



Vorlesung
Datenbanksysteme I
Wintersemester 2014/2015

Kapitel 2:
Das Relationale Modell

Vorlesung: PD Dr. Arthur Zimek
Übungen: Sebastian Goebel, Jan Stutzki

Skript © Christian Böhm
ergänzt von Arthur Zimek (2014)

http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_I



Charakteristika

- Einführungskapitel:
Viele Informationen darstellbar als Tabelle
- Die Tabelle (Relation) ist das ausschließliche Strukturierungsmittel des relationalen Datenmodells
- Edgar F. Codd, 1970.
A relational model of data for large shared data banks. Comm. of the ACM 13.06.1970
- Grundlage vieler kommerzieller und freier DBS:

ORACLE®

 DB2 / Informix

Microsoft
SQL Server


MySQL®

 PostgreSQL



Mathematische Grundlagen

Mengen:

- Eine **Menge** M ist eine Zusammenfassung von verschiedenen Objekten, den **Elementen** der Menge.
- Notation: $a \in M$
- Bedeutung: a ist Element der Menge M .
- Eine Menge kann beliebig viele Elemente enthalten, also z.B. auch gar keine (leere Menge, \emptyset , $\{\}$)



Mathematische Grundlagen

Mengen:

- Eine Menge kann **extensional** oder **intensional** definiert werden.
 - extensional: Aufzählung der Elemente (die Reihenfolge der Elemente spielt keine Rolle)
 - intensional: Charakterisierung der Menge, z.B. durch Angabe einer Bedingung, die alle Elemente und nur die Elemente der Menge erfüllen



Mathematische Grundlagen

Mengen:

- Alle Elemente einer Menge sind verschieden.
- extensionale Definition $\{1, 2, 2, 3\}$ ist redundant
- gleiche Menge: $\{1, 2, 3\}$
- Das Konzept „Menge“ kann nicht mehrfaches Vorkommen des gleichen Wertes (Duplikat) modellieren.
 - erweitertes Konzept: Multimenge
- Mengen sind nicht für die Modellierung geeignet, wenn die Reihenfolge der Elemente eine Rolle spielt.



Mathematische Grundlagen

Eigenschaften von Mengen:

Bezeichnung	Notation	Bedeutung
M ist Teilmenge von N	$M \subseteq N$	aus $a \in M$ folgt $a \in N$
M ist echte Teilmenge von N	$M \subset N$	$M \subseteq N$ und $M \neq N$
Vereinigung von M und N	$M \cup N$	$\{x x \in M \text{ oder } x \in N\}$
Schnittmenge von M und N	$M \cap N$	$\{x x \in M \text{ und } x \in N\}$
Differenz M ohne N	$M \setminus N$ oder $M - N$	$\{x x \in M \text{ und } x \notin N\}$
M und N sind disjunkt	$M \cap N = \emptyset$	M und N haben keine gemeinsamen Elemente
Kardinalität einer Menge M	$ M $	Anzahl der Elemente von M



Mathematische Grundlagen

Domäne (Domain):

- Ein Wertebereich (oder Typ)
- Logisch zusammengehörige Menge von Werten
- Beispiele:
 - $D_1 = \text{Integer}$
 - $D_2 = \text{String}$
 - $D_3 = \text{Date}$
 - $D_4 = \{\text{rot, gelb, grün, blau}\}$
 - $D_5 = \{1, 2, 3\}$
- Kann *endliche* oder *unendliche* Kardinalität $|\dots|$ haben:
 - $|D_4| = 4$; $|D_5| = 3$;
 - $|D_1| = \text{unendlich}$; ebenso $|D_2|$ und $|D_3|$.



