE CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION]

Prüfbedingungen für CHECK-Klausel:

- <Vergleichsoperator> <Wert-Ausdruck>
mit:
<Vergleichsoperator>: = | < | <= | > | >= | <>
<Wert-Ausdruck>: <konkreter Wert> | <arithm. Ausdruck> | <skalare SELECT-Anweisung>
- **[NOT] IN** (<Werte-Aufzählung> | <SELECT-Anweisung>)
- **[NOT] BETWEEN** <Untergrenze> AND <Obergrenze>
- **[NOT] LIKE** <Textmuster> [ESCAPE <Sonderzeichen>]

- <Attribut-Name> WITH OPTIONS <Attribut-Integritätsbedingungen>
- **UNIQUE** (<Liste Attribut-Namen>)
- **CHECK** (<Selektionsbedingung 1> [AND|OR ... AND|OR <Selektionsbedingung n>])
- **PRIMARY KEY** (<Liste Attribut-Namen>)
- **FOREIGN KEY** (<Liste Attribut-Namen>) REFERENCES <Tabellen-Name> [<Liste Attribut-Namen>]
[ON DELETE CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION]
[ON UPDATE CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION]

Änderung einer Tabellenstruktur

```
ALTER TABLE <Tabellen-Name>
ADD [COLUMN] <Attribut-Name> <Datentyp> [<Attribut-Integritätsbedingungen>] |
DROP [COLUMN] <Attribut-Name> CASCADE | RESTRICT
```

☞ **Tabelle löschen:**

```
DROP TABLE <Tabellen-Name> CASCADE | RESTRICT
```

1.3. Views und Indexe

➤ View = **Sicht eines Benutzers** auf die Datenbank

- Daten aus Basisrelationen und anderen Views extrahieren
- Inhalt einer View nicht permanent in Datenbank gespeichert (**virtuelle Relation**)

☞ **View definieren:**

```
CREATE VIEW <View-Name> [(<Liste Attribut-Namen> )]  
AS <SELECT-Anweisung>
```

☞ **View löschen:**

```
DROP VIEW <View-Name> CASCADE | RESTRICT
```

➤ Index = **Attributwertverzeichnis**

- dient zum **schnellen Zugriff** auf einen bestimmten Datensatz
- liefert zu einem bestimmten Attributwert die Speicheradresse des zugehörigen Datensatzes
- Index kann über einzelnes Attribut oder über eine Attributkombination definiert werden
- Index im SQL-Standard nicht vorgesehen, aber in Datenbanksystemen unentbehrlich

☞ **Index definieren:**

```
CREATE [UNIQUE] INDEX <Index-Name>  
ON <Tabellen-Name> (<Attribut-Name 1> [ASC | DESC], ...,  
                      <Attribut-Name n> [ASC | DESC] )
```

Bsp.:

☞ **Index löschen:**

```
DROP INDEX <Index-Name>
```

Datenabfrage (Suchen, Aggregieren, Schachtelungen, Verbund)

2. Datenabfrage

- ☞ SELECT-Anweisung mit den relationalen Operatoren **Selektion, Projektion, Verbund**
- ☞ Formulierung einer Abfrage mit SQL:

```
SELECT      [DISTINCT]    * | <Attribut 1> [AS <neuer Attribut-Name 1>]
              [, ..., <Attribut n> [AS <neuer Attribut-Name n>] ]
  FROM        <Tabelle 1> [[AS] <neuer Tabelle-Name 1>]
              [, ..., <Tabelle m> [[AS] <neuer Tabelle-Name m>] ]
  [WHERE       <Verknüpfungs- und Selektionsbedingungen>]
  [GROUP BY   <Liste Attribut-Namen> [HAVING <Auswahl-Bedingung>]]
  [ORDER BY   <Attribut-Name 1> [ASC | DESC] [, ..., <Attribut-Name p> [ASC | DESC]]]
```

