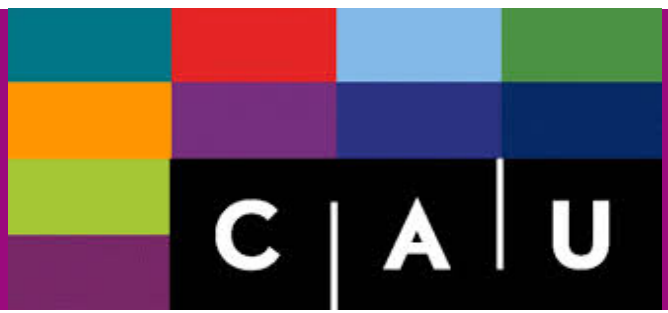
Informationssyteme[1]

Kapitel 1: Das Relationale Datenmodell

Wintersemester 2020/21

Prof. Dr. Peer Kröger
Institut für Informatik
Arbeitsgruppe Informationssysteme und Data Mining



[1] Die folgenden Folien basieren in großen Teilen auf Material zur Vorlesung "Datenbanksysteme", die ich in meiner Zeit an der LMU München mehrmals gehalten habe. Das Material ist von den damaligen Kollegen maßgeblich (mit-)gestaltet worden, insbes. von Prof. Dr. C. Böhm.

Übersicht

1. Grundlagen
2. Relationen
3. Schlüssel
4. SQL: Data Definition Language

Agenda

1. Grundlagen
2. Relationen
3. Schlüssel
4. SQL: Data Definition Language

Charakteristika

Das relationale Modell ...

- ... basiert auf dem Konzept der Tabelle (Relation) als das ausschließliche Mittel zur Sturkturierung von Daten
- ... wurde von Edgar F. Codd 1970 vorgestellt
- ... ist die Grundlage vieler kommerzieller und freier DBS:

ORACLE

IBM DB2 / Informix

Microsoft SQL Server

MySQL

PostgreSQL

Idee

- Eine Tabelle speichert Informationen zu bestimmten Objekten (der realen Welt)
- Objekte haben typw. Eigenschaften (Attribute), dies sind die Spalten der Tabelle
- Die Attribute haben (ggfls. unterschiedliche) Typen wie Zahlen, Zeichen(-ketten), ...
- Wir starten mit einer mathematischen Formalisierung dieser Konzepte
- Beispiel: Tabelle von Studenten:

Name	**Semester**	**Hauptfach**	**Spezialfach**
Ann	3	Informatik	Java
Bob	1	Informatik	PHP
Charly	4	Kunst	Piano
Debra	2	Sport	Schwimmen

Domain

- Zunächst brauchen wir eine Menge von Datentypen (in unserem Kontext "Domains" genannt), die die Eigenschaften der Attribute kennzeichnen

Domain

Eine Domain ist eine Menge semantisch zusammengehörigen Werten (d.h. ein Datentyp).

Beispiele für Domains:

- Integer
- String
- {red, green, blue, yellow, black, white}

- In unserem Kontext sind Domains endlich

Kartesisches Produkt

- Die mathematische Formalisierung einer Tabelle basiert auf dem Kartesischen Produkt (Kreuzprodukt) von Domains

Kartesisches Produkt

Das Kartesisches Produkt $D_1 \times \ldots \times D_k$ von k Mengen $D_1, \ldots, D_k$ ist die Menge aller möglichen Kombinationen der Elemente der k Mengen.