Mathematische Grundlagen

Tupel, kartesisches Produkt (auch: Kreuzprodukt):

- Die Elemente einer Menge können selbst zusammengesetzt sein aus verschiedenen Mengen.
- Ein geordnetes Paar (Tupel) (x, y) besteht aus zwei Werten x und y , wobei x die erste und y die zweite Komponente ist.
- Das kartesische Produkt $M \times N$ zweier Mengen M und N ist die Menge aller geordneten Paare mit erster Komponente aus M und zweiter Komponente aus N :

$$M \times N := \{(x, y) | x \in M \text{ und } y \in N\}$$



Mathematische Grundlagen

***n*-Tupel, kartesisches Produkt von *n* Mengen:**

- Die Konzepte „Tupel“ und „kartesisches Produkt zweier Mengen“ lassen sich leicht auf eine beliebige Anzahl *n* von Mengen verallgemeinern
- Das allgemeine kartesische Produkt über *n* Mengen ist die **Menge aller geordneten *n*-Tupel mit den Komponenten aus diesen Mengen:**

$$M_1 \times M_2 \times \cdots \times M_n := \{(a_1, a_2, \dots, a_n) \mid a_1 \in M_1, a_2 \in M_2, \dots, a_n \in M_n\}$$

- Sind alle Mengen identisch ($M_i = M_j$ für alle $1 \leq i, j \leq n$), schreibt man für $M_1 \times M_2 \times \cdots \times M_n$ auch M^n .



Mathematische Grundlagen

kartesisches Produkt, Beispiele:

- ($k = 2$):

$$D_1 = \{1, 2, 3\}, D_2 = \{a, b\}$$

$$D_1 \times D_2 = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

- ($k = 3$):

$$D_1 = D_2 = D_3 = \mathbb{N}$$

$$D_1 \times D_2 \times D_3 = \mathbb{N}^3 = \{(1, 1, 1), (1, 1, 2), (1, 1, 3), \dots, (1, 2, 1), \dots\}$$



Mathematische Grundlagen

Potenzmenge

- Viele Objekte werden durch Mengen beschrieben.
- Der Wertebereich einer Datenmenge solcher Objekte ist dann eine Menge, die Mengen enthält.
- Eine spezielle Form solcher Mengen ist die **Potenzmenge**.
- Die Potenzmenge einer Grundmenge U ist die Menge aller Teilmengen von U , geschrieben $\wp(U)$.

– Beispiel:

$$U = \{d, f, s\}$$

$$\wp(U) = \{\emptyset, \{d\}, \{f\}, \{s\}, \{d, f\}, \{d, s\}, \{f, s\}, \{d, f, s\}\}$$

- Für eine Menge der Kardinalität n – welche Kardinalität hat ihre Potenzmenge?



Mathematische Grundlagen

Relation

- Eine n -stellige **Relation** ist eine Menge von n -Tupeln.
- Prädikate können als Charakterisierung von Relationen aufgefasst werden
 - Beispiel: $<$
 - $< \subseteq \mathbb{N} \times \mathbb{N}$
 - $(1, 2) \in <$
 - $(2, 1) \notin <$
- Eine Relation R ist erfüllt (wahr) für alle Tupel und nur für die Tupel a mit $a \in R$
 - Schreibweise: Ra
 - Für zweistellige Relationen schreibt man auch xRy (z.B., $x < y$)
- Sei $R \subseteq M \times N$. Ist $M = N$, so heißt R homogen.



Mathematische Grundlagen

Eigenschaften zweistelliger homogener Relationen

Sei $R \subseteq M \times M$ (d.h., $R \in \wp(M \times M)$)

- R ist **reflexiv**, wenn für alle $x \in M$ gilt:

$$xRx$$

- R ist **symmetrisch**, wenn für alle $x, y \in M$ gilt:

$$xRy \Rightarrow yRx$$

- R ist **antisymmetrisch**, wenn für alle $x, y \in M$ gilt:

$$xRy \text{ und } yRx \Rightarrow x = y$$

- R ist **transitiv**, wenn für alle $x, y, z \in M$ gilt:

$$xRy \text{ und } yRz \Rightarrow xRz$$

- R ist **alternativ**, wenn für alle $x, y \in M$ gilt:

$$xRy \text{ oder } yRx$$



Mathematische Grundlagen

Ordnungsrelation

Bestimmte Kombinationen von Eigenschaften qualifizieren eine homogene Relation zur **Ordnungsrelation**

Sei $R \in \wp(M \times M)$

- R ist eine **partielle Ordnung**, wenn R reflexiv, antisymmetrisch und transitiv ist.
- R ist eine **totale Ordnung**, wenn R eine alternative partielle Ordnung ist.
 - Beispiel: Ordnungsrelation für ganze Zahlen:
 $\leq \in \wp(\mathbb{Z} \times \mathbb{Z})$
 - Ist diese Ordnungsrelation total?