- <Attribut>: <Attribut-Name> | <arithm. Ausdruck> | <konkreter Wert> | <skalare Subabfrage>
- <Tabelle>: <Tabellen-Name> | <Subabfrage>

- Auswertungsreihenfolge einer SQL-Abfrage-Anweisung:

Schritt	Anweisungsteil	Bedeutung
1	FROM	Auswahl der für die Auswertung erforderlichen Tabellen
2	WHERE	Angabe der Verknüpfungs- und Selektionsbedingungen für die Tabellen des FROM-Teils
3	GROUP BY	Angabe der Attribute, nach denen die Datensätze gruppiert werden sollen (gleiche Attributwerte kommen in eine Gruppe)
4	HAVING	Auswahl bestimmter Gruppen (nur in Verbindung mit GROUP BY)
5	SELECT	Auswahl der Attribute der Ergebnistabelle
6	ORDER BY	Angabe der Sortierreihenfolge für die Ergebnissätze

2.1.4. Ausgabe berechneter Attribute

- ☞ im SELECT-Teil **an Stelle von Attributname auch Formel möglich**, die numerische Attribute der im FROM-Teil genannten Relationen und Konstanten mittels der arithmetischen Operatoren +, -, *, / verknüpft
- ☞ Formel kann **Aggregationsfunktionen** (s.u.) enthalten
- ☞ Mit dem Zusatz „**AS <Attribut-Name>**“ kann der Formel als Attribut der Ergebnisrelation ein sinnvoller Name zugewiesen werden.

2.1.5. Ausgabe spezieller Datensätze einer Relation

☞ Selektionsbedingungen für den WHERE-Teil:

Vergleich	<<Attribut-Name> <arithm. Ausdruck>> <Vergleichsoperator> <Wert-Ausdruck>
Prüfung Mengenelement	<<Attribut-Name> <arithm. Ausdruck>> [NOT] IN (<Werte-Aufzählung> <Subabfrage>)
Prüfung Bereich	<<Attribut-Name> <arithm. Ausdruck>> [NOT] BETWEEN <Untergrenze> AND <Obergrenze>
Prüfung Ähnlichkeit	<<Attribut-Name> <arithm. Ausdruck>> [NOT] LIKE <Textmuster> [ESCAPE <Sonderzeichen>] <i>mit möglichen Platzhaltern im Textmuster:</i> % : Prozentzeichen für 0, 1 oder mehrere Zeichen _ : Unterstrich für 1 Zeichen

2.2. Ausgabe aggregierter Werte

2.2.1. Aggregationsfunktionen

Feststellung der Anzahl Datensätze in einer Tabelle	COUNT (*)
Feststellung der Anzahl Werte in einer Spalte	COUNT ([DISTINCT] <Attribut-Name>)
Berechnung der Summe der Werte in einer Spalte	SUM ([DISTINCT] <Attribut-Name>)
Berechnung des Durchschnitts der Werte in einer Spalte	AVG ([DISTINCT] <Attribut-Name>)
Berechnung des Minimums der Werte in einer Spalte	MIN (<Attribut-Name>)
Berechnung des Maximums der Werte in einer Spalte	MAX (<Attribut-Name>)

- Beseitigung mehrfach genannter Datensätze bei COUNT, SUM und AVG durch Angabe von DISTINCT
- SUM und AVG nur für numerische Attribute,
COUNT, MIN, MAX auch für alphanumerische Attribute
- bei COUNT (*) Nullwerte ohne Auswirkungen (alle Datensätze werden gezählt),
bei anderen Aggregationsfunktionen werden Nullwerte bei Auswertung ausgeschlossen
- Aggregationsfunktionen nur im SELECT- und im HAVING-Teil erlaubt, aber nicht im WHERE-Teil
- Vor der Anwendung von Aggregationsfunktionen im SELECT-Teil wird zunächst der WHERE-Teil und der GROUP BY-Teil ausgewertet.
- Wird bei einer Anfrage im SELECT-Teil eine Aggregationsfunktion verwendet und gibt es keinen GROUP BY-Teil, so dürfen im SELECT-Teil die Attribute nur als Argumente der Aggregationsfunktionen verwendet werden.
- Im SELECT-Teil können mit der AS-Klausel den Ausdrücken mit Aggregationsfunktionen ‚Attribut‘-Namen für die Ergebnisrelation zugewiesen werden.

