Datenbanken 1

Einführung

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

database research group
http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 6. März 2018

Inhalt

Inhalt

Alle Infos zu Vorlesung und Proseminar:

http://dbresearch.uni-salzburg.at/teaching/2018ss/db1/

Inhaltsübersicht Datenbanksysteme/1

1. **Einführung in Datenbanksysteme**
   - Gebiet, Terminologie, Datenbanksysteme
   - Kapitel 1 in Kemper und Eickler
2. **Datenbankentwurf**
   - Datenbank Entwurf, ER Modell
   - Kapitel 2 in Kemper und Eickler
3. **Das relationale Modell**
   - Relationales Modell, relationale Algebra
   - Kapitel 3 in Kemper und Eickler

Inhaltsübersicht Datenbanksysteme/2

4. **SQL**
   - Schemadefinition, Datenmanipulation, Datenabfrage, JDBC
   - Kapitel 4 + 5 in Kemper und Eickler
5. **Relationale Entwurfstheorie**
   - Funktionale Abhängigkeit, Normalformen, Zerlegung von Relationen
   - Kapitel 6 in Kemper und Eickler
6. **Vertiefende Konzepte**
   - Indexstrukturen, Transaktionen
   - Auszüge aus Kapitel 7 und 9 in Kemper und Eickler

Inhalt

Daten sind wertvoll

- Unternehmen: Information über Kunden, Lieferanten, Waren, Bestellungen, Buchhaltung, Marktstudien, usw.
- Verwaltung: Meldedaten, Führerschein, Strafregister, Krankenversicherung, etc.
- Wissenschaft: Messdaten, fachspezifische Datensammlungen (z.B. Protein-Eigenschaften), Analyseergebnisse
- Privat: Telefonnummern, Email-Kontakte, Online-Zugänge, Familienfotos, MP3-Sammlung
- ...

Daten verwalten ist schwierig

Anforderungen an Datenverwaltung:

- Daten müssen (möglichst schnell) zugänglich sein
- Einzelne Fakten müssen verknüpft werden können
- Daten müssen geändert werden können
- Mehrere Benutzer sollen gleichzeitig lesen und ändern können
- Daten müssen konsistent bleiben
- Daten dürfen nicht verloren gehen
- Daten müssen vor unberechtigtem Zugriff geschützt sein

Ansatz ohne Datenbanksystem:

- Daten in isolierten Dateien speichern
- nach Bedarf Programme zum Einfügen, Auslesen und Ändern der Daten schreiben

Probleme der Datenverwaltung/1

- Redundanz und Inkonsistenz
  - Redundanz: ein Fakt ist mehrmals gespeichert
  - bei Änderungen müssen alle Kopien geändert werden
  - Inkonsistenz: nicht alle Kopien wurden geändert, d.h., es existieren widersprüchliche Daten
  - isolierte Dateien: habe ich alle relevanten Dateien geändert?
  - Bespiel: Adresse wurde nur im Fachbereich geändert, auf Universitätsebene hingegen nicht.
  - Ziel: Redundanz kontrollieren und Inkonsistenz vermeiden.
- Beschränkte Zugriffsmöglichkeiten
  - Verknüpfungen logisch verwandter Daten erzeugt deutlichen Mehrwert
  - isolierte Dateien: verschiedene Verwalter und Formate, eigenes Programm für jede Verknüpfung
  - Bespiel: freien Hörsaal für Datenbank-Vorlesung finden (Hörsäle mit Kapazität, Veranstaltungskalender, Teilnehmerzahl der Vorlesung)
  - Ziel: Alle Daten im System lassen sich flexibel miteinander verknüpfen.

Probleme der Datenverwaltung/2

- Integritätsverletzung
  - Änderungen können zu unerlaubten Zuständen (aus der Sicht der Anwendung) führen
  - oft sind Verknüpfungen zwischen Daten erforderlich, um Integritätsverletzungen zu entdecken
  - Beispiel: Student schreibt sich in Bachelor-Projekt ein, bevor er genug Kreditpunkte gesammelt hat.
  - Ziel: Integritätsregeln formulieren und Verletzungen nicht zulassen.
- Sicherheitsprobleme
  - Nicht alle Benutzer sollen alle Daten sehen.
  - Nur ausgewählte Benutzer sollen bestimmte Daten ändern dürfen.
  - Granularität: Informationsteil, auf den sich der Zugang bezieht, z.B. ganzes Objekt, gewisse Eigenschaften des Objektes
  - Beispiel: Studenten dürfen ihre eigenen Noten sehen, aber nicht die anderer. Eigene Noten dürfen nicht verändert werden.
  - Ziel: Lese- und Schreibrechte flexibel und in feiner Granularität an Benutzer vergeben.

Probleme der Datenverwaltung/3

- Probleme des Mehrbenutzerbetriebs
  - Viele Anwender greifen zugleich auf Daten zu.
  - Beispiel: Flugreservierungssystem
  - Keine Kontrolle: Unerwünschte Anomalien, z.B. “lost updates” = meine Änderungen werden von einem anderen Benutzer überschrieben
  - Dateisysteme bieten nur sehr rudimentäre Kontrollmechanismen, z.B., Sperren auf Dateiebene
  - Rudimentäre Kontrolle: Ineffizient, ein einziger Benutzter kann Datei blockieren.
  - Ziel: Effizienter Mehrbenutzerbetrieb ohne Anomalien.
- Umgang mit Fehlern / Datenverlust
  - Verlust von Daten kann für Unternehmen existenzbedrohend sein.
  - Dateisysteme bieten Backups, aber alles nach Backup geht verloren.
  - isolierte Dateien: Konsistenz zwischen Dateien ist im Fehlerfall nicht garantiert, da sich Dateien unabhängig ändern können
  - Beispiel: Stromausfall oder Systemabsturz während Bankomatbehebung
  - Ziel: Garantien gegen Datenverlust auch im Fehlerfall

Probleme der Datenverwaltung/4

- Effizienz
  - Große Datenmengen erfordern effiziente Algorithmen für Suche, Verknüpfung und Änderung.
  - isolierte Dateien erfordern individuelle Programme für einzelne Anfragen und/oder Datentypen.
  - sehr aufwändig und möglicherweise ineffizient, da die Wahl der Algorithmen von den Daten abhängt, die sich ändern können
  - Ziel: Automatisch effiziente Algorithmen in Abhängigkeit von Anfrage und Daten wählen.
- Hohe Entwicklungskosten
  - Zumindest einem Teil oben genannter Probleme muss sich jeder Anwendungsentwickler stellen.
  - Rad ständig neu erfinden ist zeit- und kostenintensiv
  - Ziel: Komfortable Schnittstelle, die Datenverwaltungsprobleme transparent löst.

Warum Datenbankverwaltungssysteme?

- DBMS lösen Probleme der Datenverwaltung:
  - Unkontrollierte Redundanz wird vermieden.
  - Daten lassen sich flexibel miteinander verknüpfen.
  - Definierte Integritätsregeln können erzwungen werden.
  - Flexible Vergabe von Benutzerrechten.
  - Effiziente Mehrbenutzerkontrolle vermeidet Anomalien.
  - Ausgefeilte Recovery-Komponente schützt vor Datenverlust.
  - Anfrageoptimierung sorgt transparent für effiziente Ausführung.
- Fast alle Unternehmen verwenden Datenbanksysteme, weil es kaum eine Alternative gibt.

Datenbankanwendungen

- Traditionelle Anwendungen:
  - Datenbanken mit Zahlen und Wörtern
- Neuere Anwendungen:
  - Multimedia Datenbanken
  - Geographische Informationssysteme (GIS)
  - Data Warehouses
  - Echtzeit Datenbanken
  - Aktive Datenbanken
  - Viele andere Anwendungen
- Beispiele:
  - Banken (Konten)
  - Unternehmen (Lager, Verkauf)
  - Reservierungssysteme
  - Universität (Studenten, Vorlesungen, Räume)
  - Online Verkäufe (www.amazon.com)
  - Online Zeitungen (www.salzburg.com)

Wann braucht man kein DBMS?

- Hauptgründe gegen DBMS:
  - hohe Anfangsinvestition und möglicherweise zusätzlicher Hardware-Bedarf
  - Overhead für Allgemeinheit, Sicherheit, Mehrbenutzerkontrolle, Recovery, Integrationskontrolle
- DBMS möglicherweise nicht nötig, wenn:
  - einfache Datenbank und Anwendung, die klar definiert ist und sich voraussichtlich nicht ändern wird
  - kein Mehrbenutzerbetrieb
- DBMS nicht geeignet:
  - zwingende Echtzeitanforderungen, die DBMS nicht garantieren kann
  - Daten können aufgrund ihrer Komplexität nicht (nur schwer) modelliert werden
  - spezielle Operationen, die von DBMS nicht unterstützt werden

Datenbankforschung

- Konferenzen
  - SIGMOD – seit 1975
  - VLDB – seit 1975
  - ICDE – seit 1985
  - EDBT – seit 1988
- Zeitschriften
  - ACM Trans. on Database System (TODS) – seit 1976
  - The VLDB Journal (VLDBJ) – seit 1992
  - IEEE Trans. on Knowledge and Data Engineering (TKDE) – seit 1989
  - Information Systems (IS) – seit 1975
- DBLP Bibliographie (Michael Ley, Uni Trier, Germany)
  - ursprünglich für Datenbankforschung, jetzt allgemein Informatik
  - `http://dblp.uni-trier.de/db/`
- DBWorld Mailing Liste
  - `http://www.cs.wisc.edu/dbworld/`

Produkte

- Kommerzielle Produkte
  - Oracle
  - Microsoft SQL Server
  - IBM DB2
  - Teradata
  - Sybase Adaptive Server Enterprise
  - IBM Informix
  - PC "DBMSs": Access, dBase, ...
  - ...
- Open Source Produkte
  - PostgreSQL
  - MySQL
  - MonetDB
  - ...

Wir verwenden PostgreSQL für die Übungen.

Zusammenfassung

- Funktionierende Datenverwaltung ist "mission critical"
- Datenverwaltung wirft Probleme auf:
  - Konsistenz
  - effizienter und flexibler Zugriff
  - Integrität
  - Sicherheit
  - Mehrbenutzerbetrieb
  - Datenverlust
- Datenbanksysteme lösen Probleme transparent für Benutzer
- etabliertes und aktives Forschungsgebiet seit 40 Jahren
- Milliardenumsatz mit Datenbankprodukten

Inhalt

Grundlegende Definitionen/1

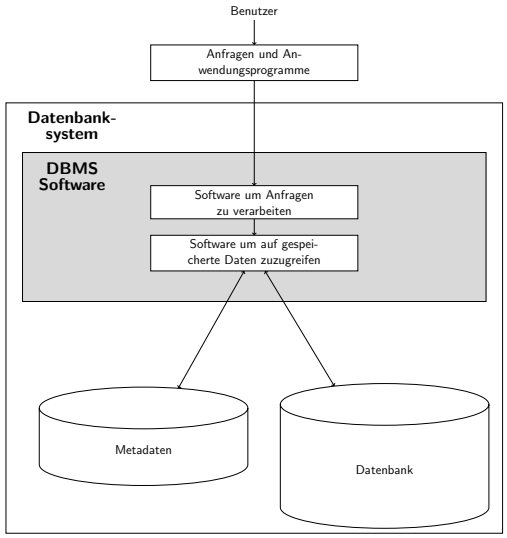
Über **Daten**, **Information** und **Wissen**:

- **Daten** sind Fakten die gespeichert werden können:
  - Buch(Herr der Ringe, 3, 10)
- **Information** = Daten + Bedeutung
  - Buch:
    - Titel = Herr der Ringe,
    - Band = 3,
    - Preis EUR = 10
- **Wissen** = Information + Anwendung

Grundlegende Definitionen/2

- **Mini-Welt**: Jener Teil der realen Welt der uns interessiert
- **Daten**: Bekannte Fakten über die Miniwelt die gespeichert werden können
- **Datenbank (DB)**: Eine Menge von Daten die miteinander in Beziehung stehen
- **Metadaten**: Information über die Struktur einer Datenbank (selbst als Datenbank organisiert)
- **Datenbankverwaltungssystem (DBMS, database management system)**: Ein Software Paket das die Erstellung, Pflege und Abfrage von Datenbanken unterstützt
- **Datenbanksystem (DBS, database system)**: DBMS + DB + Metadaten

Grundlegende Definitionen/3



Datenmodel

- "Infrastruktur" zur Modellierung der realen Welt:
  - **Datendefinitionssprache**: beschreibt Datenobjekte und Integritätsbedingungen
  - **Datenmanipulationssprache**: anwendbare Operatoren und deren Wirkung
- Analog zu Programmiersprache:
  - Typenkonstruktoren (Typendefinition)
  - Sprachkonstrukte (if-then, Schleifen, etc.)

DDL und Schema

- Datendefinitionssprache (DDL, data definition language) beschreibt:
  - **Schema**: Struktur der Datenobjekte (Typen, Gruppierung elementarer Typen) und Beziehung zwischen den Datenobjekten
  - **Integritätsbedingungen**: Einschränkung der zulässigen Daten; müssen zu jedem Zeitpunkt erfüllt sein
- Datenbankverzeichnis (Katalog): speichert Metadaten
  - Schema und Integritätsbedingungen
  - weitere Metadaten wie Zugriffsrechte, Statistiken über Datenverteilung

Haupteigenschaften des Datenbankansatzes/3

Beispiel eines Datenbankverzeichnisses (stark vereinfacht):

Tabellen

RelationName	NrOfColumns
Studenten	4
Vorlesungen	4
Module	5
Notenblätter	3
Voraussetzungen	2

Spalten

ColumnName	DataType	BelongsToRelation
Name	CHARACTER(30)	Studenten
StudentNr	CHARACTER(4)	Studenten
Class	INTEGER(1)	Studenten
...	...	...

- PostgreSQL 9.2: 72 Objekte im Datenbankkatalog
- Oracle: über 1000 Objekte im Datenbankkatalog

Datenmanipulationssprache

- Die Datenmanipulationssprache (DML, data manipulation language) besteht aus:
  - **Anfragesprache**: beantwortet Anfragen, lässt Daten unverändert
  - **eigentliche Manipulationssprache**: einfügen, löschen, ändern von Daten

Anfragesprachen

- Sprache um Information aus der Datenbank zu holen
- Kategorien von Sprachen:
  - **Imperativ**[1]: spezifiziert **wie** etwas gemacht wird; kann als Grundlage für die Anfrageoptimierung verwendet werden (weil das Vorgehen bzw. die Reihenfolge angegeben wird)
  - **Deklarativ**: spezifiziert **was** gemacht wird; nicht geeignet für die Anfrageoptimierung
- Reine Sprachen:
  - Relationale Algebra (imperativ)
  - Tupelkalkül (deklarativ)
  - Domänenkalkül (deklarativ)
- Reine Sprachen sind die Basis für Sprachen, die in der Praxis verwendet werden.

[1] "imperativ" und "prozedural" werden manchmal synonym verwendet

Integrierte Übung 1.1

- Geben Sie Beispiele aus der realen Welt für imperative bzw. deklarative Vorgehensweisen.

SQL

- Die Standardsprache von Datenbanksystemen ist SQL (Structured Query Language); "Intergalactic data speak" [Michael Stonebraker].
- SQL beinhaltet sowohl eine DDL als auch eine DML.
- SQL ist eine deklarative Sprache und wurde von IBM als praktische Anfragesprache zur relationalen Algebra vorgeschlagen.

Schema vs. Instanz/1

- Datenbankschema:
  - Die Beschreibung einer Datenbank.
  - Beinhaltet die Beschreibung der Datenbankstruktur, der Datentypen, und der Integritätsbedingungen auf der Datenbank.
- Das Datenbankschema ändert sich selten.
- Das Datenbankschema wird auch als intensionale Ebene bezeichnet.

Schema vs. Instanz/2

Beispiel: Datenbankschema

Studenten

Name	StudNr	Hauptfach

Vorlesungen

VorlesungsName	VorlesungsNr	ECTS	Institut

Voraussetzungen

VorlesungsNr	VoraussetzungsNr

Module

ModulNr	VorlesungsNr	Semester	Jahr	Dozent

Notenblätter

StudNr	ModulNr	Note

Schema vs. Instanz/3

- Datenbankinstanz:
  - Daten die zu einem gegebenen Zeitpunkt in der Datenbank gespeichert sind
  - auch Datenbankausprägung, Datenbankzustand oder extensionale Ebene genannt
  - Der Begriff "Instanz" wird auch für einzelne Komponenten verwendet (Instanz eines Tupels, Instanz einer Tabelle)
- Gültige Datenbankinstanz: Eine Instanz die sämtliche Strukturen und Integritätsbedingungen erfüllt.
- Eine Datenbankinstanz ändert sich jedesmal wenn die Datenbank geändert wird.

Schema vs. Instanz/4

Beispiel: Datenbankinstanz

Vorlesungen

VorlesungsName	VorlesungsNr	ECTS	Institut
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Databases	CS3360	3	CS

Module

ModulNr	VorlesungsNr	Semester	Jahr	Dozent
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

Voraussetzungen

VorlesungsNr	VoraussetzungsNr
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Notenblätter

StudNr	ModulNr	Note
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

Einordnung der Datenmodelle

- Konzeptionelle Datenmodelle (high-level)
  - Konzepte möglichst nahe an der Benutzersicht
  - keine Datenmainpulationssprache, da nur Schema beschrieben wird, keine Instanzen
  - Beispiele: Entity-Relationship-Modell (ER), Unified Modeling Language (UML)
- Logische Datenmodelle
  - konzentriert sich auf Darstellung der Instanzen
  - geeignet zur Implementierung der Datenbank
  - Beispiele: relationales Modell, objektorientiertes Modell
- Physische Datenmodelle (low-level)
  - Konzepte möglichst nahe an internen Datenstrukturen
  - abhängig von internem Design der Datenbank
  - systemspezifisch, in Handbuch beschrieben

Logische Datenmodelle

- Satzorientierte Datenmodelle: Netzwerkmodell, hierarchisches Modell
  - hauptsächlich historische Bedeutung
  - interessant für Legacy-Systeme (z.B. hierarchisch: IMS von IBM, Netzwerk: UDS von Siemens)
- Relationales Modell:
  - speichert Daten in Tabellen
  - elegantes mathematisches Modell
  - deklarative und imperative Abfragesprachen
- Objektorientiertes und objekt-relationales Modell:
  - Antwort auf Anwendungen mit komplexen Datentypen und Objektorientierung der Programmiersprachen
  - objektorientierte Datenbanken gibt es kaum noch, aber Aspekte leben in objekt-relationalen Datenbanken weiter (z.B. PostgreSQL)

Datenbankbenutzer/1

Datenbank Benuzer haben unterschiedliche Aufgaben:

- Abfrage und Änderung des Datenbankinhaltes
- Entwurf der Datenbank
- Entwicklung und Unterhalt von Datenbankanwendungen
- Verwaltung des Datenbanksystems

Wir unterscheiden:

- Endbenutzer
- Datenbankdesigner
- Anwendungsprogrammierer
- Datenbankadministratoren

Datenbankbenutzer/2

- Endbenutzer: Verwenden die Datenbank für Anfragen, Berichte, und Änderungen.
- Endbenutzer können wie folgt kategorisiert werden:
  - **naive Benutzer**: umfasst den Grossteil der Endbenutzer
    - Verwenden genau definierte Funtionen in der Form von vorgefertigten Transaktionen auf der Datenbank
    - Beispiele: Bankomaten, Reservierungssyteme, Webformulare
  - **fortgeschrittene Benutzer**:
    - Analysten, Wissenschaftler und Ingenieure die vertraut mit den Fähigkeiten des Systems sind
    - Schreiben keine Programme, formulieren jedoch Anfragen anhand einer Anfragesprache

Datenbankbenutzer/3

- Anwendungsprogrammierer: Betten die Anfragesprache in eine Programmiersprache ein und stellen Endbenutzern einfach zu bedienende Programme zur Verfügung, welche komplexe Anfragen bewältigen.
  - erstellen von Webanwendungen
  - erstellen von Anwendungssoftware mit Datenbankzugriff
- Datenbankdesigner:
  - Verantwortlich für den Inhalt, die Strukturen, die Integritätsbedingungen, die Funktionen und Transaktionen. Datenbankdesigner müssen mit Endbenutzern kommunizieren und deren Bedürfnisse kennen.
- Datenbankadministratoren:
  - Verantwortlich für die Autorisierung des Datenbankzugriffs, der Koordination und Überwachung der Benutzung, der Beschaffung von Soft- und Hardware, Backup, Kontrolle der Effizienz der Operationen

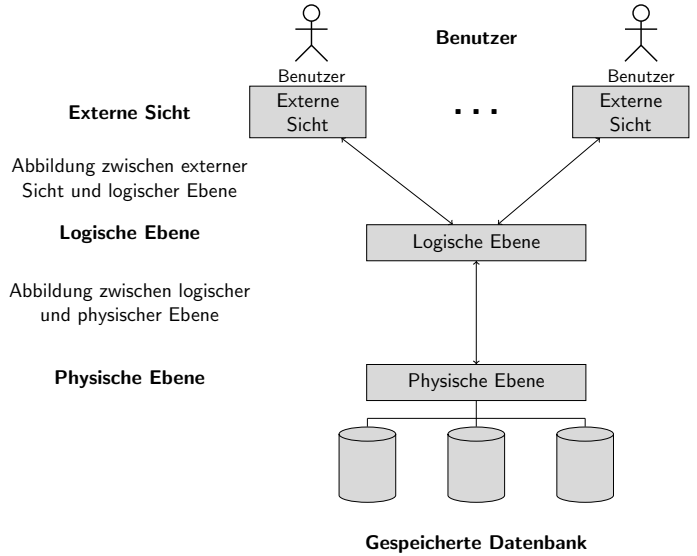
Die ANSI/SPARC Drei-Ebenen Architektur/1

- Die ANSI/SPARC Architektur wurde vorgeschlagen um folgende Charakteristiken einer Datenbank zu unterstützen:
  - Unterschiedliche Sichten auf die Daten
  - Datenunabhängigkeit
- Definiert ein Datenbankschema auf drei Ebenen:
  - **Physische Ebene**: beschreibt die physischen Speicherstrukturen (z.B. Tabellen) und Zugriffspfade (z.B. Indizes).
    - verwendet typischerweise ein physisches Datenmodell
  - **Logische Ebene**: beschreibt die Strukturen und Integritätsbedingungen für die gesamte Datenbank und deren Benutzer
    - verwendet ein konzeptionelles oder logisches Datenmodell
  - **Externe Sicht**: beschreibt unterschiedliche Sichten (views) auf die Datenbank.
    - verwendet das gleiche Datenmodell wie die logische Ebene

Die ANSI/SPARC Drei-Ebenen Architektur/2

- Abbildungen zwischen den verschiedenen Ebenen sind notwendig um Anfragen und Daten transformieren zu können.
  - Anwendungen beziehen sich auf die externe Sicht und werden durch das Datenbanksystem auf die logische und physische Ebene abgebildet um ausgewertet zu werden.
  - Daten die aus der physischen/logischen Ebene kommen werden umformatiert, damit sie der externen Sicht des Benutzers entsprechen.

Die ANSI/SPARC Drei-Ebenen Architektur/3



Datenunabhängigkeit

- Logische Datenunabhängigkeit:
  - Die Möglichkeit das logische Schema zu ändern ohne die externen Sichten und zugehörigen Anwendungen ändern zu müssen.
  - Beispiel: Objekte und deren Eigenschaften umbenennen, neue Eigenschaften hinzufügen
- Physische Datenunahängigkeit:
  - Die Möglichkeit die physische Ebene zu ändern, ohne die logische Ebene ändern zu müssen.
  - Beispiel: Speicherstruktur ändern oder neue Indices erstellen um die Effizienz zu verbessern.
- Vorteile der Datenunabhängigkeit:
  - nach der Änderung einer tieferen Ebene müssen nur die Beziehungen zwischen dieser und der darüberliegenden Ebene nachgeführt werden
  - die weiter darüberliegenden Ebenen werden nicht geändert
  - Anwendungsprogramme müssen nicht geändert werden, da sie auf die oberste Ebene zugreifen

Zusammenfassung

- Datenmodelle: Modellierung der Miniwelt
  - DDL: Data Description Language
  - DML: Data Modification Language
- Anfragesprachen (Teil von DML):
  - imperativ / deklarativ
  - reine Sprachen / praktische Sprachen
- SQL ist Standardsprache: DDL und DML
- Drei-Ebenen Architektur
  - externe, logische und interne Ebene

Datenbanken 1

Datenbankentwurf

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

database research group
http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 6. März 2018

Inhalt

Literatur und Quellen

Lektüre zum Thema "Datenbankentwurf":

- Kapitel 2 (außer 2.7.3, 2.13) aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. 9. Auflage, Oldenbourg Verlag, 2013.

Literaturquellen

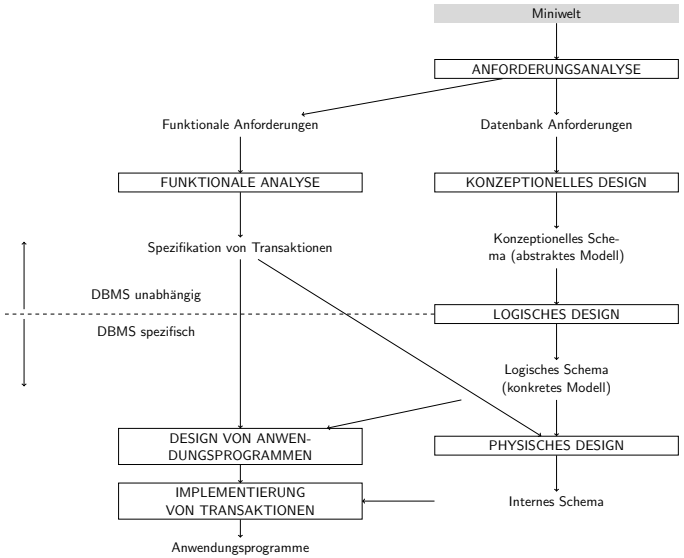
- Peter P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. ACM TODS 1(1): 9-36 (1976)
- Silberschatz, Korth, and Sudarashan: Database System Concepts, McGraw Hill, 2006.
- Elmasri and Navathe: Fundamentals of Database Systems. Fourth Edition, Pearson Addison Wesley, 2004.