Mathematische Grundlagen

Funktion

- Eine **Funktion** ist eine Abbildung von einer Menge D auf eine Menge B
- Funktionen können wir als Relationen mit speziellen Eigenschaften auffassen: als **rechtseindeutige** Beziehung zwischen D und B
- Formal:

Eine Funktion f ist eine 2-stellige Relation $f \subseteq D \times B$ mit:

$$(x, y) \in f \text{ und } (x, z) \in f \Rightarrow y = z$$

(d.h., einem Element aus D ist höchstens ein Element aus B eindeutig zugeordnet)



Mathematische Grundlagen

Funktion

- Die Menge D heißt Definitionsbereich von f
- Die Menge B heißt Bildbereich von f
- Schreibweisen mit $(x, y) \in D \times B$:

$$(x, y) \in f$$

$$y = f(x)$$

$$f(x) = y$$

$$f: x \mapsto y$$

Signatur: $f: D \rightarrow B$



Relation in der Datenbank

- Es gibt endliche und unendliche Relationen.
- In Datenbanksystemen: Nur endliche Relationen
Unendlich: Nicht darstellbar.
- Die einzelnen Domänen lassen sich als **Spalten einer Tabelle** verstehen und werden als **Attribute** bezeichnet
- Die Elemente der Relation heißen Tupel:
(1,a), (2,a), (3,a) sind drei Tupel vom Grad $k = 2$
- Relation ist eine **Menge** von Tupeln, d.h. die Reihenfolge der Tupel **spielt keine Rolle**: $\{(0,a), (1,b)\} = \{(1,b), (0,a)\}$
- Die Reihenfolge der Attribute **ist** von Bedeutung:
 $\{(a,0), (b,1)\} \neq \{(0,a), (1,b)\}$



Relationen-Schema

Alternative Definition in DBS:

Relation ist Ausprägung eines **Relationen-Schemas**.

- Geordnetes Relationenschema:
 - k -Tupel aus Domänen (Attribute)
 - Attribute werden anhand ihrer **Position** im Tupel referenziert
 - Attribute können zusätzlich einen Attributnamen haben

$$R = (A_1: D_1, \dots, A_k: D_k)$$

- Domänen-Abbildung (ungeordnetes Relationen-Schema):
 - Relationenschema R ist **Menge** von Attributnamen:
 - Jedem Attributnamen A_i ist Domäne D_i zugeordnet:
 - Attribute werden anhand ihres **Namens** referenziert

$$R = \{A_1, \dots, A_k\} \text{ mit } \text{dom}(A_i) = D_i, 1 \leq i \leq k$$



Relationen-Schema

- Beispiel: Städte-Relation

Städte	Name	Einwohner	Land
	München	1.211.617	Bayern
	Bremen	535.058	Bremen
	Passau	49.800	Bayern

- Als geordnetes Relationenschema:

Schema: $R = (\text{Name: String, Einwohner: Integer, Land: String})$

Ausprägung: $r = \{(\text{München}, 1.211.617, \text{Bayern}), (\text{Bremen}, 535.058, \text{Bremen}), (\text{Passau}, 49.800, \text{Bayern})\}$

- Als Relationenschema mit Domänenabbildung:

Schema: $R = \{\text{Name, Einwohner, Land}\}$

mit $\text{dom}(\text{Name}) = \text{String}$, $\text{dom}(\text{Einwohner}) = \text{Integer}$, ...

