Einführung in Datenbanken

Christoph Draxler
draxler@phonetik.uni-muenchen.de

29. Juni 2018

Definition

Eine Datenbank (engl. *database*) ist eine selbstständige, auf Dauer und für flexiblen und sicheren Gebrauch ausgelegte Datenorganisation, umfassend einen Datenbestand (auch *Datenbasis* genannt) und die dazugehörige Datenverwaltung [Zeh89].

Charakteristische Eigenschaften I

Strukturierung der Daten der Datenbestand hat einen sichtbaren Aufbau.

Trennung von Daten und Anwendung die Daten werden unabhängig von Anwendungen gehalten

- es gibt öffentliche Schnittstellen
- die interne Datenorganisation ist verborgen

Charakteristische Eigenschaften II

Datenintegrität die Daten sind

- frei von Widersprüchen (*Datenkonsistenz*),
- geschützt vor Verlust oder Verfälschung (*Datensicherheit*),
- nur kontrolliert zugreifbar (*Datenschutz*).

Zeitliche Permanenz die Daten sind auf Dauer nutzbar.

Spezifische Datensichten Benutzer sehen nur den für sie relevanten Ausschnitt der Daten.

Datenbankverwaltungssystem

Ein Datenbankverwaltungssystem (engl. *database management system*, abgekürzt *DBMS*) ist ein Soft- und Hardwarepaket, das die charakteristischen Eigenschaften einer Datenbank bietet.

Betrachtungsebenen

Eine grobe Unterteilung der für Datenbanken relevanten Betrachtungsebenen ist die folgende:

- reale Welt: Dinge, Personen, Abstrakta
- logische Sicht: Beschreibung der Dinge der realen Welt und der Beziehungen zwischen ihnen
- physische Sicht: Speicherung von Daten in einem DBMS

Datenbankbegriffe

Datenmodell (engl. *data model*) Strukturssprache zur Beschreibung von Datenbeständen. Beispiele von Datenmodellen sind das hierarchische, netzwerkartige, Entitätale, deduktive oder objekt-orientierte Datenmodell.

Datendefinition (engl. *data definition*) Beschreibung der Daten in der Datendefinitionssprache (engl. *data definition language*, abgekürzt *DDL*) eines Datenmodells.

Datenmanipulation (engl. *data manipulation*) Operationen über dem Datenbestand in einem Datenmodell. Beispiele von Operationen sind das Löschen, Einfügen oder Ändern von Daten. Befehle zur Datenmanipulation werden in der Datenmanipulationssprache (engl. *data manipulation language*, abgekürzt *DML*) des Datenbanksystems formuliert.

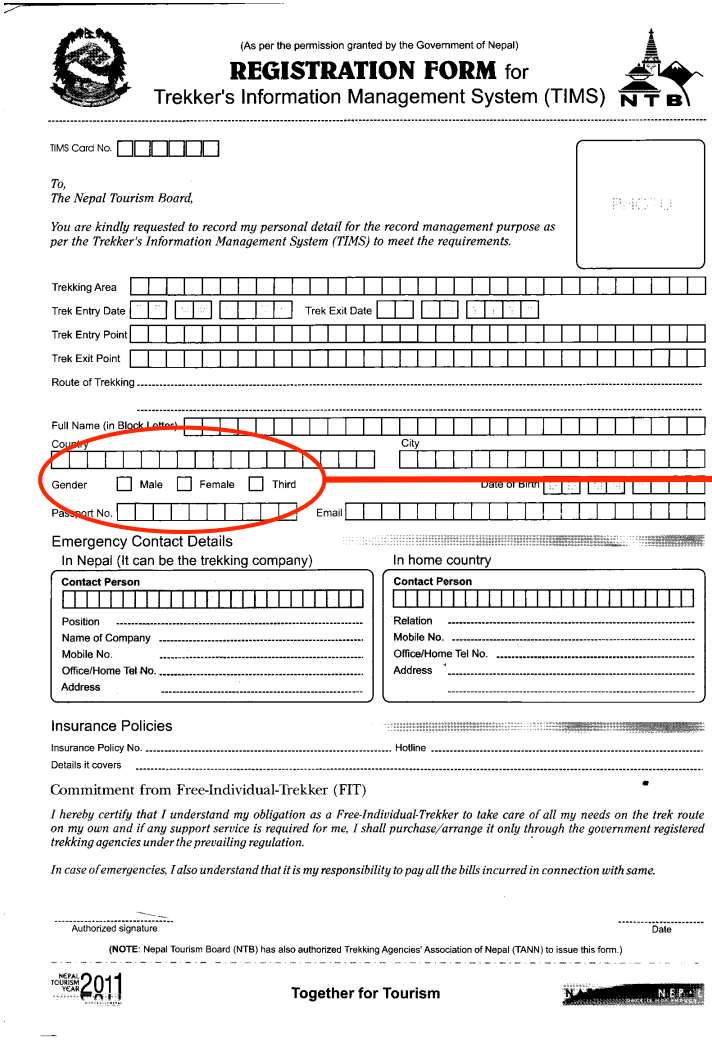
Datenbankbegriffe (Forts.)

Datenkonsistenz (engl. *consistency*) Widerspruchsfreiheit der Daten. Beispiele sind

- Wertebereichskonsistenz: "männlich, weiblich", oder Monatsziffern von 1 bis 12
- logische Konsistenz: eine Person kann nur genau eine Sozialversicherungsnummer haben
- und weitere Formen von Konsistenz.

Gilt *Widerspruchsfreiheit* immer?

Widerspruchsfreiheit



(As per the permission granted by the Government of Nepal)

REGISTRATION FORM for
Trekker's Information Management System (TIMS)

NTB

TIMS Card No.

To,
The Nepal Tourism Board,

You are kindly requested to record my personal detail for the record management purpose as per the Trekker's Information Management System (TIMS) to meet the requirements.

Trekking Area
Trek Entry Date Trek Exit Date
Trek Entry Point
Trek Exit Point
Route of Trekking

Full Name (in Block Letter)
Country City
Gender Male Female Third Date of Birth
Passport No. Email

Emergency Contact Details

In Nepal (It can be the trekking company)

Contact Person
Position
Name of Company
Mobile No.
Office/Home Tel No.
Address

In home country

Contact Person
Relation
Mobile No.
Office/Home Tel No.
Address

Insurance Policies

Insurance Policy No. Hotline
Details it covers

Commitment from Free-Individual-Trekker (FIT)

I hereby certify that I understand my obligation as a Free-Individual-Trekker to take care of all my needs on the trek route on my own and if any support service is required for me, I shall purchase/arrange it only through the government registered trekking agencies under the prevailing regulation.

In case of emergencies, I also understand that it is my responsibility to pay all the bills incurred in connection with same.

Authorized signature Date

(**NOTE**: Nepal Tourism Board (NTB) has also authorized Trekking Agencies' Association of Nepal (TANN) to issue this form.)

NEPAL TOURISM YEAR 2011

Together for Tourism

Full Name (in Block Letter)

Country

Gender Male Female Third

Passport No.

Nein – Widerspruchsfreiheit ist z.B. kultur- oder kontextabhängig!

Datenbankbegriffe (Forts.)

Transaktion (engl. *transaction*) Abfolge von Datenmanipulationen, die eine Datenbank von einem konsistenten Zustand wieder in einen konsistenten Zustand überführt. Eine Transaktion ist atomar, d.h. sie kann entweder ganz oder gar nicht ausgeführt werden. Das bedeutet, dass eine unterbrochene Transaktion den ursprünglichen Zustand wieder herstellen muß.

Datenmodellierung und Datenbankentwurf

Datenmodellierung Abbildung eines Ausschnitts der realen Welt in eine Datenbeschreibung (Datenmodell)

Datenbankentwurf Implementation des Datenmodells in einem Datenbankverwaltungssystem

Datenmodellierung und Datenbankentwurf

Vorgehen in drei Phasen:

Modellierung unabhängig vom verwendeten DBMS

Implementierung abhängig vom verwendeten DBMS

Optimierung in der Anwendung

Innerhalb der Phasen sind Iterationen möglich, Rücksprünge in vorangegangene Phasen evtl. notwendig

Werkzeuge zur Datenmodellierung

Anforderungen

- *Semantisch reich* für eine hinreichend genaue Abbildung des Weltausschnitts von Interesse (Miniwelt)
- *Syntaktisch beschränkt* für eine semi-automatische oder automatische Transformation in ein *logisches* Datenbankschema

Entity-Relationship-Modell

Das Entity-Relationship (oder ER-) Modell wurde vorgeschlagen von [Che76]. Im ER-Modell stehen Dinge von Interesse (engl. *entities*) in Beziehungen (engl. *Relationship*) zueinander. Das ER-Modell wurde erweitert zu

- Extended ER Model, z.B. [TYF86]
- Semantic Data Model u. ä.

und es ist zentraler Bestandteil der umfassenderen Modellierungstechnik UML (Unified Modelling Language) [BRJ98], [BD04].

ER-Modell (Forts.)

Das ER-Modell ist im wesentlichen eine einfache Diagrammtechnik zur Datenmodellierung.

- Entität (engl. *entity*) ist ein "Ding" der realen Welt (meist als Rechteck dargestellt).
- Beziehung (engl. *Relationship*) besteht zwischen Entitäten (meist als Raute mit – evtl. gerichteten – Verbindungslinien zwischen Entitäten dargestellt)
- Kardinalitäten (engl. *cardinalities*) quantifizieren Beziehnungen (meist an die Verbindungslinien geschrieben, z.B. 1-1, 1-m, n-m)

Konditionale Beziehungen werden durch ein "c" in der Angabe der Kardinalität dargestellt, z.B. 1-mc.

Präzisierung

Entitäten und Beziehungen haben Eigenschaften, sog. Attribute (engl. *attributes*), die sie beschreiben.
Die Menge aller Entitäten die durch die gleichen Eigenschaften beschrieben werden kann, ist eine Entitätsmenge (engl. *entity set*).
Jede Entität einer Entitätsmenge ist durch ihre Attributwerte (engl. *attribute values*) beschrieben.

Attribute

- zusammengesetzte Attribute haben Werte, die in kleinere Bestandteile zerlegt werden können
- atomare Attribute können nicht weiter zerlegt werden
- abgeleitete Attribute werden (z.B. aus anderen Attributen) berechnet
- mehrwertige Attribute bestehen aus mehreren Attributwerten, einwertige Attribute haben genau einen Attributwert
- notwendige Attribute müssen einen Wert haben, optionale Attribute können auch keinen Wert annehmen
- es gibt einen ausgezeichneten Attributwert, den Nullwert

Schlüsselattribute

Schlüsselattribute (engl. *key attributes*) können in

- Identifikationsschlüssel,
- Sortierschlüssel und
- Suchschlüssel

unterteilt werden.

Schlüssel

Identifikationsschlüssel (engl. *identification* oder *primary key*) identifizieren Entitäten

Sortierschlüssel (engl. *sort key*) erlauben die Sortierung von Entitäten

Suchschlüssel (engl. *search key*) oder ermöglichen die effiziente Suche nach und den effizienten Zugriff auf einzelne Entitäten

Erweiterungen des ER-Modells

Entitäten, die Attribute einer anderen Entität und darüber hinaus noch eigene Attribute haben, können als abhängige, spezialisierte, oder Unter-Entitäten aufgefasst werden.

Diese Unter-Entitäten können überlappend sein, z.B. "Personen sind Sprecher und/oder Transkribierer" oder disjunkt, z.B. "Sprecher sind weibliche oder männliche Sprecher".

Spezialisierung/Generalisierung

Spezialisierung ist die Erweiterung der Beschreibung von Entitäten durch zusätzliche Attribute. Sie ist möglich durch

- Rollen (engl. *roles*), oder
- Vererbung (engl. *inheritance*) von Attributen.

Generalisierung ist das Gegenteil der Präzisierung: die gemeinsamen Attribute mehrerer Entitäten werden in einer neuen Entität zusammengefasst.

Rolle

Rollen werden in semantischen Datenmodellen verwendet. Eine Entität erhält eine bestimmte Rolle, und in dieser Rolle wird sie durch zusätzliche Attribute beschrieben.

- *Student* ist eine Rolle der Entität *Person*.
- In der Rolle *Student* erhält die Entität *Person* die zusätzlichen Attribute *Matrikelnummer*, *Fachkombination*, *Anzahl Fachsemester*, usw.

Vererbung

In objekt-orientierten Datenmodellen *erbt* eine Entität Attribute von übergeordneten Entitäten.

- Objekt *Person* vererbt die Attribute *Name*, *Vorname*, *Geburtsdatum* an ein Objekt *Student*.
- *Student* hat also die geerbten Attribute *Name*, *Vorname*, *Geburtsdatum*, und die *eigenen*, *zusätzlichen* Attribute *Matrikelnummer*, *Fachkombination*, *Anzahl Fachsemester*, usw.