Danksagung Die Vorlage zu diesen Folien wurde entwickelt von:

- Michael Böhlen, Universität Zürich, Schweiz
- Johann Gamper, Freie Universität Bozen, Italien

Inhalt

Der Datenbank Entwurfsprozess im Überblick



Beispiel: NAWI Datenbank

- Aufgrund der folgenden **Anforderungen** soll ein konzeptionelles Datenbankschema für eine NAWI Fakultätsdatenbank erstellt werden:
  - Die NAWI ist in Fachbereiche gegliedert. Jeder Fachbereich hat einen Namen, eine Nummer und einen Mitarbeiter, der den Fachbereich führt. Die Fachbereichsleitung beginnt ab einem bestimmten Datum. Ein Fachbereich kann an unterschiedlichen Standorten untergebracht sein.
  - Jeder Fachbereich ist für eine Anzahl von Projekten verantwortlich. Jedes Projekt hat einen eindeutigen Namen, eine eindeutige Nummer und wird an einem einzigen Standort durchgeführt.
  - Von jedem Mitarbeiter erfassen wir Sozialversicherungsnummer, Adresse, Lohn, Geschlecht und Geburtsdatum. Jeder Mitarbeiter arbeitet für nur einen Fachbereich, kann aber an mehreren Projekten arbeiten. Die Anzahl der Wochenstunden pro Projekt werden erfasst. Jeder Mitarbeiter hat einen direkten Vorgesetzten.
  - Jeder Mitarbeiter kann eine Anzahl von abhängigen Personen haben. Von jeder abhängigen Person erfassen wir Name, Geschlecht, Geburtstag und Art der Beziehung.

Das ER-Modell

- ER steht für **Entity-Relationship**
- Das ER-Modell hat **drei Hauptkonstrukte**:
  - Entitäten (entities)
  - Attribute (attributes)
  - Beziehungen (relationships)
- ER-Modell ist **konzeptionelles Datenmodell**
  - Datendefinitionssprache (DDL)
  - keine Datenmanipulationssprache (DML)
  - beschreibt Schema, nicht Instanzen
  - ähnliches Modell: Klassendiagramme in UML
- **Entwurfsprozess** ist eine *schrittweise Verfeinerung*
  - der erste Entwurf ist typischerweise nicht komplett (was gut ist)
  - die Entwürfe werden iterativ verfeinert

Tools für die Datenmodellierung

- **Tools für konzeptionelle Modellierung**
  - unterstützen Erstellung konzeptioneller Modelle
  - bilden konzeptionelles Modell auf relationales Modell ab
  - Beispiele: ERWin, Rational Rose, ER/Studio
- **Vorteile:**
  - dient als Dokumentation der Anforderungsanalyse
  - einfache Benutzerschnittstelle: graphische Unterstützung durch Editor
  - einfache graphische Modelle sind sehr intuitiv
- **Nachteile:**
  - Graphische Modelle werden schnell komplex und mehrdeutig
- **Einfache Zeichentools**
  - Graphikprogramme mit Erweiterung für ER-Diagramme
  - Beispiele: dia (Zeichenprogramm), tikz-er (Latex Package)

Inhalt

Entitäten und Attribute

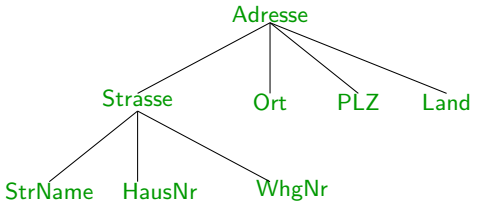
- **Entitäten** sind konkrete Gegenstände oder Konzepte der Miniwelt.
  - Bsp: "Fachbereich für Computerwissenschaften", "Mitarbeiter John Smith", "Projekt SyRA"
- **Attribute** sind Eigenschaften der Entitäten.
  - Bsp: Mitarbeiter John Smith hat Attribute Name, SVN, Geschlecht, Geburtstag, ...
  - Eine Entität hat einen Wert für jedes seiner Attribute.
    - Bsp: Mitarbeiter John Smith hat folgende Werte für seine Attribute:
      Name = 'John Smith',
      Adresse = '731, Fondren, Houston, TX',
      Geburtstag = '09-JAN-55'
  - Jedes Attribut hat eine zugehörige **Domäne** (Wertemenge, Datentyp).
    - Bsp: Name ist eine Zeichenkette, Geburtstag ist vom Typ Datum

Attribute/1

- **Einfache Attribute**
  - Jede Entität hat einen einfachen, atomaren Wert für das Attribut
    - Bsp: SVN, Geschlecht.
- **Zusammengesetzte Attribute**
  - Attribute sind aus mehreren Komponenten zusammengesetzt.
  - Notation: Attribut(Komponente1, Komponente2, . . . , KomponenteN)
  - Beispiele:
    - Adresse(Strasse, HausNr, WhgNr, PLZ, Ort, Land)
    - Name(Vorname, Nachname)
- **Mehrwertige Attribute**
  - Eine Entität kann mehrere Werte für ein Attribut haben.
  - Notation: { Attribut }
  - Bsp: { Telefonnummer }, { Farbe }
- **Abgeleitete Attribute**
  - Attribute können abgeleitet (berechnet) sein.
  - Bsp: AnzahlMitarbeiter kann berechnet werden

Attribute/2

- Zusammengesetzte und mehrwertige Attribute können **beliebig verschachtelt** werden.
- **Beispiel**: zumsammengesetztes, mehrwertiges Attribut Abschlüsse:
  { Abschlüsse (Institution, Jahr, Diplom, Fachgebiet) }
  - mehrere Abschlüsse sind möglich
  - jeder Abschluss hat vier Attribute: Institution, Jahr, Diplom, Fachgebiet
- **Beispiel**: hierarchisch zusammengesetztes Attribut Adresse:
  Adresse(Strasse(StrName, HausNr, WhgNr), Ort, PLZ, Land)



Entitätstypen und Schlüsselattribute

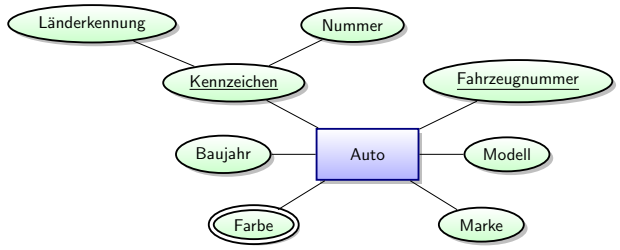
- **Entitätstyp**: fasst Entitäten mit den gleichen Attributen zusammen.
  - Bsp: Entitätstypen Mitarbeiter und Projekte
- **Schlüsselattribut**: Attribut eines Entitätstyps, für den jede Entität einen eindeutigen Wert hat.
  - Bsp: Schlüsselattribut SVN von Mitarbeiter
- Ein Schlüsselattribut **kann zusammengesetzt sein**.
  - Kennzeichen mit den Komponenten (Länderkennung, Nummer) ist ein Schlüssel des Auto Entitätstyps.
- Ein Entitätstyp kann **mehr als einen Schlüssel** haben.
  - Der Auto Entitätstyp hat zwei mögliche Schlüssel:
    - Fahrzeugnummer
    - Kennzeichen (Länderkennung, Nummer)
- Schlüsselattribute werden unterstrichen.

Darstellung von Entitätstypen in ER-Diagrammen

- **ER-Diagramm**: graphische Darstellung der ER-Modellierung
- Entitätstyp wird als Rechteck dargestellt.
- Attribut wird als Oval dargestellt.
  - Attribut ist mit einem Entitätstyp verbunden
  - **zusammengesetzten Attribute**: Komponenten werden mit zusammengesetzem Attribut verbunden
  - **mehrwertige Attribute**: werden in doppelten Ovalen dargestellt
  - **abgeleitete Attribute**: werden als gepunktete Ovale dargestellt
  - **Schlüssel**: werden unterstrichen

ER-Diagramm – Beispiel

- **Bsp**: Entitätstyp Auto mit Attributen Kennzeichen(Länderkennung, Nummer), Fahrzeugnummer, Marke, Modell, Baujahr, { Farbe }



Integrierte Übung 2.1

Gegeben ist ein Entitätstyp R mit Attributen A, B, C, D und E. Schlüssel die beiden Attribute B und E (gemeinsam), A (für sich genommen), sowie C (für sich genommen).

Stellen Sie den Entitätstyp R mit Hilfe der ER-Notation dar.

Entitätsmengen

- Eine **Entitätsmenge** besteht aus der Menge aller Entitäten eines bestimmten Entitätstypen.
- Für eine Entitätsmenge und einen Entitätstyp wird der **gleiche Name** verwendet (z.B. Auto).
- Eine Entitätsmenge stellt den **aktuellen Zustand** eines Teiles der Datenbank dar.
- Entitätsmenge Auto:

$Auto_1$
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004, {red, black})

$Auto_2$
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

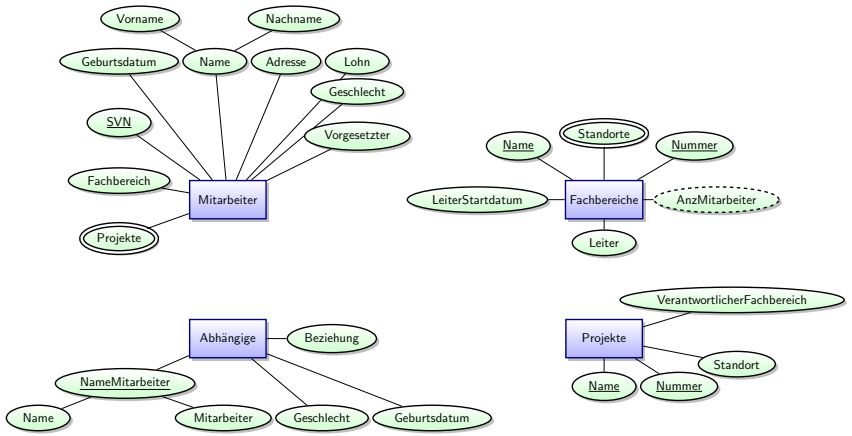
$Auto_3$
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

.
.
.

Entitätstypen der NAWI Datenbank/1

- Ausgehend von der Anforderungsanalyse identifizieren wir vier **Entitätstypen** der NAWI Datenbank:
  - Fachbereiche
  - Projekte
  - Mitarbeiter
  - Abhängige
- Die **Attribute** werden aus den Anforderungen abgeleitet.
- **Richtlinien** für die Bestimmung von Entitätstypen und Attributen:
  - *Substantive* in einer Beschreibung werden als Entitätstypen abgebildet.
  - *Substantive die Entitätstypen beschreiben* werden als Attribute abgebildet.

Entitätstypen der NAWI Datenbank/2



Inhalt

Beziehungen und Beziehungstypen/1

- **Beziehung** erstellt Zusammenhang mit spezifischer Bedeutung zwischen mehreren Entitäten:
  - Mitarbeiter John Smith arbeitet an Projekt SyRA
  - Mitarbeiter Andreas Uhl leitet den Fachbereich für Computerwissenschaften
- **Beziehungstypen** gruppieren Beziehungen des gleichen Typs:
  - der arbeitenAn Beziehungstyp zwischen Mitarbeiter und Projekten
  - der leiten Beziehungstyp zwischen Mitarbeiter und Fachbereichen
- **Ordnung** des Beziehungstyps: Anzahl der involvierten Entitätstypen
  - sowohl leiten als auch arbeitenAn sind *binäre* Beziehungstypen
  - binäre Beziehungstypen sind weitaus die häufigsten

Beziehungen und Beziehungstypen/2

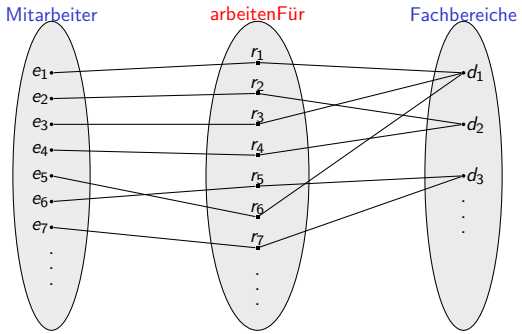
- **Beziehungstyp:**
  - identifiziert Namen der Beziehung und involvierten Entitätstypen
  - identifiziert Einschränkungen (constraints) für Beziehungen
  - beschreibt eine Beziehung auf Schemaebene
- **Beziehungsmenge**:
  - Menge von Beziehungen, die in der Datenbank dargestellt sind
  - beschreibt Beziehungen auf Instanzebene
  - Beziehungsmenge R für Entitätsmengen $E_1, E_2, \ldots, E_n$ ist definiert als:

$$R \subseteq E_1 \times E_2 \times \ldots \times E_n$$

- **Element der Beziehungsmenge** (Beziehung):
  - stellt Beziehung zwischen Entitäten her
  - genau 1 Entität jeder involvierten Entitätsmenge ist Teil der Beziehung
  - Beziehung r für Entitätsmengen $E_1, E_2, \ldots, E_n$ ist definiert als:

$$r \in E_1 \times E_2 \times \ldots \times E_n$$

Die arbeitenFür Beziehung

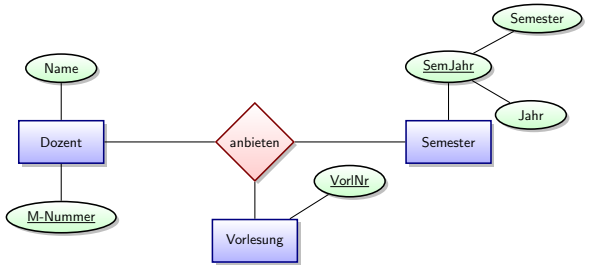


Beispiele für

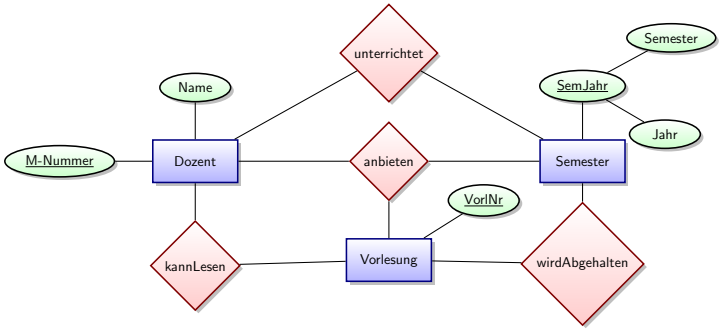
- Entitätsmenge: Mitarbeiter $= \{e_1, e_2, e_3, \ldots\}$
  Fachbereiche $= \{d_1, d_2, d_3, \ldots\}$
- Entität: e_1, e_6, d_3
- Beziehungsmenge: arbeitenFür
- Beziehung: $r_1 = (e_1, d_1)$, $r_5 = (e_6, d_3)$

Beziehungen höherer Ordnung/1

- Beziehungstypen 2. Ordnung sind **binäre** Beziehungstypen.
- Beziehungstypen 3. Ordnung sind **ternäre** und jene der n-ten Ordnung sind **n-wertige** Beziehungstypen.



Beziehungen höherer Ordnung/2

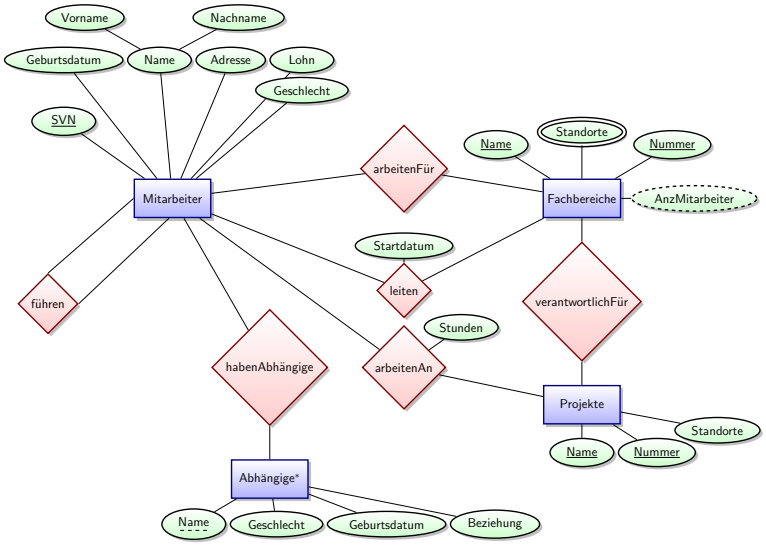


- *n*-wertige Beziehung ist **nicht äquivalent** zu *n* binären Beziehungen:
  - Beziehung unterrichtet kann von ternärer Beziehung anbieten hergeleitet werden
  - Beziehung kannLesen kann *nicht* von ternären Beziehung anbieten hergeleitet werden
  - Beziehung anbieten kann nicht aus unterrichtet, kannLesen, wirdAngeboten hergeleitet werden

Beziehungen in der NAWI Datenbank/1

- Aus den Anforderungen lassen sich **6 Beziehungstypen** ableiten
- Alle Beziehungen sind **binär** (d.h. stellen eine Beziehung zwischen zwei Entitäten her)
- Beziehungstypen mit **involvierten Entitätstypen**:
  - arbeitenFür (zwischen Mitarbeiter und Fachbereiche)
  - leiten (zwischen Mitarbeiter und Fachbereiche)
  - verantwortlichFür (zwischen Fachbereiche und Projekte)
  - arbeitenAn (zwischen Mitarbeiter und Projekte)
  - führen (zwischen Mitarbeiter (als Unterstellter) und Mitarbeiter (als Vorgesetzter))
  - habenAbhängige (zwischen Mitarbeiter und Abängige)
- In ER-Diagrammen werden **Beziehungstypen wie folgt dargestellt**:
  - ein Rhombus wird verwendet um einen Beziehungstypen darzustellen
  - der Rhombus ist mit den involvierten Entitätstypen verbunden

Beziehungstypen in der NAWI Datenbank/2



* Die Entität Abhängige hat keinen vollständigen Schlüssel. Diese Art von Entitäten heißt *existenzabhängig* und wird später behandelt.

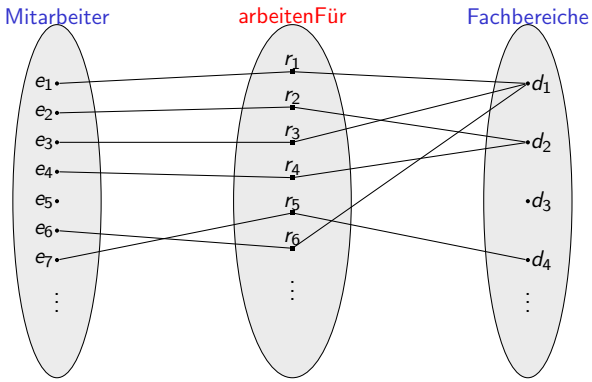
Diskussion von Beziehungstypen

- Im verfeinerten Entwurf werden einige **Attribute** von Entitätstypen **durch Beziehungen dargestellt**:
  - Leiter eines Fachbereichs -> leiten
  - Projekte von Mitarbeiter -> arbeitenAn
  - Fachbereich von Mitarbeiter -> arbeitenFür
  - usw.
- Zwischen Entitätstypen können **mehrere Beziehungstypen** existieren:
  - leiten und arbeitenFür sind unterschiedliche Beziehungstypen zwischen Mitarbeiter und Fachbereiche
  - diese Beziehzungstypen haben unterschiedliche Bedeutung und unterschiedliche Beziehungsinstanzen

Funktionalitäten von Beziehungstypen

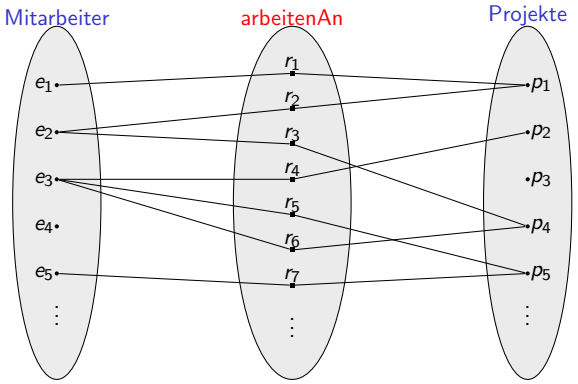
- **Funktionalitäten** schränken die Anzahl der möglichen Kombinationen von Entitäten in einer Beziehungsmenge ein.
- **Kardinalitätseinschränkung** spezifiziert *Obergrenze* für Häufigkeit einer Entität in Beziehungen
  - 1:1 Beziehung
  - 1:N Beziehung (analog eine N:1 Beziehung)
  - M:N Beziehung
- **Teilnahmebeschränkung** spezifiziert *Untergrenze* für Häufigkeit einer Entität in Beziehungen
  - 0 (optionale Teilnahme)
  - 1 oder mehr (zwingende Teilnahme)

N:1 Beziehung



- Mitarbeiter:Fachbereiche = N:1
- Ein Mitarbeiter arbeitet für maximal 1 Fachbereich.
- Ein Fachbereich beschäftigt maximal N Mitarbeiter.

M:N Beziehung



- Mitarbeiter:Projekte = M:N
- Ein Mitarbeiter arbeitet an maximal N Projekten.
- Ein Projekt wird von maximal M Mitarbeitern bearbeitet.

Notation für Funktionalitäten/1

Kardinalitätseinschränkungen binärer Beziehungen:

- **Notation**: durch Beschriftung der Kanten mit Zahlen

- **Interpretation**:
  - für einen bestimmten Mitarbeiter gibt es 1 Fachbereich, mit dem er in der arbeitetFür Beziehung stehen kann
  - für einen bestimmten Fachbereich gibt es *N* Mitarbeiter, mit denen er in der arbeitetFür Beziehung stehen kann
- **n-wertiger Beziehungstyp**:
  - *Kardinalitätseinschränkung* spezifiziert, wie oft eine Entität für eine konkrete Instanz aller anderen Entitäten vorkommen darf
- **Richtlinie für Leserichtung**: links nach rechts, oben nach unten
  - Ein Mitarbeiter arbeitet für 1 Fachbereich.
  - umgekehrte Leserichtung: Verb wird geändert, z.B. Ein Fachbereich beschäftigt *N* Mitarbeiter.

Formale Definition von Funktionalitäten

- Beziehungen mit Funktionalitäten definieren **partielle Funktionen** zwischen Entitätsmengen.
  - eine Funktion $F : X \to Y$ ist *partiell*, wenn nicht jedem Element von X ein Element von Y zugeordnet werden muss
  - eine nicht-partielle Funktion ist *total*

- Gegeben eine **n-wertige Beziehung** R zwischen den Entitätsmengen $E_1, E_2, \ldots E_n$ mit den jeweiligen Kardinalitätseinschränkungen $K_1, K_2, \ldots, K_n$, wobei K_i ein eindeutiger Buchstabe zugeordnet ist (z.B. M, N), oder $K_i = 1$. R definiert folgende partielle Funktion:

$$R : E_1 \times E_2 \times \ldots \times E_{k-1} \times E_{k+1} \times \ldots \times E_n \to E_k$$

für jedes k, $1 \leq k \leq n$ mit $K_k = 1$.

Beispiel: Formale Definition von Funktionalitäten

- Die $1 : M : N$-Beziehung "prüfen" zwischen Professor, Student, und Vorlesung definiert die partielle Funktion:

$$\text{prüfen} : \text{Student} \times \text{Vorlesung} \to \text{Professor}$$

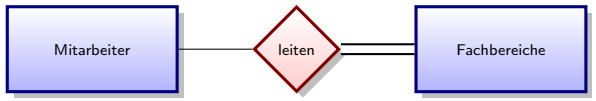
- Die $1 : 1 : N$-Beziehung "betreuen" zwischen Professor, Seminarthema und Student definiert folgende partielle Funktionen:

$$\text{betreuen} : \text{Student} \times \text{Seminarthema} \to \text{Professor}$$
$$\text{betreuen} : \text{Student} \times \text{Professor} \to \text{Seminarthema}$$

Notation für Funktionalitäten/2

Teilnahmebeschränkung einer Beziehung:

- **Notation**:
  - total (zwingend); Notation: doppelte Linie
  - partiell (optional); Notation: einfache Linie



- **Interpretation**:
  - Jeder Fachbereich muss eine Beziehung leiten eingehen, d.h., jeder Fachbereich wird von einem Mitarbeiter geleitet.
  - Ein Mitarbeiter muss nicht unbedingt die Beziehung leiten eingehen, d.h., muss keinen Fachbereich leiten.

- Nicht alle real existierenden Einschränkungen lassen sich mit Hilfe von ER-Diagrammen modellieren.

Integrierte Übung 2.2

In einem Flugreservierungssytem gibt es folgende Beziehungen:

- Flüge transportieren Passagiere
- Passagiere haben Sitze reserviert
- Gates fertigen Flüge ab
- Flüge verfügen über Sitze

Stellen Sie die entsprechenden Entitäten und Beziehungstypen mit deren Funktionalitäten dar.

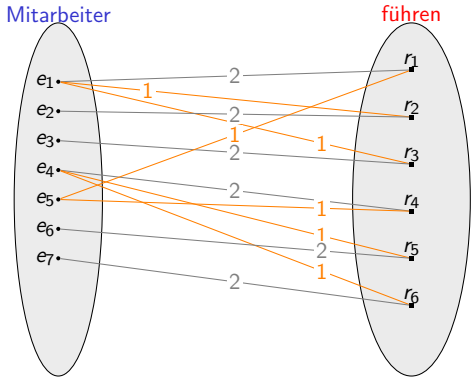
Rekursive Beziehungen/1

- In einem rekursiven Beziehungstypen kommt der gleiche Beziehungstyp in **unterschiedlichen Rollen** vor.
- **Beispiel:** Beziehungstyp führen zwischen Mitarbeitern. Mitarbeiter können in der Rolle des *Vorgesetzten* oder des *Unterstellten* auftreten
- Eine Beziehung (Element der Beziehungsmenge) verbindet dann zwei Entitäten derselben Entitäsmenge, z.B.
  - einen Mitarbeiter in der Rolle des *Vorgesetzten*
  - einen Mitarbeiter in der Rolle des *Unterstellten*

Rekursive Beziehungen/2

- Zwei Rollen eines Entitätstypen werden in Beziehungen **wie zwei verschiedene Entitätstypen** betrachtet, das heißt:
  - eine Entität kann in einer Beziehung (Element der Beziehungsmenge) mehrfach in verschiedenen Rollen aufscheinen.
  - die Funktionalität bezieht sich auf die Rolle, d.h., eine Entität kann in einer Rolle mit "1" beschränkt sein, in einer anderen Rolle mit *N*.
- In **ER-Diagrammen** geben wir die Rollennamen an, um die unterschiedlichen Rollen des Entitätstyps zu identifizieren.