2.2.2. Ausgabe aggregierter Werte mit Gruppierung

- ☞ Datensätze können vor Aggregationen gruppiert werden
 - (alle Datensätze mit gleichen Werten bei den im GROUP BY-Teil genannten Attributen kommen in eine Gruppe)
 - die Aggregationen werden dann jeweils über die Datensätze einer Gruppe gebildet
- ☞ in Ergebnisrelation erscheint je Gruppe ein Ergebnissatz
- ☞ im SELECT-Teil nur Gruppierungsattribute und Aggregationsfunktionen erlaubt
- ☞ WHERE-Teil wird vor GROUP BY-Teil ausgewertet

2.3. Verwendung von geschachtelten Abfragen

- Suchabfragen über mehrere Relationen ohne Verbund-Operator
 - ☞ Sub-Abfrage oder Sub-Select
- im WHERE-Teil sind bei Selektionsbedingungen Subabfragen möglich
=> (möglicherweise mehrfach) **geschachtelte Datenabfrage**
 - ☞ innere Abfrage wird zuerst ausgewertet und deren Ergebnis in der äußeren Abfrage verwendet
- bei Selektionsbedingungen mit
 - Vergleichsoperatoren $=, <, \leq, >, \geq, \neq$ muss Subabfrage genau einen Einzelwert als Ergebnis liefern
 - IN-Operator (Mengenelement-Prüfung) kann eine Subabfrage eine Menge von Ergebniswerten liefern

2.4. Verknüpfung von Tabellen mit Verbund-Operator

- Datensätze aus mehreren Relationen zu einem neuen Datensatz kombinieren
- mittels **Verbund-Operator (Join-Operator)**
 - ☞ im FROM-Teil stehen die zu verknüpfenden Relationen
 - ☞ im WHERE-Teil stehen die Verknüpfungsbedingungen
 - ☞ einzelner SQL-Verbund verknüpft zwei Relationen
 - ☞ falls zu verknüpfende Relationen gleichnamige Attribute aufweisen:
Attribute mit führenden Relationennamen (Präfixe) ergänzen
 - ☞ spezieller Verbund: **kartesisches Produkt** (s. Bsp. 24)
 - im WHERE-Teil fehlen die Verknüpfungsbedingungen
 - jeden Datensatz der einen Relation mit jedem Datensatz der anderen Relation kombinieren

2.4.1. Verknüpfung von Relationen mit Bedingungen

- ☞ Verknüpfungsbedingungen stehen im WHERE-Teil
- ☞ Vergleich Attributwert von Datensatz der Relation A mit Attributwert von Datensatz der Relation B
 - Kombination der Datensätze zu neuem Datensatz, falls Vergleichsbedingung erfüllt
- ☞ Formulierung der Vergleichsbedingungen mit =, <, <=, >, >=, <>
- ☞ Gleichheitsverbund oder **Equi-Join**: Verknüpfung mit „=“-Operator
- ☞ natürlicher Verbund oder **Natural Join**:
Gleichheitsverbund über gleichnamige Attribute
 - in Ergebnistabelle wird eine der beiden identischen Spalten gestrichen
- ☞ mehrere Verknüpfungsbedingungen mit AND und OR kombinierbar, auch Kombination mit Selektionsbedingungen möglich

2.4.2. Neue Formulierungsformen für Verknüpfungsbedingungen

- ☞ Verknüpfungsbedingungen auch im FROM-Teil formulierbar (statt im WHERE-Teil)

Kartesisches Produkt	FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>] CROSS JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] [AS <Ergebnistabellen-Name 1>] [CROSS JOIN <Tabelle 3> [[AS] <Tab.-Name 3 neu>] [AS <Ergebnistabellen-Name 2>] CROSS JOIN ...]
Verbund mit Bedingungen	FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>] [INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] ON <Verknüpfungsbedingungen 1> [AS <Ergebnistabellen-Name 1>] [[INNER] JOIN <Tabelle 3> [[AS] <Tab.-Name 3 neu>] ON <Verknüpfungsbedingungen 2> [AS <Ergebnistabellen-Name 2>] [INNER] JOIN ...]