Ausprägung: $r = \{t_1, t_2, t_3\}$

mit $t_1(\text{Name}) = \text{München}$, $t_1(\text{Einwohner}) = 1.211.617, \dots$



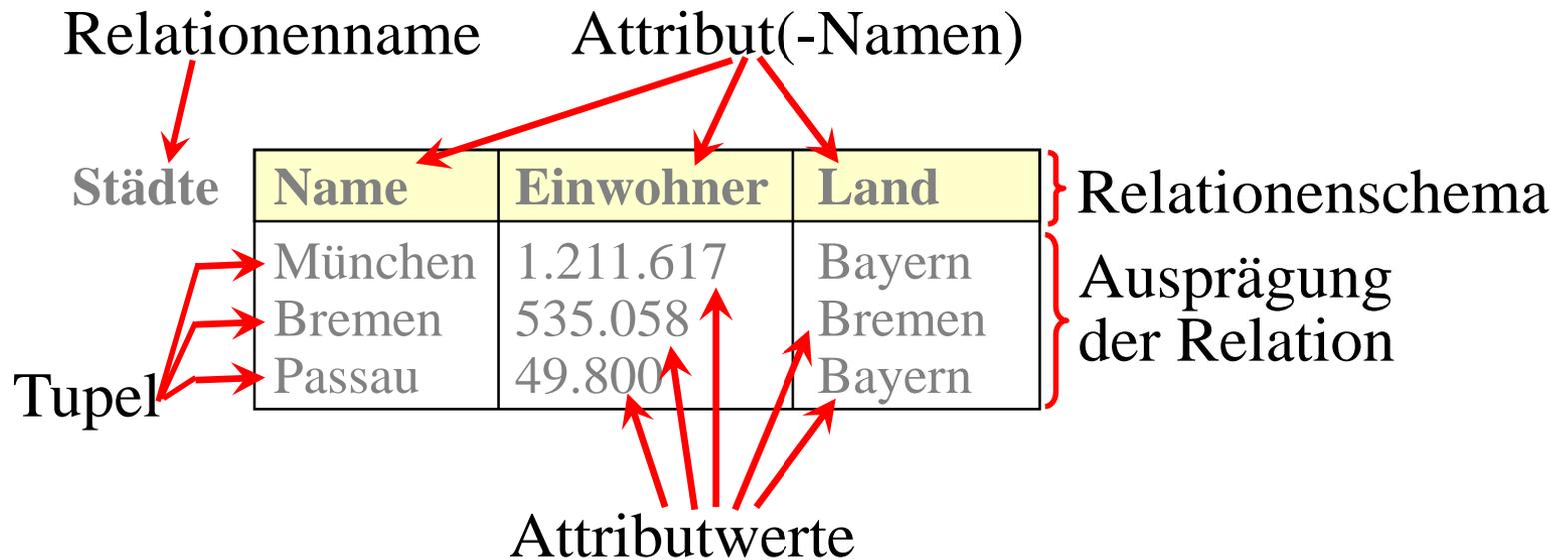
Diskussion

- Vorteil von geordnetem Relationenschema:
 - Prägnanter aufzuschreiben.
Wichtig z.B. beim Einfügen neuer Tupel:
 $t_3 = (\text{Passau}, 49.800, \text{Bayern})$
vergleiche: $t_3(\text{Name}) = \text{Passau}$; $t_3(\text{Einwohner}) = \dots$
- Nachteil von geordnetem Relationenschema:
 - Einschränkungen bei logischer Datenunabhängigkeit:
Applikationen sensibel bzgl. Einfügung neuer Attribute (nur am Ende!)
- Definitionen prinzipiell gleichwertig
- Wir verwenden beide Ansätze



Begriffe

- Relation: Ausprägung eines Relationenschemas
- Datenbankschema: Menge von Relationenschemata
- Datenbank: Menge von Relationen (Ausprägungen)





Duplikate

- Relationen sind Mengen von Tupeln.
Konsequenzen:
 - Reihenfolge der Tupel irrelevant (wie bei math. Def)
 - Es gibt keine Duplikate (gleiche Tupel) in Relationen:
 $\{(0,a), (0,a), (0,a), (1,b)\} = \{(0,a), (1,b)\}$
- Frage: Gilt dies auch für die Spalten beim ungeordneten Relationenschema $R = \{A_1, \dots, A_k\}$?
 - Reihenfolge der Spalten ist **irrelevant**
(das ist gerade das besondere am ungeordneten RS)
 - Duplikate **treten nicht auf, weil alle Attribut-Namen verschieden sein müssen**



Schlüssel

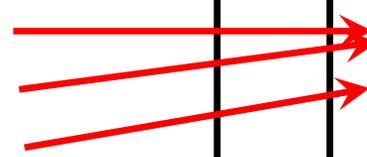
- Tupel müssen eindeutig identifiziert werden
- Warum? Z.B. für Verweise:

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	
002	Mayer	Hugo	
003	Müller	Anton	

Abteilungen

ANr	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing



- Objektidentifikation in Java:
Mit Referenz (Adresse im Speicher)
- Im relationalen Modell werden Tupel anhand von Attributwerten identifiziert
- Ein/mehrere Attribute als **Schlüssel** kennzeichnen
- Konvention: Schlüsselattribut(e) unterstreichen!