Rekursive Beziehungen/3



- Annotation 1 steht für die Rolle des Vorgesetzten
- Annotation 2 steht für die Rolle des Unterstellten
- e_1 ist der Vorgesetzte von e_2
- e_1 ist der Unterstellte von e_5

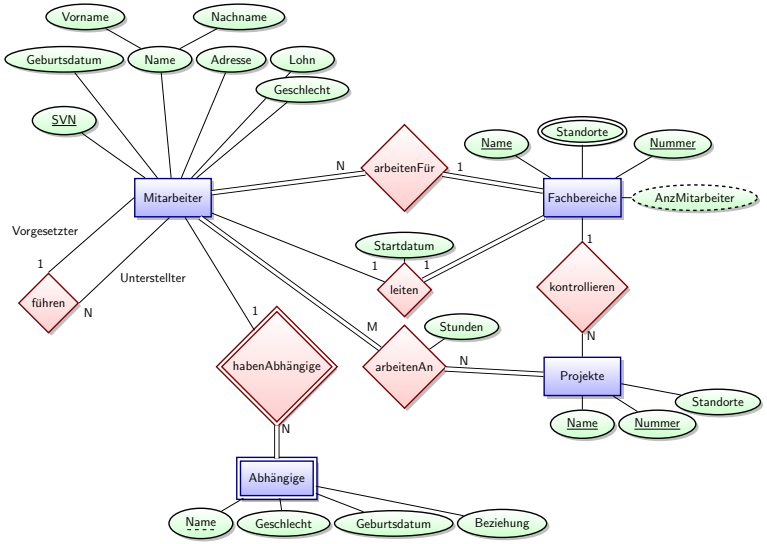
Attribute eines Beziehungstyps

- Ein **Beziehungstyp** kann **Attribute** haben.
  - Beispiel: Wochenstunden von arbeitenAn
  - der Wert von Wochenstunden gibt für jede Beziehung an, für viele Wochenstunden der Mitarbeiter auf das Projekt angestellt ist
  - der Wert von Wochenstunden hängt von den involvierten Entitäten (Mitarbeiter, Projekt) ab
- Attribut des Beziehungstypen **zum Entitätstypen verschieben**?
  - in 1:1 Beziehungen können die Attribute zu einem der beiden Entitätstypen verschoben werden
  - in 1:N Beziehungen können die Attribute zum Entitätstyp auf der N-Seite verschoben werden
  - in M:N Beziehungen können die Attribute nicht verschoben werden
- **Beachte**: Attribut beim Entitätstyp muss auch für Entitäten, die nicht in Beziehung stehen, einen Wert haben.

Existenzabhängige Entitätsypen

- **Existenzabhänger Entitätstyp** (weak entity type):
  - hat keinen (vollständigen) Schlüssel
  - muss **übergeordneten Entitätstypen** haben
  - geht identifizierende Beziehung mit übergeordnetem Entitätstyp ein
- **Schlüssel**: Existenzabhängige Entitäten werden wie folgt identifiziert:
  - den partiellen Schlüssel des existenzabhängigen Entitätstypen und
  - die übergeordnete Entität
- **Beispiel:**
  - eine Entität von Abhängige wird identifiziert durch Name der abhängigen Person *und* SVN des dazugehörigen Mitarbeiters
  - Name von Abhängige ist ein *partieller Schlüssel* (im ER-Schema unterstrichen mit gepunkteter Linie)
  - Abhängige ist ein *existenzabhängiger Entitätstyp*
  - Mitarbeiter ist der *übergeordnete Entitätstyp*

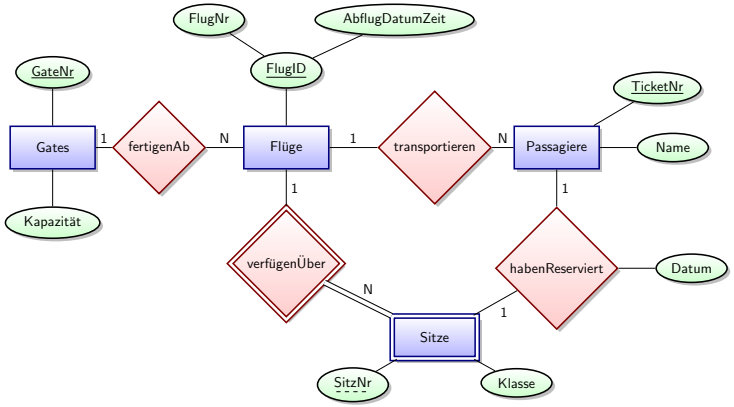
Das NAWI ER-Diagramm



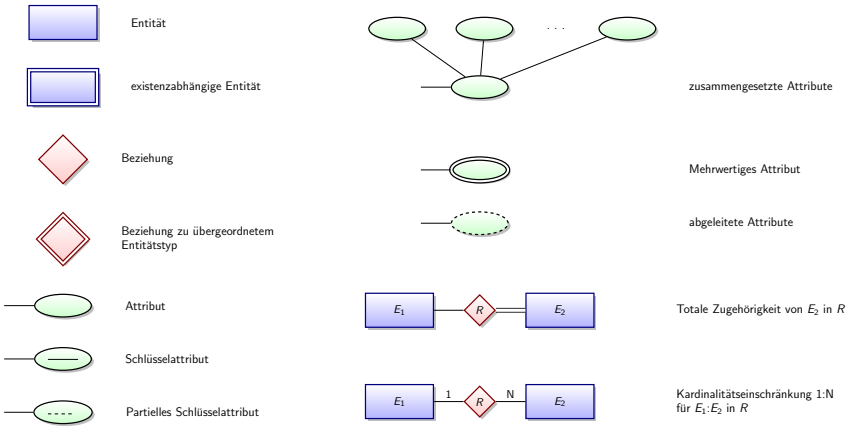
Integrierte Übung 2.3

Interpretieren Sie folgendes ER-Schema einer Flugreservierung:

- Beschreiben Sie die Entitäts- und Beziehungstypen mit Attributen, Schlüssel und Funktionalitäten.
- Überlegen Sie, welche Instanzen das ER-Schema (nicht) erlaubt.



Zusammenfassung der ER-Notation



Inhalt

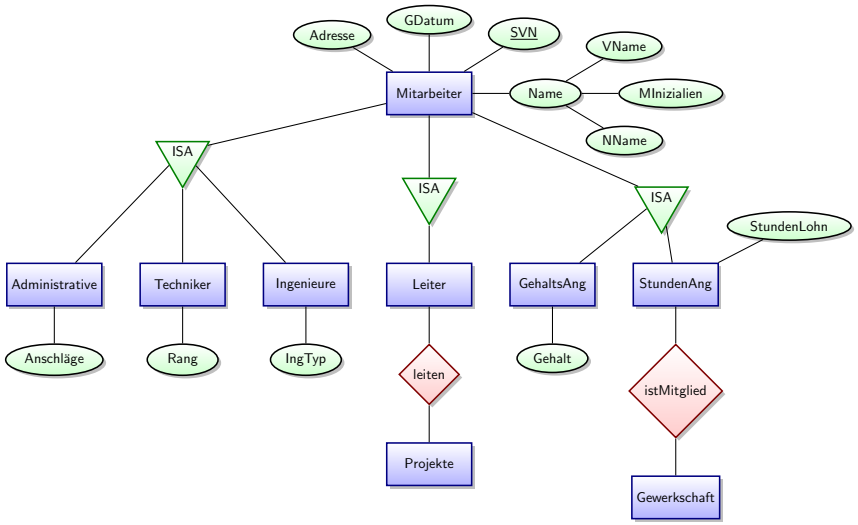
Unter- und Obertypen/1

- Entitätstypen können **sinnvolle Untergruppen** haben.
- Beispiel: Mitarbeiter können weiter unterteilt werden in
  - Administrative, Techniker, Ingenieure, ...
    – abhängig von der Aufgabe der Mitarbeiter
  - Leiter
    – Mitarbeiter die Leiter sind
  - GehaltsAng, StundenAng
    – abhängig von der Vertragsart
- Jede Unterteilung ist eine **Teilmenge** von Mitarbeiter
- Erweiterte ER-Diagramme stellen diese Unterteilungen anhand von **Untertypen** und **Obertypen** dar.

Unter- und Obertypen/2

- Verhältnis zwischen Mitarbeiter und Teilmengen wird **Obertyp/Untertyp Beziehung** genannt:
  - Mitarbeiter/Administrative
  - Mitarbeiter/Techniker
  - Mitarbeiter/Leiter
  - ...
- Obertyp/Untertyp Beziehungen auch als **IS-A Beziehung** bezeichnet:
  - Administrative IS-A Mitarbeiter
  - Techniker IS-A Mitarbeiter
  - ...
- **Notation:** Dreieck mit Bezeichnung "ISA"

Beispiel: Unter- und Obertypen



Unter- und Obertypen/3

- Die **Entität eines Untertypen ist dasselbe Objekt** in der realen Welt wie das des Obertypen:
  - die Untertypen-Entität ist dasselbe Objekt in einer *speziellen Rolle*
  - eine Entität kann nicht als reiner Untertyp existieren, es ist immer auch eine Entität des Obertypen
- Eine Entität kann in **mehreren Untertypen** vorkommen.
- Beispiel: Ein Mitarbeiter der Techniker ist und mit Gehalt angestellt ist, gehört zu den Untertypen: Techniker und GehaltsAng

Vererbung

- Die Entitäten der Untertypen **erben** vom Obertypen
  - alle Attribute
  - alle Beziehungstypen
- **Beispiel**:
  - Administrative (sowie Techniker und Ingenieure) erben die Attribute Name, SVN, . . . , von Mitarbeiter
  - Jeder Administrative hat Werte für die geerbten Attribute.

Spezialisierung und Generalisierung

- **Spezialisierung:**
  - aus einem Obertype eine Menge von Untertypen generieren
  - die Untertypen basieren auf unterschiedlichen Charakteristiken
  - Bsp: Administrative, Ingenieur, Techniker ist eine Spezialisierung von Mitarbeiter aufgrund der verrichteten Arbeit
  - unterschiedliche Spezialisierungen eines Obertyps sind möglich
- **Generalisierung:**
  - umgekehrter Prozess zu Spezialisierung
  - verschiedene Typen mit gemeinsamen Eigenschaften werden zu Obertyp zusammengefasst
  - Bsp: PKW und Lastwagen werde zu Fahrzeug generalisiert
    - PKW und Lastwagen werden Untertypen des Obertyps Fahrzeug
    - Fahrzeug ist Generalisierung von PKW und Lastwagen, PKW und Lastwagen sind Spezialisierung von Fahrzeug

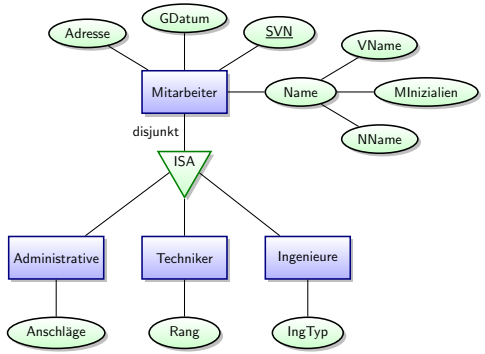
Einschränkungen der Spezialisierung/1

- **Zwei Arten** von Einschränkungen auf Spezialisierung:
  - disjunkte Spezialisierung
  - vollständige Spezialisierung
- **Disjunkte Spezialisierung:**
  - Entität kann zu *höchstens einem* Untertyp gehören
  - Notation: Verbindungslinie zu Obertyp mit “disjunkt” bezeichnen
  - nicht-disjunkte Spezialisierung wird *überlappend* genannt (keine Beschriftung auf Verbindungslinie)
- **Vollständige Spezialisierung:**
  - Entität muss zu *mindestens einem* Untertyp gehören, d.h. *keine* Entität kann *nur* zu Obertyp gehören
  - Notation: Doppelte Linie als Verbindung zu Obertyp
  - nicht-vollständige Spezialisierung wird *partiell* genannt (einfache Verbindungslinie)

Einschränkungen der Spezialisierung/2

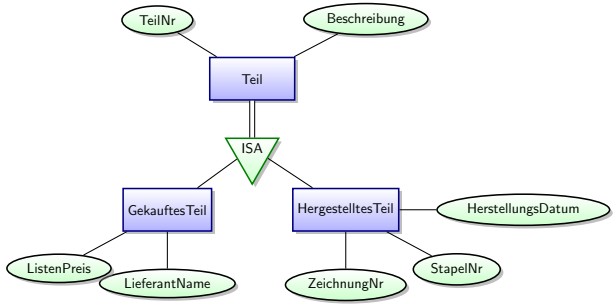
- **Alle Kombinationen** erlaubt:
  - überlappend/partiell (Standard): Entität *kann* zu einem oder mehreren Untertypen gehören
  - disjunkt/partiell: Entität *kann* zu maximal einem Untertypen gehören
  - überlappend/vollständig: Entität *muss* zu einem oder mehreren Untertypen gehören
  - disjunkt/vollständig: Entität *muss* zu genau einem Untertypen gehören

Beispiel: Disjunkte partielle Spezialisierung



- Ein Mitarbeiter muss weder Administrativer, Techniker, noch Ingenieur sein (partielle Spezialisierung).
- Ein Mitarbeiter kann nicht Administrativer und Techniker, Administrativer und Ingenieur, oder Techniker und Ingenieur zugleich sein (disjunkte Spezialisierung).

Beispiel: Überlappende vollständige Spezialisierung



- Es gibt kein Teil, das weder hergestellt noch gekauft ist (vollständige Spezialisierung).
- Ein Teil kann zugleich hergestellt und gekauft sein (überlappende Spezialisierung).

Zusammenfassung

- Das ER-Modell hat **drei Hauptkonstrukte**:
  - Entitäten (mit Entitätstypen und Entitätsmengen)
  - Attribute (einfach, zusammengesetzt, mehrwertig, usw.)
  - Beziehungen (mit Beziehungstypen und Beziehungsmengen)
- **Funktionalitäten** schränken die möglichen Beziehungen ein:
  - Kardinalitätseinschränkung: 1:1, 1:N, M:N
  - Teilnahmebeschränkung: zwingende oder optionale Teilnahme
- **Generalisierung und Spezialisierung**:
  - Obertypen erlauben Zusammenfassung ähnlicher Entitäten
  - jede Entität eines Untertyps ist auch im Obertyp
  - Einschrängkung der Spezialisierung: disjunkt, vollständig

Datenbanken 1

Das Relationale Modell

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

database research group
http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 9. April 2018

Inhalt

Literatur und Quellen

Lektüre zum Thema "Relationales Modell":

- Kapitel 3 (3.1-3.3) aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. 9. Auflage, Oldenbourg Verlag, 2013.

Literaturquellen

- Elmasri and Navathe: Fundamentals of Database Systems. Fourth Edition, Pearson Addison Wesley, 2004.
- Silberschatz, Korth, and Sudarashan: Database System Concepts, McGraw Hill, 2006.

Danksagung Die Vorlage zu diesen Folien wurde entwickelt von:

- Michael Böhlen, Universität Zürich, Schweiz
- Johann Gamper, Freie Universität Bozen, Italien

Inhalt

Das Relationale Modell/1

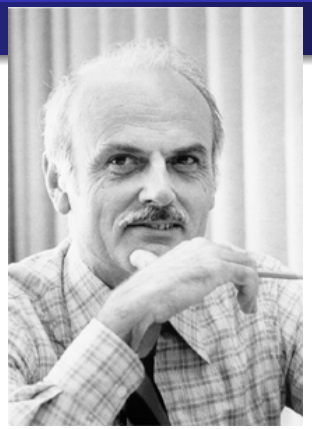
- Relationale Modell:
  - logisches Datenmodell
  - basiert auf Relationen
- Relation: mathematisches Konzept, das auf Mengen basiert.
- Die Stärke des relationalen Modells ist die formale Grundlage durch Relationen (und Mengen).
- In der Praxis wird der SQL Standard verwendet. Der SQL Standard unterscheidet sich vom formalen Modell in einigen Punkten (wir gehen später auf diese Unterschiede ein).

Das Relationale Modell/2

- Das relationale Modell wurde von E. Codd von IBM Research in folgendem Artikel eingeführt:

  A Relational Model for Large Shared Data Banks, Communications of the ACM, June 1970
- Dieser Artikel hat das Feld der Datenbanksysteme revolutioniert.
- Codd erhielt hierfür den ACM Turing Award.

Das Relationale Modell/3



- Edgar Codd, a mathematician and IBM Fellow, is best known for creating the relational model for representing data that led to today's 12 billion database industry.
- Codd's basic idea was that relationships between data items should be based on the item's values, and not on separately specified linking or nesting.
- The idea of relying only on value-based relationships was quite a radical concept at that time, and many people were skeptical. They didn't believe that machine-made relational queries would be able to perform as well as hand-tuned programs written by expert human navigators.

`http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html`

Schema

- $sch(R) = [A_1, A_2, \ldots, A_n]$ ist das Schema der Relation.
- Eckige Klammern [...] werden für eine Liste von Werten verwendet; eine Liste ist geordnet.
- R ist der Name der Relation.
- $A_1, A_2, \ldots, A_n$ sind die Attribute.
- Kurzschreibweise: Für die Definition einer Relation R mit Schema $sch(R) = [A_1, A_2, \ldots, A_n]$ schreiben wir kurz:

$$R[A_1, A_2, \ldots, A_n]$$

- Beispiel: Für die Relation $Kunden[KundenName, KundenStrasse, KundenOrt]$ gilt $sch(Kunden) = [KundenName, KundenStrasse, KundenOrt]$

Die Relation Kunden

- Schema: $sch(Kunden) = [KundenName, KundenStrasse, KundenOrt]$

Kunden

KundenName	KundenStrasse	KundenOrt
Meier	Zeltweg	Brugg
Steger	Ringstr	Aarau
Marti	Seeweg	Brugg
Kurz	Marktplatz	Luzern
Egger	Weststr	Brugg
Staub	Bahnhofstr	Brugg
Gamper	Bahnhofstr	Chur
Ludwig	Baugasse	Brugg
Wolf	Bahnhofstr	Brugg
Koster	Magnolienweg	Brugg
Kunz	Fliedergasse	Brugg
Pauli	Murtenstr	Biel

Domäne

- Eine Domäne ist eine Menge von atomaren Werten.
  - Beispiel: Alter einer Person ist eine positive Ganzzahl.
- Zu jeder Domäne gehört ein Datentyp (oder Format):
  - Telefonnummer hat Format: 0dd ddd dd dd, wobei d eine Ziffer ist.
  - Für ein Datum existieren verschiedene Formate: z.B. yyyy-mm-dd oder dd.mm.yyyy
- Der reservierte Wert `null` gehört zu jeder Domäne:
  - wird für fehlende Werte verwendet
  - Nullwerte machen die Definition von Operationen komplexer

Attribute

- Attributwert: Attribut nimmt für jedes Tupel einen Wert an
  - mögliche Werte durch Domäne bestimmt
  - $dom(A)$ ist die Domäne von Attribut A
- Atomar: Attributwerte müssen atomar sein
  - also "einfach" im Sinne des ER-Modells
  - zusammengesetzte oder mehrwertige Attribute sind nicht erlaubt
  - "Pink Floyd" ist atomar, "Pink Floyd – Wish you were here" ist nicht atomar
- Attributname: spezifizieren Rolle der entsprechenden Domäne in Relation:
  - Name ist eindeutig innerhalb einer Relation
  - wird verwendet, um die Werte dieses Attributs zu interpretieren
- Beispiel: Die Domäne *Datum* wird für die Attribute *Rechnungsdatum* und *Zahlungstermin* mit unterschiedlichen Bedeutungen verwendet.
  - $dom(Rechnungsdatum) = Datum$
  - $dom(Zahlungstermin) = Datum$

Tupel

- Ein Tupel ist eine geordnete Menge (d.h. eine Liste) von Werten
- Eckige Klammern [...] werden verwendet um Tupel darzustellen
- Jeder Wert eines Tupels muss aus der entsprechenden Domäne stammen
- Beispiel: Tupel der Relation *Kunden*
  - Schema: $sch(Kunden) = [KundenName, KundenStrasse, KundenOrt]$
  - Tupel: $[Meier, Zeltweg, Brugg]$

Instanz (Ausprägung)

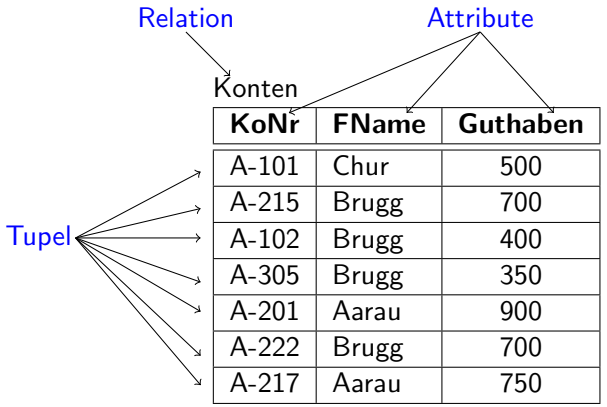
- Der Name einer Relation R wird auch als Bezeichner für die Instanz einer Relation verwendet
- Instanz einer Relation R mit Schema $sch(R) = [A_1, A_2, \ldots, A_n]$ ist eine Untermenge des Kreuzprodukts der Domänen der Attribute:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

  R ist also eine Menge von Tupeln $[v_1, v_2, \ldots, v_n]$, sodass jedes $v_i \in D_i$
- Geschweifte Klammern $\{...\}$ werden für Mengen verwendet
- Beispiel:

$$\begin{aligned} D_1 &= dom(KuName) = \{Ludwig, Koster, Marti, Wolf, \ldots\} \\ D_2 &= dom(KuStrasse) = \{Bahnhofstr, Baugasse, Seeweg, \ldots\} \\ D_3 &= dom(KuOrt) = \{Brugg, Luzern, Chur, \ldots\} \\ R &= \{\ [Ludwig, Bahnhofstr, Brugg], [Koster, Baugasse, Brugg], \\ &\quad\ [Marti, Seeweg, Brugg], [Wolf, Weststr, Brugg]\ \} \\ &\subseteq D_1 \times D_2 \times D_3 \end{aligned}$$

Beispiel einer Relation



KoNr	FName	Guthaben
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-201	Aarau	900
A-222	Brugg	700
A-217	Aarau	750

Eigenschaften von Relationen

- Relationen sind ungeordnet, d.h., die Ordnung der Tupel ist nicht relevant.
- Die Attribute eines Schemas $sch(R) = [A_1, ..., A_n]$ und die Werte in einem Tuple $t = [v_1, ..., v_n]$ sind geordnet.

Konten

KoNr	FName	Guthaben
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-201	Aarau	900
A-222	Brugg	700
A-217	Aarau	750

$=$

Konten

KoNr	FName	Guthaben
A-305	Brugg	350
A-201	Aarau	900
A-222	Brugg	700
A-102	Brugg	400
A-217	Aarau	750
A-101	Chur	500
A-215	Brugg	700

Integrierte Übung 3.1

1. Ist $R = \{[Tom, 27, ZH], [Bob, 33, Rome, IT]\}$ eine Relation?

2. Was ist der Unterschied zwischen einer Menge und einer Relation? Geben Sie ein Beispiel, das den Unterschied illustriert.

Integrierte Übung 3.2

1. Illustrieren Sie die folgenden Relationen graphisch:
   $sch(R) = [X, Y]; R = \{[1, a], [2, b], [3, c]\}$
   $sch(S) = [A, B, C]; S = \{[1, 2, 3]\}$

2. Bestimmen Sie die folgenden Objekte:
   - Das 2. Attribut der Relation R?
   - Das 3. Tupel der Relation R?
   - Das Tuple in der Relation R mit dem kleinsten Wert von Attribut X?

Datenbank

- Eine Datenbank ist eine Menge von Relationen.
- Beispiel: Die Informationen eines Unternehmens werden in mehrere Teile aufgespaltet:
  - *Konten*: speichert Informationen über Konten
  - *Kunde*: speichert Informationen über Kunden
  - *Kontoinhaber*: speichert welche Kunden welche Konten besitzen
- Warum nicht alle Informationen in eine Relation speichern?
  - Beispiel: $sch(Bank) = [KoNr, Guthaben, KuName, \ldots]$
  - Redundanz: Wiederholung von Informationen, z.B. zwei Kunden mit demselben Konto
  - Nullwerte: z.B. für einen Kunden ohne Konto

Datenbank mit Relationen Konten und Kontoinhaber

Konten

KoNr	**FName**	**Guthaben**
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-201	Aarau	900
A-222	Brugg	700
A-217	Aarau	750

Kontoinhaber

KName	**KoNr**
Staub	A-102
Gamper	A-101
Gamper	A-201
Ludwig	A-217
Wolf	A-222
Koster	A-215
Kunz	A-305
Mair	A-101

Integrierte Übung 3.3

1. Welche Art von Objekt ist $X = \{\{[3]\}\}$ im relationalen Modell?

2. Sind DB1 und DB2 identische Datenbanken?
   $DB1 = \{\{[1, 5], [2, 3]\}, \{[4, 4]\}\}$
   $DB2 = \{\{[4, 4]\}, \{[2, 3], [1, 5]\}\}$

Zusammenfassung des relationalen Modells

- Eine **Domäne** D ist eine Menge von atomaren Werten.
  - Telefonnummern, Namen, Noten, Geburtstage, Institute
  - jede Domäne beinhaltet den reservierten Wert `null`
- Zu jeder Domäne wird ein **Datentyp** oder Format spezifiziert.
  - 5-stellige Zahlen, yyyy-mm-dd, Zeichenketten
- Ein **Attribut** A_i beschreibt die Rolle einer Domäne innerhalb eines Schemas.
  - TelephonNr, Alter, Institutsname
- Ein **Schema** $sch(R) = [A_1, ..., A_n]$ besteht aus einer Liste von Attributen.
  - $sch(Angestellte) = [Name, Institut, Lohn]$,
    $sch(Institute) = [InstName, Leiter, Addresse]$
- Ein **Tupel** t ist eine Liste von Werten $t = [v_1, ..., v_n]$ mit $v_i \in dom(A_i)$.
  - $t = [Tom, SE, 23K]$
- Eine **Relation** $R \subseteq D_1 \times ... \times D_n$ mit dem Schema $sch(R) = [A_1, ..., A_n]$ ist eine Menge von n-stelligen Tupeln.
  - $R = \{[Tom, SE, 23K], [Lene, DB, 33K]\} \subseteq NAMEN \times INSTITUTE \times INTEGER$
- Eine **Datenbank** DB ist eine Menge von Relationen.
  - $DB = \{R, S\}$
  - $R = \{[Tom, SE, 23K], [Lene, DB, 33K]\}$
  - $S = \{[SE, Tom, Boston], [DB, Lena, Tucson]\}$

Integritätsbedingungen

- Integritätsbedingungen (constraints) sind Einschränkungen auf den Daten, die alle Instanzen der Datenbank erfüllen müssen.
- Klassen von Integritätsbedingungen im relationalen Modell :
  - Schlüssel
  - Domänenintegrität
  - Referentielle Integrität
- Integritätsbedingungen garantieren eine gute Datenqualität.