Vererbung ordnet Objekte hierarchisch.

Beschränkungen des ER-Modells

Das ER-Modell erfasst einige für die Verwaltung eines Datenbestands wichtige Eigenschaften nicht, z.B. bestimmte Integritätsbedingungen, oder die Propagation von Änderungen.

Vorgehen bei der Datenmodellierung

Identifizierung von

- Entitäten
  - existieren unabhängig voneinander
  - sind in der Regel durch Substantive zu bezeichnen: z.B. *Sprecher*, *Wort*, *Buch*, *Computer*
- und Beziehungen
  - bestehen nur zwischen Entitäten, sie sind also abhängig
  - sind häufig durch Verben zu bezeichnen: z.B. *lesen*, *aufnehmen*, *ausleihen*, *speichern*

Modellierung

Wichtig: Es gibt in der Regel nicht "die" richtige Modellierung!
z.B.

- Entitäten einer Modellierung können in anderen Modellierungen Beziehungen sein und umgekehrt.
- 1-n Beziehungen können durch mehrwertige Attribute erfasst werden
- konditionale Beziehungen können durch Attribute mit Nullwerten erfasst werden
- 1-1 Beziehungen können auch durch Attribute erfasst werden.

Schema vs. Instanzierung

Ein ER-Modell ist ein konzeptionelles Schema einer Datenbank, d.h. es beschreibt die Struktur der Daten und ihre Beziehungen zueinander als logische Sicht. Die *Ausprägung* oder *Instanz* einer Datenbank sind die in der Datenbank gespeicherten Werte.

Normalisierung

- Die Normalisierung ist ein wesentlicher Schritt der Datenmodellierung.
- Bei der Normalisierung sucht man nach *Redundanz* und sog. *funktionalen Abhängigkeiten* und versucht, diese durch geeignete Umformung der Entitäten und Attribute zu vermeiden.

Redundanz

Ein Eintrag ist *redundant*, wenn er in der Datenbank mehrfach gespeichert ist. Redundanz ist aus mindestens zwei Gründen unerwünscht:

- sie verursacht sog. Änderungsanomalien, und
- sie führt zu einer schlechteren Ausnutzung des Speicherplatzes.

Änderungsanomalien sind inkonsistente Datenbankzustände, die durch Einfügen, Löschen oder Ändern von Datenbankeinträgen hervorgerufen werden. Eine typische solche Anomalie ist z.B. die Änderung nur an einem von mehreren redundanten Einträgen.

Änderungsanomalien

In der Tabelle enthält das Attribut `Anschrift` nicht-atomare, doppelte (und zudem falsch formatierte) Werte.

Vorgang	Datum	Signatur	Titel	Benutzer	Anschrift
1352	22.1.13	MAI	Logic Programming	ST442	80799 München, Schellingstr. 3
1352	22.1.13	CLO	Programming in Prolog	ST442	Schellingstr. 3, 80799 München
1353	22.1.13	ZEH	Informationssysteme	DOZ387	Oettingenstr. 67, 80538 München

`Anschrift` enthält unterschiedlich formatierte Werte; korrigiert man die Formatierung, dann gibt es doppelte Einträge.

Normalisierung und Normalformen

Ziel der Normalisierung ist, Redundanz und damit Anomalien zu vermeiden.

- sie ist ein notwendiger Schritt bei der Datenmodellierung, denn sie liefert eine formale Beschreibung der vorhandenen Redundanz in der Datenbank
- sie kann für jedes Datenmodell durchgeführt werden; meist wird sie am Beispiel des Relationalen Modells erklärt.

Grundlage der Normalisierung sind *Normalformen* (häufig NF abgekürzt).

Im Zuge der Normalisierung werden Attribute und Entitäten solange zerlegt, bis die Datenbank den vorgegebenen Normalformen genügt.

1. Normalform

Die erste Normalform bezieht sich auf die Struktur einzelner Attribute.

1.Normalform Eine Entität ist in 1. Normalform, wenn alle ihre Attribute *atomar* sind, d.h. die Attributwerte können nicht weiter zerlegt werden.

Beispiel IPS-Bibliothek

Ein Student des IPS und ein Dozent der Computerlinguistik leihen in einem Ausleihvorgang Bücher aus.

Vorgang	Datum	Signatur	Titel	Benutzer	Anschrift
1352	22.1.13	MAI	Logic Programming	ST442	Schellingstr. 3, 80799 München
1352	22.1.13	CLO	Programming in Prolog	ST442	Schellingstr. 3, 80799 München
1353	22.1.13	ZEH	Informationssysteme	DOZ387	Oettingenstr. 67, 80538 München

Diese Entität ist nicht in 1.NF, da das Attribut `Anschrift` offensichtlich nicht atomar ist.

Beispiel 1. Normalform

`Anschrift` gliedert sich in Straße, Postleitzahl, und Ort.

Vorgang	Datum	Signatur	Titel	Benutzer	Straße	PLZ	Ort
1352	22.1.13	MAI	Logic Programming	ST442	Schellingstr. 3	80799	München
1352	22.1.13	CLO	Programming in Prolog	ST442	Schellingstr. 3	80799	München
1353	22.1.13	ZEH	Informationssysteme	DOZ387	Oettingenstr. 67	80538	München

Diese Entität ist in 1. NF mit der Attributkombination Vorgang und Signatur als Schlüssel.

Funktionale Abhängigkeit

Ein Attribut bzw. eine Attributskombination R.B ist *funktional abhängig* von einem Attribut bzw. einer Attributskombination R.A der Entität R, wenn es zu jedem Wert von R.A höchstens einen Wert R.B gibt.

Anders gesagt: R.B ist funktional abhängig von R.A, wenn R.A den Wert von R.B bestimmt.

Die funktionale Abhängigkeit wird wie folgt geschrieben:
$R.A \rightarrow R.B$

Funktionale Abhängigkeit

- Ob eine funktionale Abhängigkeit vorliegt, ist nicht immer einfach zu beurteilen.
- Die Abhängigkeit kann in der modellierten Welt nicht bestehen, während sie in den Einträgen der Datenbank durchaus zu finden ist.

Beispiel: Funktionale Abhängigkeit

In den Daten besteht anscheinend eine funktionale Abhängigkeit:

Straße	Ort
Leopoldstraße	München
Kurfürstendamm	Berlin
Gänsemarkt	Hamburg

In dieser Entität scheint es eine funktionale Abhängigkeit zwischen Straße und Ort zu geben.
Unser *Weltwissen* sagt uns aber, dass eine solche funktionale Abhängigkeit in der realen Welt nicht besteht: es gibt viele Straßennamen, die in mehr als einem Ort vorkommen.

Formen der funktionalen Abhängigkeit

voll funktional abhängig R.B ist voll funktional abhängig von R.A, wenn es nicht bereits von Teilen von R.A funktional abhängig ist.

transitiv funktional abhängig R.C ist transitiv abhängig von R.A, wenn $R.A \rightarrow R.B$ und $R.B \rightarrow R.C$, nicht aber $R.B \rightarrow R.A$.

2. Normalform

Eine Entität ist in 2. Normalform, wenn sie in 1. NF ist und jedes nicht zum Identifikationsschlüssel gehörige Attribut voll von diesem abhängig ist.

Beispiel 2. Normalform

Vorgang	Datum	Signatur	Titel	Benutzer	Straße	PLZ	Ort
1352	22.1.13	MAI	Logic Programming	ST442	Schellingstr. 3	80799	München
1352	22.1.13	CLO	Programming in Prolog	ST442	Schellingstr. 3	80799	München
1353	22.1.13	ZEH	Informationssysteme	DOZ387	Oettingenstr. 67	80538	München

Die Entität ist nicht in 2. NF, denn es gilt:

- $\mathit{Signatur} \rightarrow \mathit{Titel}$
- $\mathit{Vorgang} \rightarrow \mathit{Datum}$, $\mathit{Vorgang} \rightarrow \mathit{Benutzer}$, $\mathit{Vorgang} \rightarrow \mathit{Straße}$, $\mathit{Vorgang} \rightarrow \mathit{PLZ}$, $\mathit{Vorgang} \rightarrow \mathit{Ort}$
- $\mathit{Benutzer} \rightarrow \mathit{Straße}$, $\mathit{Benutzer} \rightarrow \mathit{PLZ}$, $\mathit{Benutzer} \rightarrow \mathit{Ort}$, $\mathit{PLZ} \rightarrow \mathit{Ort}$

3. Normalform

Eine Entität ist in 3. Normalform, wenn sie in 2. NF ist und kein Attribut, das nicht zum Identifikationsschlüssel gehört, transitiv von diesem abhängt.

2. Normalform

Für die 2. Normalform wird die Entität in drei Entitäten zerlegt.

Vorgang	Datum	Benutzer	Straße	PLZ	Ort
1352	22.1.13	ST442	Schellingstr. 3	80799	München
1353	22.1.13	DOZ387	Oettingenstr. 67	80538	München

Vorgang	Signatur
1352	MAI
1352	CLO
1353	ZEH

Signatur	Titel
MAI	Logic Programming
CLO	Programming in Prolog
ZEH	Informationssysteme

2. Normalform

Die drei Entitäten sind nun in 2. NF. In der ersten Entität sind nun alle Attribute voll funktional abhängig vom Schlüssel Vorgang. Daneben bestehen aber auch transitive Abhängigkeiten:

- $Vorgang \rightarrow Benutzer, Benutzer \rightarrow Straße$
- $Vorgang \rightarrow Benutzer, Benutzer \rightarrow PLZ$
- $Vorgang \rightarrow Benutzer, Benutzer \rightarrow Ort$
- $Benutzer \rightarrow PLZ, PLZ \rightarrow Ort$

3. Normalform

Für die 3. NF müssen die transitiven Abhängigkeiten beseitigt werden. Dazu wird die erste Entität erneut zerlegt:

Vorgang	Datum	Benutzer
1352	22.1.13	ST442
1353	22.1.13	DOZ387

Benutzer	Straße	PLZ
ST442	Schellingstr. 3	80799
DOZ387	Oettingenstr. 67	80538

PLZ	Ort
80799	München
80538	München

3. Normalform

Diese Datenbank, bestehend aus fünf Entitäten, ist in 3. Normalform.

Ausleihvorgang Vorgang, Datum, Benutzer

Benutzer Benutzer, Straße, PLZ

PLZVerzeichnis PLZ, Ort

Ausleihe Vorgang, Signatur

Katalog Signatur, Titel

Beispiel: Phonetische Sprachaufnahmen

Ein Protokollbogen für Sprachaufnahmen am Institut hat folgende Felder:

sprecher	g.	alter	akzent	datei	gräße	stadt	land	nuts	ebene	label
4823	m	14	SN	AAA482332	274476	Taucha	DE-SN	DED32	ORT	PC
4823	m	14	SN	AAA482354	405548	Taucha	DE-SN	DED32	ORT	Lars
4823	m	14	SN	AAA482365	405548	Taucha	DE-SN	DED32	ORT	noch
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	ist
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	alt
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	Jahr
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	noch
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	obwohl
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	Zähne
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	Kleinkir
4740	f	14	SN	AAA4740P06	249900	Chemnitz	DE-SN	DED11	ORT	das
4742	m	17	SN	AAA474234	164908	Chemnitz	DE-SN	DED11	ORT	Achtung
4742	m	17	SN	AAA474234	164908	Chemnitz	DE-SN	DED11	ORT	Rechner
					...					
4743	f	14	SN	AAA474321	188460	Chemnitz	DE-SN	DED11	ORT	sechzig
4743	f	14	SN	AAA474337	275500	Chemnitz	DE-SN	DED11	ORT	Rechner

Offensichtlich sind hier viele Daten redundant – aber welche?

Normalformen: Beispiel

In diesem Beispiel wird der Protokollbogen als Entität aufgefasst. Diese Entität wird nun Schritt für Schritt normalisiert. Üblicherweise führt man die Normalisierung allein auf der Datenbankstruktur, d.h. ohne konkreten Inhalt durch. Hier ist auf den nächsten Folien der Datenbankinhalt zur besseren Vergleichbarkeit dargestellt.

Die Entität ist in 1. Normalform, denn sie enthält nur atomare Attribute.