Schlüssel

Beispiel: PNr und ANr werden Primärschlüssel:

Mitarbeiter

<u>PNr</u>	Name	Vorname	Abteilung
001	Huber	Erwin	
002	Mayer	Hugo	
003	Müller	Anton	

Abteilungen

<u>ANr</u>	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing

- Damit müssen diese Attributswerte eindeutig sein!
- Verweis durch Wert dieses Schlüsselattributs:

Mitarbeiter

<u>PNr</u>	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01
003	Müller	Anton	02

Abteilungen

<u>ANr</u>	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing



Zusammengesetzter Schlüssel

- Oft ist ein einzelnes Attribut nicht ausreichend, um die Tupel eindeutig zu identifizieren
- Beispiel:

Lehrveranstaltung

<u>VNr</u>	Titel	<u>Semester</u>
012	Einführung in die Informatik	WS 2001/02
012	Einführung in die Informatik	WS 2002/03
013	Medizinische Informationssyst.	WS 2001/02
...

- Schlüssel: (VNr, Semester)
- Anmerkung: Warum ist dies ein schlechtes DB-Design?
Nicht redundanzfrei:
Der Titel ist mehrfach in der Datenbank gespeichert.
→ hierzu mehr in Kapitel 5+6



Schlüssel: Formale Definition

Definition:

- Eine Teilmenge S der Attribute eines Relationenschemas R ($S \subseteq R$) heißt **Schlüssel**, wenn gilt:

1) Eindeutigkeit

Keine Ausprägung von R kann zwei verschiedene Tupel enthalten, die sich in **allen** Attributen von S gleichen (d.h., es gibt eine Funktion $S \rightarrow R$).

2) Minimalität

Es existiert keine **echte** Teilmenge $T \subset S$, die bereits die Bedingung der Eindeutigkeit erfüllt (d.h., es gibt **keine** Funktion $T \rightarrow R$ mit $T \subset S$).



Schlüssel: Formale Definition

Manche Lehrbücher definieren in noch formalerer Notation:

1) Eindeutigkeit:

\forall möglichen Ausprägungen r und Tupel $t_1, t_2 \in r$ gilt:

$$t_1 \neq t_2 \Rightarrow t_1[S] \neq t_2[S].$$

2) Minimalität:

\forall Attributmengen T , die (1) erfüllen, gilt:

$$T \subseteq S \Rightarrow T = S.$$

Hierbei bezeichne $t[S]$ ein Tupel t eingeschränkt auf die Attribute aus S (alle anderen Attribute gestrichen).

Wir schreiben später auch $\pi_S(t)$ für $t[S]$ (*Projektion*, s. Kap. 3)



Superschlüssel / Minimale Menge

- Eine Menge $S \subseteq R$ heißt Superschlüssel (oder Oberschlüssel, engl. Superkey), wenn sie die Eindeutigkeitseigenschaft erfüllt
- Der Begriff des Superschlüssels impliziert keine Aussage über die Minimalität
- In der Mathematik wird allgemein eine Menge M als **minimale Menge bezüglich einer Eigenschaft B** bezeichnet, wenn es keine echte Teilmenge von M gibt, die ebenfalls B erfüllt.
- Damit können wir auch definieren:
Ein Schlüssel ist ein minimaler Superschlüssel
(minimale Menge $S \subseteq R$ mit Eindeutigkeits-Eigenschaft)



Schlüssel: Beispiele

- Gegeben sei die folgende Relation:

Lehrveranst.	LNr	VNr	Titel	Semester
$(t_1=)$	1	012	Einführung in die Informatik	WS 2001/02
$(t_2=)$	2	012	Einführung in die Informatik	WS 2002/03
$(t_3=)$	3	013	Medizinische Informationssysteme	WS 2001/02
...