Schlüssel/1

- $K \subseteq R$ ist eine Teilmenge der Attribute von R
- K ist ein Superschlüssel von R falls die Werte von K ausreichen um ein Tupel jeder möglichen Relation R eindeutig zu identifizieren.
  - Mit "jeder möglichen" meinen wir eine Relation, die in der Miniwelt, die wir modellieren, existieren könnte.
  - Beispiel: $\{KuName, KuStrasse\}$ und $\{KuName\}$ sind Superschlüssel von *Kunde*, falls keine zwei Kunden den gleichen Namen haben können.

KuName	KuStrasse
N. Jeff	Binzmühlestr
N. Jeff	Hochstr

KuName ist *kein* Schlüssel

KuName	KuStrasse
N. Jeff	Binzmühlestr
T. Hurd	Hochstr

KuName ist ein Schlüssel

Schlüssel/2

- K ist ein Kandidatschlüssel falls K minimal ist
  Beispiel:
  - $\{KuName\}$ ist ein Kandidatschlüssel für *Kunde* weil diese Menge ein Superschlüssel ist und keine Untermenge ein Superschlüssel ist.
  - $\{KuName, KuStrasse\}$ ist kein Kandidatschlüssel weil eine Untermenge, nämlich $\{KuName\}$, ein Superschlüssel ist.
- Primärschlüssel: ein Kandidatschlüssel der verwendet wird um Tupel in einer Relation zu identifizieren.
  - Als Primärschlüssel sollte ein Attribut ausgewählt werden, dessen Wert sich nie ändert (oder zumindest sehr selten).
  - Beispiel: *email* ist eindeutig und ändert sich selten

Domänenintegrität

- Die Domänenintegrität garantiert, dass alle Attributwerte aus der entsprechenden Domäne stammen.
- Nullwerte: sind standardmäßig erlaubt da Teil der Domäne
- Primärschlüssel dürfen nicht `null` sein
  - falls der Primärschlüssel aus mehreren Attributen besteht darf keines dieser Attribute `null` sein
  - andere Attribute der Relation, selbst wenn sie nicht zum Primärschlüssel gehören, können ebenfalls Nullwerte verbieten

ID	Name	KuStrasse
1	N. Jeff	Binzmühlestr
`null`	T. Hurd	Hochstr

ID kann nicht Primärschlüssel sein

ID	Name	KuStrasse
1	N. Jeff	Binzmühlestr
2	T. Hurd	Hochstr

ID kann Primärschlüssel sein

Referentielle Integrität

- Fremdschlüssel: Attribute im Schema einer Relation, die Primärschlüssel einer anderen Relation sind.
  - Beispiel: *KuName* und *KoNr* der Relation *Kontoinhaber* sind Fremdschlüssel von *Kunde* bzw. *Konten*.
- Rekursion: Nicht-Primärschlüssel Attribute können auch Fremdschlüssel zum Primärschlüssel in derselben Relation sein.
- Erlaubte Werte für Fremdschlüssel:
  - Werte, die als Primärschlüssel in der referenzierten Relation vorkommen
  - `null` Werte (alle oder kein Attribut des Fremdschlüssels)
- Graphischen Darstellung eines Schemas: gerichteter Pfeil vom Fremdschlüsselattribut zum Primärschlüsselattribut.

ID	KuName	KuStrasse
1	N. Jeff	2
2	T. Hurd	4

StrassenNr	Strasse
2	Binzmühlestr
3	Hochstr

KuStrNr kann kein Fremdschlüssel sein weil StrassenNr 4 nicht existiert.

Integrierte Übung 3.4

- Bestimmen Sie die Schlüssel der Relation *R*:

R

X	Y	Z
1	2	3
1	4	5
2	2	2

Integrierte Übung 3.5

- Bestimmen Sie mögliche Superschlüssel, Kandidatschlüssel, Primärschlüssel und Fremdschlüssel für die Relationen *R* und *S*:

R

A	B	C
a	d	e
b	d	c
c	e	e

S

D	E
d	a
e	a
a	a

mögliche Superschlüssel:

mögliche Kandidatschlüssel:

mögliche Primärschlüssel:

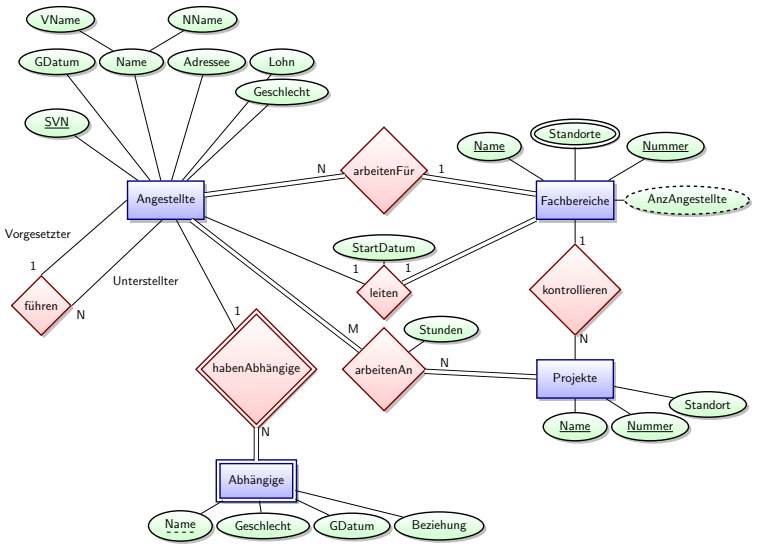
mögliche Fremdschlüssel:

Inhalt

Algorithmus ER-Schema → Relationales Modell

- Algorithmus um ein konzeptionelles ER-Schema (fast) automatisch in ein relationales Schema abzubilden.
  - Schritt 1: unabhängige Entitätstypen
  - Schritt 2: existenzanhängige Entitätstypen
  - Schritt 3: Beziehungstypen
  - Schritt 4: mehrwertige Attribute
  - Schritt 5: n-wertigen Beziehungstypen
  - Schritt 6: Spezialisierung/Generalisierung

Beispiel: ER Schema der NAWI Datenbank



Schritt 1: Abbildung unabhängiger Entitätstypen

(a) Entitätstyp: Für jeden unabhängigen Entitätstypen E erstellen wir eine Relation R.

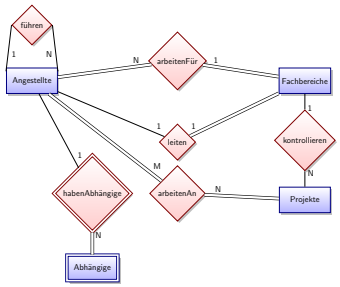
(b) Attribute: Die Attribute von R sind
- alle einfachen Attributen von E
- alle einfachen Komponenten von zusammengesetzten Attributen

(c) Primärschlüssel: Ein Schlüsselattribut von E wird als Primärschlüssel für R ausgewählt.

→ Falls der ausgewählte Schlüssel von E zusammengesetzt ist, besteht der Primärschlüssel aus allen einfachen Komponenten.

Beispiel: Abbildung unabhängiger Entitätstypen

- **Beispiel:** Wir erstellen Relationen Angestellte, Fachbereiche, Projekte.
  - SVN, FNummer, und PNummer sind die Primärschlüssel



Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn]
Fachbereiche[FName, FNummer]
Projekte[PName, PNummer, PStandort]

Schritt 2: Abbildung existenzabhängiger Entitätstypen

(a) Existenzabhängiger Entitätstyp: Für jeden existenzabhängigen Entitätstypen W mit übergeordnetem Entitätstypen E erstellen wir eine Relation R.

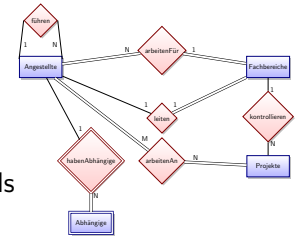
(b) Attribute von R sind alle einfachen Attribute bzw. einfachen Komponenten zusammengesetzter Attribute von W.

(c) Fremdschlüssel: Der Primärschlüssel der Relation des übergeordneten Entitätstypen E wird als Fremdschlüssel zu R hinzugefügt.

(d) Primärschlüssel von R besteht aus der *Kombination* der
- Primärschlüssel der übergeordneten Entitätstypen
- des partiellen Schlüssels des existenzabhängigen Entitätstypen

Beispiel: Abbildung existenzabhängiger Entitätstypen

- **Beispiel:** Der existenzabhängigen Entitätstypen Abhängige wird auf Relation Abhängige abbildet.
- Primärschlüssel SVN von Angestellte wird als Fremdschlüssel zu Relation Abhängige hinzugefügt (umbenannt auf AngSVN).
- Der Primärschlüssel von Abhängige ist die Kombination { AngSVN, AbhName }, weil AbhName ein partieller Schlüssel von Abhängige ist.



Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn]
Fachbereiche[FName, FNummer]
Projekte[PName, PNummer, PStandort]
Abhängige[AngSVN, AbhName, Geschlecht, GDatum, Beziehung]

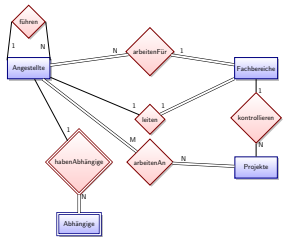
Schritt 3: Abbildung von Beziehungen

Drei mögliche Ansätze für Beziehung zwischen Entitätstypen S und T:

1. Zusammengefasste Relationen: nur für **1:1**
   - beteiligte Entitätstypen werden in einzige Relation zusammengelegt
   - keine Nullwerte falls S und T totale Beziehung eingehen
2. Fremdschlüssel: 1:1, **1:N**
   - eine der beteiligten Entitäten wird ausgewählt, z.B. S (N-Seite im Falle von 1:N)
   - Primärschlüssel von T wird als Fremdschlüssel zu S hinzugefügt
   - keine Nullwerte falls S totale Beziehung eingeht
3. Neue Beziehungsrelation: 1:1, 1:N, **M:N**
   - neue Relation R mit den Primärschlüsseln von S und T als Fremdschlüssel
   - Primärschlüssel:
     - 1 : 1-Beziehung: einer der beiden Fremdschlüssel
     - 1 : N-Beziehung: Fremdschlüssel der N-Seite
     - M : N-Beziehung: beide Fremdschlüssel
   - keine Nullwerte

Beispiel: Abbildung von 1:1 Beziehungstyp

- **Beispiel**: Der 1:1 Beziehungstyp leiten wird mithilfe eines Fremdschlüssels abgebildet. Fachbereiche übernimmt die Rolle von S, weil die Teilnahme in der Beziehung total ist.



Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn]
Fachbereiche[FName, FNummer, LeiterSVN, StartDatum]
Projekte[PName, PNummer, PStandort]
Abhängige[AngSVN, AbhName, Geschlecht, GDatum, Beziehung]

Integrierte Übung 3.6

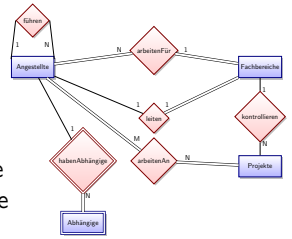
Illustrieren Sie die Probleme die auftreten, wenn der Beziehungstyp leiten

(a) durch einen Fremdschlüssel in der Relation Angestellte abgebildet wird

(b) durch Zusammenfassen der Relationen Angestellte und Fachbereiche abgebildet

Beispiel: Abbildung von 1:N Beziehungstyp

- **Beispiel:** Abbildung des N:1 Beziehungstyps Angestellte arbeitenFür Fachbereiche:
  - Angestellte entspricht der Relation S.
  - Primärschlüssel FNummer von Fachbereiche wird Fremdschlüssel der Relation Angestellte
- Weitere 1:N Beziehungstypen:
  - Angestellte/Vorgesetzte führen Angestellte/Unterstellte: Primärschlüssel von Angestellte als Fremdschlüssel VorgSVN zu Angestellte hinzufügen.
  - Fachbereiche kontrollieren Projekte: Primärschlüssel von Fachbereiche als Fremdschlüssel zu Projekte hinzufügen.



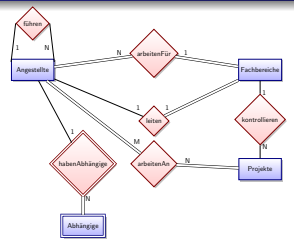
Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn, VorgSVN, FNummer]
Fachbereiche[FName, FNummer, LeiterSVN, StartDatum]
Projekte[PName, PNummer, PStandort, FNummer]
Abhängige[AngSVN, AbhName, Geschlecht, GDatum, Beziehung]

Wie werden Attribute von Beziehungstypen abgebildet?

- Beziehung zwischen Relationen S und T soll abgebildet werden.
- Zusammengefasste Relationen: nur 1:1
  - S und T verschmelzen zu R
  - Attribute[1] des Beziehungstypen werden als Attribute zu R hinzugefügt
- Fremdschlüssel: 1:1, 1:N
  - S erhält Fremdschlüssel (N-Seite im Falle von 1:N)
  - Attribute[1] des Beziehungstypen werden als Attribute zu S hinzugefügt
- Neue Beziehungsrelation: 1:1, 1:N, M:N
  - neue Relation R wird erstellt
  - Attribute[1] des Beziehungstypen werden als Attribute zu R hinzugefügt

[1] alle einfachen Attribute bzw. einfachen Komponenten zusammengesetzter Attribute

Beispiel: Abbildung von M:N Beziehungstyp mit Attributen



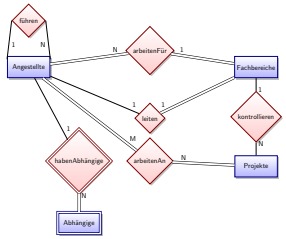
- **Beispiel:** Für den M:N Beziehungstyp arbeitenAn erstellen wir eine Relation ArbeitenAn.
- Die Primärschlüssel der Relationen Projekte und Angestellte werden als Fremdschlüssel zur Relation ArbeitenAn hinzugefügt.
- Attribut *Stunden* der Relation ArbeitenAn bildet das gleichnamige Attribut des Beziehungstypen ab.
- Der Primärschlüssel von ArbeitenAn ist die Kombination der Fremdschlüssel: { AngSVN, PNummer }.

Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn, VorgSVN, FNummer]
Fachbereiche[FName, FNummer, LeiterSVN, StartDatum]
Projekte[PName, PNummer, PStandort, FNummer]
Abhängige[AngSVN, AbhName, Geschlecht, GDatum, Beziehung]
ArbeitenAn[AngSVN, PNummer, Stunden]

Schritt 4: Abbildung mehrwertiger Attribute

- Neue Relation: Für jedes mehrwertige Attribut A erstellen wir eine neue Relation R.
- Attribute: Das mehrwertige Attribut A wird zur Relation R als (einfaches) Attribut hinzugefügt. Falls das mehrwertige Attribut A zusammengesetzt ist, werden alle einfachen Komponenten von A als (einfache) Attribute hinzugefügt.
- Fremdschlüssel: Primärschlüssel K der Relation, die den Entitäts- oder Beziehungstyp von A abbildet.
- Primärschlüssel: Kombination von A und K. Falls das mehrwertige Attribut zusammengesetzt ist, sind alle einfachen Komponenten Teil des Primärschlüssels.

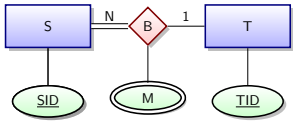
Beispiel 1: Abbildung mehrwertiger Attribute



- **Beispiel:** das mehrwertige Attribut Standorte des Entitätstyps Fachbereiche.
- Eine neue Relation FBStandorte mit Attribut Standort wird erstellt.
- FNummer der Relation Fachbereiche ist Fremdschlüssel in FBStandorte.
- Der Primärschlüssel von FBStandorte sind die Attribute { FNummer, Standort }.

Angestellte[VName, NName, SVN, GDatum, Adresse, Geschlecht, Lohn, VorgSVN, FNummer]
Fachbereiche[FName, FNummer, LeiterSVN, StartDatum]
Projekte[PName, PNummer, PStandort, FNummer]
Abhängige[AngSVN, AbhName, Geschlecht, GDatum, Beziehung]
ArbeitenAn[AngSVN, PNummer, Stunden]
FBStandorte[FNummer, Standort]

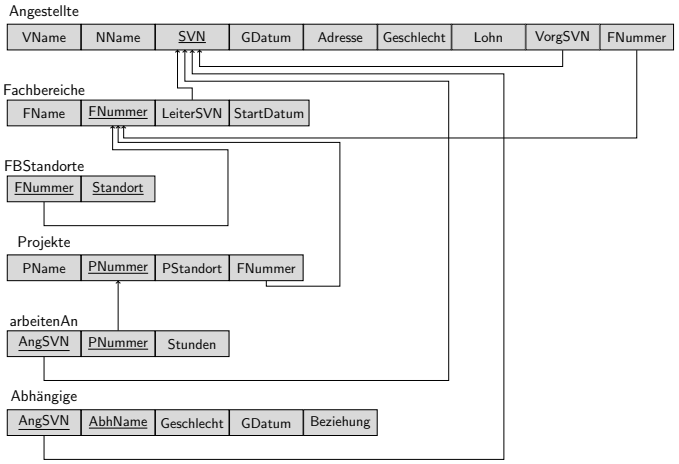
Beispiel 2: Abbildung mehrwertiger Attribute



- **Beispiel:** mehrwertiges Attribut M der 1:N Beziehung B
- 1:N Beziehung wird als Fremdschlüssel in S modelliert
- mehrwertiges Attribut wird durch neue Relation MB modelliert

S[SID, TID]
T[TID]
MB[SID,M]

Beispiel: Vollständige NAWI Datenbank

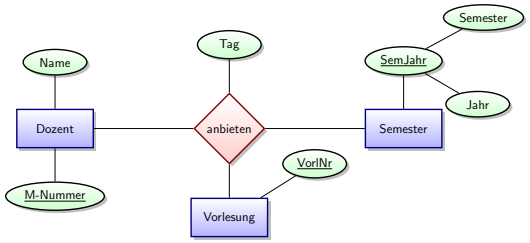


Schritt 5: Abbildung von n-wertigen Beziehungstypen.

- Neue Relation: Für jeden n-wertigen Beziehungstypen ($n > 2$) erstellen wir eine neue Relation R.
- Fremdschlüssel: Die Primärschlüssel der Relationen der involvierten Entitätstypen sind Fremdschlüssel in R.
- Primärschlüssel: Kombination aller Fremdschlüssel.
- Attribute: Alle einfachen Attribute bzw. einfachen Komponenten zusammengesetzter Attribute des M:N Beziehungstypen werden als Attribute zu R hinzugefügt.

Beispiel: Abbildung von n-wertigen Beziehungstypen.

- **Beispiel:** Der 3-wertige Beziehungstyp anbieten



- Der Beziehungstyp wird durch eine neue Relation Anbieten abgebildet.
- Der Primärschlüssel ist die Kombination der drei Fremdschlüssel: { M-Nummer, Jahr, Semester, VorlNr }

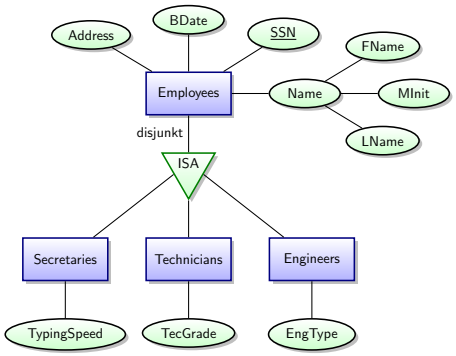
Dozent[M-Nummer, ...]
Semester[Jahr, Semester, ...]
Vorlesung[VorlNr, ...]
Anbieten[M-Nummer, Jahr, Semester, VorlNr, Tag, ...]

Schritt 6: Abbildung von Spezialisierung/Generalisierung

- Notation:
  - Untertyp: $U_1, U_2, \ldots, U_m$
  - Obertyp: O mit Attributen $k, a_1, a_2, \ldots, a_n$
  - k ist Primärschlüssel des Obertypen O
- Umsetzung:
  - Relation R für Obertyp O mit Attributen $attr(R) = \{\underline{k}, a_1, \ldots, a_n\}$.
  - Relation R_i für Untertypen U_i, $1 \leq i \leq m$, mit den Attributen $attr(R_i) = \{\underline{k}\} \cup \{\text{Attribute von } U_i\}$.
  - Attribute k der Relationen R_i sind Fremdschlüssel auf Attribut k in R.
- Kann für alle Arten der Spezialisierung verwendet werden:
  - vollständig und partiell
  - disjunkt und überlappend
- Einschränkung: vollständig und/oder disjunkt wird nicht erzwungen

Beispiel: Abbildung von Spezialisierung

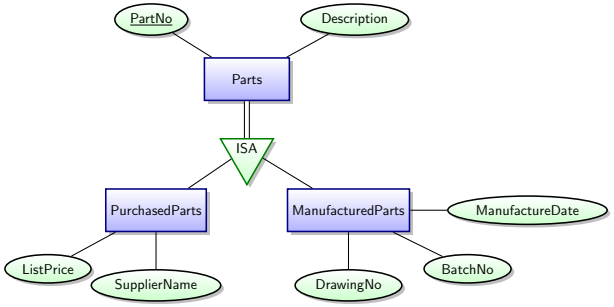
- **Beispiel:** Spezialisierung von Employees



Employees[SSN, FName, MInit, LName, BDate, Address]
Secretaries[SSN, TypingSpeed]
Technicians[SSN, TecGrade]
Engineers[SSN, EngType]

Integrierte Übung 3.7

- Bilden Sie folgendes ER-Diagramm in Relationen ab.



Zusammenfassung der Abbildungen

Abbildung zwischen dem ER und dem relationalem Modell

ER Modell	**Relationales Modell**
Entitätstyp	Entitätsrelation
1:1 oder 1:N Beziehungstyp	Fremdschlüssel (oder Beziehungsrelation)
M:N Beziehungstyp	Beziehungsrelation mit 2 Fremdschlüsseln
n-wertige Beziehungstyp	Beziehungsrelation mit *n* Fremdschlüsseln
(Einfaches) Attribut	Attribut
zusammengesetztes Attribut	Menge von einfachen Attributen
Mehrwertiges Attribut	Relation mit Fremdschlüssel
Schlüsselattribut	Primärschlüssel
Spezialisierung	Relation für Ober- und Untertypen

Datenbanken 1

Relationale Algebra

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 9. April 2018

Inhalt

Literatur und Quellen

Lektüre zum Thema "Relationale Algebra":

- Kapitel 3 (3.4) aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. 9. Auflage, Oldenbourg Verlag, 2013.
- Kapitel 6 (6.1) aus Silberschatz, Korth, and Sudarashan: Database System Concepts, McGraw Hill, 2011.

Literaturquellen

- Elmasri and Navathe: Fundamentals of Database Systems. Fourth Edition, Pearson Addison Wesley, 2004.

Danksagung Die Vorlage zu diesen Folien wurde entwickelt von:

- Michael Böhlen, Universität Zürich, Schweiz
- Johann Gamper, Freie Universität Bozen, Italien

Inhalt

Relationale Algebra

- Die relationale Algebra ist eine prozedurale Anfragesprache.
- Besteht aus sechs (notwendigen) Operatoren:
  - Selektion: σ
  - Projektion: π
  - Mengenvereinigung: $\cup$
  - Mengendifferenz: $-$
  - Kartesisches Produkt: $\times$
  - Umbenennung: ρ (Hilfsoperation)
- Die relationale Algebra ist abgeschlossen:
  - Argumente der Operatoren sind (ein oder zwei) Relationen.
  - Ergebnis der Operatoren ist wieder eine Relation.