Sprachaufnahmen										
sprecher	geschl.	alter	akzent	datei	größe	stadt	land	nuts	ebene	label
4823	m	14	SN	AAA482332	274476	Taucha	DE-SN	DED32	ORT	PC
4823	m	14	SN	AAA432354	405548	Taucha	DE-SN	DED32	ORT	Lars
4740	f	14	SN	AAA4740P07	249900	Chemnitz	DE-SN	DED11	ORT	noch
...										
4743	f	14	SN	AAA474337	275500	Chemnitz	DE-SN	DED11	ORT	Rechner

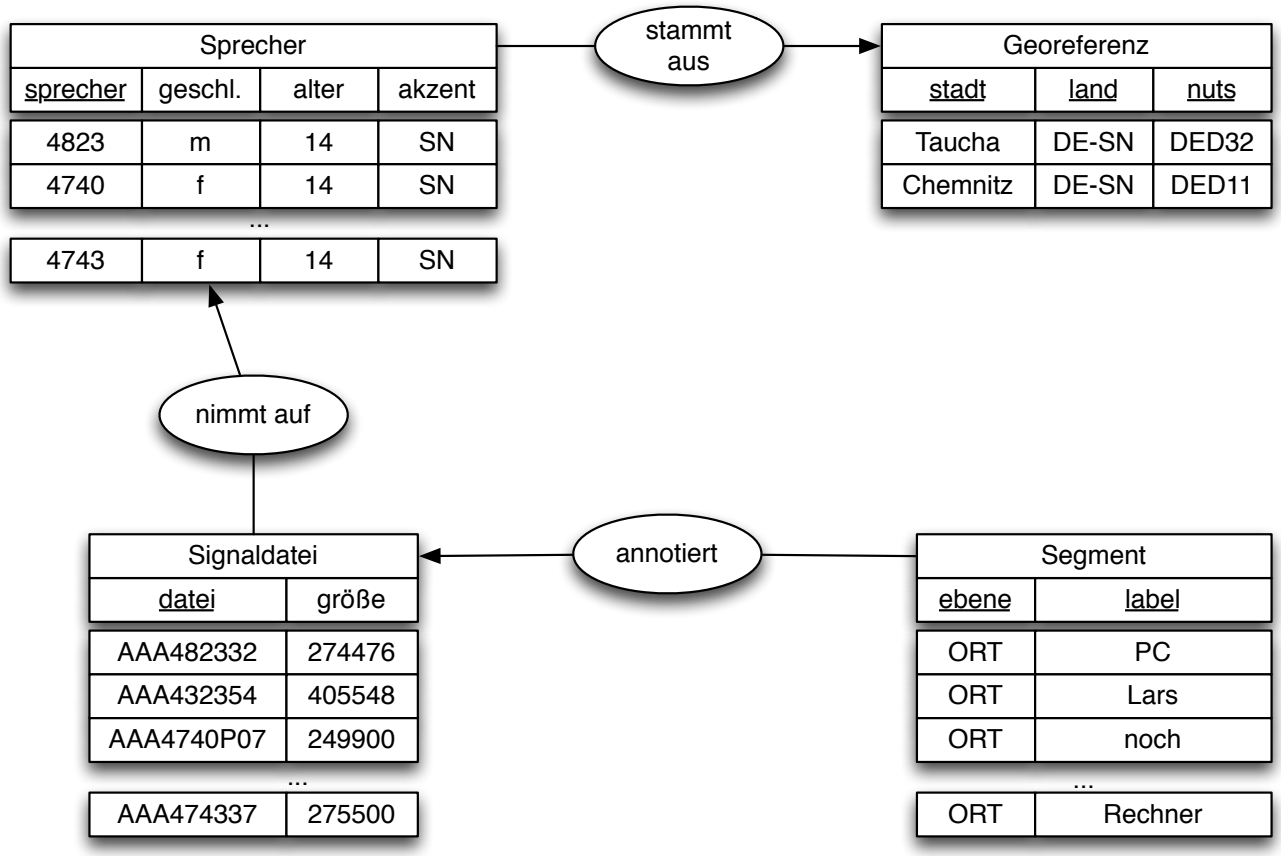
Normalformen (Forts.)

Die Entität ist nicht in 2. Normalform, weil es einen zusammengesetzten Schlüssel aus `sprecher`, `datei`, `stadt`, `nuts` und `label` gibt, aber einzelne Attribute schon von Teilen des Schlüssels abhängen, z.B. `land` und `nuts` von `stadt`.

Das Attribut `label` ist kein Schlüssel, weil uns das Weltwissen sagt, dass ein Wort in mehreren Äußerungen gesprochen werden kann.

Normalisierung (Forts.)

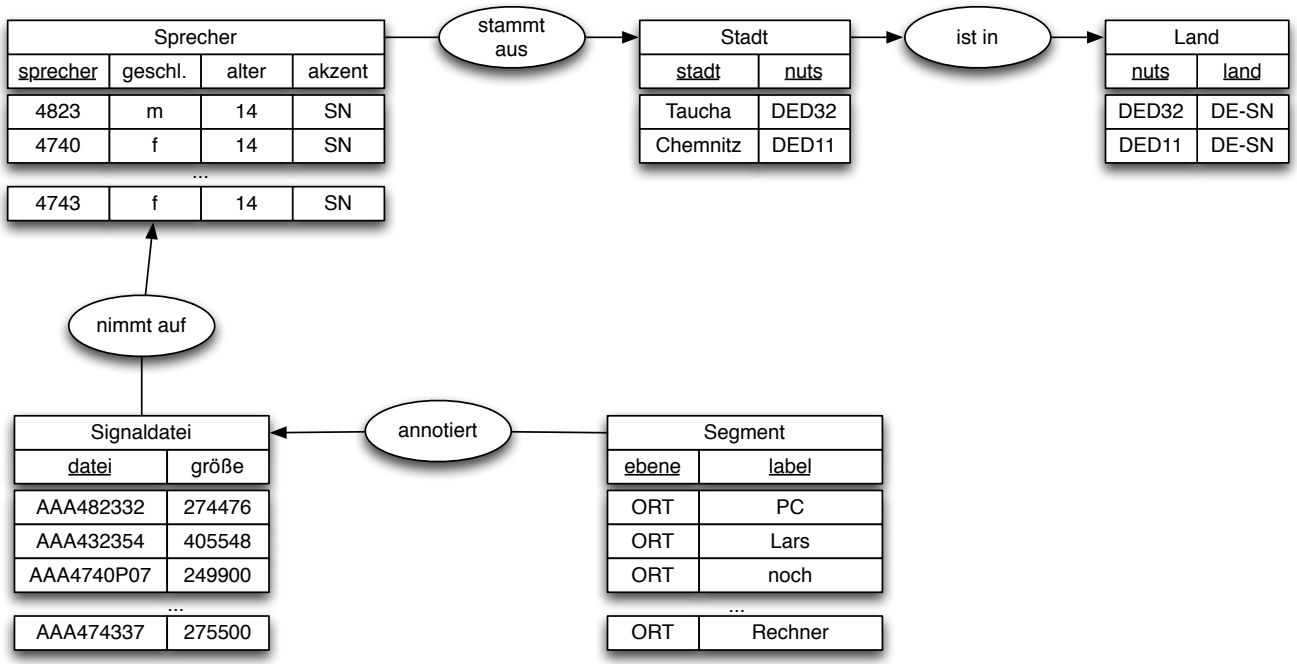
Die Entität wird in vier kleinere Entitäten aufgeteilt:



Die Schlüsselattribute sind jetzt `sprecher` für die Entität `Sprecher`, `stadt,land,nuts` für `Georeferenz`, `datei` für `Signaldatei` und `ebene,label` für `Segment`.

Normalisierung (Forts.)

In der Entität `Georeferenz` ist das Attribut `land` abhängig vom Attribut `nuts`, das wiederum abhängig ist von `stadt`. Um die 3. Normalform zu erreichen muss diese Entität also nochmals in zwei Entitäten unterteilt werden.



Die Entität `Stadt` hat die Attribute `stadt,nuts` als Schlüssel, die Entität `Land` die Attribute `nuts,land`.

Normalisierung Zusammenfassung

Durch die Normalisierung sind aus einer Entität mit 11 Attributen fünf Entitäten mit vier Beziehungen dazwischen geworden; die Anzahl der Attribute hat sich auf 12 erhöht.

- funktionale Abhängigkeiten und damit Redundanzen sind eliminiert
- genaugenommen müsste die Entität `Segment` ein eigenes Schlüsselattribut bekommen, weil die Kombination von Annotationsebene und Label nicht eindeutig sein muss – ein Wort kann mehrfach in einer Äußerung vorkommen.

Relationales Datenmodell

Das relationale Datenmodell (auch *Relationenmodell*, engl. relational data model)

- wurde 1970 von [Cod70] vorgeschlagen
- ist gekennzeichnet durch seine einheitlichen und einfachen Datenstrukturen

Die meisten kommerziellen Datenbanksysteme basieren auf dem relationalen Datenmodell

- Die Sprache SQL hat sich als Standardsprache für relationale Datenbanken etabliert

Literatur zu Datenbanken (Auswahl):
[EN02, KE04, FWBRB07, Har01]

Terminologie

Wertebereich (engl. *domain*) ist eine Menge *atomarer* Werte

Attribut ist ein Rollenname (engl. *role name*) eines Wertebereichs

Nullwert ist ein ausgezeichneter Attributwert, der anzeigt, dass ein Wert unbekannt ist.

Beispiele

Wertebereiche

- sind formal oder durch Aufzählung definiert
- z.B. natürliche, ganze oder Gleitkommazahlen, Zeichenketten fester oder variabler Länge, ...
- oder Telefonnummern, Postleitzahlen, ...

Attribute sind *benannte* Wertebereiche

- z.B. Privat-, Arbeits-, Mobil- und Faxnummer sind jeweils Telefonnummern

Terminologie

Schema (engl. *relation schema*), geschrieben als $R(A_1, ..., A_n)$, ist eine Menge von Attributen A_i.

Relation (oder *Tabelle* (engl. *relation* oder *relational table*) des Relationenschemas $R(A_1, ..., A_n)$ ist eine Untermenge von n-Tupeln $r = x_1, ..., x_n$, wobei ein Tupel eine geordnete Liste von n Werten x_i aus den Wertebereichen D_i ist.

$$r(R) \subset D_i \times ... \times D_n$$

Aus der Mengeneigenschaft ergibt sich, dass kein Tupel mehr als einmal vorkommt.

Kardinalität Anzahl der Tupel einer Relation

Terminologie

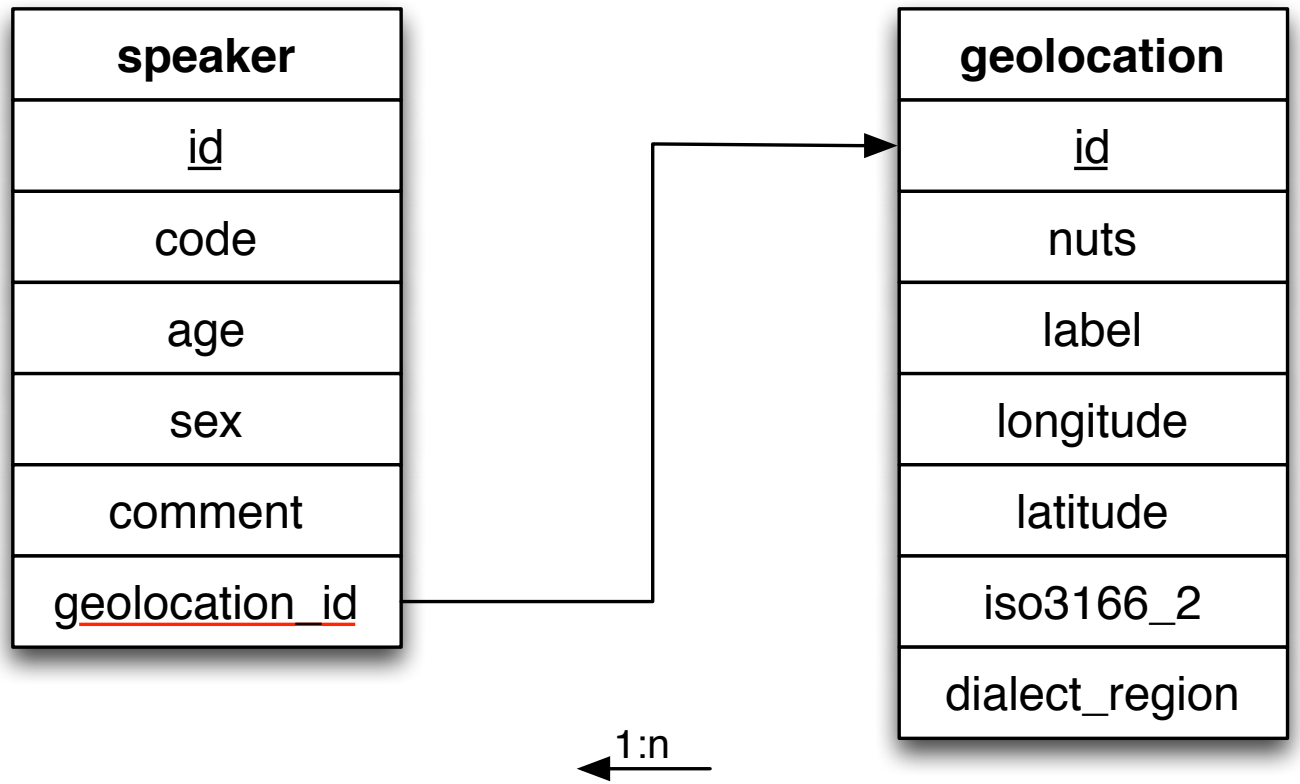
Schlüssel (engl. *key*) Untermenge von Attributen für die gilt, dass keine zwei Tupel der Relation denselben Attributwert haben. Auch *Primärschlüssel* (engl. *primary key*) genannt.