- $\{VNr\}$ ist kein Schlüssel
Nicht eindeutig: $t_1 \neq t_2$ aber $t_1[VNr] = t_2[VNr] = 012$
- $\{Titel\}$ ist kein Schlüssel
(gleiche Begründung)
- $\{Semester\}$ ist kein Schlüssel
Nicht eindeutig: $t_1 \neq t_3$ aber $t_1[Semester] = t_3[Semester]$



Schlüssel: Beispiele

Lehrveranst.	LNr	VNr	Titel	Semester
$(t_1=)$	1	012	Einführung in die Informatik	WS 2001/02
$(t_2=)$	2	012	Einführung in die Informatik	WS 2002/03
$(t_3=)$	3	013	Medizinische Informationssyst.	WS 2001/02

- $\{\text{LNr}\}$ ist Schlüssel !!!
Eindeutigkeit: Alle $t_i[\text{LNr}]$ sind paarweise verschieden,
d.h. $t_1[\text{LNr}] \neq t_2[\text{LNr}]$, $t_1[\text{LNr}] \neq t_3[\text{LNr}]$, $t_2[\text{LNr}] \neq t_3[\text{LNr}]$
Minimalität: Trivial, weil 1 Attribut kürzeste Möglichkeit
- $\{\text{LNr}, \text{VNr}\}$ ist kein Schlüssel (aber Superschlüssel)
Eindeutigkeit: Alle $t_i[\text{LNr}, \text{VNr}]$ paarweise verschieden.
Nicht minimal, da **echte** Teilmenge $\{\text{LNr}\} \subset \{\text{LNr}, \text{VNr}\}$ (\neq) die
Eindeutigkeit bereits gewährleistet, s.o.



Schlüssel: Beispiele

Lehrveranst.	LNr	VNr	Titel	Semester
$(t_1=)$	1	012	Einführung in die Informatik	WS 2001/02
$(t_2=)$	2	012	Einführung in die Informatik	WS 2002/03
$(t_3=)$	3	013	Medizinische Informationssyst.	WS 2001/02

- $\{VNr, Semester\}$ ist **Schlüssel !!!**

Eindeutigkeit: Alle $t_i[VNr, Semester]$ paarw. verschieden:

- $t_1 [VNr, Semester] = (012, WS 2001/02)$
 - $t_2 [VNr, Semester] = (012, WS 2002/03)$
 - $t_3 [VNr, Semester] = (013, WS 2001/02)$
- } \neq
} \neq) \neq

Minimalität:

Weder $\{VNr\}$ noch $\{Semester\}$ gewährleisten Eindeutigkeit (siehe vorher). Dies sind alle echten Teilmengen.



Primärschlüssel

- Minimalität bedeutet **nicht**:
Schlüssel mit den wenigsten Attributen
- Sondern Minimalität bedeutet:
Keine überflüssigen Attribute sind enthalten
(d.h. solche, die zur Eindeutigkeit nichts beitragen)
- Manchmal gibt es mehrere verschiedene Schlüssel
 - {LNR}
 - {VNr, Semester} → **Schlüsselkandidat** (SQL: **unique**)
- Später ist wichtig, *alle* Schlüsselkandidaten zu ermitteln.
- Man wählt einen dieser Kandidaten aus als sogenannten **Primärschlüssel** (SQL: **primary key**)
- Ein Attribut, das auf einen Schlüssel einer anderen Relation verweist, heißt **Fremdschlüssel** (SQL: **foreign key**)



Schlüssel: Semantische Eigenschaft

- Die Eindeutigkeit bezieht sich **nicht** auf die aktuelle Ausprägung einer Relation R
- Sondern immer auf die **Semantik** der realen Welt

Mitarbeiter	PNr	Name	Gehalt
	001	Müller	1700 €
	002	Mayer	2172 €
	003	Huber	3189 €
	004	Schulz	2171 €

- Bei der aktuellen Relation wären sowohl {PNr} als auch {Name} und {Gehalt} eindeutig.
- Aber es ist möglich, dass mehrere Mitarbeiter mit gleichem Namen und/oder Gehalt eingestellt werden
- {PNr} ist **für jede mögliche** Ausprägung eindeutig