Syntaktische Konventionen

- Es ist hilfreich bei der Namensgebung systematisch zu sein.
- Wir verwenden folgende Regeln.
  - Tabellennamen: Großschreibung und Plural
    Beispiele: Vorlesungen, Studenten, Module, R, S
  - Attributnamen: Großschreibung und Singular
    Beispiele: Semester, Jahr, Name, A, B
  - Konstanten (Werte):
    - Numerische Werte: 12, 17.6
    - Zeichenketten: durch Hochkommas begrenzen
      Beispiele: 'Martin', 'Mehr als ein Wort'
- Es gibt keinen einheitlichen Standard; verschiedene Lehrbücher verwenden verschiedene Notationen

Elementare Operatoren

- Selektion σ
- Projektion π
- Mengenvereinigung $\cup$
- Mengendifferenz $-$
- Kartesisches Produkt $\times$
- Umbenennung ρ

Selektion

- Notation: $\sigma_p(R)$ (sigma)
- Selektionsprädikat p ist aus folgenden Elementen aufgebaut:
  - Attributnamen der Argumentrelation R oder Konstanten als Operatoren
  - arithmetische Vergleichsoperatoren ($=, <, \leq, >, \geq$)
  - logische Operatoren: $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
- $p(t)$, $t \in R$ heißt: Prädikat p ist für Tupel t aus Relation R erfüllt.
- Definition: $t \in \sigma_p(R) \Leftrightarrow t \in R \wedge p(t)$
- Beispiel: $\sigma_{FiName='Brugg'}(Konten)$
- Beispiel: $\sigma_{A=B \wedge D>5}(R)$

R

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D>5}(R)$

A	B	C	D
α	α	1	7
β	β	23	10

Projektion

- Notation: $\pi_{A_1,\ldots,A_k}(R)$ (pi)
- $A_1, A_2, \ldots, A_k$ sind Attribute von R und heißen Projektionsliste
- Definition: $t \in \pi_{A_1,\ldots,A_k}(R) \Leftrightarrow \exists x(x \in R \wedge t = x[A_1, \ldots, A_k])$, wobei $x[A_1, A_2, \ldots, A_k]$ ein neues Tupel bezeichnet, welches für die Werte von A_i, $1 \leq i \leq k$, die Werte der entsprechenden Attribute von x annimmt (alle Attribute A_i müssen in x vorkommen)
- Beachte: Allfällige Duplikate (identische Tupel), die sich aus der Projektion ergeben, müssen entfernt werden.
- Beispiel: $\pi_{KoNr,Guthaben}(Konten)$
- Beispiel: $\pi_{A,C}(R)$

R

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\pi_{A,C}(R)$

A	C
α	1
β	1
β	2

Mengenvereinigung

- Notation: $R \cup S$
- Definition: $t \in (R \cup S) \Leftrightarrow t \in R \vee t \in S$
- $R \cup S$ ist nur definiert, wenn r und s das gleiche Schema haben (union compatible). Namensdifferenzen können durch explizites Umbenennung der Attribute eliminiert werden (s. weiter unten).
- Beispiel: $\pi_{KuName}(Kontoinhaber) \cup \pi_{KuName}(Kreditnehmer)$
- Beispiel: $R \cup S$

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

$R \cup S$

A	B
α	1
α	2
β	1
β	3

Mengendifferenz

- Notation: $R - S$
- Definition: $t \in (R - S) \Leftrightarrow t \in R \wedge t \notin S$
- Die Argumentrelationen der Mengendifferenz müssen das gleiche Schema haben (union compatible).
- Beispiel: $R - S$

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

$R - S$

A	B
α	1
β	1

Kartesisches Produkt (Kreuzprodukt)

- Notation: $R \times S$
- Definition: $t \in (R \times S) \Leftrightarrow \exists x, y(x \in R \wedge y \in S \wedge t = x \circ y)$
- $\circ$ bezeichnet die Konkatenation von Tupeln: $[1, 2] \circ [5] = [1, 2, 5]$
- Die Attribute von R und S müssen unterschiedliche Namen haben.
- Beispiel: $R \times S$

R

A	B
α	1
β	2

S

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$R \times S$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Umbenennung

- Erlaubt es den Namen der Relation und der Attribute eines algebraischen Ausdrucks E zu spezifizieren.
- Wird auch verwendet um Namenskonflikte aufzulösen (z.B., in Mengenvereinigung oder Kreuzprodukt)
- Verschiedene Variationen (E ist ein relationaler Ausdruck):
  - $\rho_R(E)$ ist eine Relation mit Namen R.
  - $\rho_{R[A_1,...,A_k]}(E)$ ist eine Relation mit Namen R und Attributnamen $A_1, ..., A_k$.
  - $\rho_{[A_1,...,A_k]}(E)$ ist eine Relation mit Attributnamen $A_1, ..., A_k$.
- Beispiel: $\rho_{S[X,Y,U,V]}(R)$

R

A	B	C	D
α	α	1	7
β	β	23	10

S

X	Y	U	V
α	α	1	7
β	β	23	10

Zusammengesetzte Ausdrücke

- Geschachtelte Ausdrücke: Da die relationale Algebra abgeschlossen ist, d.h. das Resultat eines Operators der relationalen Algebra ist wieder eine Relation, ist es möglich Ausdrücke zu schachteln.
- Beispiel: $\sigma_{A=C}(R \times S)$

R

A	B
α	1
β	2

S

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$R \times S$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$\sigma_{A=C}(R \times S)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Integrierte Übung 4.1

- Identifizieren und korrigieren Sie Fehler in den nachfolgenden relationalen Algebra Ausdrücken. Relation R hat Schema $sch(R) = [A, B]$.
- $\sigma_{R.A>5}(R)$
- $\sigma_{A,B}(R)$
- $R \times R$

Integrierte Übung 4.2

- Identifizieren und korrigieren Sie Fehler in den nachfolgenden relationalen Algebra Ausdrücken. Relation *Pers* hat Schema $sch(Pers) = [Name, Alter, Stadt]$.
- $\sigma_{Name='Name'}(Pers)$
- $\sigma_{Stadt=Zuerich}(Pers)$
- $\sigma_{Alter>'20'}$

Beispiel: Banken

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Fremdschlüssel:

- $\pi_{\text{FiName}}(\text{Konten}) \subseteq \pi_{\text{FiName}}(\text{Filialen})$
- $\pi_{\text{FiName}}(\text{Kredite}) \subseteq \pi_{\text{FiName}}(\text{Filialen})$
- $\pi_{\text{KuName}}(\text{Kontoinhaber}) \subseteq \pi_{\text{KuName}}(\text{Kunden})$
- $\pi_{\text{KoNr}}(\text{Kontoinhaber}) \subseteq \pi_{\text{KoNr}}(\text{Konten})$
- $\pi_{\text{KuName}}(\text{Kreditnehmer}) \subseteq \pi_{\text{KuName}}(\text{Kunden})$
- $\pi_{\text{KrNo}}(\text{Kreditnehmer}) \subseteq \pi_{\text{KrNr}}(\text{Kredite})$

Anfragebeispiele/1

- Jene Kredite die größer als \$1200 sind.

  $\sigma_{Betrag>1200}(Kredite)$

- Die Nummern jener Kredite die größer als \$1200 sind.

  $\pi_{KrNr}(\sigma_{Betrag>1200}(Kredite))$

- Die Namen aller Kunden die einen Kredit oder ein Konto (oder beides) haben.

  $\pi_{KuName}(Kreditnehmer) \cup \pi_{KuName}(Kontoinhaber)$

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Anfragebeispiele/2

- Die Namen aller Kunden die einen Kredit bei der Brugg Filiale haben.
  - Anfrage 1

    $\pi_{KuName}(\sigma_{FiName='Brugg'}($
    $\sigma_{KrNo=KrNr}(Kreditnehmer \times Kredite)))$

  - Anfrage 2

    $\pi_{KuName}(\sigma_{KrNo=KrNr}($
    $(\sigma_{FiName='Brugg'}(Kredite)) \times Kreditnehmer))$

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Anfragebeispiele/3

- Die Namen aller Kunden die einen Kredit bei der Brugg Filiale haben, aber kein Konto bei der Bank.

$\pi_{KuName}(\sigma_{FiName='Brugg'}($
$\sigma_{KrNo=KrNr}(Kreditnehmer \times Kredite)))$
$-$
$\pi_{KuName}(Kontoinhaber)$

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Integrierte Übung 4.3

- Gegeben: Relation $R[A] = \{[1], [2], [3]\}$. Schreiben Sie einen relationalen Algebra Ausdruck der den größten Wert in R bestimmt.

Definition von relationalen Algebra Ausdrücken

- Ein elementarer Ausdruck der relationalen Algebra ist eine Relation in der Datenbank (z.B. Konten).
- Falls E_1 und E_2 relationale Algebra Ausdrücke sind, dann lassen sich weitere relationale Algebra Ausdrücke wie folgt bilden:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p(E_1)$, p ist ein Prädikat in E_1
  - $\pi_s(E_1)$, s ist eine Liste mit Attributen aus E_1
  - $\rho_x(E_1)$, x ist der Name für E_1

Notationsvarianten der Relationalen Algebra

- Im Laufe der Zeit sind unterschiedliche Notationen entstanden.
- Notation von Kemper&Eikler (Lehrbuch) unterscheidet sich wie folgt.
- Qualifizierte Attributnamen
  - Attributnamen werden durch Voranstellen des Relationsnamen eindeutig gemacht (wo nötig), z.B., $R.B$, $S.B$
  - Kreuzprodukt $R \times S$ ist auch dann erlaubt, wenn R und S gleichnamige Attribute haben
  - Beispiele: Gegeben $R[A, B]$, $S[B, C]$
    - $sch(R \times S) = [A, R.B, S.B, C]$
    - $\sigma_{R.B=S.B}(R \times S)$ ist syntaktisch korrekt
- Umbenennung mit Zuordnung
  - Syntax von ρ unterscheidet sich für Relationen und Attribute
  - Relation: $\rho_R(E)$ benennt relationalen Ausdruck E mit R
  - Attribut: $\rho_{A \leftarrow B}(R)$ benennt Attribut A zu B um ($A \in sch(R)$)

In der Prüfung ist die Notation aus der Vorlesung zu verwenden.

Zusammenfassung: Elementare Operatoren

- Relationale Algebra ist prozedural und abgeschlossen.
- Elementare Operatoren:
  - unär: Selektion σ, Projektion π, Umbenennung ρ
  - binär: Mengenvereinigung $\cup$, Mengendifferenz $-$, Kreuzprodukt $\times$
- Ein relationaler Ausdruck kann sein:
  - ein elementarer Ausdruck (Relation)
  - eine Kombination von relationalen Ausdrücken, die über relationale Operatoren verbunden sein müssen

Zusätzliche Operatoren der Relationalen Algebra

- Neben den elementaren Operatoren gibt es zusätzliche Operatoren:
  - Mengendurchschnitt $\cap$
  - Join $\bowtie$
  - Zuweisung $\leftarrow$
- Die zusätzlichen Operatoren machen Algebra nicht ausdrucksstärker:
  - man kann die zusätzlichen Operatoren mithilfe der elementaren Operatoren ausdrücken
  - deshalb sind die zusätzlichen Operatoren *redundant*
- Formulierung häufiger Anfragen wird zum Teil erheblich vereinfacht.

Mengendurchschnitt

- Notation: $R \cap S$
- Definition: $t \in (R \cap S) \Leftrightarrow t \in R \wedge t \in S$
- Voraussetzung: R und S haben das gleiche Schema
- Beachte: $R \cap S = R - (R - S)$
- Beispiel: $R \cap S$

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

$R \cap S$

A	B
α	2

Theta Join (Verbund)/1

- Notation: $R \bowtie_\theta S$
- Annahme: R und S sind Relationen. θ ist ein Prädikat über den Attributen von R und S.
- $R \bowtie_\theta S$ ist eine Relation mit einem Schema das aus allen Attributen von $sch(R)$ und allen Attributen von $sch(S)$ besteht.
- Beachte: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- Beispiel:
  - $sch(R) = [A, B, D]$ und $sch(S) = [X, Y, Z]$
  - $R \bowtie_{A=Z} S$
  - Schema des Resultats ist $[A, B, D, X, Y, Z]$
  - Äquivalent zu: $\sigma_{A=Z}(R \times S)$

R

A	B	D
α	1	a
β	2	a
γ	4	b

S

X	Y	Z
1	a	α
3	a	β
3	b	ϵ

$R \bowtie_{A=Z} S$

A	B	D	X	Y	Z
α	1	a	1	a	α
β	2	a	3	a	β

Theta Join (Verbund)/2

- Beispiel:
  - $sch(R) = [A, B, D]$ und $sch(S) = [X, Y, Z]$
  - $R \bowtie_{A=Z \wedge B<X} S$
  - Schema des Resultats ist $[A, B, D, X, Y, Z]$
  - Äquivalent zu: $\sigma_{A=Z \wedge B<X}(R \times S)$

R

A	B	D
α	1	a
β	2	a
γ	4	b

S

X	Y	Z
1	a	α
3	a	β
3	b	ϵ

$R \bowtie_{A=Z \wedge B<X} S$

A	B	D	X	Y	Z
β	2	a	3	a	β

Natürlicher Join

- Notation: $R \bowtie S$
- Annahme: R und S sind Relationen.
- Der natürliche Join verlangt, dass Attribute die sowohl in R als auch S vorkommen identische Werte haben.
- Das Resultat von $R \bowtie S$ ist eine Relation mit einem Schema das alle Attribute von R enthält und alle Attribute von S die nicht in R vorkommen.
- Beispiel:
  - $R \bowtie S$ mit $sch(R) = [A, B, D]$ und $sch(S) = [B, D, E]$
  - Schema des Resultats ist $[A, B, D, E]$
  - Äquivalent zu: $\pi_{A,B,D,E}(\sigma_{B=Y \wedge D=Z}(R \times \rho_{[Y,Z,E]}(S))$

R

A	B	D
α	1	a
β	2	a

S

B	D	E
1	a	α
3	a	β

$R \bowtie S$

A	B	D	E
α	1	a	α

Semi- und Anti-Join

- Semi-Join: $R \ltimes S$
  - alle Tupel von R die in einem natürlichen Join mit S mindestens einen Join-Partner finden.
  - $R \ltimes S = \pi_{sch(R)}(R \bowtie S)$
- Anti-Join: $R \triangleright S$
  - alle Tupel von R die in einem natürlichen Join mit S keinen Join-Partner finden.
  - $R \triangleright S = R - (R \ltimes S)$

Zuweisung

- Die Zuweisung ($\leftarrow$) erlaubt es, komplexe Ausdrücke in kleinere übersichtliche Blöcke aufzubrechen.
  - links von $\leftarrow$ steht eine Variable
  - rechts von $\leftarrow$ steht ein relationaler Algebra Ausdruck
  - das Resultat rechts von $\leftarrow$ wird der Variablen links von $\leftarrow$ zugewiesen
  - komplexe Ausdrücke werden als Sequenz von Zuweisungen geschrieben

Bankbeispiel Anfragen/1

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Das Konto (bzw. die Konten) mit dem höchsten Kontostand.
- Lösung:
  1. Bestimmen jener Konten die **nicht** den höchsten Kontostand haben (indem man jedes Konto mit allen anderen Konten vergleicht)

$$K \leftarrow \pi_{KoNr}(\sigma_{Guthaben<Guth}(Konten \times \rho_{[Nr,Fil,Guth]}(Konten)))$$

  2. Mit Hilfe der Mengendifferenz werden jene Konten bestimmt die im ersten Schritt nicht gefunden wurden.

$$Result \leftarrow \pi_{KoNr}(Konten) - K$$

Bankbeispiel Anfragen/2

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Alle Kunden die sowohl ein Konto als auch einen Kredit haben.

  $\pi_{KuName}(Kreditnehmer) \cap \pi_{KuName}(Kontoinhaber)$

- Name und Kreditbetrag aller Kunden die einen Kredit haben.

  Lösung 1: $\pi_{KuName,Betrag}(Kreditnehmer \bowtie_{KrNo=KrNr} Kredite)$

  Lösung 2: $\pi_{KuName,Betrag}(\rho_{[KuName,KrNr]}(Kreditnehmer) \bowtie Kredite)$

Bankbeispiel Anfragen/3

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Kunden die sowohl ein Konto bei der Filiale Chur als auch der Filiale Lanquart haben.
  - Lösung:

    $\pi_{KuName}(\sigma_{FiName='Chur'}(Kontoinhaber \bowtie Konten))$
    $\cap$
    $\pi_{KuName}(\sigma_{FiName='Lanquart'}(Kontoinhaber \bowtie Konten))$

Zusammenfassung: Zusätzliche Operatoren

- Zusätzliche Operatoren der relationalen Algebra:
  - Mengendurschnitt $\cap$
  - Join (theta, natural) $\bowtie$
  - Zuweisung $\leftarrow$
- Zusätzliche Operatoren verändern nicht die Ausdrucksstärke der relationalen Algebra, vereinfachen aber die Anfragen.
- Besonders der Join Operator spielt eine große Rolle in der effizienten Implementierung der relationalen Algebra in Systemen.

Operatoren der Erweiterten Relationalen Algebra

Die erweiterten Operatoren erhöhen die Ausdrucksstärke der relationalen Algebra.

- Verallgemeinerte Projektion π
- Gruppierung und Aggregation γ
- Äußerer Join (outer join) $⟗$, $⟕$, $⟖$

Verallgemeinerte Projektion

- Erlaubt arithmetische Funktionen in der Projektionsliste:

$$\pi_{F_1,F_2,\ldots,F_n}(E)$$

- E ist ein relationaler Ausdruck.
- $F_1, F_2, \ldots, F_n$ sind jeweils arithmetische Ausdrücke, welche Konstanten und Attribute des Schemas von E enthalten.
- Beispiel: Gegeben eine Relation $Kredite[Kunde, Limit, KreditBetrag]$, finde heraus, wieviel jeder Kunde noch ausgeben darf:

$$\pi_{Kunde,Limit-KreditBetrag}(Kredite)$$

Aggregationsfunktionen

- Aggregationsfunktionen erhalten eine Multimenge von Werten als Argument und liefern als Ergebnis einen einzigen Funktionswert.
  - avg: Durchschnitt
  - min: kleinster Wert
  - max: größter Wert
  - sum: Summe aller Werte
  - count: Anzahl der Werte (Kardinalität der Menge/Multimenge)
- Elemente der Argumentmenge und Funktionswert sind atomar, nicht Tupel.
- Multimenge (Menge mit Duplikaten): k-fache Werte gehen k-fach in die Berechnung ein.
- Beispiele: ($\{...\}_m$ ist eine Multimenge)
  - $\min(\{3,1,5,5\}_m) = 1$
  - $\text{count}(\{3,1,5,5\}_m) = 4$
  - $\text{avg}(\{3,1,5,5\}_m) = 3.5$

Gruppierung

- Partitionierung der Tupel einer Relation gemäß ihrer Werte in einem oder mehreren Attributen.
- Gruppe (Partition): Alle Tupel mit identischen Werten in allen Gruppierungsattributen.
- Hauptzweck: Aggregation auf Teilen einer Relation (Gruppen)
- Beispiel: Gegeben Relation $R = \{[1,2,3],[1,2,5],[1,4,3],[2,3,5],[2,4,5]\}$ mit Schema $sch(R) = [A,B,C]$.
  - Gruppierung nach Attribut A ergibt die Gruppen $\{[1,2,3],[1,2,5],[1,4,3]\}$ und $\{[2,3,5],[2,4,5]\}$
  - Gruppierung nach den Attributen A, C ergibt die Gruppen $\{[1,2,3],[1,4,3]\}, \{[1,2,5]\}, \{[2,3,5],[2,4,5]\}$

Gruppierungsoperator

- Die Gruppierung der relationalen Algebra:

$$\gamma_{G_1,G_2,\ldots,G_m;F_1(A_1),F_2(A_2),\ldots,F_n(A_n)}(R)$$

  R ist eine Relation:
  - Gruppierungsattribute: $G_1, G_2 \ldots, G_m$ ist eine Liste von Attributen aus R, über die gruppiert wird (kann leer sein)
  - Aggregationsfunktionen: F_i ist eine Aggregationsfunktion
  - Aggregierte Attribute: A_i ist ein Attribut von R
- Leere Attributeliste: Gruppe besteht aus der gesamten Relation R.
- Ergebnis: Relation mit $m + n$ Attributen
  - Anzahl der Tupel entspricht Anzahl der Gruppen (ein Tupel pro Gruppe)
  - die Werte der ersten m Attribute des Tupels einer Gruppe entsprechen $G_1, G_2, \ldots, G_m$ (Wert gleich für alle Tupel in der Gruppe)
  - die letzten n Attribute entsprechen den Funktionsergebnissen von F_i über die (Multi-)menge aller Werte von A_i in der Gruppe

Beispiel: Gruppierungsoperator

- Relation R, $Res \leftarrow \rho_{[SumC]}(\gamma_{sum(C)}(R))$

r

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

Res

sumC
27

- Gesamteinlagen pro Filiale:
  $Res \leftarrow \rho_{[FiName,SumEinlagen]}(\gamma_{FiName;sum(Guthaben)}(Konten))$

Konten

FiName	*KoNr*	*Guthaben*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

Res

FiName	*SumEinlagen*
Perryridge	1300
Brighton	1500
Redwood	700

Äußerer Join (Outer Join)

- Erweiterung des Join Operators, welche Informationsverlust verhindert.
- Berechnet Join und fügt die Tupel, die keinen Join-Partner haben, zum Join-Ergebnis hinzu.
- Varianten:
  - (Voller) äußerer Join ($R \mathbin{⟗} S$): erhält Tupel von R und S
  - Linker äußerer Join ($R \mathbin{⟕} S$): erhält nur Tupel von R (linke Relation)
  - Rechter äußerer Join ($R \mathbin{⟖} S$): erhält nur Tupel von S (rechte Relation)
- Verwendet `null` Werte, um die neuen Attribute der Tupel ohne Join-Partner zu füllen.
- Analog zum "normalen" (inneren) Join gibt es einen natürlichen und einen theta äußeren Join.

Beispiel: Äußerer Join/1

- Beispiel Relationen:

Kredite

KrNr	*FiName*	*Betrag*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Kreditnehmer

KuName	*KrNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Join (auch "innerer" Join genannt)

Kredite $\bowtie$ *Kreditnehmer*

KrNr	*FiName*	*Betrag*	*KuName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Beispiel: Äußerer Join/2

- Beispiel Relationen:

Kredite

KrNr	*FiName*	*Betrag*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Kreditnehmer

KuName	*KrNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Linker äußerer Join (erhält Tupel der linken Relation)

Kredite $\mathbin{⟕}$ *Kreditnehmer*

KrNr	*FiName*	*Betrag*	*KuName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Beispiel: Äußerer Join/3

- Beispiel Relationen:

Kredite

KrNr	*FiName*	*Betrag*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Kreditnehmer

KuName	*KrNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Rechter äußerer Join (erhält Tupel der rechten Relation)

Kredite ⟕ *Kreditnehmer*

KrNr	*FiName*	*Betrag*	*KuName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Beispiel: Äußerer Join/4

- Beispiel Relationen:

Kredite

KrNr	*FiName*	*Betrag*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Kreditnehmer

KuName	*KrNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- (Vollständiger) äußerer Join (erhält Tupel beider Relationen)

Kredite ⟗ *Kreditnehmer*

KrNr	*FiName*	*Betrag*	*KuName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Zusammenfassung: Erweiterte Relationale Algebra

- Erweiterte Relationale Algebra ist ausdrucksstärker als elementare relationale Algebra.
- Verallgemeinerte Projektion π: Arithmetik in Projektionsliste
- Gruppierung und Aggregation γ: Berechnung über Gruppen von Attributwerten
- Äußerer Join ⟖, ⟕, ⟗: Tupel-erhaltender Join

Änderung der Datenbank

- Der Inhalt der Datenbank kann mithilfe folgenden Operatoren verändert werden:
  - Löschen (delete)
  - Einfügen (insert)
  - Ändern (update)
- All diese Operationen verwenden den Zuweisungsoperator.

Löschen

- Ausdruck ähnlich einer Anfrage, wobei die Ergebnistupel von der Datenbank entfernt werden.
- Nur ganze Tupel können entfernt werden; Werte einzelner Attribute können nicht entfernt werden.
- Löschen wird in der relationalen Algebra folgendermaßen ausgedrückt:

$$R \leftarrow R - E$$

wobei R eine Relation ist und E ein Ausdruck der relationalen Algebra.

Beispiel: Löschen

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Lösche alle Konten in der Filiale Domplatz:

$$R_1 \leftarrow \sigma_{FiName='Domplatz'}(Konten)$$
$$Konten \leftarrow Konten - R_1$$
$$R_2 \leftarrow \pi_{KuName,KoNr}(R_1 \bowtie Kontoinhaber)$$
$$Kontoinhaber \leftarrow Kontoinhaber - R_2$$

Einfügen

- Es gibt zwei Möglichkeiten, um Daten in die Relation einzufügen:
  - die einzufügenden Tupel explizit angeben
  - eine Anfrage schreiben deren Ergebnis eingefügt werden soll
- Einfügen wird in der relationalen Algebra folgendermaßen ausgedrückt:

$$R \leftarrow R \cup E$$

wobei R eine Relation und E ein relationaler Ausdruck sind.

- Wird ein einzelnes, explizites Tupel eingefügt, ist E eine konstante Relation die nur ein Tupel enthält.

Beispiel: Einfügen/1

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Füge folgende Information in die Datenbank ein: Kunde Smith eröffnet ein neues Konto mit Nummer A-973 auf der Domplatz Filiale und legt 1200 EUR ein.

$$Konten \leftarrow Konten \cup \{[\text{'A-973'}, \text{'}Domplatz\text{'}, 1200]\}$$
$$Kontoinhaber \leftarrow Kontoinhaber \cup \{[\text{'}Smith\text{'}, \text{'A-973'}]\}$$

Beispiel: Einfügen/2

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

- Alle Kreditnehmer der Domplatz Filiale erhalten ein Konto mit 200 EUR Guthaben geschenkt, wobei die Kontonummer des neuen Kontos identisch mit der jeweiligen Kreditnummer ist.

$$R_1 \leftarrow \sigma_{FiName='Domplatz'}(Kreditnehmer \bowtie_{KrNo=KrNr} Kredite)$$
$$Konten \leftarrow Konten \cup \rho_{KoNr,FiName,Guthaben}(\pi_{KrNr,FiName}(R_1) \times \{[200]\})$$
$$Kontoinhaber \leftarrow Kontoinhaber \cup \rho_{KuName,KoNr}(\pi_{KuName,KrNr}(R_1))$$

Änderung

- Änderungen erlauben, einzelne Werte eines Tupels zu ändern, ohne alle Werte ändern zu müssen.
- Kann durch Löschen und Einfügen ausgedrückt werden.
  - in realen Systemen ist die Änderungsoperation jedoch oft viel schneller
  - deshalb gibt es einen eigenen Operator
- In relationaler Algebra werden Änderungen in der Relation R durch Ersetzten der Relation R durch einen relationalen Ausdruck E ausgedrückt:

$$R \leftarrow E$$

- Oft ist E eine erweiterte Projektion über $R[A_1, A_2, \ldots, A_n]$:

$$R \leftarrow \rho_{[A_1,A_2,\ldots,A_n]}\pi_{F_1,F_2,\ldots,F_n}(R)$$

wobei F_i
  - entweder A_i ist, falls Attribut A_i nicht geändert werden soll
  - oder eine Funktion, die einen neuen Wert für A_i festlegt.