Fremdschlüssel (engl. *foreign key*) Untermenge von Attributen, die in einer Relation vorkommt und in einer weiteren Relation Primärschlüssel ist. Auch Sekundärschlüssel (engl. *secondary key*) genannt

Schlüsselkandidat (engl. *key candidate*) Untermenge von Attributen, die die Schlüsseleigenschaft erfüllt.

Darstellung im Diagramm

In Diagrammen werden der Primärschlüssel mit einer durchgezogenen Linie, Fremdschlüssel mit Pfeilen zwischen Relationen dargestellt.



Eine konsequente Namensgebung hilft bei der Orientierung. So verweist in dieser Datenbank `geolocation_id` stets auf den Schlüssel `id` in der Relation `geolocation`.

Integritätsbedingungen

Entitätsintegrität der Primärschlüssel einer Relation kann kein Nullwert sein.

Referentielle Integrität zu jedem Fremdschlüssel gibt es ein Tupel mit dem entsprechenden Primärschlüssel.

Beispiel

Ein Sprecher in der Phonetik ist eine Person, von der Geschlecht, Alter und ein Sprechercode bekannt sind.

- Relation `speaker`
- Attribute `sex`, `age`, `code` mit den Wertebereichen `[m,f]`, `[1..150]` und `[AAA0...ZZZ9]`
- neues Schlüsselattribut `id`

Ein Tupel `speaker(id, age, sex, code)` ist dann z.B. (48790244, 57, m, GRE5), die Relation `speaker` ist dann die Menge aller solcher Tupel.

ER-Diagramm und Relationales Datenmodell

Ein ER-Diagramm kann mechanisch in ein Relationenschema konvertiert werden.

- Entitäten werden direkt zu Relationen
- Primärschlüssel werden übernommen bzw. angelegt, wo notwendig
- Beziehungen zwischen Entitätsmengen werden über Fremdschlüsselattribute in bestehenden bzw. neu angelegten Relationen abgebildet.

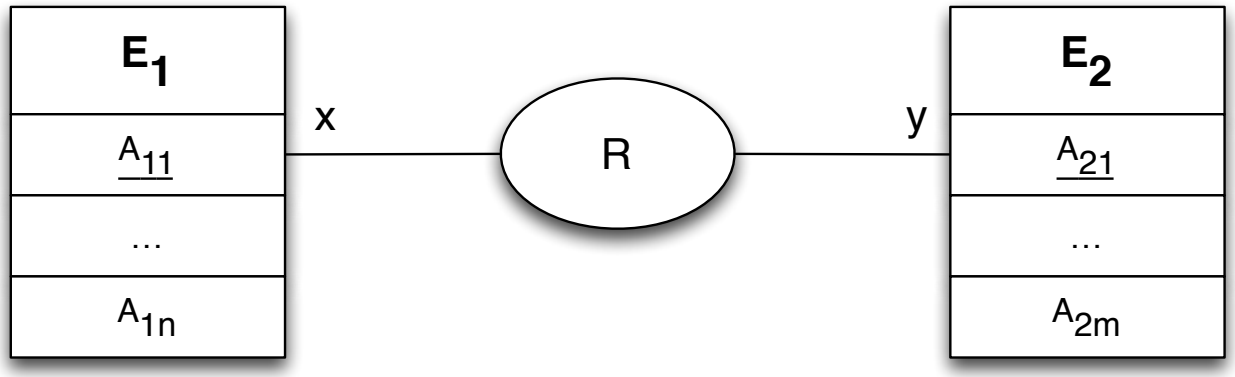
ER-Diagramm und Relationales Datenmodell

Es ist sinnvoll, Namenskonventionen zu beachten:

- Entitätsnamen werden zu Relationennamen
- bei n:m Beziehungen heißen die neuen Relationen wie die Beziehung
- Attributnamen bleiben erhalten
- künstliche bzw. neu angelegte Primärschlüssel heißen `id`
- Fremdschlüsselattribute heißen wie die Relation auf die sie zeigen, gefolgt von `'_id'`

Einheitliche Kleinschreibung und sprechende Namen verwenden

ER-Diagramm und Relationales Datenmodell



Mit den Kardinalitäten x:y

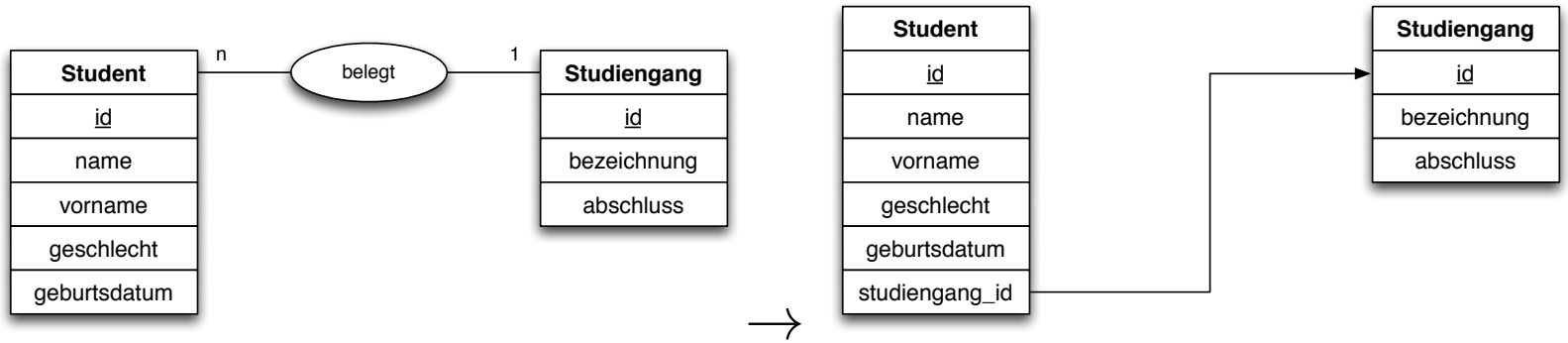
- 1:1 Attribute A_{1i} werden Attribute von E_2 oder umgekehrt (oder E_1 und E_2 erhalten als Fremdschlüssel den Primärschlüssel der anderen Relation).
- 1:n Primärschlüssel von E_1 wird Fremdschlüssel in E_2
- n:m neue Relation R mit den Primärschlüsseln von E_1 und E_2 zusammen als komplexem Primärschlüssel und einzeln als Fremdschlüssel

Beispiel: Universität I

1 : *n* Beziehung zwischen Student und Studiengang

- Ein Student studiert genau einen Studiengang.
- Ein Studiengang wird von 0 bis *n* Studenten studiert.

Die Relation Student bekommt ein zusätzliches Attribut `studiengang_id`, das als Fremdschlüssel auf das Schlüsselattribut `id` der Relation Studiengang verweist.

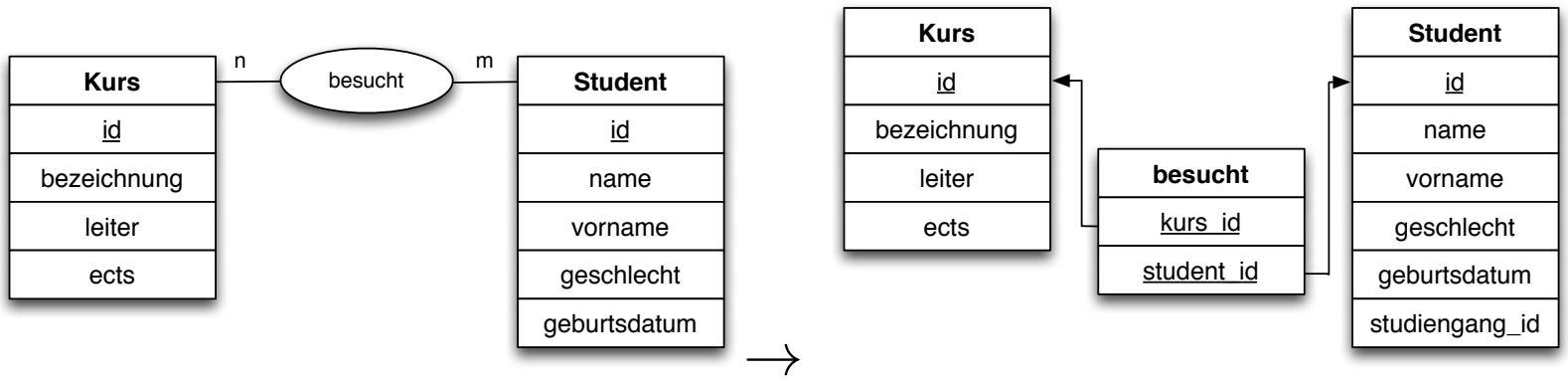


Beispiel: Universität II

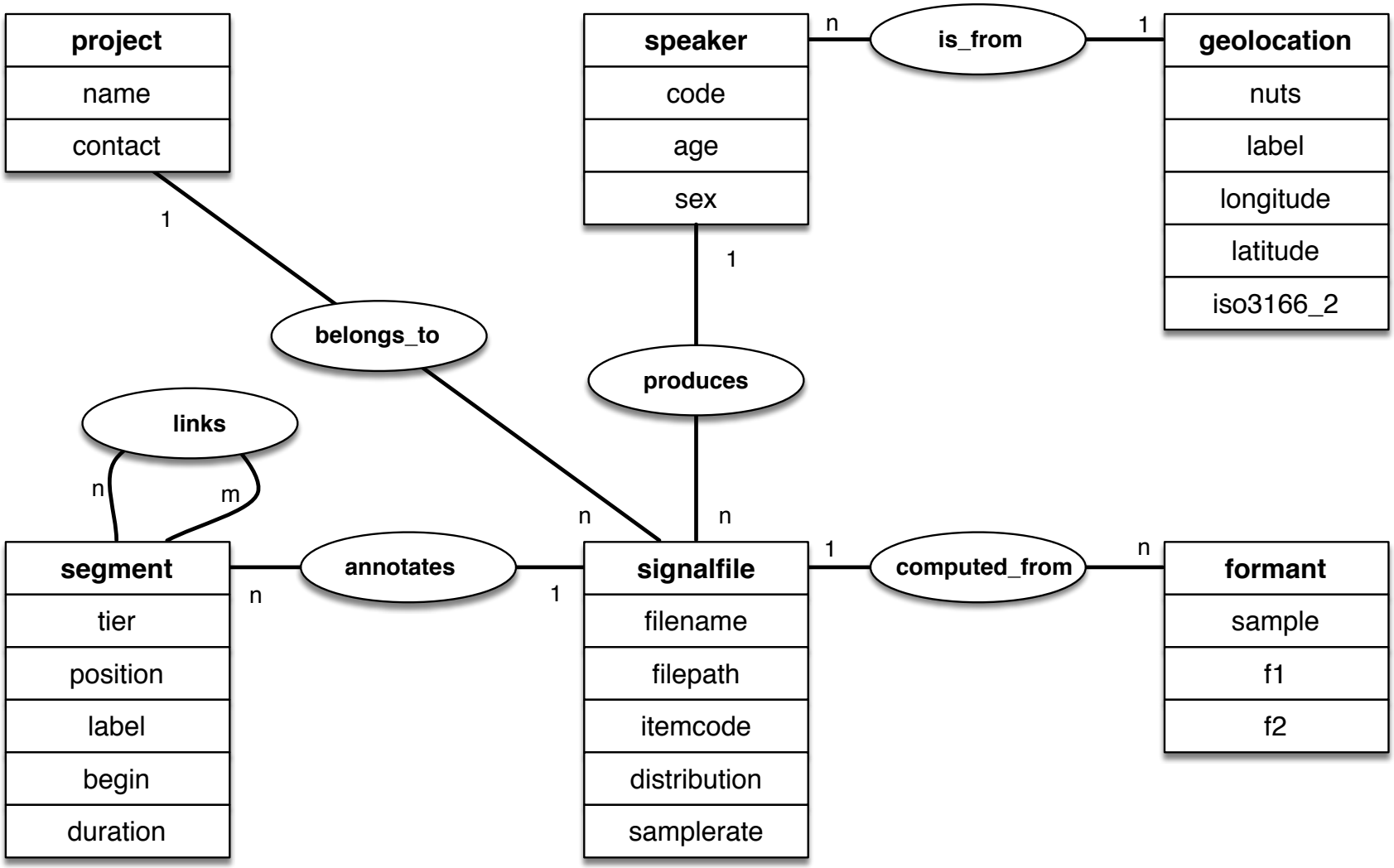
$n : n$ Beziehung zwischen Student und Kurs

- Ein Student besucht 0 bis n Kurse.
- Ein Kurs wird von 0 bis n Studenten besucht.