Tabellendefinition in SQL

- Definition eines Relationenschemas:

```
CREATE TABLE n  
(  
    a1 d1 c1,  
    a2 d2 c2,  
    ...  
    ak dk ck  
)
```

← *n* Name der Relation

← Definition des ersten Attributs

← Definition des Attributs Nr. *k*

- hierbei bedeuten...
 - *a_i* der Name des Attributs Nr. *i*
 - *d_i* der Typ (die Domain) des Attributs
 - *c_i* ein optionaler Constraint für das Attribut
- Wirkung: Definition eines Relationenschemas mit einer leeren Relation als Ausprägung.



Basis-Typen in SQL

Der SQL-Standard kennt u.a. folgende Datentypen:

- **integer** oder auch **integer4**, **int**
- **smallint** oder **integer2**
- **float** (p) oder auch **float**
- **decimal** (p, q) und **numeric** (p, q)
mit p Stellen, davon q Nachkommast.
- **character** (n), **char** (n) für Strings fester Länge n
- **character varying** (n), **varchar** (n): variable Strings
- **date**, **time**, **timestamp** für Datum und Zeit



Zusätze bei Attributdefinitionen

- Einfache Zusätze (Integritätsbedingungen) können unmittelbar hinter einer Attributdefinition stehen:
 - **not null**: Das Attribut darf nicht undefiniert sein
*in DBS: undefinierte Werte heissen **null**-Werte*
 - **primary key**: Das Attribut ist Primärschlüssel
(nicht bei zusammengesetzten Schlüsseln)
 - **unique**:
Das Attribut ist Schlüsselkandidat
 - **references** $t_1(a_1)$:
Ein Verweis auf Attribut a_1 von Tabelle t_1
 - **default** w_1 : Wert w_1 ist Default, wenn unbesetzt.
 - **check** f :
Die Formel f wird bei jeder Einfügung überprüft, z.B.:
check $A \leq 100$

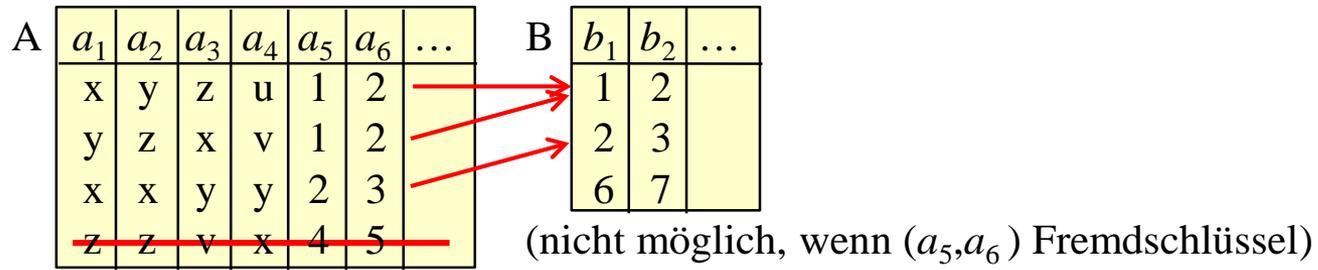


Integritätsbedingungen

- Zusätze, die keinem einzelnen Attribut zugeordnet sind, stehen mit Komma abgetrennt in extra Zeilen
 - **primary key** (A_1, A_2, \dots) :
Zusammengesetzter Primärschlüssel
 - **unique** (A_1, A_2, \dots) :
Zusammengesetzter Schlüsselkandidat
 - **foreign key** (A_1, A_2, \dots) **references** $t_1 (B_1, B_2, \dots)$
Verweis auf zusammengesetzten Schlüssel in Rel. T_1
Anmerkung: Fehlt die Angabe (B_1, B_2, \dots) hinter t_1 so wird automatisch (A_1, A_2, \dots) eingesetzt.
 - **check** f
- Anmerkung: SQL ist case-insensitiv:
Im Ggs. zu Java hat die Groß-/Kleinschreibung weder bei Schlüsselworten noch bei Bezeichnern Bedeutung