Beispiel: Änderung

- Auszahlung der Zinsen von 5% auf alle Konten:

$$Konten \leftarrow \rho_{[KoNr,FiName,Guthaben]}\pi_{KoNr,FiName,Guthaben*1.05}(Konten)$$

- Zahle 6% Zinsen für alle Konten mit mehr als 10.000 EUR Guthaben und 5% für alle anderen Konten:

$$Konten \leftarrow$$
$$\rho_{[KoNr,FiName,Guthaben]}($$
$$\pi_{KoNr,FiName,Guthaben*1.06}(\sigma_{Guthaben>10000}(Konten)))$$
$$\cup$$
$$\rho_{[KoNr,FiName,Guthaben]}($$
$$\pi_{KoNr,FiName,Guthaben*1.05}(\sigma_{Guthaben\leq 10000}(Konten)))$$

Zusammenfassung

- Relationale Manipulationssprache
  - Löschen, Einfügen, Ändern
  - Wird durch Zuweisungsoperator ($\leftarrow$) und Ausdrücken der relationalen Algebra ausgedrückt.

Zusammenfassung

Relationale Algebra:

- Elementare Operatoren: notwendig
- Zusätzliche Operatoren: redundant (können durch elementare Operatoren ausgedrückt werden)
- Erweiterte Operatoren: erhöhen die Ausdrucksstärke
- Manipulationssprache: Zuweisungsoperator und relationale Algebra

Datenbanken 1
SQL

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

database research group
http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 12. Juni 2018

Inhalt

Literatur und Quellen

Lektüre zum Thema "SQL":

- Kapitel 4 aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. 8. Auflage, Oldenbourg Verlag, 2011.

Literaturquellen

- Elmasri and Navathe: Fundamentals of Database Systems. Fourth Edition, Pearson Addison Wesley, 2004.
- Silberschatz, Korth, and Sudarashan: Database System Concepts, McGraw Hill, 2006.

Danksagung Die Vorlage zu diesen Folien wurde entwickelt von:

- Michael Böhlen, Universität Zürich, Schweiz
- Johann Gamper, Freie Universität Bozen, Italien

Inhalt

Geschichte/1

- Die IBM Sprache Sequel wurde als Teil des System R Projekts am IBM San Jose Research Laboratory entwickelt.
- Umbenannt in Structured Query Language (SQL)
- ANSI und ISO standard SQL:
  - SQL-86, SQL-89: erste Standards, sehr ähnlich
  - SQL-92 (auch SQL2): größere Revision
    - entry level: entspricht in etwa SQL-89
    - weiters gibt es: intermediate level, full level
  - SQL:1999 (auch SQL3) – Rekursion, reguläre Ausdrücke, Trigger u.A.
  - SQL:2003 – Bug fixes zu SQL:1999, erste XML Unterstützung, WINDOW Funktionen, MERGE Befehl
  - SQL:2006 – Verbesserte XML Unterstützung, Einbindung von XQuery
  - SQL:2008 – viele kleinere Zusätze und Verbesserungen
  - SQL:2011 – Temporal Database Funktionalität
- Kommerzielle Systeme bieten:
  - einen Grossteil der Funktionen von SQL-92
  - eine Anzahl von Funktionen von späteren Standards
  - zusätzliche, proprietäre Funktionen

Geschichte/2

- Don Chamberlin holds a Ph.D. from Stanford University.
- He worked at IBM Almaden Research Center doing research on database languages and systems.
- He was a member of the System R research team that developed much of today's relational database technology.
- He designed the original SQL database language (together with Ray Boyce, 1947–1974).

`http://researcher.watson.ibm.com/researcher/view.php?person=us-dchamber`

Modell und Terminologie

- SQL verwendet die Begriffe Tabelle, Spalte und Zeile.
- Vergleich der Terminologie:

SQL	Relationale Algebra
Tabelle	Relation
Spalte	Attribut
Zeile	Tupel
Anfrage	relationaler Algebra Ausdruck

- In einer Tabelle kann die gleiche Zeile mehrmals vorkommen.
- Zwischen den Zeilen der Tabelle besteht keine Ordnung.

DDL, DML und DCL

SQL besteht aus drei unterschiedlichen Teilen:

- DDL – Date Definition Language (Datendefinitionssprache): Schema erstellen, z.B. CREATE TABLE
- DML – Data Manipulation Language (Datenmanipulationssprache), weiter unterteilt in
  - Anfragesprache: Anfragen, die keine Daten ändern, z.B. SELECT
  - Sonstige DML-Befehle: Anfragen, die Daten ändern können, z.B. UPDATE, INSERT, DELETE, COMMIT
- DCL – Data Control Language (Datenkontrollsprache): Zurgriffsrechte verwalten, z.B. GRANT

Inhalt

Datendefinitionssprache

Erlaubt die Spezifikation unterschiedlicher Eigenschaften einer Tabelle, zum Beispiel:

- Das Schema einer Tabelle.
- Die Domäne zu jeder Spalte.
- Integritätsbedingungen, welche alle Instanzen erfüllen müssen.
- Indexe (Schlagwortverzeichnisse), die für Tabellen aufgebaut werden sollen.
- Die physische Speicherstruktur jeder Tabelle.

Vordefinierte Domänen in SQL

char(n)	Zeichenkette von maximal n Zeichen; nicht genutzte Zeichen werden mit Leerzeichen aufgefüllt.
varchar(n)	Zeichenkette von maximal n Zeichen; variable Speicherlänge
integer	Eine ganze Zahl (maximale Grösse ist maschinenabhängig).
smallint	Eine kleine ganze Zahl (max. Grösse maschinenabhängig).
numeric(p,d)	Festkommazahl mit einer Präzision von p Ziffern, wovon d von diesen Ziffern rechts vom Komma stehen.
real, double precision	Gleitkommazahl mit einfacher bzw. doppelter Genauigkeit. Die Genauigkeit ist maschinenabhängig.
float(n)	Gleitkommazahl mit einer Genauigkeit von mindestens n binären Ziffern.

Create Table

- Eine SQL Tabelle wird mit dem Befehl **create table** definiert:

  create table R(
  A_1 D_1, A_2 $D_2, \ldots, A_n$ D_n,
  (Integritätsbedingung$_1$),
  $\ldots$,
  (Integritätsbedingung$_k$))

  - R ist der Name der Tabelle
  - $A_i, 1 \leq i \leq n$, ist eine Spalte der Tabelle
  - D_i ist die Domäne von Spalte A_i
  - D_i ist von **not null** gefolgt, falls Spalte A_i keine *null*-Werte erlaubt

- Beispiel:

  create table *Filialen*(
  FiName varchar(15) **not null**,
  TlfNr varchar(10),
  Umsatz integer)

Integritätsbedingungen

- Bedingungen auf Spalten: **not null**, **check** (Bedingung auf Spalte)
- Bedingungen auf Tabelle:
  - **unique** $(A_1, \ldots, A_n)$
  - **primary key** $(A_1, \ldots, A_n)$
  - **foreign key** $(A_1, \ldots, A_n)$ **references** $T(B_1, \ldots, B_n)$
  - **check** (Bedingung auf eine oder mehrere Spalten)
- Beispiel: *KoNr* als Primärschlüssel der Tabelle *Konten* definieren:

  create table *Konten*(
  KoNr integer, *FiName* varchar(30), *Guthaben* integer,
  check (*Guthaben* >= 0),
  primary key (*KoNr*))
- Beispiel: *KoNum* als Fremdschlüssel in der Tabelle *Kontoinhaber*:

  create table *Kontoinhaber*(
  KuName varchar(15), *KoNum* integer,
  foreign key (*KoNum*) **references** Konten(KoNr))

Notation/1

- SQL ist eine umfangreiche Sprache und stellt verschiedene syntaktische Konstrukte zur Verfügung, um Tabellen und Integritätsbedingungen zu definieren.
- Oft gibt es mehrere Möglichkeiten, um etwas auszudrücken.
- Die genaue Syntax hängt auch vom Datenbanksystem und oft sogar von der verwendeten Version ab.
- Bei Syntaxproblemen ist die genaue Syntax nachzuschlagen (Manual, Web, Forum).
- Wir verwenden einen kleinen Kern von SQL, der allgemein und mehrheitlich unabhängig vom Datenbanksystem und der Version ist.

Notation/2

- Groß- und Kleinschreibung von reservierten Wörtern:
  - In SQL ist Gross- und Kleinschreibung von reservierten Wörtern irrelevant (z.B. SELECT, select, SeLEct).
  - Im Programmcode werden reservierte Wörter meistens groß geschrieben (z.B. SELECT).
  - In den Vorlesungsunterlagen verwenden wir Fettschrift für reservierte Wörter (z.B. **select**).
- Groß- und Kleinschreibung von Bezeichnern:
  - In Bezeichnern kann Gross- und Kleinschreibung eine Rolle spielen (z.B. Tabellennamen in MySQL Linux).
  - Gross- und Kleinschreibung ist relevant, falls man den Bezeichner unter Anführungszeichen stellt (select "KundenName").
  - PostgreSQL verwenden doppelte Hochkommas für Bezeichner ("abcde"), MySQL erlaubt wahlweise Backticks (`abcde`) oder doppelte Hochkommas.
- Das Ende eines SQL Befehls wird oft durch einen Strichpunkt markiert

  select * from Konten;

Drop und Alter Table

- Der **drop table** Befehl löscht alle Informationen einer Tabelle von der Datenbank, z.B. **drop table** *Filialen*
- Der **alter table** Befehl wird verwendet, um neue Spalten zu einer Tabelle hinzuzufügen. Die Werte für die neue Spalte sind:
  - x, falls **default** x für die Spalte spezifiziert ist,
  - ansonsten *null*

  Beispiel: Spalte *AnzMitarbeiter* als neuen Spalte vom Typ **integer** in Tabelle *Filialen* einfügen (neue Werte sind *null*)

  alter table *Filialen* **add** *AnzMitarbeiter* **integer**
- Der **alter table** Befehl kann auch verwendet werden, um eine Spalte von einer Tabelle zu löschen:

  alter table *Filialen* **drop** *TlfNr*

  wobei *TlfNr* der Name einer Spalte von Tabelle *Filialen* ist.

Zusammenfassung: DDL

- SQL DDL erlaubt
  - das Schema einer Tabelle zu definieren
  - jeder Spalte eine Domäne zuzuordnen
  - Integritätsbedingungen für Spalten anzugeben

  Viele weitere Möglichkeiten, z.B., Indices festlegen.
- Vordefinierte Domänen: **varchar**, **integer**, **float**, ...
- Integritätsbedingungen:
  not null, **unique**, **primary key**, **foreign key**, **check**
  SQL kennt noch viele weitere Integritätsbedingungen.
- Schema kann nachträglich mit **alter table** geändert werden.
- Tabellen können mit **drop table** gelöscht werden.

Inhalt

Ausdrücke und Prädikate

- Ausdrucksstarke Ausdrücke und Prädikate (Bedingungen) machen Computersprachen anwenderfreundlich.
- Datenbankfirmen messen sich anhand der angebotenen Ausdrücke und Prädikate (sowohl Funktionalität als auch Geschwindigkeit).
- Die effiziente Auswertung von Prädikaten ist ein wichtiger Aspekt on Datenbanksystemen.
- Beispiel: 1 Milliarde Tupel und die folgenden Prädikate:

  Nachname = 'Miller'
  Nachname like 'Ester%'
  Nachname like '%mann'
  length(Nachname) < 5

  - Eine alphabetische Ordnung unterstützt die effiziente Evaluierung des 1. und 2. Prädikats nicht aber des 3. und 4. Prädikats.
  - Das ist einer der Gründe warum die Definition von Prädikaten und Funktionen durch den Benutzer limitiert war/ist.

Struktur von SQL Anfragen/1

- SQL basiert auf Relationen und relationalen Operatoren mit gewissen Änderungen und Erweiterungen (z.B. Duplikate).
- SQL ist sehr weit verbreitet in der Geschäftswelt.
- SQL ist weit mehr als einfache select-from-where Anfragen wie z.B.:

```
select *
from Kunden
where KundenName = 'Bohr'
```

- Viele Benutzer/Programmierer...
  - unterschätzen SQL
  - verstehen nicht die Konzepte, die sich hinter der Syntax verbergen
  - verstehen nicht, wie mit einer deklarativen Sprache und mit Mengen zu arbeiten ist (dies braucht eine gewisse Übung)

Struktur von SQL Anfragen/2

- Eine typische SQL Anfrage hat folgende Form:

select Teil
from Teil
where Teil
group Teil
having Teil
} *Anfrage–spezifikation*

union

select Teil
from Teil
where Teil
group Teil
having Teil
} *Anfrage–spezifikation*

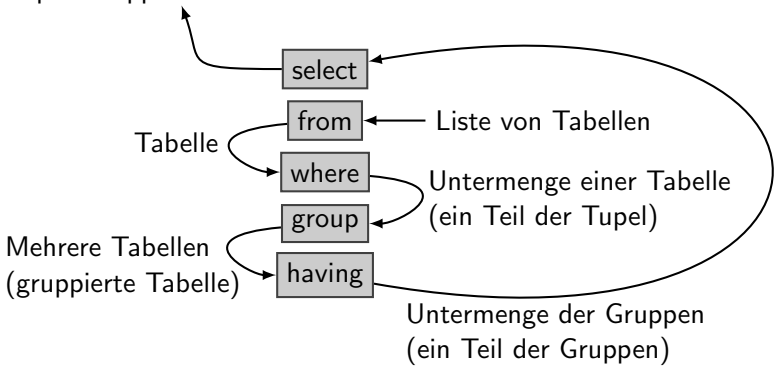
order Teil

} *Anfrage–ausdruck*

- Das Resultat einer SQL Anfrage ist eine (virtuelle) Tabelle.

Illustration: Evaluierung einer Anfragespezifikation/1

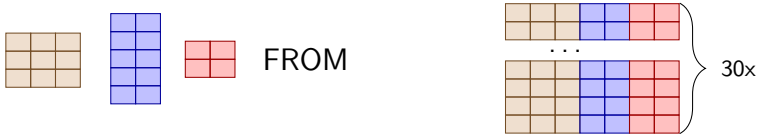
Berechnet eine Zeile pro Gruppe; oft wird eine Aggregation pro Gruppe berechnet



Aggregation: min, max, sum, count, avg einer Menge von Werten.

Illustration: Evaluierung einer Anfragespezifikation/2

1. FROM: bilden des Kreuzprodukts aller Tabellen im **from** Teil



2. WHERE: eliminiert Tupel die die Bedingung im **where** Teil nicht erfüllen

3. GROUP BY: gruppiert Tupel gemäss den Spalten im **group** Teil

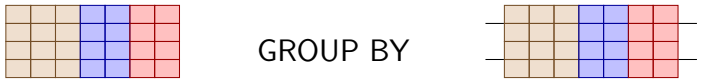


Illustration: Evaluierung einer Anfragespezifikation/3

4. HAVING: eliminiert Gruppen welche die Bedingung des **having** Teil nicht erfüllen

5. SELECT: evaluiert die Ausdrücke im **select** Teil und produziert ein Ergebnistuple für jede Gruppe

Konzeptionelle Evaluierung eines Anfrageausdrucks

1. Bilden des Kreuzprodukts aller Tabellen im **from** Teil.
2. Eliminierung aller Tupel die den **where** Teil nicht erfüllen.
3. Gruppierung der verbleibenden Tupel gemäss **group** Teil.
4. Eliminierung der Gruppen die den **having** Teil nicht erfüllen.
5. Evaluierung der Ausdrücke im **select** Teil.
6. Für jede Gruppe wird genau ein Resultattupel berechnet
7. Duplikate werden eliminiert falls **distinct** spezifiziert wurde.
8. Anfragespezifikationen werden unabhängig ausgewertet und anschliessend werden die Teilresultate durch die angegebenen **Mengenopertionen** (union, except, intersect) kombiniert.
9. Sortierung des Resultats gemäss **order** Teil.

Der from Teil

- Der **from** Teil listet die Tabellen, die in der Anfrage involviert sind.
  - Entspricht dem kartesischen Produkt in der relationalen Algebra.
- Kartesisches Produkt von *Kreditnehmer* und *Kredite*
  from *Kreditnehmer*, *Kredite*
- Kartesisches Produkt von *Kreditnehmer* und *Kredite* mit Umbenennung:
  from *Kreditnehmer* **as** *T*, *Kredite* **as** *S*
- Umbenennung wird notwendig, wenn die gleiche Tabelle mehrmals im **from** Teil vorkommt.
  from *Kredite* **as** *K*1, *Kredite* **as** *K*2

Der where Teil/1

- Der **where** Teil spezifiziert Bedingungen, die Ergebnistupel erfüllen müssen.
- Input: Der **where** Teil arbeitet mit der virtuellen Tabelle, die der **from** Teil produziert und behält alle Zeilen, welche die Bedingung erfüllen.
- Beispiel: Kredite der Brugg Filiale, die grösser als $1200 sind.

  from *Kredite*
  where *FiName* = 'Brugg' **and** *Betrag* > 1200

KrNr	*FiName*	*Betrag*
L-260	Brugg	1700

- Der **where** Teil entspricht dem Selektionsprädikat.
- Prädikate können über die logischen Operatoren **and**, **or**, und **not** verknüpft werden.

Der where Teil/2

- Der **where** Teil kann verwendet werden, um Join- oder Selektionsbedingungen zu spezifizieren.
- Selektionsbedingung: schränkt Attributwerte einer einzigen Tabelle aus dem **from** Teil ein.
  - **from** Filialen **where** FiName = 'Brugg'
  - **from** Filialen, Kredite **where** Betrag > 12000
- Joinbedingung: verknüpft Tupel von zwei Tabellen durch Prädikate, die jeweils Attribute beider Tabellen enthalten.
  - **from** Kreditnehmer, Kredite **where** KrNo = KrNr

Integrierte Übung 5.1

Übersetzen Sie die folgenden Ausdrücke der relationalen Algebra in äquivalente SQL Fragmente:

1. $R \times S$
2. $(R \times S) \times T$
3. $\sigma_{A>5}(R)$
4. $\sigma_{A>5}(\sigma_{B=4}(R))$
5. $\sigma_{A=X}(R \times S)$
6. $\sigma_{A>5}(R) \times \sigma_{X=7}(S)$

Der group Teil

- Der **group** Teil partitioniert eine Tabelle in nicht-überlappende Teilmengen von Tupeln (=Gruppen).
- Input: Der **group** Teil nimmt die Tabelle, die der **where** Teil produziert hat und berechnet darauf die Gruppen.
- Konzeptionell gibt **group** mehrere Tabellen zurück.
- Beispiel: Konten gruppiert nach Filialen.

  from *Konten*
  group by *FiName*

Konten

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Der having Teil/1

- Input: Der **having** Teil nimmt eine gruppierte Tabelle und berechnet eine neue gruppierte Tabelle (mit ausgewählten Gruppen).
- Die **having** Bedingung wird auf jede Gruppe angewandt; nur jene Gruppen, welche die Bedingung erfüllen werden zurückgegeben.
- Die **having** Bedingung kann sich nur auf gruppierte oder aggregierte Attribute beziehen (weil die Bedingung entweder alle oder kein Tupel einer Gruppe auswählen muss).
- Alles oder nichts: Der **having** Teil gibt nie individuelle Tupel einer Gruppe zurück (entweder die gesamte Gruppe oder nichts).

Der having Teil/2

- Filialen mit mehr als einem Konto:

  from *Konten*
  group by *FiName*
  having count(KoNr) > 1

- Dieser **having** Teil gibt alle Gruppen mit mehr als einem Tupel zurück:

Konten

KoNr	*FiName*	*Guthaben*
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Integrierte Übung 5.2

- Welche der folgenden SQL Fragmente sind korrekt?

```
from Konten
group by FiName
having Guthaben < 730
```

```
from Konten
group by FiName
having FiName = 'Chur'
or FiName = 'Aarau'
```

```
from Konten
group by FiName
having sum(Guthaben) < 1000
```

Konten

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Der select Teil/1

- Der **select** Teil spezifiziert die Spalten, die im Resultat vorkommen sollen.
- Entspricht der Projektion in der relationalen Algebra.
- Beispiel: Namen aller Kunden:

```
select KuName
from Kunden
```

- Äquivalente Anfrage in relationaler Algebra (Beachte: *KuName* ist Primärschlüssel und hat keine Duplikate):

$$\pi_{KuName}(Kunden)$$

Der select Teil/2

- SQL erlaubt Duplikate in Tabellen und im Resultat einer Anfrage.
- Duplikate können in SQL durch **distinct** eliminiert werden.
- Beispiel: Die Namen aller Filialen, die Kredite vergeben:
  - SQL:
    1. **select** *FiName* **from** *Kredite*
    2. **select distinct** *FiName* **from** *Kredite*
  - Relationale Algebra:

$$\pi_{FiName}(Kredite)$$

- SQL 1 ist *nicht* äquivalent zu $\pi_{FiName}(Kredite)$:
  - durch die Projektion entstehen Duplikate (mehrere Tupel von *Kredite* können denselben Wert für *FiName* haben)
  - relationale Algebra: die Duplikate im Ergebnis werden eliminiert
  - SQL: Duplikate werden nicht eliminiert
- SQL 2 ist äquivalent zu $\pi_{FiName}(Kredite)$:
  - **select distinct** eliminiert Duplikate im Ergebnis

Der select Teil/3

- Im **select** Teil können Aggregationsfunktionen verwendet werden:
  - **avg**: Durchschnittswert
  - **min**: kleinster Wert
  - **max**: grösster Wert
  - **sum**: Summe aller Werte
  - **count**: Anzahl Werte
- Die Aggregatfunktionen verarbeiten alle Zeilen einer Gruppe und berechnen einen aggregierten Wert für diese Gruppe.
- Falls es einen **group** Teil gibt, dürfen im **select** Teil nur folgende Attribute vorkommen:
  - gruppierte Attribute: kommen im **group** Teil vor
  - aggregierte Attribute: beliebiges Attribut als Argument einer Aggregatfunktion
- Falls der **group** Teil fehlt und Aggregationsfunktionen verwendet werden, bildet die gesamte Tabelle die einzige Gruppe.

Der select Teil/4

- Der Stern * im **select** Teil bedeutet "alle Spalten"

  select *
  from *Kredite*

- **count**(*) berechnet die Anzahl der Tupel pro Gruppe
  - **count**($*$) zählt Tupel, auch wenn diese nur *null*-Werten speichern
  - **count**(A) zählt nur Attributwerte von A, die *nicht null* sind
- Beispiel:

R

A
3
3
null

select *
from R

A
3
3
null

select count(*)
from R

count
3

select count(A)
from R

count
2

The select Teil/5

- Das durchschnittliche Guthaben auf den Konten der Brugg Filiale.

  select avg(*Guthaben*)
  from *Konten*
  where *FiName* = 'Brugg'

- Anzahl der Tupel in der Kunden Tabelle.

  select count($*$)
  from *Kunden*

- Die Anzahl der Konten pro Filiale.

  select count(KoNr), FiName
  from *Konten*
  group by *FiName*

Integrierte Übung 5.3

Formulieren Sie die folgenden Anfragen in SQL:

1. Die Nummern jener Kredite, deren Betrag grösser als 1200 ist

2. Die Namen aller Kunden, die einen Kredit bei der Brugg Filiale haben

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Integrierte Übung 5.4

Formulieren Sie die folgenden Anfragen in SQL:

1. Von jeder Filiale das grösste Guthaben.

2. Von jeder Filiale das grösste und kleinste Guthaben.

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Konten

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Anfrageausdruck/1

- Die Mengenoperationen **union**, **intersect**, und **except** entsprechen den relationalen Operatoren $\cup$, $\cap$, $-$
- Keine Duplikate: Jeder der Operatoren wird auf Tabellen ohne Duplikate angewandt und gibt ein Resultat ohne Duplikate zurück.
- Um Duplikate zu bewahren werden erweiterte Mengenoperationen verwendet: **union all**, **intersect all**, und **except all**.

  Annahme: ein Tupel kommt m mal in R und n mal in S vor. In diesem Fall kommt das Tupel:
  - $m + n$ mal in R **union all** S vor
  - $min(m, n)$ mal in R **intersect all** S vor
  - $max(0, m - n)$ mal in R **except all** S vor
- Union compatibility:
  - Im Unterschied zur relationalen Algebra müssen die Attributnamen in den Schemata nicht übereinstimmten.
  - Die Typen der entsprechenden Spalten müssen jedoch kompatibel sein.

Anfrageausdruck/2

- Alle Kunden die Kredite oder Konten haben:

  (**select** *KuName* **from** *Kontoinhaber*)
  union
  (**select** *KuName* **from** *Kreditnehmer*)

- Kunden die sowohl einen Kredite wie auch ein Konto haben:

  (**select** *KuName* **from** *Kontoinhaber*)
  intersect
  (**select** *KuName* **from** *Kreditnehmer*)

- Kunden die ein Konto aber keinen Kredit haben:

  (**select** *KuName* **from** *Kontoinhaber*)
  except
  (**select** *KuName* **from** *Kreditnehmer*)

Notation

- Um Namenskonflikte aufzulösen können qualifizierte Bezeichner verwendet werden:
  - T.C anstatt C
  - T.C bedeutet Spalte C aus Tabelle T
- Tabellen (und Spalten) können mit **as** umbenannt werden:
  - **from** Kunden **as** K
  - **select** max(Lohn) **as** GroessterLohn
- Eigenheiten realer Systeme:
  - In MySQL und PostgreSQL kann **as** in **from** und **select** Teil weggelassen werden
  - In Oracle muss **as** im **from** Teil weggelassen werden und kann im **select** weggelassen werden
  - Oracle verwendet MINUS statt EXCEPT für Mengendifferenz.
  - In MySQL existiert keine Mengendifferenz (EXCEPT) und kein Mengendurchschnitt (INTERSECT).