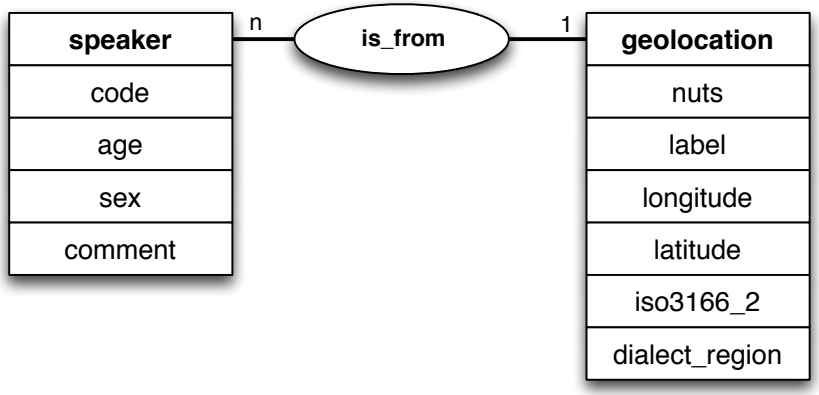
Es wird eine neue Relation *besucht* eingefügt, die zwei Fremdschlüsselattibute bekommt, die auf die Schlüsselattribute der Relationen *Student* und *Kurs* verweisen.



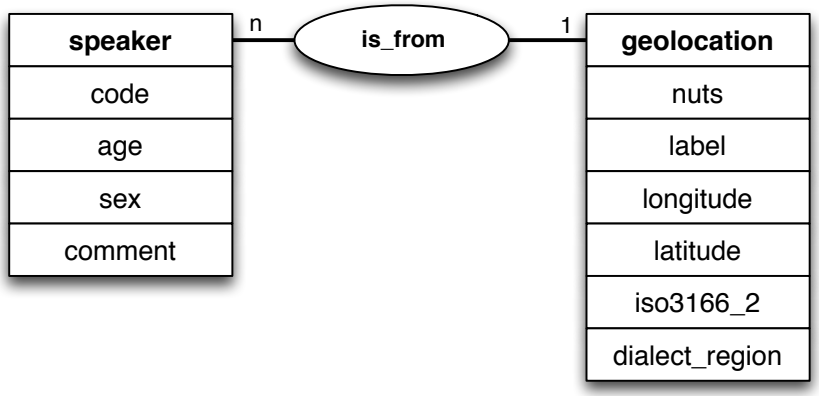
Beispiel: phonetische Datenbank



Beispiel: speaker und geolocation

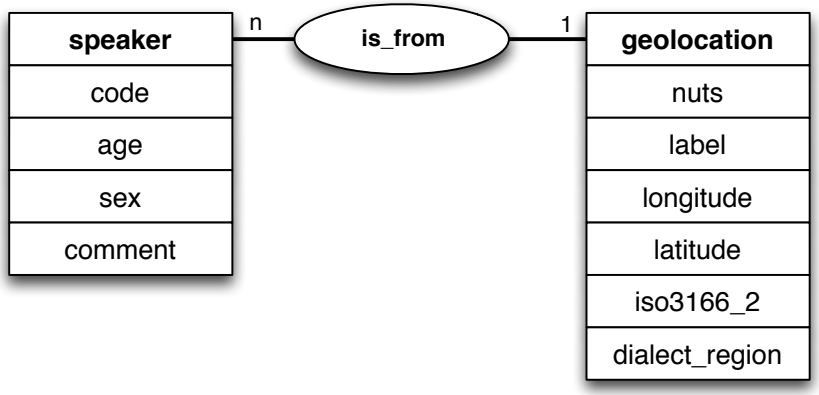


Beispiel: speaker und geolocation

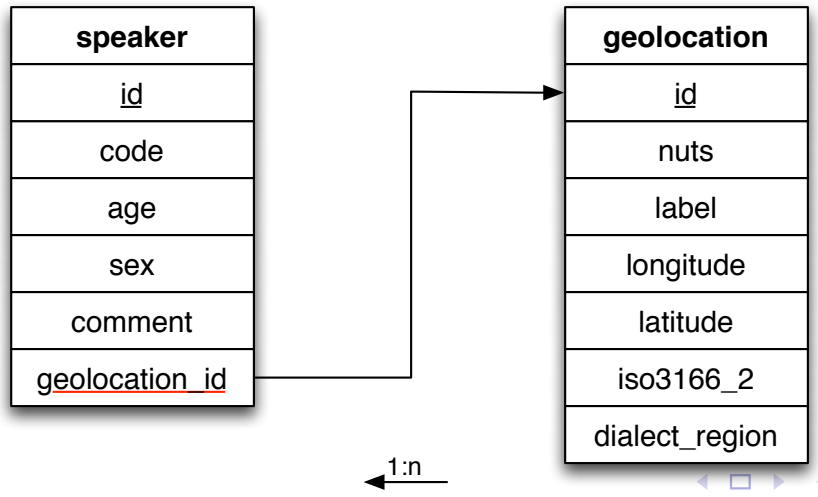


- `speaker` bekommt einen Primärschlüssel `id`
- `geolocation` bekommt einen Primärschlüssel `id`
- die Beziehung *is_from* wird durch einen Fremdschlüssel `geolocation_id` in `speaker` abgebildet

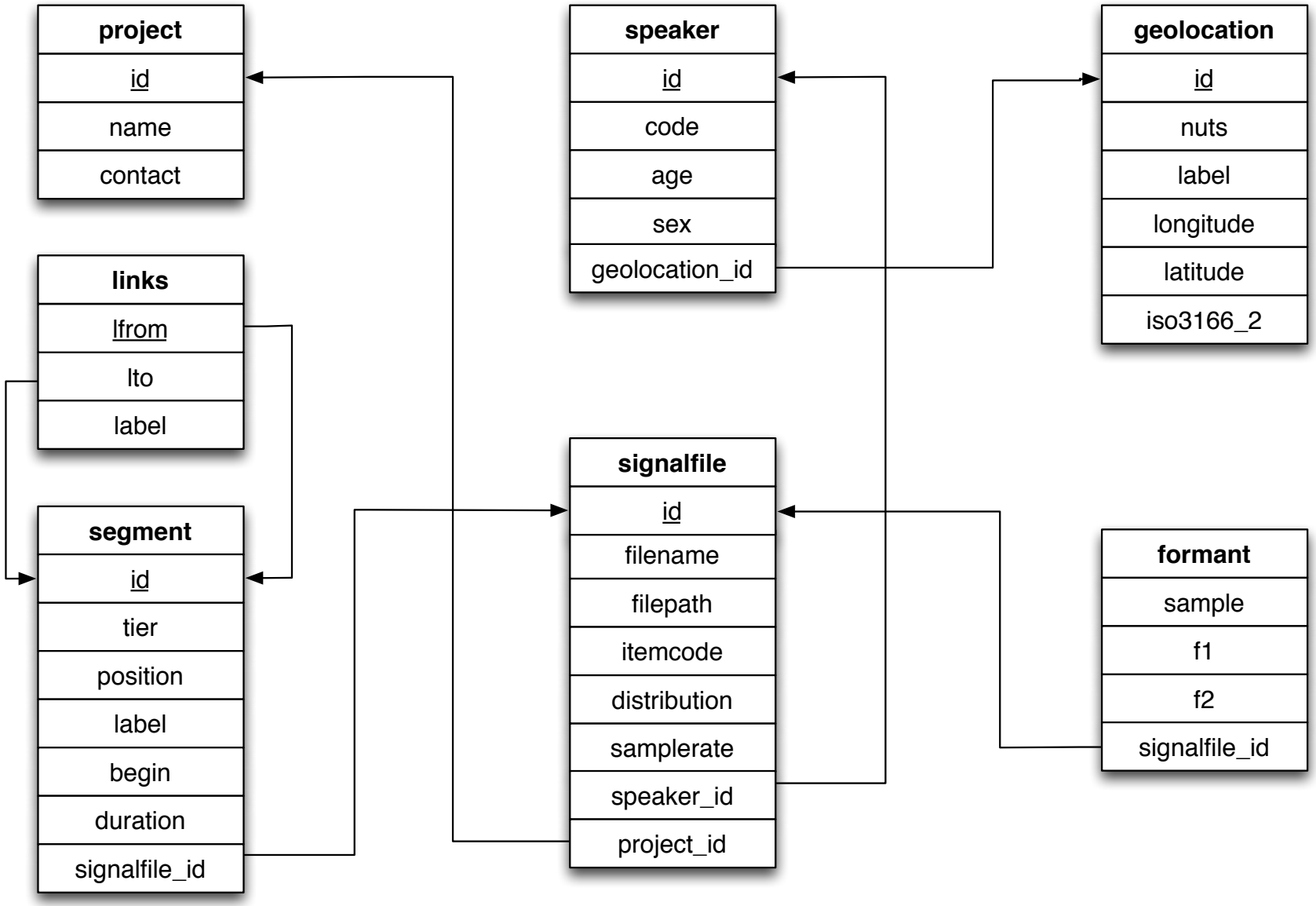
Beispiel: speaker und geolocation



- `speaker` bekommt einen Primärschlüssel `id`
- `geolocation` bekommt einen Primärschlüssel `id`
- die Beziehung *is_from* wird durch einen Fremdschlüssel `geolocation_id` in `speaker` abgebildet



phonetische Datenbank im Relationenmodell



Relationale Algebra

Die relationale Algebra umfasst eine Reihe von Operatoren, die über Relationen definiert sind. Es gibt

- unäre $R \rightarrow R$, und
- binare $R \times R \rightarrow R$ Operatoren

Das Ergebnis der Anwendung eines relationalen Operators ist wieder eine Relation. Daher können relationale Operatoren geschachtelt werden.

Beispiel: Relation speaker

speaker				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010
48790258	m	18	HAN1	1044010
48790263	f	54	HAN6	1044010
48790261	f	19	HAN4	1044010
48790289	m	19	HUS3	1297516
48790287	f	20	HUS1	1297516

Beispiel: Relation `geolocation`

geolocation					
id	nuts	label	longitude	latitude	iso3166_2
1044000	DE801	Greifswald	13.4086	54.0809	DE-MV
1297546	DED3	Leipzig	12.374	51.34	DE-SN
1297552	DE27B	Marktoberdorf	10.618	47.779	DE-BY
1044010	DE929	Hannover	9.74959	52.3936	DE-NI
1297594	DE232	Regensburg	12.096	49.017	DE-BY
1297609	DEG0B	Schmalkalden	10.451	50.721	DE-TH
1297619	DEA5A	Siegen	8.024	50.874	DE-NW
1297626	DE21M	Traunstein	12.612	47.858	DE-BY
1297516	DE927	Husum	9.051	54.477	DE-SH
1297637	DEC01	Völklingen	6.853	49.253	DE-SL

Relationale Operatoren

Die ersten drei relationalen Operatoren sind die bekannten Mengenoperatoren

Vereinigung $R_1 \cup R_2$ enthält alle Elemente aus R_1 oder R_2

Durchschnitt $R_1 \cap R_2$ enthält alle Elemente, die in R_1 und in R_2 vorkommen

Differenz $R_1 \setminus R_2$ enthält die Elemente, die nur in R_1 und nicht in R_2 vorkommen

Diese Operatoren sind nur für sog. *vereinigungsverträgliche* oder *kompatible* Relationen definiert, d.h. Relationen mit den gleichen Attribute.

Mengenoperatoren: Vereinigung $\cup$

speaker1				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010

$\cup$

speaker2				
id	sex	age	code	geolocation_id
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010

Mengenoperatoren: Vereinigung $\cup$

`speaker1` und `speaker2` sind *vereinigungsverträglich*, weil sie die gleichen Attribute haben.

Mengenoperatoren: Vereinigung $\cup$

`speaker1` und `speaker2` sind *vereinigungsverträglich*, weil sie die gleichen Attribute haben.