- Bemerkung; die Mengen $D_1, \ldots, D_k$ sind in unserem Kontext die (ggfls. unterschiedlichen) Domains

Kartesisches Produkt

Beispiele ($k = 2$):

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

Beispiele ($k = 3$):

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

Agenda

1. Grundlagen
2. Relationen
3. Schlüssel
4. SQL: Data Definition Language

Relation

Relation

Eine (k-stellige) Relation R ist Teilmenge des kartesischen Produktes von k Domains $D_1, ..., D_k$:

$$R \subseteq D_1 \times \ldots \times D_k.$$

Beispiele ($k = 2$):

Mit D_1 und D_2 von oben:

- $R_1 = \{(1, a), (2, b)\}$
- $R_2 = \{\}$ (empty set)
- $R_3 = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\} = D_1 \times D_2$

Relation

- Wir kennen das Konzept der Relationen natürlich in erster Linie aus der Mathematik, z.B. ist die Relation $R_{\leq} \subseteq \mathbb{N} \times \mathbb{N}$ mit

$$\begin{aligned} R_{\leq} \quad = \quad & \{(1,1),(1,2),(1,3),... \\ & (2,2),(2,3),(2,4),... \\ & (3,3),(3,4),(3,5),...\} \end{aligned}$$

  auch als "Kleiner-Gleich"-Relation bekannt (und wir benutzen üblicherweise als Abkürzung das Zeichen "$\leq$" für $R_{\leq}$)
- Es gibt endliche und unendliche Relationen (letzteres ist nur möglich, wenn mindestens eine Domain unendlich ist — und in unserem Kontext nicht relevant)
- Die Anzahl der Tupel $|R|$ einer Relation R heißt Kardinalität von R

Relation und Tabelle

- Wie bereits erwähnt ist die Relation die mathematische Formalisierung einer Tabelle
- Jede Domain D_i ist eine Spalte in der Tabelle, auch Attribut genannt
- Die Zeilen der Tabelle sind die Elemente der Relation und heißen Tupel
- Achtung: eine Relation ist eine Teilmenge aus einem entsprechenden Kreuzprodukt, d.h.
  - die Reihenfolge der Tupel (Zeilen) ist nicht relevant
  - Duplikate (von Tupeln/Zeilen) sind nicht erlaubt
  - die Reihenfolge der Attribute ist sehr wohl von Bedeutung, (z.B. sind die Tupel $(1, a)$ und $(a, 1)$ unterschiedlich, da von unterschiedlichen Kreuzprodukten)

Relationen-Schema

- Basierend auf dieser Formalisierung wird eine Tabelle (Relation) in DBS als Ausprägung eines Relationen-Schemas definiert
- Grundsätzlich definieren die k Domains der Attribute das Schema der Relation
- In der Literatur gibt es zwei Alternative formale Definitionen des Begriffs Relationen-Schema (siehe nächste Folie)

Relationen-Schema

Alternative 1:

Geordnetes Relationen-Schema

Ein geordnetes Relationen-Schema $R = (A_1 : D_1, \ldots, A_k : D_k)$ ist ein k-Tupel aus Domains D_i mit zugeordneten Attributnamen A_i. Attribute werden anhand ihrer Position im Tupel referenziert.

Alternative 2:

Ungeordnetes Relationen-Schema (Domänen-Abbildung)

Ein ungeordnetes Relationen-Schema R ist eine Menge von k Attributnamen A_i und zugeordneter Domäne $D_i = dom(A_i)$. Attribute werden anhand ihrers Namens referenziert:

Ausprägung eines Relationen-Schemas

- Eine konkrete Tabelle ist eine Ausprägung des Relationen-Schemas der Tabelle
- Beispiel: Tabelle

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

- Schema $R = (\text{Name} : \text{String}, \text{Einwohner} : \text{Integer}, \text{Land} : \text{String})$
- Ausprägung $r = \{(\text{München}, 1.211.617, \text{Bayern}), (\text{Bremen}, 535.058, \text{Bremen}), (\text{Passau}, 49.800, \text{Bayern})\}$

Relationen-Schema

Beispiel: Relationen-Schema

- Relation:

Städte

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

- Als geordnetes Relationen-Schema:
  - Schema: $R =$(Name: String, Einwohner: Integer, Land: String)
  - Ausprägung: $r = \{$(München, 1.211.617, Bayern), (Bremen, 535.058,Bremen), (Passau, 49.800, Bayern)$\}$
- Als ungeordnetes Relationen-Schema mit Domänen-Abb.:
  - Schema: $R = \{$Name, Einwohner, Land$\}$ mit dom(Name) = String, dom(Einwohner) = Integer, ...
  - Ausprägung: $r = \{t_1, t_2, t_3\}$ mit t_1(Name) = München, t_1(Einwohner) = 1.211.617, ...