Integrierte Übung 5.5

- Formulieren Sie folgende Anfrage in SQL:

  Bestimmen Sie das größte Guthaben von Filialen, welche ein Guthabenvolumen (Summe aller Guthaben in einer Filiale) von mehr als 2000 haben.

Filialen[FiName, Stadt, Umsatz]
Kunden[KuName, Strasse, Ort]
Konten[KoNr, FiName, Guthaben]
Kredite[KrNr, FiName, Betrag]
Kontoinhaber[KuName, KoNr]
Kreditnehmer[KuName, KrNo]

Konten

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Integrierte Übung 5.6

- Identifizieren Sie Probleme der folgenden SQL Anfrage:

  Bestimmen Sie für jede Filiale die Konten mit dem grössten Guthaben.

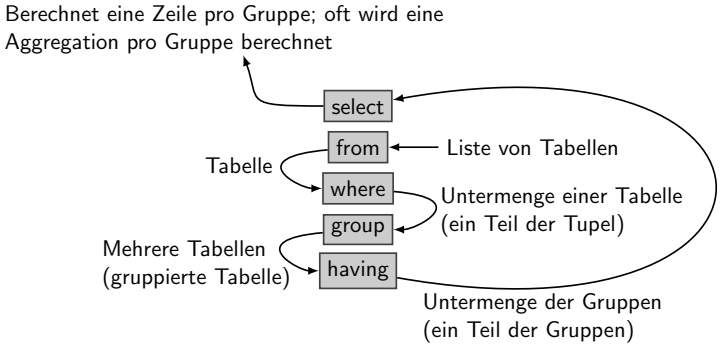
```
select max(Guthaben), KoNr, FiName
from Konten
group by FiName
```

Konten

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-222	Brugg	700
A-201	Aarau	900
A-217	Aarau	750

Zusammenfassung: Grundstruktur von SQL

- Anfrageausdruck verbindet Anfragespezifikationen mit **union**,**except**,**intersect**
- Konzeptionelle Auswertung von Anfragespezifikation muss verstanden werden:



Nullwerte/1

- Es ist möglich, dass Attribute einen Nullwert *null* haben.
- *null* steht für einen unbekannten Wert oder einen Wert der nicht existiert oder einen Wert der zurückgehalten wird oder ...
- Das Prädikat **is null** muss verwendet werden um auf Nullwerte zu prüfen.
  - Beispiel: Alle Kredite, für die der Betrag ein Nullwert ist.

```
select KrNr
from Kredite
where Betrag is null
```

- Arithmetische Ausdrücke ergeben immer *null*, falls ein Teil *null* ist.
  - $5 + null$ ergibt *null*
  - $0 * null$ ergibt *null*

Nullwerte/2

- Intuition: Nullwerte sind Platzhalter für unterschiedliche Werte.

Konten (ohne Nullwerte)

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	Brugg	700
A-102	Brugg	400
A-305	Brugg	350
A-201	Aarau	900
A-222	Brugg	700
A-217	Aarau	750

Konten (mit Nullwerten)

KoNr	*FiName*	*Guthaben*
A-101	Chur	500
A-215	null	700
A-102	null	null
A-305	Brugg	350
A-201	null	900
A-222	Brugg	700
A-217	Aarau	750

- Nullwerte sind also nicht als Variable mit Name *null* zu verstehen. Insbesondere ist $(null = null)$ nicht wahr.

Nullwerte/3

- SQL verwendet dreiwertige Logik mit dem zusätzlichen Wahrheitswerte *unknown*.
- Jeder Vergleich mit *null* ergibt (den dritten logischen Wert) *unknown*
  Beispiele: $5 < null$ oder $null <> null$ oder $null = null$
- Wahrheitswerte logischer Verknüpfungen sind wie erwartet:
  - **OR** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - **AND** (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*
    (*unknown* **and** *unknown*) = *unknown*
  - **NOT** (**not** *unknown*) = *unknown*
- *unknown* als Ergebnis des Prädikates im **where** bzw. **having** Teil wird gleich behandelt wie *false* (d.h., Tupel bzw. Gruppe wird nicht zurückgegeben).

Nullwerte/4

Aggregatfunktionen:

- Ignorieren Nullwerte in den aggregierten Attributen.
- Ausnahme: **count(*)** zählt die Anzahl der Zeilen in einer Tabelle.
- Beispiel: Die Anzahl vergebener Kredite?

  select count(*Betrag*)
  from *Kredite*
  - Die SQL Anfrage zählt keine Kredite mit einem Nullwert als Betrag.
  - Das Resultat ist 0 falls alle Kreditbeträge null sind.

Gruppierung:

- **group** betrachtet alle Nullwerte als wären sie identisch.
- Nullwerte in aggregierten Attributen werden als Gruppe zusammengefasst.
- Beispiel: $R[A, B, C] = \{[1, null, 100], [1, null, 200], [null, null, 100]\}$ gruppiert nach den Attributen A und B ergibt die Gruppen
  - $\{[1, null, 100], [1, null, 200]\}$
  - $\{[null, null, 100]\}$

Duplikate/1

- Für Tabellen mit Duplikaten muss definiert werden, wie oft ein Tuple im Resultat einer Anfrage vorkommt (d.h. die reine Mengenlehre ist nicht mehr ausreichend).
- Beispiel:

R

A	B
1	4
1	2
1	3
1	3

S

X
1
1

select A **from** R

A
1
1
1
1

select * **from** R, S

A	B	X
1	4	1
1	2	1
1	3	1
1	3	1
1	4	1
1	2	1
1	3	1
1	3	1

select A **from** R **except all select** X **from** S

A
1
1

Duplikate/2

- Um SQL abbilden zu können, wird die relationale Algebra auf Multimengen (Mengen mit Duplikaten) erweitert.
- Beispiele:

  $\sigma_p(R)$ Für ein Tupel t das c mal in R vorkommt gilt: Falls t das Selektionsprädikat p erfüllt, dann sind c Kopien von t in $\sigma_p(R)$, andernfalls keine.

  $\pi_A(R)$ Für jede Kopie eines Tupels t von R gibt es ein eine Kopie des Tupels $t.[A]$ in $\pi_A(R)$.

  $R_1 \times R_2$ Wenn es c_1 Kopien von t_1 in R_1 gibt und c_2 Kopien von t_2 in R_2, dann gibt es $c_1 * c_2$ Kopien des Tupels $t_1 \circ t_2$ in $R_1 \times R_2$.

Duplikate/3

- SQL-Anfrage

  select $A_1, A_2, \ldots, A_n$
  from $R_1, R_2, \ldots, R_m$
  where p

  ist äquivalent zu Ausdruck der Relationalen Algebra mit Multimengen:

$$\pi_{A_1,A_2,\ldots,A_n}(\sigma_p(R_1 \times R_2 \times \ldots \times R_m))$$

Ordnung der Tupel

- Die Zeilen einer Tabelle sind nicht geordnet.
- **order by** Teil: Das Ergebnis einer Anfrage lässt sich mit **order by** ordnen.
- Beispiel: Alphabetisch geordnete Liste aller Namen von Kunden die einen Kredit von der Brugg Filiale haben.

  select distinct *KuName*
  from *Kreditnehmer*, *Kredite*
  where *KrNo* = *KrNr* **and** *FiName* = 'Brugg'
  order by *KuName*

- Sortierung: Es ist möglich zwischen **desc** (absteigende Sortierung) oder **asc** (aufsteigende Sortierung, Default) auszuwählen.
  - Beispiel: **order by** *KuName* **desc**

Integrierte Übung 5.7

- Erklären Sie das Resultat des folgenden SQL Befehls

  select count(*) **as** Cnt1,
  count(Umsatz) **as** Cnt2
  from Filiale

Cnt1	Cnt2
123	87

Integrierte Übung 5.8

1. Was macht folgende Anfrage?
   select * **from** *PC* **where** *SpeedGHz* > 1 **or** *SpeedGHz* < 4
   Wie könnte eine äquivalente, bessere Anfrage lauten.

2. Was ergibt folgende Anfrage?
   select * **from** *R* **where** *X* <> *null*

3. Was ergibt folgende Anfrage für eine Tabelle $R[X]$?
   select * **from** *R* **group by** *X*

Zusammenfassung: Nullwerte, Duplikate, Ordnung

- Nullwerte: Wert nicht vorhanden.
  - Platzhalter für unterschiedliche Werte
  - dreiwertige Logik mit *unknown*
  - Aggregatfunktionen ignorieren Nullwerte (außer **count**(∗))
- Duplikate:
  - SQL erlaubt Duplikate
  - relationale Algebra für Multimengen erforderlich
- Ordnung:
  - Tupel in Tabelle sind nicht sortiert
  - Ergebnis einer Anfrage kann mit **order by** sortiert werden

Inhalt

Geschachtelte Anfragen

- In SQL können **select**-Anweisungen geschachtelt werden.
- Eine Unteranfrage ist ein Anfrageausdruck der innerhalb einer anderen Anfrage geschachtelt ist:
  - im **from**-Teil: sogenannte "abgeleitete Tabelle"
  - im **where**-Teil: typischerweise Mengenvergleiche, Tests auf Mengenzugehörigkeit und Kardinalitäten von Mengen

Abgeleitete Tabellen

- SQL erlaubt eine Unteranfrage im **from** Teil (anstelle eines Tabellennamens kann eine SQL Anfrage verwendet werden).
- Das ist wichtig für die Geschlossenheit einer Sprache.
- Eine abgeleitete Tabelle wird durch einen Anfrageausdruck definiert.
- Den durchschnittlichen Kontostand von Filialen die einen durchschnittlichen Kontostand von mehr als $1200 haben.

```
select FiName, AvgGuthaben
from FilialeAvg
where AvgGuthaben > 1200

select FiName, AvgGuthaben
from (select FiName, avg(Guthaben) as AvgGuthaben
      from Konten
      group by FiName) as FilialeAvg
where AvgGuthaben > 1200
```

Geschachtelte Anfragen im WHERE-Teil

1. Unteranfragen im **where** Teil können folgende Konstrukte verwenden:
   - **exists**, **not exists**
   - **in**, **not in**
   - $=$ **some**, $<$ **some**, $<>$ **some** usw.
     any ist ein Synonym für **some**
   - $=$ **all**, $<$ **all**, $<>$ **all** usw.

   Beispiele:
   - **select** * **from** Kr **where** KrNr **in** (**select** KrNo **from** KrNe)
   - **select** * **from** Kr **where** KrNr $=$ **some** (**select** KrNo **from** KrNe)
   - **select** * **from** Kr **where** KrNr $<>$ **all** (**select** KrNo **from** KrNe)
2. Weiters kann die Unteranfrage über einen Operator verknüpft sein.
   - Die Unteranfrage darf nur eine einzige Zeile zurückliefern.
   - Typischerweise berechnet die Unteranfrage eine Aggregationsfunktion.

   Beispiel:
   - **select** * **from** Kr **where** Betrag $=$ (**select avg**(Betrag) **from** Kr)

Anfragen mit EXISTS

- Die **exists** (und **not exists**) Unteranfragen werden oft verwendet. **exists** ist erfüllt falls die Unteranfrage nicht leer ist.
  - **exists** $(q) \Leftrightarrow q \neq \emptyset$
  - **not exists** $(q) \Leftrightarrow q = \emptyset$
- Bsp: Kontoinhaber die auch Kreditnehmer sind?
  - **select** KuName
    from Kontoinhaber **as** KI
    where exists (**select** *
    from Kreditnehmer **as** KN
    where KI.KuName $=$ KN.KuName)
- Bsp: Kontoinhaber die nicht Kreditnehmer sind?
  - **select** KuName
    from Kontoinhaber **as** KI
    where not exists (**select** *
    from Kreditnehmer **as** KN
    where KI.KuName $=$ KN.KuName)

Integrierte Übung 5.9

- Gegeben ist Tabelle R wie folgt:

R
A
1
2
3

Geben Sie einen SQL Befehl, der den grössten Wert in R mithilfe einer geschachtelten Anfrage bestimmt. Der SQL Befehl soll ohne Aggregationsfunktion en auskommen.

Anfragen mit IN

- a **in** (R)
  - a ist ein Ausdruck, z.B. ein Attributname oder eine Konstante
  - R ist eine Anfrage und liefert gleich viele Spalten zurück wie der Ausdruck a (eine Spalte, falls a ein Attributname)
  - ist wahr, falls mindestens ein Ergebnistupel von R gleich a ist
- a **not in** (R)
  - ist wahr, falls kein Ergebnistupel von R gleich mit a ist

Beispiele: Anfragen mit IN

- Alle Kunden die sowohl ein Konto als auch einen Kredit haben.

```
select KuName
from Kreditnehmer
where KuName in (select KuName
                 from Kontoinhaber)
```

Bestimmt alle Zeilen in der Tabelle *Kreditnehmer* deren Kundennanme auch in der Tabelle *Kontoinhaber* vorkommt

- Alle Kunden die einen Kredit aber kein Konto haben.

```
select KuName
from Kreditnehmer
where KuName not in (select KuName
                     from Kontoinhaber)
```


Integrierte Übung 5.10

- Sind die folgenden SQL Befehle äquivalent?
  - **select** A **from** R, S **where** R.A = S.X
  - **select** A **from** R **where** A **in** (**select** X **from** S)

Anfragen mit SOME

- a <comp> **some** $(R) \Leftrightarrow \exists t \in R(a$ <comp> $t)$
  wobei <comp> eines der folgenden Prädikate sein kann:
  $<, \leq, \geq, >, =, \neq$
- Beispiele:

$(5 < \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array})$ = true (Bedeutung: 5 < ein Tupel in der Tabelle)

$(5 < \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array})$ = false

$(5 = \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array})$ = true

$(5 \neq \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array})$ = true (weil $0 \neq 5$)

Beispiel: Anfragen mit SOME

- Alle Konten, deren Betrag höher ist als der Umsatz einer Filiale in Salzburg.

```
select distinct KoNr
from Konten, Filiale
where Konten.Betrag > Filiale.Umsatz and
      Filiale.Stadt = 'Salzburg'
```

- Die gleiche Anfrage wie oben aber mit > **some** Konstrukt

```
select KoNr
from Konten
where Betrag > some (select Umsatz
                     from Filiale
                     where Stadt = 'Salzburg')
```

SOME vs. IN/1

- $=$ **some** und **in** sind äquivalent.
- Beispiel: Kontoinhaber die auch Kreditnehmer sind?

```
select KuName
from Kontoinhaber as KI
where KI.KuName in (select KN.KuName
                    from Kreditnehmer KN)

select KuName
from Kontoinhaber as KI
where KI.KuName = some (select KN.KuName
                        from Kreditnehmer KN)
```


SOME vs. IN/2

- $\neq$ **some** und **not in** sind nicht äquivalent.
- Beispiel: Kontoinhaber die nicht Kreditnehmer sind?

Richtig:

```
select KuName
from Kontoinhaber as KI
where KI.KuName not in (select KN.KuName
                        from Kreditnehmer KN)
```

Falsch:

```
select KuName
from Kontoinhaber as KI
where KI.KuName <> some (select KN.KuName
                         from Kreditnehmer KN)
```


Anfragen mit ALL

- a <comp> **all** $(R) \Leftrightarrow \forall t \in R\ (a$ <comp> $t)$

$(5 < \textbf{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$

$(5 < \textbf{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$

$(5 = \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$

$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$

$(\neq \textbf{all}) \equiv$ **not in**
Aber: $(= \textbf{all}) \not\equiv$ **in**

Beispiel: Anfragen mit ALL

- Die Namen aller Filialen die ein grösseres Guthaben als alle Banken in Aarau haben.

```
select FiName
from Filiale
where Umsatz > all
                (select Umsatz
                 from Filiale
                 where FiOrt = 'Aarau')
```

EXISTS statt SOME/ANY, IN, ALL

- Die Konstrukte **in**, **all**, **any**, **some** können unübersichtlich und schwer zu interpretieren werden.
- Beispiel: Nullwerte und leere Mengen müssen berücksichtigt werden.

  $(5 < \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline \text{null} \\ \hline \end{array}) = \text{false}$

  $(5 > \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline \text{null} \\ \hline \end{array}) = \text{unknown}$

  $(5 <> \textbf{all}\ \emptyset) = \text{true}$
- Mithilfe von **exists** können alle Anfragen ausgedrückt werden, die **in**, **all**, **any**, **some** verwenden.
- Manche Implementierungen schreiben alle geschachtelten Anfragen im **where** Teil in **exists**-Anfragen um.

Zusammenfassung

- Geschachtelte Anfragen sind Anfragen mit Unteranfragen.
- Unteranfragen im **where** Teil können folgende Konstrukte verwenden:
  - (**not**) **exists**
  - (**not**) **in**
  - **some**/**any**
  - **all**
- Alle Unteranfragen können mit (**not**) **exists** ausgedrückt werden (empfohlen).
- Eine abgeleitete Tabellen ist eine Unteranfragen im **from** Teil.

Inhalt

Löschen von Tupeln

- Löschen aller Konten der Brugg Filiale.

```
delete from Konten
where FName = 'Brugg'
```

- Löschen aller Kredite zu denen kein Kreditnehmer erfasst ist.

```
delete from Kredite
where KrNr not in ( select KrNo
                    from Kreditnehmer )
```

- **delete** vs. **drop**:
  - "**delete from** Konten" löscht alle Zeilen der Tabelle *Konten*, das Schema bleibt jedoch erhalten
  - "**drop table** Konten" löscht alle Zeilen, Schemadefinition, Indexes usw. der Tabelle *Konten*

Einfügen von Tupeln/1

- Neues Tupel zur Tabelle *Konten[KoNr, FiName, Guthaben]* hinzufügen:

```
insert into Konten
    values ('A-9732', 'Brugg', 1200)
```

- Ein neues Tupel zur Tabelle Konten hinzufügen. Das Guthaben soll **null** sein.

```
insert into Konten
    values ('A-9732', 'Brugg', null)
```


Einfügen von Tupeln/2

- Nur die ersten zwei Werte werden gesetzt:

```
insert into Konten
    values ('A-9732', 'Brugg')
```

- Nicht angegebene Werte sind *null* bzw. erhalten den Wert, der mit **default** festgelegt wurde.
- Ein Wert in der Mitte wird ausgelassen:

```
insert into Konten(KoNr, Guthaben)
    values ('A-9732', 1200)
```

Äquivalente Anfrage (falls *FiName* keinen Default-Wert hat):

```
insert into Konten
    values ('A-9732', null, 1200)
```


Einfügen von Tupeln/3

- Außer konstanten Tupeln kann auch das Ergebnis einer Anfrage eingefügt werden.
- Beispiel: Füge *Kredite* als Konten mit negativem Guthaben in die Tabelle *Konten* ein:

```
insert into Konten
    select KrNr, FiName, −Betrag from Kredite
```


Ändern von Tupeln

- Die Guthaben aller Konten mit Guthaben über $10,000 um 6% erhöhen. Die Guthaben aller anderen Konten um 5% erhöhen.
  - Kann mit Hilfe zweier **update** Befehle erreicht werden:

```
update Konten
set Guthaben = Guthaben ∗ 1.06
where Guthaben > 10000

update Konten
set Guthaben = Guthaben ∗ 1.05
where Guthaben ≤ 10000
```

- Die Ordnung dieser Befehle ist wichtig. Die umgekehrte Reihenfolge der SQL Befehle ist falsch.

Zusammenfassung

- Die Daten einer Tabelle können durch folgende Befehle geändert werden:
  - **delete**: Zeilen löschen
  - **insert**: neue Zeilen einfügen
  - **update**: Werte in einer oder mehrerer Zeilen ändern

Inhalt

Sichten (Views)

- Nutzen von Sichten
- Erstellung und Verwendung von Sichten
- Behandlung von Sichten durch das DBMS
- Temporäre Sichten

Nutzen von Sichten

- In manchen Fällen ist es wünschenswert, dass
  - nicht alle Benutzer das gesamte logische Modell sehen (d.h. alle Tabellen der Datenbank)
  - Benutzer auf berechnete Tabellen zugreifen können (statt auf die tatsächlich gespeicherten Basistabellen)
- Beispiel: Ein Benutzer braucht Zugang zu Kundenname, Kreditnummer und Name der Filiale, soll aber den Kreditbetrag nicht sehen. Dieser Benutzer sollte eine Relation sehen, die in SQL so ausgedrückt wird:

  select *KuName*, *Kredite.KrNr*, *FiName*
  from *Kredite*, *Kreditnehmer*
  where *Kreditnehmer.KrNo* = *Kredite.KrNr*

- Eine Sicht (**view**) stellt einen Mechanismus zur Verfügung um:
  - Daten vor bestimmte Benutzergruppen zu verstecken
  - Benutzern Zugang zu Ergebnissen (komplexer) Anfragen zu geben

Erstellen von Sichten

- Eine Sicht wird durch den Befehl **create view** erstellt:

  create view $v(A_1, A_2, \ldots, A_n)$ **as** <Anfrageausdruck>

  - wobei v der Name der Sicht ist
  - <Anfrageausdruck> ein gültiger SQL Ausdruck, der n Spalten liefert
  - A_i den Namen der i-ten Spalte festlegt
- Spaltennamen optional: Falls die Spaltennamen im Anfrageausdruck eindeutig sind und keine Funktionen enthalten, müssen keine neuen Namen angegeben werden.
- Eine Sicht ist eine virtuelle Tabelle; der Name v der Sicht kann in Anfragen wie eine Tabelle verwendet werden.
- Die Sichtdefinition (Name und Anfrageausdruck) wird als Metadaten in der Datenbank gespeichert.

Beispiel: Erstellen von Sichten

- Eine Sicht auf Filialen und deren Kunden:

```
create view Alle_Kunden as
    (select FiName, KuName
    from Kontoinhaber, Konten
    where Kontoinhaber.KoNr = Konten.KoNr)
    union
    (select FiName, KuName
    from Kreditnehmer, Kredite
    where Kreditnehmer.KrNo = Kredite.KrNr)
```

- Finde alle Kunden der Filiale 'Brugg':

```
select KuName
from Alle_Kunden
where FiName = 'Brugg'
```


Auswertung von Sichten (View Expansion)

- Die Definition einer Sicht ist in den Metadaten definiert.
- View Expansion: Bei der Auswertung einer Anfrage, wird der Name der Sicht durch den entsprechenden Anfrageausdruck ersetzt.
- View Expansion wird durch folgenden Algorithmus beschrieben:

  repeat
  finde alle Sichten v_i im Anfrageausdruck e_1
  ersetzte v_i in e_1 durch den Anfrageausdruck von v_i
  until e_1 enthält keine Sichten mehr
- Für Sichten die nicht rekursiv sind terminiert dieser Algorithmus.

Änderbarkeit von Sichten

- Eine Sicht ist änderbar (update-fähig), wenn das Datenbanksystem die Umkehrabbildung von der Sicht zu den Basistabellen herstellen kann.
- In SQL-92 sind Sichten not updatable (nicht änderbar), wenn die Anfrage in der Sichtdefinition eine der folgenden Bedingungen erfüllt:
  1. das Schlüsselwort **distinct** wird benutzt
  2. ein **group by** Teil wird benutzt
  3. ein **having** Teil wird benutzt
  4. die **select** Liste enthält Ausdrücke, die verschieden von Spaltennamen sind, oder Spaltennamen kommen mehrfach vor
  5. der **from** Teil enthält mehr als eine Sicht/Tabelle oder eine nicht änderbare Sicht
- Theoretisch könnte die Umkehrabbildung auch für Sichten erstellt werden, die laut SQL nicht änderbar sind:

  SQL änderbare Sichten $\subset$ theoretisch änderbare Sichten $\subset$ alle Sichten

Beispiel

Korrekte Anfrage für *Integrierte Übung 5.6.*

Bestimmen Sie für jede Filiale die Konten mit dem größten Guthaben.

```
with
Max_Guthaben_Filiale (FiName, MaxG) as (
      select FiName, max(Guthaben)
      from Konten
      group by FiName
)
select K.KoNr, K.FiName, M.MaxG
from Konten K, Max_Guthaben_Filiale M
where M.MaxG = K.Guthaben and
      M.FiName = K.FiName;
```


Integrierte Übung 5.11

Betrachten Sie folgenden DDL Befehl:

```
create view v as
  select FiName, KuName
  from Konten ko, Kontoinhaber ki
  where ko.KoNr = ki.KoNr
```

Warum wird folgender DML Befehl abgewiesen?

```
update v
set FiName = 'Brugg'
where KuName = 'Tschurtschenthaler';
```


Temporäre Sichten mit **with**

- Der **with** Teil ermöglicht die Definition von temporären Sichten, welche nur innerhalb desselben Anfrageausdrucks gültig sind.
- Beispiel: Finde alle Konten mit dem maximalen Kontostand:

```
with
Max_Kontostand (Wert) as (
      select max(Guthaben)
      from Konten
)
select KoNr
from Konten, Max_Kontostand
where Konten.Guthaben = Max_Kontostand.Wert
```


Beispiel: Temporäre Sichten mit **with**

- Finde alle Filialen, in denen das Gesamtguthaben der Konten über dem durchschnittlichen Gesamtguthaben aller Filialen liegt.

```
with
Filiale_Guthaben (FiName, Wert) as (
      select FiName, sum(Guthaben)
      from Konten
      group by FiName
),
Filiale_Guthaben_Avg (Wert) as (
      select avg(Wert)
      from Filiale_Guthaben
)
select FiName
from Filiale_Guthaben, Filiale_Guthaben_Avg
where Filiale_Guthaben.Wert > Filiale_Guthaben_Avg.Wert
```

Inhalt

DCL: Data Control Language

- Autorisierung und Zugriffsrechte
- Befehl **grant**
- Befehl **revoke**

Literatur:
Kemper&Eikler. Datenbanksysteme – Eine Einführung. Kapitel 12.2.