Ergebnis:

speaker				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010

Mengenoperatoren: Durchschnitt ∩

speaker1				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010

∩

speaker2				
id	sex	age	code	geolocation_id
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010

Mengenoperatoren: Durchschnitt ∩

Ergebnis:

speaker				
id	sex	age	code	geolocation_id
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010

Mengenoperatoren: Differenz \

speaker1				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010

\

speaker2				
id	sex	age	code	geolocation_id
48790245	m	51	GRE6	1044000
48790262	m	58	HAN5	1044010
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010

Mengenoperatoren: Differenz \

Ergebnis:

speaker				
id	sex	age	code	geolocation_id
48790244	f	57	GRE5	1044000

Relationale Operatoren

Die nächsten drei Operatoren sind die eigentlichen relationalen Operatoren. Sie definieren die Auswahl von Zeilen und Spalten einer Relation bzw. die Verknüpfung zweier Relationen.

Selektion $\sigma_{A_i \oplus v_i} R$ wählt alle Tupel der Relation R aus, die der Bedingung $A_i \oplus v_i$ genügen, mit $\oplus$ einem Vergleichsoperator und v_i einem Wert

Projektion $\pi_{A_i} R$ wählt aus der Relation R nur die Attribute A_i aus

Join-Operation

Mit dem *Verbund* oder *Join* werden Tabellen miteinander verknüpft. Die Ergebnistabelle bestet dabei aus allen Spalten beider Relationen.

Verbund, Join $R[X \oplus Y]Q$ verlängert ein Tupel aus R um ein Tupel aus Q wenn der Vergleich $\oplus$ von X und Y gilt (meist ist X ein Attribut aus R und Y eines aus Q).

Beim *equi-Join* ist der Vergleichsoperator das $=$, beim *natürlichen Join* werden doppelte Attribute aus der Ergebnisrelation gelöscht.

Relationale Operatoren: Selektion σ

"Alle Tupel in `speaker` mit `age = 18`."

$\sigma_{age=18} speaker$

speaker				
id	sex	age	code	geolocation_id
48790260	m	18	HAN3	1044010
48790259	f	18	HAN2	1044010
48790258	m	18	HAN1	1044010

Relationale Operatoren: Selektion σ

Die Bedingungen können komplex sein.

"Alle Tupel mit `age = 18 und sex = 'f'` in `speaker`."

$\sigma_{age=18 \wedge sex='f'} speaker$

speaker				
id	sex	age	code	geolocation_id
48790259	f	18	HAN2	1044010

Relationale Operatoren: Projektion π

"Nur die Attribute `sex`, `age` und `code` aus `speaker`"

$\pi_{sex,age,code}\, speaker$

speaker		
sex	age	code
f	57	GRE5
m	51	GRE6
m	58	HAN5
m	18	HAN3
f	18	HAN2
m	18	HAN1
f	54	HAN6
f	19	HAN4
m	19	HUS3
f	20	HUS1

Relationale Operatoren: Projektion π

"Die Attribute `sex` und `age` aus `speaker`"

$\pi_{sex,age} speaker$

speaker	
sex	age
f	57
m	51
m	58
m	18
f	18
f	54
f	19
m	19
f	20

Fällt etwas auf?

Relationale Operatoren: Projektion π

"Das Attribut `sex` aus `speaker`"

$\pi_{sex}\, speaker$

speaker
sex
f
m

Und jetzt? Warum hat die Ergebnisrelation nur zwei Tupel?

Relationale Operatoren: Projektion π

"Das Attribut `sex` aus `speaker`"

$\pi_{sex} speaker$

speaker
sex
f
m

Und jetzt? Warum hat die Ergebnisrelation nur zwei Tupel?

Richtig: Relationen sind Mengen, und Mengen enthalten keine Duplikate.

Relationale Operatoren: Join $[X \oplus Y]$

Verbindet zwei Relationen über eine Vergleichsoperation.

1. für jedes Tupel aus der ersten Relation suche alle Tupel aus der zweiten Relation, für die der Vergleich wahr ist
2. für jedes Tupel aus der zweiten Relation füge das erste Tupel vorne an und
3. schreibe das neue Tupel in die Ergebnisrelation

Relationale Operatoren: Join $[X \oplus Y]$

Verbindet zwei Relationen über eine Vergleichsoperation.

1. für jedes Tupel aus der ersten Relation suche alle Tupel aus der zweiten Relation, für die der Vergleich wahr ist
2. für jedes Tupel aus der zweiten Relation füge das erste Tupel vorne an und
3. schreibe das neue Tupel in die Ergebnisrelation

Fragen: was passiert, wenn der Vergleich immer wahr ist? Was passiert, wenn er nie wahr ist?

Relationale Operatoren: Join [$X \oplus Y$]

Joins verknüpfen Relationen üblicherweise über Fremdschlüssel.

"`speaker` und `geolocation` über `geolocation_id` in `speaker` und `id` in `geolocation` verknüpfen"

$speaker[geolocation\_id = id]geolocation$

speaker ⊕ *geolocation*									
s.id	s.sex	s.age	s.code	g.id	g.nuts	g.label	g.longitude	g.latitude	g.iso3166_2
48790244	f	57	GRE5	1044000	DE801	Greifswald	13.4086	54.0809	DE-MV
48790245	m	51	GRE6	1044000	DE801	Greifswald	13.4086	54.0809	DE-MV
48790262	m	58	HAN5	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790260	m	18	HAN3	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790259	f	18	HAN2	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790258	m	18	HAN1	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790263	f	54	HAN6	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790261	f	19	HAN4	1044010	DE929	Hannover	9.74959	52.3936	DE-NI
48790289	m	19	HUS3	1297516	DE927	Husum	9.051	54.477	DE-SH
48790287	f	20	HUS1	1297516	DE927	Husum	9.051	54.477	DE-SH

In der Ergebnisrelation wird das Fremdschlüssel-Attribut `geolocation_id` nicht angezeigt, weil es denselben Wert wie das Primärschlüssel-Attribut `id` in der Relation `geolocation` hat.

Relationale Operatoren: Schachtelung

"Alter und Code aller männlichen Sprecher"

$\pi_{age,code}(\sigma_{sex='m'}speaker)$

speaker	
age	code
51	GRE6
58	HAN5
18	HAN3
18	HAN1
19	HUS3

Relationale Operatoren: Schachtelung

Was liefert $\sigma_{sex='m'}(\pi_{age,code}speaker)$?

Relationale Operatoren: Schachtelung

Was liefert $\sigma_{sex='m'}(\pi_{age,code}speaker)$?

Richtig: $\emptyset$

Relationale Operatoren: Schachtelung

Was liefert $\sigma_{sex='m'}(\pi_{age,code}speaker)$?

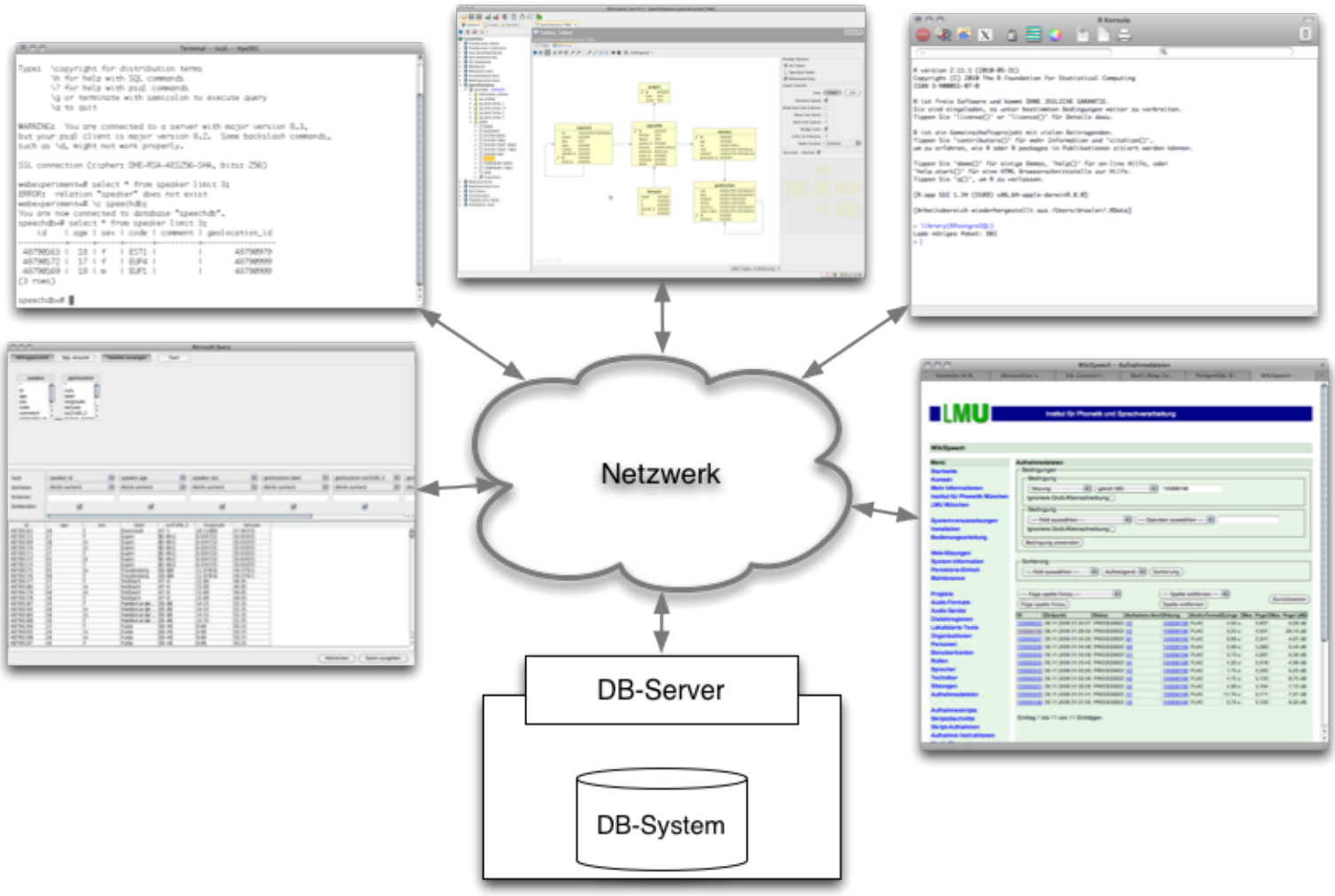
Richtig: $\emptyset$

Denn

- $\pi_{age,code}speaker$ wählt nur die Attribute age und code aus
- $\sigma_{sex='m'}$ gilt nie, weil es das Attribut sex nach der Projektion nicht mehr gibt

Konsequenz: Schachtelung der Operationen *immer* möglich, wenn die Projektion als letzte Operation ausgeführt wird.

Datenbank-Architektur



Üblicherweise ist ein Datenbanksystem als Client-Server-System installiert (Ausnahme: SQLite)

SQL Übersicht

SQL ist der Sprachstandard für relationale Datenbanksysteme

- basiert auf relationaler Algebra und Relationenkalkül
- lehnt sich an natürliches English an
- ist ein ISO-Standard
- wird laufend aktualisiert

Einleitung SQL

Datendefinition und Datenmanipulation sind getrennt

Datendefinition Anlegen, Verändern, Löschen der Datenbank*struktur*

Datenmanipulation Anlegen, Verändern, Löschen und Suchen von Datenbank*inhalten*

SQL betrachtet relationale Tabellen *nicht* als Mengen, sondern erlaubt Duplikate.

SQL Datendefinition

In der Regel darf nur der DB-Administrator Tabellen anlegen, ändern oder löschen

CREATE	legt eine neue Datenbank, Tabelle, Nutzer oder Index an
DROP	löscht eine Datenbank, Tabelle, Nutzer oder Index
ALTER	verändert eine Tabelle oder einen Index

Üblicherweise legt man zunächst eine Datenbank an und darin die relationalen Tabellen.

SQL CREATE

Anlegen einer Datenbank mit einer Tabelle

```
CREATE DATABASE speechdb;

CREATE TABLE speaker (
  id INT PRIMARY KEY,
  age INT,
  sex CHAR(1),
  code VARCHAR(10),
  comment VARCHAR(100),
  geolocation_id INT REFERENCES geolocation (id));
```

Das ';' schließt einen Befehl ab (in den meisten SQL-Terminals).

Datentypen

SQL kennt mindestens die folgenden Datentypen

smallint, int	kleine bzw. große ganze Zahlen
real, double precision	normale bzw. extra präzise Gleitkommazahlen
char(N), varchar(N)	Zeichenketten fester bzw. variabler Länge
date, time, timestamp	Datum, Uhrzeit, Zeitstempel

Darüberhinaus unterstützen die meisten Datenbanksysteme weitere Typen, z.B. `text`.