- Verknüpfung Tabelle 1 mit Tabelle 2, Ergebnisrelation mit Tabelle 3,
- **INNER** bezeichnet den Standardfall eines Verbunds: Verknüpfungsbedingungen sind zu erfüllen
 - daneben gibt es noch den **OUTER JOIN** (s.u.)

➤ Formulierung natürlicher Verbund:

Natürlicher Verbund	<ul style="list-style-type: none"> • FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>] [INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] USING (<Liste Attribut-Namen 1>) [AS <Ergebnistabellen-Name 1>] [[INNER] JOIN <Tabelle 3> [[AS] <Tab.-Name 3 neu>] USING (<Liste Attribut-Namen 2>) [AS <Ergebnistabellen-Name 2>] [INNER] JOIN ...] • FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>] NATURAL [INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] [AS <Ergebnistabellen-Name 1>] [NATURAL [INNER] JOIN <Tabelle 3> [[AS] <Tab.-Name 3 neu>] [AS <Ergebnistabellen-Name 2>] NATURAL [INNER] JOIN ...]
----------------------------	--

- im Allgemeinen Verknüpfung über **aufgelistete** gleichnamige Attribute der beteiligten Relationen
- Verknüpfung bei **NATURAL JOIN** über **alle** gleichnamigen Attribute der beteiligten Relationen

➤ Formulierung **äußerer Verbund**

- auch die Datensätze, die die Verknüpfungsbedingungen **nicht erfüllen**, kommen in die Ergebnisrelation
- drei unterschiedliche Fälle (jeweils entweder als normaler oder als natürlicher äußerer Verbund):

[NATURAL] <LEFT | RIGHT | FULL> [OUTER] JOIN

- **[natural] left [outer] join:**
 alle Datensätze aus dem inneren Verbund und jeder Datensatz der zuerst genannten (linken) Relation, der sich beim inneren Verbund nicht qualifiziert hat (wird jeweils mit Nullwerten für die Attribute der rechten Relation ergänzt)
- **[natural] right [outer] join:**
 alle Datensätze aus dem inneren Verbund und jeder Datensatz der zweiten (rechten) Relation, der sich beim inneren Verbund nicht qualifiziert hat (wird jeweils mit Nullwerten für die Attribute der linken Relation ergänzt)
- **[natural] full [outer] join:**
 alle Datensätze aus dem inneren Verbund und jeder Datensatz der beiden zu verknüpfenden Relationen, der sich beim inneren Verbund nicht qualifiziert hat (wird jeweils mit Nullwerten für die Attribute der anderen Relation ergänzt)

Datenmanipulation (Einfügen, ändern, löschen)

3. Datenmanipulation

3.1. Einfügen von Datensätzen

➤ Einfüge-Anweisung für einzelne oder mehrere Datensätze:

Einfügen einzelner Datensätze	INSERT INTO <Tabellen-Name> [<Attribut-Name 1>, ..., <Attribut-Name n>] VALUES (<Wert-Ausdruck 1> [, ..., <Wert-Ausdruck n>]) [, (<Wert-Ausdruck 2-1> [, ..., <Wert-Ausdruck 2-n>]) , ...]
Einfügen einer Menge von Datensätzen	INSERT INTO <Tabellen-Name> [<Attribut-Name 1>, ..., <Attribut-Name n>] <SELECT-Anweisung>