Relationen-Schema: Diskussion

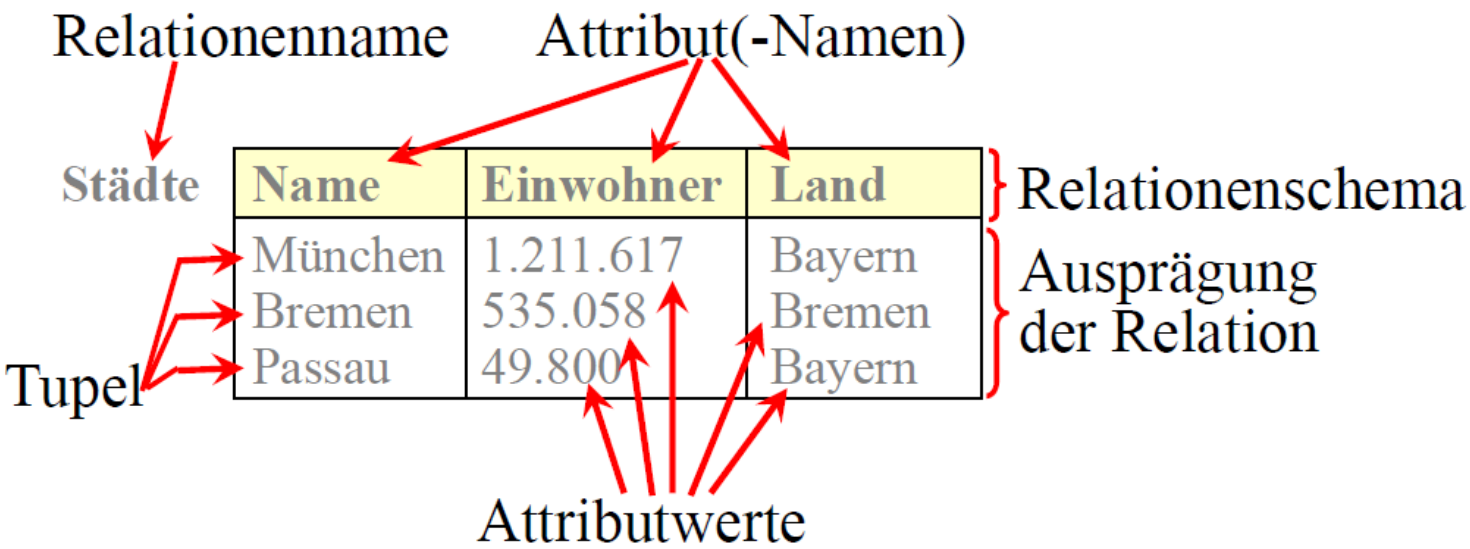
- Vorteil von geordnetem Relationen-Schema: prägnanter aufzuschreiben
  Wichtig z.B. beim Einfügen neuer Tupel: $t_3 = (\text{Passau}, 49.800, \text{Bayern})$ statt $t_3(\text{Name}) = \text{Passau}$; $t_3(\text{Einwohner}) = ...$
- Nachteil von geordnetem Relationen-Schema:
  Einschränkungen bei logischer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

Duplikate

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

Zusammenfassung Begriffe

- Relation: Ausprägung eines Relationen-Schemas
- Datenbankschema: Menge von Relationen-Schemata
- Datenbank: Menge von Relationen (Ausprägungen)