Grammatik von CREATE TABLE

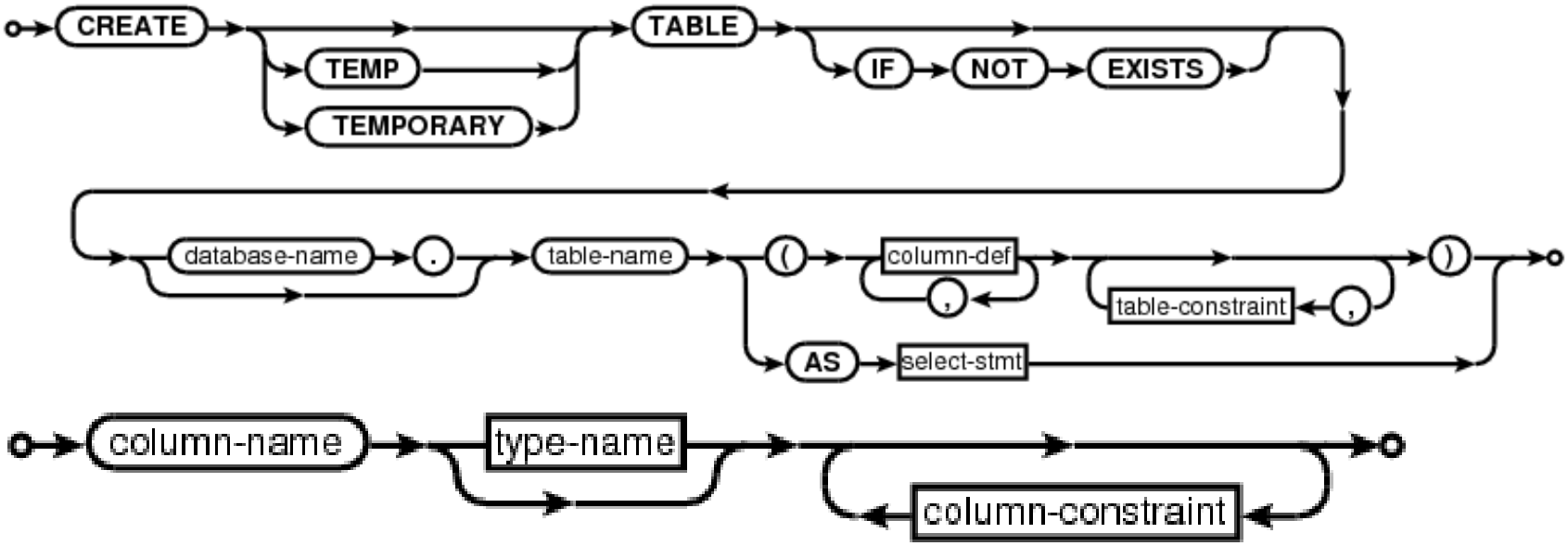
```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ]
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
...
```

aus: online-Hilfe im SQL Terminal von postgreSQL

Grafische Darstellung der Grammatik



aus: SQLite Dokumentation
`http://www.sqlite.org/lang_createtable.html`

SQL Datenmanipulation

Einfügen, Ändern und Löschen von Datensätzen

- INSERT fügt Einträge in eine Tabelle ein
- UPDATE verändert Einträge in einer Tabelle
- DELETE löscht Einträge aus einer Tabelle

Üblicherweise mit einer Bedingung in der WHERE-Klausel kombiniert

Transaktion

Eine Transaktion führt eine Datenbank von einem konsistenten wieder in einen konsistenten Zustand

- `insert`, `update` und `delete` sind gefährlich, denn sie verändern die Datenbank
- das Einbetten dieser Befehle in eine Transaktion erlaubt es, diese Aktionen rückgängig zu machen
- `begin` startet eine Transaktion
- `commit` schließt die Transaktion ab, die Änderungen werden bestätigt
- `rollback` macht alles bis zum Beginn der Transaktion rückgängig

INSERT Befehl

```
insert into speaker (id, age, sex)
values (11, 25, 'm');
```

fügt einen 25-jährigen männlichen Sprecher mit der eindeutigen ID 11 ein.

- stehen nach dem Tabellennamen keine Attribute, dann muss für jede Spalte ein Wert vorgesehen sein
- wenn Attribute nicht null sein dürfen, dann müssen für sie auch Werte vorgesehen sein

UPDATE Befehl

```
update speaker set age = 35 where id = 11;
```

ändert das Alter des Sprechers mit der ID 11 auf 35 Jahre

DELETE Befehl

```
delete from speaker where id = 11;
```

löscht alle Sprecher aus der Datenbank, die eine id von 11 haben.
Wie viele sind das?

SQL Abfragen

Abfragen berechnen Ergebnisse in der Datenbank. Sie bestehen aus den folgenden Klauseln

SELECT	Liste der zurückzugebenden Spalten
FROM	Liste der verwendeten Tabellen
WHERE	mit logischen Operatoren verknüpfte Bedingungen
ORDER BY	Sortierkriterien für die Ergebnistabelle

Alle Klauseln außer `SELECT` sind optional.

Syntax von SQL Abfragen

Die folgende *vereinfachte* und *unvollständige* Grammatik definiert die Syntax von SQL Abfragen, wie sie im Kurs verwendet werden.

```
SELECT [DISTINCT] * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ ORDER BY expression [ ASC | DESC ] [, ...] ]
    [ LIMIT count ]

where from_item is:

    table_name [ alias ]
    [ JOIN from_item ON join_condition ]

and expression is

    [[ alias | table_name ] . ] column_name | function ( arguments )
```

Prüfen Sie, ob die Abfragen in diesen Unterlagen der Grammatik entsprechen!

Einfache Abfragen

```
select * from speaker limit 5;

    id    | age | sex | code | comment | geolocation_id
----------+-----+-----+------+---------+----------------
 48790163 |  18 | f   | EST1 |         |       48790979
 48790172 |  17 | f   | EUP4 |         |       48790999
 48790169 |  18 | m   | EUP1 |         |       48790999
 48790170 |  17 | m   | EUP2 |         |       48790999
 48790171 |  17 | f   | EUP3 |         |       48790999
(5 rows)
```

gibt die ersten 5 Einträge der Tabelle `speaker` aus.

- `*` ist eine Abkürzung für 'alle Spalten'
- `limit 5` begrenzt die Anzahl Zeilen im Ergebnis auf 5

Einfache Abfragen

```
select id, age, sex from speaker limit 5;

    id     | age | sex
-----------+-----+-----
 48790163 |  18 | f
 48790172 |  17 | f
 48790169 |  18 | m
 48790170 |  17 | m
 48790171 |  17 | f
(5 rows)
```

gibt die Spalten `id`, age und `sex` der ersten 5 Einträge der Tabelle `speaker` aus.

Einfache bedingte Abfragen

```
select * from speaker
where age between 18 and 22 or sex = 'f'
```

```
    id     | age | sex | code | comment | geolocation_id
-----------+-----+-----+------+---------+----------------
 48790163 |  18 | f   | EST1 |         |       48790979
 48790172 |  17 | f   | EUP4 |         |       48790999
 48790169 |  18 | m   | EUP1 |         |       48790999
 48790171 |  17 | f   | EUP3 |         |       48790999
...
 48790314 |  17 | f   | KAS2 |         |        1044771
(1391 rows)
```

gibt alle Sprecher aus, die entweder zwischen 18 und 22 oder weiblich sind.

Aliasnamen

Im FROM-Teil kann eine Relation einen Aliasnamen bekommen.

```
select spk.id, spk.age, spk.sex
from speaker spk
where spk.age between 18 and 22 and spk.sex = 'f'
```

Damit kann man

- Attributnamen disambiguieren
- Join-Abfragen über derselben Tabelle formulieren
- Abfragen kürzer und verständlicher schreiben

Aliasnamen können frei gewählt werden. Sinnvoll sind 'sprechende' Namen, die helfen, die Abfrage zu verstehen.

Aliasnamen beim Join

Die Aliasnamen `spk` und `geo` disambiguieren das Attribut `id`, das in beiden Relationen vorkommt.

```
select spk.id, spk.sex, spk.age, geo.iso3166_2
from speaker spk
join geolocation geo on spk.geolocation_id = geo.id;
```

```
    id     | sex | age | iso3166_2
-----------+-----+-----+-----------
 48790163 | f   |  18 | AT-1
...
 48790314 | f   |  17 | DE-HE
```

Ohne Aliasnamen gibt die Datenbank die Fehlermeldung zurück, das Attribut `id` sei nicht eindeutig zuzuordnen. Mit den Aliasnamen ist diese eindeutige Zuordnung möglich.

Aliasnamen bei Joins über derselben Tabelle

Die Tabelle `geolocation` enthält unterschiedliche geo-referenzierte Einträge.

```
select poi.label, city.label
from geolocation poi
join geolocation city on poi.reference = city.id
where city.label = 'München'
```

gibt alle Einträge zurück, die auf den Ortseintrag von München verweisen.

Hierbei wird zweimal in der Tabelle `geolocation` gesucht: auf der Ebene der touristischen Attraktionen (`poi`) und der Ebene der Stadt (`city`). Eine solche Abfrage ist *nur* mit Aliasnamen zu formulieren.

Aliasnamen zur besseren Lesbarkeit

Neben der Disambiguierung verbessern Aliasnamen auch die Lesbarkeit von Abfragen.

- `poi` für engl. *point of interest* weist auf touristische Ziele
- `city` auf Städteeinträge hin

Aus der natürlichen Zuordnung von touristischen Zielen zu Städten kann man die Abfrage leichter verstehen. Sie sucht alle touristischen Ziele, die der Stadt München zugeordnet sind:

```
...poi.reference = city.id...
```

Umbenennung von Spaltentiteln

Die Lesbarkeit kann man durch eine Umbenennung der Spaltentitel noch verdeutlichen: mit `AS` hinter einem Attributnamen im Select-Teil kann der Spaltentitel gesetzt werden:

```
select poi.label as tour_ziel, city.label as stadt
from geolocation poi
join geolocation city on poi.reference = city.id
where city.label = 'München';
```

```
   tour_ziel    |  stadt
----------------+---------
 St. Ludwig     | München
 LMU München    | München
 Marienplatz    | München
 Hackerbrücke   | München
 Allianz Arena | München
(5 rows)
```

Beispiele

Die Tabelle `segment` enthält Annotationsdaten. Die Annotationsebenen sind im Attribut `tier` gespeichert. Das Attribut hat die Werte:

`Transkript` für den Wortlaut einer Äußerung

- `ORT` für die einzelnen Wörter einer Äußerung
- `KAN` für die kanonische Aussprache von Wörtern
- `MAU` für die automatisch bestimmten phonetischen Segmente der Äußerung

Die Tabelle `segment` ist über den Fremschlüssel `signalfile_id` mit der Tabelle `signalfile` verknüpft.

Suche nach phonetischen Segmenten

Gib den Dateinamen, die Annotationsebene, die Position und Dauer, das Wort sowie das Phonem der Verknüpfung von Segment und Signaldatei wieder.

```
select sig.filename, mau.tier, mau.position, mau.begin, mau.duration,
  ort.label as word, mau.label as phoneme

from signalfile sig
  join segment mau on sig.id = mau.signalfile_id
  join links l on l.lfrom = mau.id
  join segment ort on l.lto = ort.id

where mau.tier = 'MAU'
  and ort.tier = 'ORT'
  and mau.label <> '<p:>'
  and mau.position >= 0
order by sig.filename, mau.begin
limit 20;
```

Suche nach Wörtern, die mit /aI/ beginnen

```
select sig.filename, mau.tier, mau.position, mau.duration,
  ort.label as word, mau.label as phoneme

from signalfile sig
  join segment mau on sig.id = mau.signalfile_id
  join links l on l.lfrom = mau.id
  join segment ort on l.lto = ort.id

where mau.tier = 'MAU'
  and ort.tier = 'ORT'
  and mau.label = 'aI'
  and mau.position = 0

order by sig.filename, mau.begin
limit 20;
```

Die leeren Zeilen verbessern die Lesbarkeit der Abfrage.

Funktionen

die folgenden Funktionen bzw. Befehle sind sehr nützlich

`x between n and m` im WHERE-Teil prüft, ob der Wert `x` zwischen `n` und `m` liegt

`x like 'abc%'` im WHERE-Teil prüft, ob der Wert `x` mit 'abc' beginnt

`x in (a,b,...,n)` im WHERE-Teil prüft, ob x in der Menge (a, b, ..., n) vorkommt

`distinct` im SELECT-Teil eliminiert Duplikate aus dem Ergebnis

Darüberhinaus gibt es viele weitere numerische und Textfunktionen.