- pro Anweisung Einfügen in nur eine Relation möglich
- falls keine Attribute angegeben, gilt als Default-Wert die komplette Attributliste der betreffenden Relation
- im VALUES-Teil stehen einzelne einzufügende Datensätze
- angegebene Werte werden in angegebener Reihenfolge den aufgelisteten Attributen zugeordnet
- nicht aufgelistete Attribute erhalten Nullwert oder ggf. Default-Wert
- mit SELECT-Anweisung Menge von einzufügenden Datensätzen spezifizierbar

➤ beim Einfügen muss referentielle Integrität beachtet werden

- ☞ referenzierte Primärschlüsselwerte müssen vorhanden sein

3.2. Ändern von Datensätzen

```
UPDATE <Tabellen-Name>
SET    <Attribut-Name 1> = <Wert-Ausdruck 1>
      [, ..., <Attribut-Name n> = <Wert-Ausdruck n>]
[WHERE  <Selektionsbedingungen>]
```

- pro Anweisung Ändern in nur einer Relation möglich
- falls WHERE-Teil fehlt, werden alle Datensätze der betreffenden Relation geändert, ansonsten werden nur Datensätze geändert, die die Selektionsbedingungen erfüllen
- beim Ändern ist auf die referentielle Integrität zu achten

3.3. Löschen von Datensätzen

```
DELETE FROM <Tabellen-Name>
[WHERE      <Selektionsbedingungen>]
```

- pro Anweisung Löschen in nur **einer** Relation möglich
- falls WHERE-Teil fehlt, werden alle Datensätze der betreffenden Relation gelöscht, ansonsten werden nur Datensätze gelöscht, die die Selektionsbedingungen erfüllen
- beim Löschen ist auf die referentielle Integrität zu achten

Anhang: SQL-Syntax

```
CREATE TABLE <Tabellen-Name>
( <Attribut-Name 1> <Datentyp 1> [<Integritätsbedingungen Attr.1>],
  .....
  <Attribut-Name n> <Datentyp n> [<Integritätsbedingungen Attr.n>],      [<Tabellen-
Integritätsbedingungen>] )
```

Datentypen:

- INT | INTEGER
- SMALLINT
- DEC | DECIMAL (p, [q])
- NUM | NUMERIC (p, [q])
- CHAR (n)
- VARCHAR (n)
- CLOB (m [K|M|G])
- BOOLEAN
- DATE
- TIME
- TIMESTAMP

Integritätsbedingungen (Attribut):

- UNIQUE
- NOT NULL
- DEFAULT <Default-Wert>
- CHECK (<<Attribut-Name> | VALUE> <Prüfbedingung 1>
 [AND|OR ... AND|OR <<Attribut-Name> | VALUE> <Prüfbedingung n>])
- PRIMARY KEY
- REFERENCES <Tabellen-Name> [(<Attribut-Name>)]

Prüfbedingung:

- <Vergleichsoperator> <Wert-Ausdruck>
- [NOT] IN (<Werte-Aufzählung> | <SELECT-Statement>)
- [NOT] BETWEEN <Untergrenze> AND <Obergrenze>
- [NOT] LIKE <Textmuster> [ESCAPE <Sonderzeichen>]

Vergleichsoperator:

= | < | <= | > | >= | <>

Wert-Ausdruck:

<konkreter Wert> | <arithmetischer Ausdruck> | <skalares SELECT-Statement>

Tabellen-Integritätsbedingungen:

- <Attribut-Name> WITH OPTIONS <Integritätsbedingung Attribut>
- UNIQUE (<Liste Attribut-Namen>)
- CHECK (<Selektionsbedingung 1> [AND|OR ... AND|OR <Selektionsbedingung n>])
- PRIMARY KEY (<Liste Attribut-Namen>)
- FOREIGN KEY (<Liste Attribut-Namen>) REFERENCES <Tabellen-Name>
 [(<Liste Attribut-Namen>)]

```
SELECT  [DISTINCT] * | <Attribut 1> [AS <neuer Attr.-Name 1>]
           [ , ..., <Attribut n> [AS <neuer Attr.-Name n>] ]
FROM    <Tabelle 1> [[AS] <neuer Tab.-Name 1>]
           [ , ..., <Tabelle m> [[AS] <neuer Tab.-Name m>] ]
[WHERE   <Verknüpfungs- und Selektionsbedingungen>]
[GROUP BY <Liste Attribut-Namen> [HAVING <Auswahl-Bedingung>] ]
[ORDER BY <Attribut-Name 1> [ASC | DESC] [ , ..., <Attribut-Name p> [ASC | DESC]] ]
```