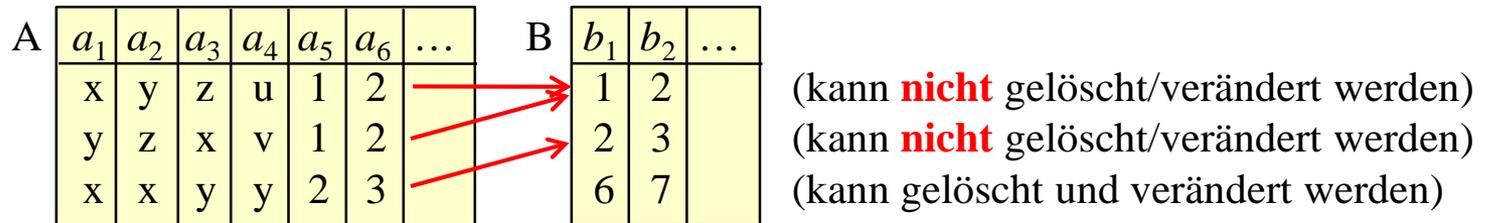
Schlüssel-Definitionen



- **primary key** (a_1, a_2) ,
definiert den Primärschlüssel.
- **unique** (a_3, a_4) ,
definiert einen weiteren Schlüsselkandidaten.
- **foreign key** (a_5, a_6) **references** B (b_1, b_2)
definiert einen Fremdschlüssel.
 - Tupel in A ohne gültigen Partner in B nicht erlaubt
 - Ohne weiteren Zusatz nicht möglich, Tupel in B, auf die durch Tupel in A verwiesen wird, zu löschen oder die Werte von b_1, b_2 zu verändern.



Schlüssel-Definitionen



Löschen eines Tupels in B mit Referenzen nicht möglich.

Es gibt aber verschiedene Zusätze:

- **foreign key (a_5, a_6) references B (b_1, b_2)
on delete cascade**

Löschen eines Tupels in B führt auch zum Löschen der entsprechenden Tupel in A

- **on update cascade**

Verändern eines Tupels in B führt zum Verändern in A

- **on delete set null**

„hängende Verweise“ werden ggf. auf **null** gesetzt.



Beispiel Tabellendefinition

- Zusammengesetzter Primärschlüssel {VNr, Semester}:

```
create table Lehrveranst  
(  
  LNr          integer          not null,  
  VNr          integer          not null,  
  Titel        varchar(50),  
  Semester     varchar(20)     not null,  
  primary key (VNr, Semester)  
)
```

- Alternative mit einfachem Primärschlüssel {LNr}:

```
create table Lehrveranst2  
(  
  LNr          integer          primary key,  
  VNr          integer          not null,  
  Titel        varchar(50),  
  Semester     varchar(20)     not null  
)
```



Beispiel Tabellendefinition

- Tabelle für Dozenten:

```
create table Dozenten  
(  
  DNr      integer      primary key,  
  Name     varchar(50),  
  Geburt   date  
)
```

- Verwendung von Fremdschlüsseln:

```
create table Haelt  
(  
  Dozent   integer      references Dozenten (DNr)  
                        on delete cascade,  
  VNr      integer      not null,  
  Semester varchar(20) not null,  
  primary key (Dozent, VNr, Semester),  
  foreign key (VNr, Semester) references Lehrveranst  
)
```



Beispiel Tabellendefinition

- Das Schlüsselwort **on delete cascade** in *Haelt* führt dazu, dass bei Löschen eines *Dozenten* auch entsprechende Tupel in *Haelt* gelöscht werden
- Weitere Konstrukte der Data Definition Language:
 - **drop table** n_1
Relationen-Schema n_1 wird mit allen evtl. vorhandenen Tupeln gelöscht.
 - **alter table** n_1 **add** $(a_1 d_1 c_1, a_2 d_2 c_2, \dots)$
 - Zusätzliche Attribute oder Integritätsbedingungen werden (rechts) an die Tabelle angehängt
 - Bei allen vorhandenen Tupeln Null-Werte
 - **alter table** n_1 **drop** (a_1, a_2, \dots)
 - **alter table** n_1 **modify** $(a_1 d_1 c_1, a_2 d_2 c_2, \dots)$