Aggregatfunktionen

Aggregatfunktionen berechnen statistische Werte aus Relationen

- z.B. Anzahl, Summe, Durchschnitt, usw.
- $f : R \rightarrow \mathbb{R}$
- sind in der Praxis sehr nützlich
- gehen über die relationale Algebra oder den Relationenkalkül hinaus

Sie können *nicht einfach* wie andere Funktionen (z.B. `substring()`) verwendet werden.

SQL-Aggregatfunktionen

`count(*)`	zählt die Tupel
`sum(x)`	summiert die Attributwerte `x`
`avg(x)`	berechnet den Durchschnittswert von `x`
`min(x)`	berechnet das Minimum,
`max(x)`	das Maximum des Attributs `x`

`sum()` und `avg()` sind nur für numerische Attribute definiert, `count()` kann um den Befehl `distinct` erweitert werden, um die Anzahl Zeilen ohne Duplikate zu zählen.

Aggregatfunktionen in SQL: Syntax

stehen im SELECT-Teil der Abfrage

- sind auch *normale* Attribute im SELECT-Teil, dann
  - *muss* die Abfrage um einen `GROUP BY`-Teil
  - *kann* die Abfrage um einen `HAVING`-Teil

  erweitert werden

`GROUP BY` gruppiert die Ergebnisrelation nach Attributen, `HAVING` wendet eine Bedingung auf die Gruppen an.

Aggregatfunktion Beispiel

Wie viele Sprecher gibt es in der Datenbank?

```
select count(*) from speaker;
```

```
 count
-------
  1956
(1 row)
```

GROUP BY ist nicht notwendig, weil neben der Aggregatfunktion `count()` kein weiteres Attribut verwendet wird.

Aggregatfunktion Beispiel

Wie viele männliche und weibliche Sprecher gibt es in der Datenbank?

```
select sex, count(*) from speaker group by sex;
```

```
 sex | count
-----+-------
 f   |  1015
 m   |   936
     |     5
(3 rows)
```

GROUP BY ist notwendig, weil neben der Aggregatfunktion `count()` auch noch das Attribut `sex` verwendet wird.

Phonemstatistik

Erstelle eine Dauerstatistik des Diphthongs /aI/ in den Wörtern 'eins', 'zwei' und 'drei'

```
select ort.label as word, mau.label as phoneme, count(mau.label),
 avg(mau.duration)::int
from segment ort
 join links l on l.lto = ort.id
 join segment mau on l.lfrom = mau.id
where ort.tier = 'ORT'
 and mau.tier = 'MAU'
 and ort.label in ('eins', 'zwei', 'drei')
 and mau.label = 'aI'
group by ort.label, mau.label
order by avg(mau.duration);
```

Was kann man über die Dauern von /aI/ sagen?

Virtuelle Tabellen

Häufig muss die gesuchte Information aus mehreren Tabellen ermittelt werden. Das ist aufwändig und fehleranfällig.

- *Virtuelle Tabellen* fassen (komplexe) Abfragen unter einem eigenen Namen zusammen
- verhalten sich (fast) wie reale Tabellen
- werden aber erst im Moment der Abfrage berechnet
- und erlauben eine feine Kontrolle der Sichtbarkeit von Attributen

Sie können den Zugriff auf die Daten regeln und den Schreibaufwand bei Abfragen ganz erheblich reduzieren.

Virtuelle Tabellen in SQL

- In SQL heißt eine virtuelle Tabelle `View`
- Die Definition einer View besteht aus zwei Teilen
  - `CREATE VIEW view_name AS`
  - `SELECT`-Abfrage
- In einer Abfrage kann eine View überall dort stehen, wo auch Relationen stehen können

Beispiel

Die View speaker_phonemes kombiniert Attribute aus 4 Relationen:

```
create view speaker_phonemes as
select spk.sex, spk.age, geo.iso3166_2 AS state, geo.label AS city,
 mau.label AS phoneme, mau.begin, mau.duration, ort.label AS word
from speaker spk
 join signalfile sig ON sig.speaker_id = spk.id
 join geolocation geo ON geo.id = spk.geolocation_id
 join segment mau ON mau.signalfile_id = sig.id
 join links ls ON ls.lfrom = mau.id
 join segment ort ON ls.lto = ort.id
where mau.tier = 'MAU'
 and ort.tier = 'ORT'
 and not mau.label = in ('<p:>', '<usb>');
```

Was macht diese View?

Beispiel der Verwendung

```
select * from speaker_phonemes;
```

sex	age	state	city	phoneme	begin	duration	word
m	19	DE-SH	Brunsbüttel	aI	134945	882	drei
m	19	DE-SH	Brunsbüttel	d	133622	661	drei
m	19	DE-SH	Brunsbüttel	r	134284	660	drei
f	14	DE-BW	Esslingen	aI	94153	2204	deiner
f	14	DE-BW	Esslingen	n	96358	1322	deiner
f	14	DE-BW	Esslingen	6	97681	1983	deiner
f	14	DE-BW	Esslingen	d	92830	1322	deiner
m	18	DE-SN	Plauen	n	104076	660	null
m	18	DE-SN	Plauen	l	106060	5291	null
m	18	DE-SN	Plauen	U	104737	1322	null
m	15	DE-BB	Cottbus	i:	110029	2865	vier
...							

Verwendung mit Aggregatfunktion

Minimale, Durchschnitts- und maximale Dauer der Diphthonge /OY, aU, aI/ nach Geschlecht

```
select phoneme, min(duration), avg(duration)::int, max(duration), sex
from speaker_phonemes
where phoneme in ('OY', 'aI', 'aU')
   and sex in ('m', 'f')
   and duration > 0
group by sex, phoneme
order by phoneme, sex;
```

```
 phoneme | min | avg  |  max  | sex
---------+-----+------+-------+-----
 OY      | 881 | 3263 |  8598 | f
 OY      | 881 | 3038 |  6393 | m
 aI      | 881 | 4484 | 29546 | f
 aI      | 881 | 4180 | 34176 | m
 aU      | 881 | 2872 | 10363 | f
 aU      | 881 | 2936 | 11024 | m
(6 rows)
```

Komplexe Abfragen mit Aggregatfunktionen

Suche die Grundfrequenz des Sprechers mit $id = 48790454$

```
select spk.id, spk.age, spk.sex, sig.filename, min(p.f0), avg(p.f0), max(p.f0)
from speaker spk
join signalfile sig on sig.speaker_id = spk.id
join pitch p on p.signalfile_id = sig.id
where spk.id = 48790454
group by spk.id, spk.age, spk.sex, sig.filename order by sig.filename;
```

Das Ergebnis ist

```
    id    | age | sex | filename   | min |    avg     | max
----------+-----+-----+------------+-----+------------+-----
 48790454 |   0 | m   | MNH3_1_12 |  52 | 123.75714 | 698
(1 row)
```

Zwischenstand

Mit den bisherigen SQL-Abfragen können wir

- Zeilen und Spalten aus Tabellen auswählen
- mit `JOIN` Tabellen miteinander verknüpfen
- Aggregatfunktionen für elementare Statistik ausführen

SQL ist im Wesentlichen eine deklarative Sprache, d.h. man gibt an, *was* man sucht, und nicht, *wie*.

Rekursion

Eine *rekursive* Funktion ist eine Funktion, deren Definition sich selbst aufruft.

```
function fakultaet (n) {
  if (n === 1) {
    return (1);
  } else {
    return (n * fakultaet (n-1));
  }
}
```

Diese Definition von `fakultaet()` verwendet einen Aufruf von sich selbst in der drittletzten Zeile.

Aufbau rekursiver Definitionen

Eine rekursive Definition hat generell den folgenden Aufbau:

Terminalfall hier gibt man an, unter welcher Bedingung die Rekursion beendet ist

Rekursionsschritt hier wird die rekursive Verwendung der Definition festgelegt; in der Regel wird dabei die Definition mit einem modifizierten Argument aufgerufen.

Die Modifikation des Arguments muss dazu führen, dass irgendwann der Terminalfall eintritt. Im Beispiel `fakultaet(n)` tritt der Terminalfall ein, wenn `n === 1` gilt. In jedem rekursiven Schritt wird daher das Argument `n` um 1 verkleinert, damit irgendwann 1 erreicht ist.

Anwendungsbeispiel Rekursion

Mit der Aufgabe, eine Verbindung in einem Netzwerk, z.B. der U-Bahn, zu finden, kann man Rekursion anschaulich darstellen.

```
function verbindung (a, b) {
  if (a === b) {
    return ("");
  else {
    strecke (a, c);
    return (a + verbindung (c, b));
  }
}
```

Umgangssprachlich: ich suche eine Verbindung von a nach b. Wenn a gleich b ist, dann bin ich da und brauche keine Route zurückgeben. Ansonsten suche ich nach einer Strecke von a und einem direkt verbundenen c, füge das a zur Route hinzu, und suche nach einer Verbindung von c nach b.

Geht das mit den bisherigen SQL Befehlen?

Jein: wenn ich weiß, dass es eine Verbindung über n einzelne Strecken gibt, dann kann ich die Abfrage als n Joins formulieren. Das haben wir z.B. bei der Tabellen-internen Verknüpfungen von `geolocation` gemacht.

Wenn wir nicht wissen, wieviele Schritte es sind, dann können wir das mit den bisherigen Befehlen *nicht* formulieren!

Rekursion in SQL

Rekursion ist in der relationalen Algebra nicht erlaubt. SQL erlaubt rekursive Abfragen, geht hier also über die relationale Algebra hinaus. In SQL sehen rekursive Abfragen vereinfacht dargestellt so aus:

```
WITH RECURSIVE with_query AS (
SELECT terminal_case
UNION recursive_step)
SELECT columns FROM from_with
```

Die eigentliche rekursive Definition steht hier innerhalb der Klammern; der erste `SELECT`-Ausdruck ist der Terminalfall, der Teil nach dem `UNION ALL` ist der Rekursionsschritt. `UNION` ist der SQL Befehl für die mengenmäßige Vereinigung von Tabellen.

Beispiel: finde alle zu Bayern gehörenden Einträge

Die Tabelle `geolocation` enthält Angaben zu Ländern, Bundesländern, Städten und touristischen Zielen, die miteinander verknüpft sind. So ist die LMU in München, München in Bayern, und Bayern in Deutschland.

```
with recursive g(id, label) as (
  select id, label from geolocation where label = 'Bayern'
  union select geo.id, geo.label from g, geolocation geo
    where g.id = geo.reference
)
select * from g;
```

Umgangssprachlich: suche zunächst alle Einträge, bei denen das Land 'Bayern' ist (*Terminalfall*), dann vereinige dieses mit allen Einträgen, die mit Bayern verknüpft sind – und vereinige diese wiederum mit allen, die damit verknüpft sind (*Rekursionsschritt*).

Rechteverwaltung

SQL erlaubt die Vergabe von Rechten, sog. *privileges* an Datenbank-Objekten an Nutzer

Rechte	`SELECT, UPDATE, INSERT, DELETE, ALL,...`
DB-Objekte	Tabellen, Views, Attribute, ...
Nutzer	Nutzergruppen, einzelne Benutzer

Die Rechtevergabe kann je nach Datenbanksystem unterschiedlich ausfallen. sqlite3 z.B. kennt keine Benutzer, PostgreSQL hat immer den Superuser postgres und meist weitere Nutzer mit geringeren Privilegien, usw.

Rechteverwaltung in SQL

Mit `GRANT` und `REVOKE` werden Rechte vergeben bzw. wieder widerrufen

- `GRANT privileges ON tables TO users`
- `REVOKE privileges ON tables FROM users`

`GRANT SELECT ON SEGMENT TO DB_KURS` gibt der Gruppe DB_KURS Leserechte an der Tabelle SEGMENT

Weitergabe von Rechten

Ein Nutzer kann seine Rechte auch weitergeben, aber nur, wenn

- diese nicht weitgehender sind als seine eigenen
- ihm dies vom Superuser nicht verboten wurde

Der Superuser selbst hat alle Rechte und kann allen anderen Benutzern Rechte gewähren oder sie widerrufen.

Literaturangaben


B. Brügge and A. Dutoit.
Object-Oriented Software Engineering.
Pearson Education International, 2nd edition, 2004.


G. Booch, J. Rumbaugh, and H. Jacobson.
The Unified Modeling Language.
Addison Wesley, Reading, MA, 1998.


P. S. Chen.
The Entity-Relationship Model - Toward a Unified View of Data.
Transactions on Database Systems, Vol. 1(No. 1), 1976.


E. F. Codd.
A relational model for large shared data banks.
Communications of the acm, 13(6):377–387, 1970.


R. Elmasri and S. Navathe.
Grundlagen von Datenbanksystemen.
Pearson Studium, 2002.