Agenda

1. Grundlagen
2. Relationen
3. Schlüssel
4. SQL: Data Definition Language

Schlüssel

- Tupel müssen eindeutig identifiziert werden
- In Programmiersprachen: Objekte werden durch logische/ physische Referenzen auf Speicheradressen identifiziert
- Im relationalen Modell: Tupel werden anhand von Attribut-Werten identifiziert
- Ein/mehrere Attribute als Schlüssel kennzeichnen, die diese Identifizierung gewährleisten
- Konvention: Schlüsselattribut(e) unterstreichen
- Schlüssel können insbesondere auch für Verweise auf Tupel in anderen Tabellen verwendet werden

Schlüssel: Verweis

Beispiel

- Jedem Mitarbeiter aus "Mitarbeiter" soll eine entsprechenden Abteilung in "Abteilungen" zugeordnet werden:

Mitarbeiter

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

Abteilungen

ANr	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing

- Dazu benötigen wir einen Schlüssel der Tabelle "Abteilungen", hier bestehend aus dem Attribut "ANr"
- Verweis durch Wert dieses Schlüsselattributs:

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01
003	Müller	Anton	02

Abteilungen

ANr	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing

Zusammengesetzter Schlüssel

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

Lehrveranstaltung

LNr	**Titel**	**Semester**
012	Einf. in Python	Sommer 2018
013	Machine Learning	Winter 2017
013	Machine Learning	Winter 2018

- Hier werden zwei Attribute benötigt um alle Tupel eindeutig zu identifizieren
- Im Übrigen ist dieses Schema ein schlechtes DB-Design! Warum und wie man es besser macht kommt aber leider erst später

Schlüssel: formale Definition

Im folgenden bezeichnet $t[S]$ ein Tupel t eingeschränkt auf die Attribute aus S (dies entspricht einer "Projektion" auf S, siehe später "Relationale Algebra")

Schlüssel

Eine Teilmenge[2] S der Attribute eines Relationen-Schemas 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.:
   Für alle 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: Es existiert keine echte Teilmenge $T \subset S$ ($T \neq S$), die bereits die Bedingung der Eindeutigkeit erfüllt

[2] Bemerkung: Der Teilmengenbegriff umfasst die Menge selbst ($S \subseteq S$). Für eine echte Teilmenge $T \subset S$ gilt zusätzlich $T \neq S$.

Superschlüssel

- Eine Menge $S \subseteq R$ heißt Superschlüssel (oder Oberschlüssel, engl. Superkey), wenn sie die Eindeutigkeit erfüllt
- Der Begriff des Superschlüssels impliziert keine Aussage über die Minimalität, d.h. ein Schlüssel ist ein minimaler Superschlüssel

Schlüssel: Beispiele

- Gegeben sei 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 Informationssyst.	WS 2001/02
	...	...	...	...

- $\{$VNr$\}$ ist kein Schlüssel
  (nicht eindeutig: $t_1 \neq t_2$ aber $t_1[\text{VNr}] = t_2[\text{VNr}] = 012$)
- $\{$Titel$\}$ ist kein Schlüssel
  (nicht eindeutig: $t_1 \neq t_2$ aber $t_1[\text{Titel}] = t_2[\text{Titel}]$)
- $\{$Semester$\}$ ist kein Schlüssel
  (nicht eindeutig: $t_1 \neq t_3$ aber $t_1[\text{Semester}] = t_3[\text{Semester}]$)
- $\{$LNr$\}$ ist Schlüssel
  (eindeutig: alle $t_i[\text{LNr}]$ sind paarweise verschieden; minimal: trivial, da nur ein Attribut, also echte Teilmenge ist leer)

Schlüssel: Beispiele

- Es geht weiter:

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
	...	...	...	...