Attribut:

<Attribut-Name> | <arithmetischer Ausdruck> | <konkreter Wert> | <skalare Subanfrage>

Tabelle:

<Tabellen-Name> | <Subanfrage>

Selektionsbedingungen:

- <<Attribut-Name> | <arithm. Ausdruck>> <Vergleichsoperator> <Wert-Ausdruck>
- <<Attribut-Name> | <arithm. Ausdruck>> [NOT] IN (<Werte-Aufzählung> | <Subanfrage>)
- <<Attribut-Name> | <arithm. Ausdruck>> [NOT] BETWEEN <Untergrenze> AND <Obergrenze>
- <<Attribut-Name> | <arithm. Ausdruck>> [NOT] LIKE <Textmuster> [ESCAPE <Sonderzeichen>] mit möglichen Platzhaltern im Textmuster: % und _
- [NOT] EXISTS (<Subanfrage>)

Aggregationsfunktionen:

- COUNT (*)
- COUNT ([DISTINCT] <Attribut-Name>)
- SUM ([DISTINCT] <Attribut-Name>)
- AVG ([DISTINCT] <Attribut-Name>)
- MIN (<Attribut-Name>)
- MAX (<Attribut-Name>)

Verknüpfungsbedingung:

<Attribut-Name 1> <Vergleichsoperator> <Attribut-Name 2>

FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>]
CROSS JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] [AS <Ergebnisrelation-Name 1>]
[... CROSS JOIN <Tabelle n> [[AS] <Tab.-Name n neu>] [AS <Ergebnisrelation-Name m>]]

FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>]
[INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>]
ON <Verknüpfungsbedingung 1> [AS <Ergebnisrelation-Name 1>]
[... [INNER] JOIN <Tabelle n> [[AS] <Tab.-Name n neu>]
ON <Verknüpfungsbedingung n> [AS <Ergebnisrelation-Name m>]]

- FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>]
[INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>]
USING (<Liste Attribut-Namen 1>) [AS <Ergebnisrelation-Name 1>]
[... [INNER] JOIN <Tabelle n> [[AS] <Tab.-Name n neu>]
USING (<Liste Attribut-Namen m>) [AS <Ergebnisrelation-Name m>]]
- FROM <Tabelle 1> [[AS] <Tab.-Name 1 neu>]
NATURAL [INNER] JOIN <Tabelle 2> [[AS] <Tab.-Name 2 neu>] [AS <Erg.-rel.-Name 1>]
[... NATURAL [INNER] JOIN <Tabelle n> [[AS] <Tab.-Name n neu>] [AS <Erg.-rel.-Name m>]]

[NATURAL] <LEFT | RIGHT | FULL> [OUTER] JOIN

INSERT INTO <Tabellen-Name> [(<Attribut-Name 1>, ..., <Attribut-Name n>)]
VALUES (<Wert-Ausdruck 1-1> [, ..., <Wert-Ausdruck 1-n>])
[, (<Wert-Ausdruck 2-1> [, ..., <Wert-Ausdruck 2-n>]), ...]

INSERT INTO <Tabellen-Name> [(<Attribut-Name 1>, ..., <Attribut-Name n>)] <SELECT-Statement>

```
UPDATE <Tabellen-Name>
SET   <Attribut-Name 1> = <Wert-Ausdruck 1>
      [, ..., <Attribut-Name n> = <Wert-Ausdruck n>]
[WHERE <Selektionsbedingungen>]
```

```
DELETE   FROM   <Tabellen-Name>
[WHERE <Selektionsbedingungen>]
```