- {LNr, VNr} ist kein Schlüssel (aber Superschlüssel)

  (eindeutig: alle t_i[LNr, VNr] sind paarweise verschieden; nicht minimal: {LNr}⊂{LNr, VNr} ist bereits eindeutig)

- {VNr, Semester} ist Schlüssel

  (eindeutig: alle t_i[LNr] sind paarweise 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)

  minimal: alle echte Teilmengen ({VNr} und {Semester}) sind nicht eindeutig (siehe oben))

Schlüssel: Begriffe und Schlüssel-Begriffe

- Minimalität bedeutet nicht: Schlüssel mit den wenigsten Attributen
- Minimalität bedeutet: Keine überflüssigen Attribute sind enthalten (d.h. solche, die zur Eindeutigkeit nichts beitragen)
- Manchmal gibt es mehrere verschiedene Schlüssel (in unserem Beispiel: {LNr} und {VNr, Semester}), diese heißen Schlüsselkandidaten (in SQL mit `unique` gekennzeichnet)
- Später ist es wichtig alle Schlüsselkandidaten einer Relation zu kennen
- Man wählt einen dieser Kandidaten aus als sogenannten Primärschlüssel (SQL: `primary key`)
- Attribute, die auf einen Schlüssel einer anderen Relation verweisen, heißen Fremdschlüssel (SQL: `foreign key`)

Schlüsseleigenschaft

- Die Schlüssel-Eigenschaft, insbesondere die Eindeutigkeit ist eine semantische Eigenschaft, d.h. sie bezieht sich nicht auf die aktuelle Ausprägung der Relation, sondern auf die Semantik der realen Welt (für alle möglichen Ausprägungen!)
- Beispiel:

Mitarbeiter

PersNr	**Name**	**Gehalt**
001	Müller	2450
002	Huber	2030
003	Schmid	4500
004	Meier	2800

  - In dieser aktuellen Ausprägung wären alle Attribute für sich eindeutig (und trivialerweise minimal), aber es ist (vermutlich) möglich, dass mehrere Mitarbeiter mit gleichem Namen und/oder Gehalt eingestellt werden
  - Nur {PersNr} ist für jede mögliche Ausprägung eindeutig

Agenda

1. Grundlagen
2. Relationen
3. Schlüssel
4. SQL: Data Definition Language

SQL: Allgemeines

- Die Structured Query Language (SQL) ist **DIE** Standard-Programmier-Sprache ("Datenbanksprache") für relationale DBS
- SQL wurde urspr. bei IBM von Donald D. Chamberlin und Raymond F. Boyce entwickelt
- SQL ist seit 1986 ANSI Standard und seit 1987 ISO Standard
- Der Standard kontinuierlich weiter entwickelt und um viele Features erweitert (letzter Update in 2016)
- Alle Hersteller bieten eine SQL-Schnittstelle an
- Viele Hersteller erlauben allerdings auch eigene Erweiterungen und (Syntax-)Spezifika; daher kann SQL Code manchmal nicht 1:1 über Hersteller-Grenzen portiert werden

SQL: Komponenten

Wie jede Datenbanksprache besteht SQL aus zwei Teilen:

- Data Definition Language (DDL)
  - Deklarationen zur Beschreibung des Schemas
  - Bei relationalen Datenbanken: Anlegen und Löschen von Tabellen, Integritätsbedingungen usw.
- Data Manipulation Language (DML)
  - Anweisungen zum Arbeiten mit den Daten in der Datenbank (Datenbank-Zustand)
  - Das sind insbes. Konstrukte zum reinen Lesen der DB (Anfragesprache) sowie zum Manipulieren (Einfügen, Ändern, Löschen) des Datenbankzustands

SQL ist case-insensitiv

- Wir schauen uns in diesem Kapitel einige wichtige Befehle der DDL-Komponente von SQL an
- Wir verwenden für Schlüsselwörter der Sprache Großbuchstaben, z.B. `SELECT` für den "select"-Befehl[3]
- Hinweis: SQL ist case-insensitive, d.h. `SELECT` ist äquivalent zu `select` und `SelEcT` und ...
- Wir halten uns an den SQL Standard

[3] Dieser Befehl ist Teil der DML von SQL; wir lernen ihn später noch genauer kennen.

Tabellendefinition in SQL

- In SQL werden Relationen-Schemas mit dem "create table"-Befehl definiert:

```
CREATE TABLE name
(
   attr_1 domain_1 constr_1,
   attr_2 domain_2 constr_2,
   ...
   attr_n domain_n constr_n,
)
```

  dabei gilt ...

  - `attr_i` ist der Name des i-ten Attributs
  - `domain_i` ist der Typ (Domain) des i-ten Attributs
  - `constr_i` ist ein oder mehrere optionale Constraints (Integritätsbedingungen)

- Wirkung: Definition eines Relationenschemas mit einer leeren Relation als Ausprägung

Basis-Typen in SQL

- Wie die anderen Programmiersprachen bietet auch der SQL-Standard einige vordefinierte Datentypen (es können keine eigenen definiert werden!)
  - `INTEGER`, `INTEGER4`, `INT` sowie `SMALLINT`, `INTEGER2`
  - `FLOAT`, `DECIMAL (p,q)`, `NUMERIC (p,q)`, ...
  - `CHARACTER (n)`, `CHAR (n)` (Strings fester Länge)
  - `CHARACTER VARYING (n)` `VARCHAR (n)` (Strings mit variabler aber maximaler Länge)
  - `DATE`, `TIME`, `TIMESTAMP`
- Was fällt auf:
  - keine "komplexen" Datentypen wie Arrays (Strings haben fixe (maximale) Länge), Listen, Records, Klassen, ...
  - dafür immerhin spezielle Datentypen für Datum/Zeit
  - bei allen Datentypen ist klar, wie viel Speicherplatz sie benötigen

Integritätsbedingungen in SQL

- Constraints sind einfache Integritätsbedingungen:
  - `NOT NULL`: Das Attribut darf nicht undefiniert (im DBS-Jargon "null-Werte" bzw. in SQL `null`) sein
  - `PRIMARY KEY`: Das Attribut ist Primärschlüssel (nicht, wenn Primärschlüssel aus mehreren Attributen zusammengesetzt ist)
  - `UNIQUE`: Das Attribut ist Schlüsselkandidat
  - `REFERENCES tab (attr)`: Ein Verweis auf einen Fremdschlüssel: Attribut `attr` in Tabelle `tab`
  - `DEFAULT value`: Wert `value` ist Default, wenn unbesetzt.
  - `CHECK f`: Die Formel `f` wird bei jeder Einfügung/Veränderung überprüft, z.B. `CHECK Kontostand` ≥ 0
- Diese Bedingungen werden vom System automatisch überprüft; eine Operation, die ein solches Constraint verletzt, wird das System zurückweisen

Integritätsbedingungen in SQL

- Constraints (Integritätsbedingungen), die keinem einzelnen Attribut zugeordnet sind, stehen mit Komma abgetrennt in extra Zeilen (unter der Auflistung der Attribute:
  - `PRIMARY KEY (attr_1, attr_2, ...)`: Zusammengesetzter Primärschlüssel
  - `UNIQUE (attr_1, attr_2, ...)`: Zusammengesetzter Schlüsselkandidat
  - `FOREIGN KEY ( A_1, A_2, ...) REFERENCES tab ( B_1, B_2, ...)`: Verweis auf zusammengesetzten Fremdschlüssel in Relation `tab`
    Fehlt die Angabe `(B_1, B_2, ...)`, so wird automatisch `(A_1, A_2, ...)` eingesetzt

Tabellendefinition in SQL

Beispiel: Mitarbeiter und Abteilungen in SQL

Wir wollen folgendes Relationen-Schema anlegen:

Mitarbeiter

PersNr	Nachname	Vorname	Abteilung

Abteilungen

AbtNr	Name

Wir beginnen mit Abteilungen:

```
CREATE TABLE Abteilungen (
   abtNr     INT              PRIMARY KEY,
   name      VARCHAR(40),
)
```

Tabellendefinition in SQL

Beispiel: Mitarbeiter und Abteilungen in SQL

Und weiter geht es mit:

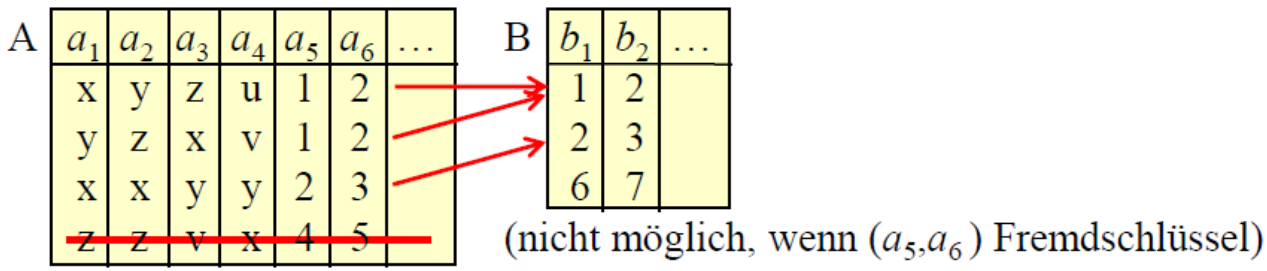
Mitarbeiter

PersNr	Nachname	Vorname	Abteilung

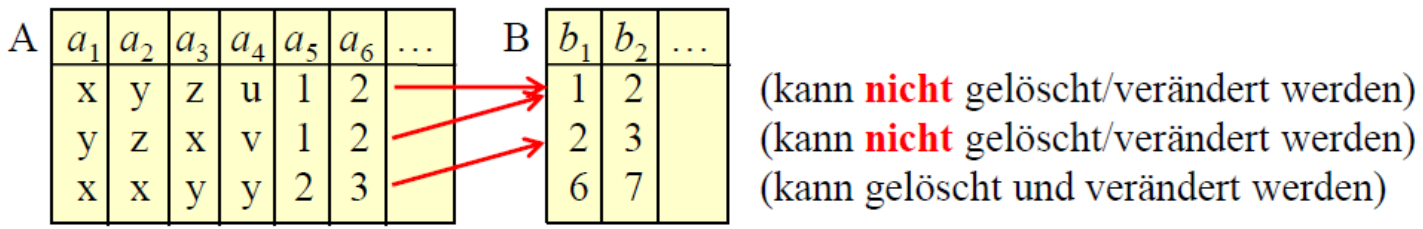
```
CREATE TABLE Mitarbeiter (
   persNr       INT          PRIMARY KEY,
   nachname     VARCHAR(50),
   vorname      VARCHAR(80),
   abteilung    INT          REFERENCES Abteilungen (abtNr),
)
```

Tabellendefinition in SQL

- Damit sind die (zunächst leeren) Tabellen angelegt und können mit Daten gefüllt werden
- Aber Vorsicht:
  Wir können keine Tupel in die Mitarbeiter-Tabelle einfügen, die keine gültige Referenz (in Abteilungen) haben



Und genauso können wir keine Abteilungen löschen, die von Mitarbeitern referenziert werden



Tabellendefinition in SQL

Beispiel: Mitarbeiter einfügen

Nehmen wir an, wir haben folgenden Zustand in der Abteilungen-Tabelle

AbtNr	**Name**
01	Special Investigations

Nun wollen wir folgenden Zustand für die Mitarbeiter-Tabelle herstellen:

PersNr	**Nachname**	**Vorname**	**Abteilung**
01	Muller	Robert	01
02	Trump	Donny	**02**

- Tupel (01, Muller, ...) kann eingefügt werden
- Tupel (02, Trump, Donny, 02) leider nicht (Sorry!)
  Wir brauchen natürlich vorher eine Ateilung mit AbtNr = 02

Tabellendefinition in SQL

Beispiel: Abteilung löschen

Nehmen wir an, wir haben folgenden Zustand

AbtNr	**Name**
01	Special Investigation
02	Controlling

PersNr	**Nachname**	**Vorname**	**Division**
01	Muller	Robert	01
02	Comey	Jimmy	01

- OK, Abteilung (02, Controlling) können wir löschen (wer braucht auch schon Controlling, mal ehrlich!?)
- Bei Abteilung (01, Special Investigation) ist das offensichtlich keine gute Idee (auch wenn über die Gründe vielleicht diskutiert werden könnte) und tatsächlich würde das (DB-)System dies ohne Zusatzinfos auch nicht zulassen

Tabellendefinition in SQL

- Was sind diese "Zusatzinfos"?
- Wenn wir Abteilungen löschen wollen, auf die aus der Mitarbeiter-Tabelle immer noch Referenzen existieren, können wir das `REFERENCES table (attr)` Constraint noch mit folgenden Zusätzen versehen:
  - `ON DELETE CASCADE`: Löschen eines Tupels in Abteilungen führt zum Löschen aller entsprechenden Mitarbeiter-Tupel, die diese Abteilung referenzieren
  - `ON DELETE SET NULL`: Löschen eines Tupels in Abteilungen führt nicht zum Löschen aller entsprechenden Mitarbeiter-Tupel; die entsprechenden "hängenden" Verweise ("dangling references") werden stattdessen auf `NULL`
  - `ON UPDATE CASCADE`: Verändern eines Tupels in Abteilungen führt zum Update aller entsprechenden Mitarbeiter (d.h. die Mitarbeiter "ziehen mit um")

Tabellendefinition in SQL

Beispiel: Mitarbeiter

Mitarbeiter-Tabelle nun mit Constraints für dangling references:

```
CREATE TABLE Mitarbeiter (
   persNr       INT          PRIMARY KEY,
   name         VARCHAR(50),
   vorName      VARCHAR(80),
   abteilung    INT          REFERENCES Abteilungen (abtNr),
                             ON DELETE SET NULL,
                             ON UPDATE CASCADE
)
```

Änderung einer Abteilung führt die Änderungen bei allen betroffenen Mitarbeitern mit; Löschen einer Abteilung löscht die Mitarbeiter nicht.

Tabellendefinition in SQL

Beispiel: Tabelle "Lehrveranst" (siehe Folie 23)

Mit zusammengesetztem Primärschlüssel {VNr, Semester}:

```
CREATE TABLE Lehrveranst (
   lNr          INT             NOT NULL,
   vNr          INT             NOT NULL,
   titel        VARCHAR(50),
   semester     VARCHAR(20),    NOT NULL,
   PRIMARY KEY(vNr, semester)
)
```

Alternative mit einfachem Primärschlüssel {LNr}:

```
CREATE TABLE Lehrveranst2 (
   lNr          INT             PRIMARY KEY,
   vNr          INT             NOT NULL,
   titel        VARCHAR(50),
   semester     VARCHAR(20),    NOT NULL,
)
```

Tabellendefinition in SQL

Beispiel: Lehrveranstaltungen ...

Dazu noch Tabellen “Dozent” und “Haelt”

```
CREATE TABLE Dozent (
   dNr          INT              PRIMARY KEY,
   name         VARCHAR(80),
   geburt       Date
)

CREATE TABLE Haelt (
   dozent       INT              REFERENCES Dozent (dNr),
                                 ON DELETE CASCADE,
   vNr          INT              NOT NULL,
   semester     VARCHAR(20),     NOT NULL,
   PRIMARY KEY (dozent, vNr, semster),
   FOREIGN KEY (vNr, semster) REFERENCES Lehrveranst2
)
```

Weitere DDL-Konstrukte in SQL

Weitere (hoffentlich selbsterklärende) Konstrukte der DDL von SQL sind u.a.:

- `DROP TABLE name`
- `ALTER TABLE name ADD (attr_1 dom_1 constr_1, attr_2 dom_2 constr_2, ...)`
- `ALTER TABLE name DROP (attr_1, attr_2, ...)`

Achtung: einmal mit `DROP` gelöschte Tabellen sind wirklich weg und nicht etwa im Papierkorb ... ;-)