Authorisierung und Zugriffrechte

- Autorisierung schränkt den Zugriff und die Änderung von Daten durch Benutzer ein.
- Beschränkungen können sich beziehen auf:
  - Objekte wie z.B. Schemas, Tabellen, Spalten, Zeilen
  - Ressourcen wie z.B. Zeit (CPU, Verbindung, oder Wartezeiten).
- Es gibt Zugriffsrechte auf verschiedenen Ebenen:
  - System: tablespace, ...
  - Schema: Cluster, Index, Trigger, Datenbank, ...
  - Tabellen: create, alter, index, references, drop, select, delete, insert, update, ...
  - View: create, select, delete, insert, update
  - Prozeduren: create, alter, drop
  - Typen: create, alter, drop, execute
- Zugriffsrechte können an Benuzter oder Rollen (Role Based Access Control) vergeben werden

Der GRANT Befehl

- Der **grant** Befehl überträgt Zugriffsrechte:
  grant <Liste von Zugriffsrechte>
  on <Tabelle oder View> **to** <Liste von Benutzern>
- <Liste von Benutzer> kann sein:
  - ein Benutzername
  - eine Rolle
  - **public**: alle gültigen Benuzter
- Vergeber eines Zugriffsrechtes müssen dieses selber besitzen (oder Administrator sein).

Einige Zugriffsrechte auf Tabellen

select: Direktes Leserecht über select-Anfragen oder indirektes Leserecht über Views.

- Beispiel: **select** Zugriffsrecht für Benuzter U_1, U_2 und $U3$ auf Relation *Filialen* vergeben:

  grant select on *Filialen* **to** U_1, U_2, U_3

insert: erlaubt Einfügen von Zeilen mit dem **insert** Befehl

update: erlaubt Ändern von Werten mit dem **update** Befehl

delete: erlaubt Löschen von Zeilen mit dem **delete** Befehl (**drop table** ist jedoch *nicht* erlaubt!)

Der REVOKE Befehl

- Der **revoke** Befehl nimmt Zugriffsrechte zurück.

  revoke <Liste von Zugriffsrechte>
  on <Tabelle oder View> **from** <Liste von Benutzern>

  - Beispiel:

    revoke select on *Filialen* **from** U_1, U_2, U_3

- <Liste von Zugriffsrechte> kann **all** sein, um alle Zugriffsrechte zurückzunehmen
- Falls <Liste von Benutzern> **public** enthält, verlieren alle Benutzer die angegebenen Rechte, außer die Rechte wurden explizit gewährt.
- Falls dasselbe Zugriffsrecht von zwei verschiedenen Benutzern gewährt wurde (also doppelt), kann es auch nach dem **revoke** Befehl erhalten bleiben.

Inhalt

Zugriff auf die Datenbank

Zugriff auf die Datenbank über Programmiersprachen:

- Embedded SQL
- Dynamic SQL
- ODBC
- JDBC

Datenbankzugriff

- API (application program interface) für die Interaktion mit einem Datenbankserver.
- API übernimmt:
  - Verbindung zu Datenbankserver herstellen (connection)
  - SQL Befehle an den Datenbankserver schicken
  - Ergebnistupel abrufen und in Programmvariablen speichern
- **Embedded SQL**: viele Sprachen erlauben die Einbettung von SQL in den Programm Code. Embedded SQL kann sein:
  - statisch (d.h. bekannt zum Zeitpunkt der Compilierung)
  - dynamisch (d.h. Code ist zum Zeitpunkt der Compilierung nicht bekannt und wird erst zur Laufzeit erzeugt)
- **ODBC** (Open Database Connectivity) ist ein Microsoft Standard und funktioniert mit C, C++, C#, und Visual Basic
- **JDBC** (Java Database Connectivity) ist von Sun Microsystems und funktioniert mit Java

JDBC

- JDBC ist ein Java API zur Kommunikation mit SQL Datenbanken
- JDBC unterstützt eine Vielzahl von Funktionen um Daten anzufragen, zu ändern und die Ergebnistupel einzulesen.
- JDBC unterstützt auch Anfragen auf die Metadaten, z.B. Namen und Typen von Spalten.
- Ablauf der Kommunikation mit der Datenbank:
  - Netzwerkverbindung herstellen (*Connection* Objekt)
  - *Statement* Objekt erzeugen (ist einer Connection zugeordnet)
  - das *Statement* Objekt wird benutzt, um Anfragen auszuführen und Ergebnisse auszulesen
  - Exceptions werden zur Fehlerbehandlung verwendet

Beispiel: JDBC/1

- Wir schreiben ein Java Programm, das sich über JDBC mit PostgreSQL Datenbank verbindet.
- Zugangsdaten:
  - Hostname: `dumbo.cosy.sbg.ac.at`
  - Port: 5432
  - Datenbankname: `ss2013`
  - Benutzername: `augsten`
  - Passwort: `xxx`
- Aufruf des Programmes

  `java -cp .:postgresql_jdbc.jar PostgresqlJDBC`

  wobei folgende Dateien im aktuellen Pfad zu finden sein müssen:
  - `PostgresqlJDBC.class`
  - `postgresql_jdbc.jar`: ein JDBC Driver für Postgresql
- Das Programm gibt die Namen aller Tabellen zurück, deren Besitzter `augsten` ist.

Beispiel: JDBC/2

```
import java.sql.*;

public class PostgresqlJDBC  {

    public static void main(String[] args) throws  Exception {

        Class.forName("org.postgresql.Driver");
        Connection conn =
            DriverManager.getConnection(
                "jdbc:postgresql://dumbo.cosy.sbg.ac.at:5432/ss2013",
                "augsten", "xxx");

        Statement stmt = conn.createStatement();

        ResultSet rset = stmt.executeQuery(
            "select tablename from pg_tables where tableowner='augsten'");

        while (rset.next())
            System.out.println(rset.getString(1));
    }

}
```

Datenbanken 1

Relationale Entwurfstheorie

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
FB Computerwissenschaften
Universität Salzburg

http://dbresearch.uni-salzburg.at

Sommersemester 2018

Version 12. Juni 2018

Inhalt

Inhalt

Ziele des Relationalen Datenbankentwurfs

- Ziel des relationalen Entwurfs sind gute Schemas in der Datenbank.
- Die Hauptschwierigkeit ist es, eine gute Gruppierung der Attribute in relationale Schemas zu finden.

Anomalien bei Datenänderung/1

- Beispiel Schema / Instanz:

  AngProj(SVN, PNum, Stunden, AName, PName, POrt)

AngProj

SVN	PNum	Stunden	AName	PName	POrt
1234	1	32.5	Schmidt	ProjektX	Salzburg
1234	2	7.5	Schmidt	ProjektY	Wien
6688	3	40.5	Mair	ProjektZ	Linz
4567	1	20.0	Huber	ProjektX	Salzburg
4567	2	20.0	Huber	ProjektY	Wien
3334	2	10.0	Wong	ProjektY	Wien
3334	3	10.0	Wong	ProjektZ	Linz
3334	10	10.0	Wong	Computerization	Innsbruck
3334	20	10.0	Wong	Reorganization	Linz

- AngProj ist kein gutes Schema, da es unter Anomalien leidet.

Anomalien bei Datenänderung/2

- Beispiel Schema:
  - AngProj(SVN, PNum, Stunden, AName, PName, POrt)
- Updateanomalie
  - Wenn der Ort eines Projektes geändert wird, muss er für alle Angestellten im Projekt geändert werden.
- Einfügeanomalie
  - Es kann kein Projekt ohne Angestellte eingefügt werden (SVN darf nicht *null* sein, da Teil des Schlüssels).
- Löschanomalie
  - Wenn ein Projekt gelöscht wird, werden als Nebeneffekt auch alle Angestellten gelöscht, die auf diesem Projekt arbeiten.

Richtlinien für den relationalen Entwurf

- **Richtlinie 1**: Jedes Tupel einer Relation sollte nur die Instanz *einer* Entität oder Beziehung darstellen.
- **Richtlinie 2**: Update-, Einfüge- und Löschanomalien sollen vermieden werden.
- **Richtlinie 3**: Die Relationen sollen möglichst wenige *null* Werte enthalten; Attribute, die *null* Werte enthalten, kommen in eine eigene Relation (zusammen mit dem Primärschlüssel).
- **Richtlinie 4**: Durch einen natürlichen Join von Relationen sollen keine zusätzlichen (d.h. falschen) Tupel erzeugt werden.

Inhalt

Übersicht

- Was sind funktionale Abhängigkeiten?
- Armstrong-Axiome
- Richtigkeit und Vollständigkeit
- Hülle und kanonische Überdeckung

Funktionale Abhängigkeiten/1

- Funktionale Abhängigkeiten (FDs – functional dependencies) werden zwischen Attributmengen $X \subseteq sch(R)$ und $Y \subseteq sch(R)$ einer Relation R definiert.
- Definition: Y ist von X funktional abhängig genau dann, wenn der Wert von X einen eindeutigen Wert von Y in R vorgibt:

$$X \to Y \Leftrightarrow \forall t_1, t_2 \in R : t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

- $X \to Y$ bedeutet, dass Y von X funktional abhängt, bzw., dass die Attribute X die Attribute Y funktional bestimmen.
- FDs definieren eine Einschränkung auf das Schema, d.h., auf alle möglichen relationalen Instanzen von R.
- Definition: Eine Menge Y ist trivial funktional abhängig von X genau dann wenn $Y \subseteq X$.

Funktionale Abhängigkeiten/2

- Wozu FDs?
  - Funktionale Abhängigkeiten werden als formales Maß für die Qualität eines relationalen Entwurfs verwendet.
  - Funktionale Abhängigkeiten und Schlüssel werden verwendet, um Normalformen für Relationen zu definieren.
- Woher kommen FDs?
  FDs ergeben sich aus der zugrundeliegenden Anwendung und werden abgeleitet von
  - der Bedeutung der Attribute,
  - der Beziehung der Attribute untereinander.
- Beispiele Funktionaler Abhängigkeiten:
  - Sozialversicherungsnummer bestimmt Angestelltenname:
    - $\{SVN\} \to \{AName\}$
  - Projektnummer bestimmt Projektname und Projektort:
    - $\{PNum\} \to \{PName, POrt\}$
  - Angestellten SVN und Projektnummer bestimmten die Anzahl der Wochenstunden, die der Angestellte auf dem Projekt arbeitet:
    - $\{SVN, PNum\} \to \{Stunden\}$

Funktionale Abhängigkeiten/3

- Schema vs. Instanz:
  - FDs sind auf dem Schema definiert und müssen für alle Instanzen gelten
  - manche FDs können aufgrund einer gegebenen Instanz ausgeschlossen werden (weil diese die FD verletzen würde)
- Notation:
  - Statt $\{A, B\}$ schreiben wir AB (oder A, B), z.B. $AB \to BCD$ statt $\{A, B\} \to \{B, C, D\}$.
  - Für eine Menge von Attributen X (z.B., $X = \{A, B, C\}$) und ein einzelnes Attribut A schreiben wir $X - A$ statt $X - \{A\}$.

Integrierte Übung 6.1

Betrachten Sie die abgebildete Instanz der Relation $R[A, B, C]$. Welche der folgenden Aussagen sind korrekt?

R

A	B	C
1	1	3
2	1	1
3	2	2
4	1	1

a. $B \rightarrow C$ gilt für die Relation R.

b. $C \rightarrow B$ gilt für die Relation R.

c. $BC \rightarrow A$ gilt in der abgebildeten Instanz von R.

d. A ist der Primärschlüssel von R.

Schlüssel (Auffrischung)

- Superschlüssel: Teilmenge der Attribute einer Relation, welche in jeder gültigen Ausprägung eindeutige Werte annimmt.
- Ein Kandidatschlüssel K ist ein Superschlüssel für den gilt, dass durch die Entfernung eines beliebigen Attributes von K die Superschlüssel-Eigenschaft von K verloren geht.
- Eine *beliebiger* Kandidatenschlüssel wird als Primärschlüssel ausgewählt.
- *Notation:* Die Attribute des Primärschlüssels werden unterstrichen: AngProj(SVN, PNum, Stunden, AName, PName, POrt)

FDs und Schlüssel

- $K \subseteq sch(R)$ ist genau dann ein Superschlüssel von R, wenn

$$K \rightarrow sch(R),$$

  d.h. K bestimmt *alle* Attribute in R.
- $K \subseteq sch(R)$ ist genau dann ein Kandidatenschlüssel von R, wenn folgendes gilt:
  1. K ist ein Superschlüssel von R, d.h.

$$K \rightarrow sch(R)$$

  2. K kann nicht mehr verkleinert werden, ohne die Superschlüssel-Eigenschaft zu verlieren, d.h.

$$\forall A \in K : (K - A) \not\rightarrow sch(R)$$

Armstrong-Axiome/1

- Für eine bestimmte Menge F von FDs können zusätzliche FDs hergeleitet werden, die immer gelten, wenn die FDs in F gelten.
- Armstrong-Axiome[1] (Inferenzregeln):
  - Reflexivität: $Y \subseteq X \models X \rightarrow Y$
  - Verstärkung: $X \rightarrow Y \models XZ \rightarrow YZ$
  - Transitivität: $X \rightarrow Y, Y \rightarrow Z \models X \rightarrow Z$
- Notation:
  - $A \models B$ heißt: von A kann B hergeleitet werden
  - XZ steht für $X \cup Z$
- Die Armstrong-Axiome sind korrekt und vollständig:
  - Diese Regeln sind gültig (korrekt) und alle anderen gültigen Regeln können von diesen Regeln abgeleitet werden (vollständig).

[1] William W. Armstrong: Dependency Structures of Data Base Relationships, pages 580-583. IFIP Congress, 1974.

Beispiel: Armstrong-Axiome

Zeige oder widerlege folgende Herleitungen:

$$X \to Y, X \to W, WY \to Z \models \mathbf{X \to Z}$$

Lösung:

Verstärkung: $X \to Y \models XX \to XY = X \to XY$
$X \to W \models XY \to WY$

Transitivität: $X \to XY, XY \to WY \models X \to WY$
$X \to WY, WY \to Z \models \mathbf{X \to Z}$

Integrierte Übung 6.2

Zeige oder widerlege folgende Herleitungen:

1. $X \to Y, Z \subseteq Y \models X \to Z$

2. $XY \to Z, Y \to W \models XW \to Z$

Armstrong-Axiome/2

- Folgende zusätzliche Inferenzregeln werden oft verwendet:
  - Dekompositionsregel: $X \to YZ \models X \to Y, X \to Z$
  - Vereinigungsregel: $X \to Y, X \to Z \models X \to YZ$
  - Pseudotransitivitätsregeln: $X \to Y, WY \to Z \models WX \to Z$

- Diese zusätzlichen Inferenzregeln (und alle anderen möglichen Inferenzregeln) lassen sich aufgrund der Vollständigkeit der Armstrong-Axiome aus diesen ableiten.

Hülle/1

- Die Hülle F^+ (closure) der Menge F von FDs ist die Menge aller FDs die von F hergeleitet werden können.
- Die Hülle $\mathcal{H}(F, X)$ einer Menge von Attributen X bezüglich F ist die Menge aller Attribute, welche von X funktional abhängig sind.
- F^+ und $\mathcal{H}(F, X)$ können durch wiederholte Anwendung der Armstrong-Axiome berechnet werden.

Hülle/2

- Die Attribut-Hülle $\mathcal{H}(F, X)$ der Attributmenge X bezüglich F kann auch durch folgenden Algorithmus berechnet werden.

  $\mathcal{H}\ (F, X)$
  $Erg := X$
  while (Änderungen an Erg) **do**
  foreach FD $A \to B$ **in** F **do**
  if $A \subseteq Erg$ **then** $Erg := Erg \cup B$
  return Erg
- Input: Menge F von FDs, Attributmenge X
- Output: Attributhülle von X bezüglich F

Integrierte Übung 6.3

Gegeben die Relation $R[A, B, C, D]$ mit der Menge $F = \{AB \to D, B \to A, C \to B\}$ von FDs.

- Berechnen Sie die Attributhülle von C.
- Bestimmen Sie alle Kandidatenschlüssel von R.

Überdeckung und Äquivalenz

- F ist eine Überdeckung von G (F covers G) wenn jede FD in G von F hergeleitet werden kann, i.e., $G^+ \subseteq F^+$.
- Zwei Mengen F und G von FDs sind äquivalent genau dann wenn $F^+ = G^+$.
- Gleichbedeutende Definitionen von äquivalent:
  - Jede FD in F kann von G hergeleitet werden und jede FD in G kann von F hergeleitet werden.
  - F ist eine Überdeckung von G und G ist eine Überdeckung von F.

Membership und Äquivalenz

- Membership-Problem: Ist $X \to Y$ in F^+?
- Das Membership-Problem für $X \to Y$ und einer Menge F von FDs kann folgendermaßen ausgedrückt werden:

$$X \to Y \in F^+ \Leftrightarrow Y \subseteq \mathcal{H}(F, X)$$

- Der Algorithmus zur Berechnung der Attributhülle kann also für das Membership-Problem angewandt werden.
- Membership-Algorithmus zur Äquivalenz zwischen F und G:
  1. Teste für alle FDs in F ob sie in G^+ sind.
  2. Teste für alle FDs in G ob sie in F^+ sind.
  3. F und G sind genau dann äquivalent, wenn alle Membership-Tests erfolgreich waren.

Beispiel: Äquivalenz/1

Betrachte $F = \{A \to C, AC \to D, E \to AD, E \to H\}$ und $G = \{A \to CD, E \to AH\}$. Sind F und G äquivalent?

Zeigen oder widerlegen Sie die Äquivalenz indem Sie die folgenden Aussagen überprüfen:

1. F ist eine Überdeckung von G (mithilfe der Armstrong-Axiome)
2. G ist eine Überdeckung von F (mithilfe von Membership-Tests)

Beispiel: Äquivalenz/2

1. F is Überdeckung von G:
   Für jedes $X \to Y \in G$ überprüfen wir: $X \to Y \in F^+$?

   $A \to CD \in F^+$:
   $A \to C \overset{\text{Verstärkung}}{\models} A \to AC$
   $A \to AC, AC \to D \overset{\text{Transitivität}}{\models} A \to D$
   $A \to C, A \to D \overset{\text{Vereinigung}}{\models} A \to CD$

   $E \to AH \in F^+$:
   $E \to H, E \to AD \overset{\text{Vereinigung}}{\models} E \to HAD$
   $E \to HAD \overset{\text{Dekomposition}}{\models} E \to AH, E \to D$

2. G ist Überdeckung von F:
   Für jedes $X \to Y \in F$ überprüfen wir: $X \to Y \in G^+$?

   $A \to C$:
   $C \in \mathcal{H}(G, A) = \{A, C, D\}$
   $\Rightarrow A \to C \in G^+$
   $AC \to D$:
   $D \in \mathcal{H}(G, AC) = \{A, C, D\}$
   $\Rightarrow AC \to D \in G^+$

   $E \to AD$:
   $AD \in \mathcal{H}(G, E) = \{E, A, H, C, D\}$
   $\Rightarrow E \to AD \in G^+$
   $E \to H$:
   $H \in \mathcal{H}(G, E) = \{E, A, H, C, D\}$
   $\Rightarrow E \to H \in G^+$

Kanonische Überdeckung

- Zu einer gegebenen Menge F von FDs nennt man F_c eine kanonische Überdeckung wenn folgende drei Eigenschaften erfüllt sind:
  1. $F_c^+ = F^+$ (d.h., F_c und F sind äquivalent)
  2. In F_c existieren keine FDs $X \to Y$, bei denen X oder Y überflüssige Attribute enthalten, d.h., es muss gelten:
     - Keine FD $X \to Y$ in F_c kann durch $X' \to Y$ mit $X' \subset X$ ersetzt werden ohne die Äquivalenz zu F zu verletzen.
     - Keine FD $X \to Y$ in F_c kann durch $X \to Y'$ mit $Y' \subset Y$ ersetzt werden ohne die Äquivalenz zu F zu verletzen.
  3. Jede linke Seite einer funktionalen Abhängigkeit in F_c ist einzigartig.
- Die kanonische Überdeckung ist sozusagen eine minimale Menge von FDs welche noch äquivalent ist zu F.
- Jedes Menge F von FDs hat eine kanonische Überdeckung.
- Es kann mehrere kanonische Überdeckungen geben.

Algorithmus für Kanonische Überdeckung/1

Eine kanonische Überdeckung der Menge F von FDs kann folgendermaßen berechnet werden.

1. Linksreduktion: Führe für alle $X \to Y \in F$ eine Linksreduktion durch.
   - Linksreduktion für $X \to Y$: Überprüfe für alle einzelnen Attribute $A \in X$:
     $$Y \subseteq \mathcal{H}(F, X - A)$$
     Falls dies gilt, ist A überflüssig und $X \to Y$ wird in F durch $(X - A) \to Y$ ersetzt:
     $$F := F - \{X \to Y\} \cup \{(X - A) \to Y\}$$
2. Rechtsreduktion: Führe für alle (verbleibenden) $X \to Y \in F$ eine Rechtsreduktion durch.
   - Rechtsreduktion für $X \to Y$: Überprüfe für alle $B \in Y$ ob
     $$B \in \mathcal{H}(F - \{X \to Y\} \cup \{X \to (Y - B)\}, X)$$
     Falls dies gilt, ist B auf der rechten Seite überflüssig und kann eliminiert werden, d.h., $X \to Y$ wird in F durch $X \to (Y - B)$ ersetzt.

Algorithmus für Kanonische Überdeckung/2

3. Entfernen von leeren Mengen: Entferne alle FDs der Form $X \to \emptyset$, die möglicherweise durch die Rechtsreduktion entstanden sind.
4. Vereinigung: Fasse mittels der Vereinigungsregel FDs der Form $X \to Y_1$,$X \to Y_2$,...,$X \to Y_n$ zusammen zu $X \to (Y_1 \cup Y_2 \cup \ldots \cup Y_n)$.

Eigenschaften des Algorithmus:

- Dieser Algorithmus erzeugt eine der möglichen kanonischen Überdeckungen.
- Je nach Ordnung der FDs können andere kanonische Überdeckungen herauskommen.

Beispiel: Kanonische Überdeckung berechnen

Gegeben die Menge $F = \{A \to B, A \to C, AB \to E, B \to ED\}$ von FDs.
Bestimme die kanonische Überdeckung F_c von F.

1. Linksreduktion: $F = \{A \to B, A \to C, B \to E, B \to ED\}$
   - wenn links nur 1 Attribut steht kann nie reduziert werden
   - $AB \to E$ könnte optional auch um B reduziert werden
2. Rechtsreduktion: $F = \{A \to B, A \to C, B \to \emptyset, B \to ED\}$
   - jedes der fünf Attribute auf der rechten Seite muss überprüft werden
   - statt $B \to E$ könnte optional $B \to ED$ rechts um E reduziert werden
3. Entfernen von leeren Mengen: $F = \{A \to B, A \to C, B \to ED\}$
4. Vereinigung: $F = \{A \to BC, B \to ED\}$

Die Kanonische Überdeckung ist $F_c = \{A \to BC, B \to ED\}$

Inhalt

Zerlegungen und deren Eigenschaften

- Zerlegung (decomposition) von Relationen
- Verlustlosigkeit (lossless join decomposition)
- Abhängigkeitsbewahrung (dependency preservation)

Relationale Entwurf durch Zerlegung

- Sinn der Zerlegung einer Relation ist das Vermeiden von Redundanzen
- Zerlegung: Schema einer Relation R in Schemas $Z = \{R_1, R_2, \ldots, R_n\}$ zerlegen, sodass

$$sch(R) = sch(R_1) \cup sch(R_2) \cup \ldots \cup sch(R_n)$$

- Für eine Instanz von R berechnen sich die neuen Relationen R_i:

$$\begin{aligned} R_1 &= \pi_{sch(R_1)}(R) \\ R_2 &= \pi_{sch(R_2)}(R) \\ \ldots & \\ R_n &= \pi_{sch(R_n)}(R) \end{aligned}$$

- Korrektheitskriterien für Zerlegungen von R in $Z = \{R_1, R_2, \ldots, R_n\}$:
  - Verlustlosigkeit: für jede Instanz muss R aus $R_1, R_2, \ldots, R_n$ rekonstruierbar sein.
  - Abhängigkeitsbewahrung: die FDs von R müssen auf $R_1, R_2, \ldots, R_n$ übertragbar sein.

Integrierte Übung 6.4

Gegeben eine Relation $R[A, B, C]$ mit der Instanz $R = \{(\alpha, x, 0), (\beta, y, 2), (\gamma, z, 1), (\alpha, y, 2)\}$.

Zerlege R in $R_1[A, B]$, $R_2[B, C]$.

Verlustlosigkeit

- Verlustlosigkeit (lossless join decomposition):
  Eine Zerlegung von R mit FDs F in $R_1, R_2, \ldots, R_n$ ist genau dann verlustlos, wenn für jede Instanz von R die F erfüllt gilt:

$$\pi_{sch(R_1)}(R) \bowtie \pi_{sch(R_2)}(R) \bowtie \ldots \bowtie \pi_{sch(R_n)}(R) = R$$

- Beachte: Das Wort "verlustlos" bezieht sich auf Information, nicht die Anzahl der Tupel. Im Gegenteil: Joins auf nicht verlustlose Zerlegungen erzeugen zusätzliche (falsche) Tupel.
- Satz: R_1 und R_2 sind eine verlustlose Zerlegung von R bezüglich der FDs F genau dann wenn
  - $(sch(R_1) \cap sch(R_2)) \rightarrow sch(R_1)$ ist in F^+ oder
  - $(sch(R_1) \cap sch(R_2)) \rightarrow sch(R_2)$ ist in F^+
- Intuition: Die Join-Attribute zwischen R_1 und R_2 sollen entweder für R_1 oder R_2 einen Schlüssel darstellen, d.h., alle natürlichen Joins sind Fremdschlüssel-Joins.

Integrierte Übung 6.5

Gegeben eine Relation $R[A, B, C]$ mit der Instanz $R = \{(\alpha, x, 0), (\beta, y, 2), (\gamma, z, 1), (\alpha, y, 2)\}$ und den funktionalen Abhängigkeiten $FD = \{AC \rightarrow B, B \rightarrow C\}$.

R wird in $Z = \{R_1[A, B], R_2[B, C]\}$ zerlegt.

a. Überprüfe, ob für diese Instanz von R Verlustlosigkeit gilt.
b. Ist die Zerlegung Z für R mit F für alle Instanzen verlustlos?
c. Was passiert mit der Zerlegung, wenn das Tupel $(\alpha, y, 2)$ durch $(\alpha, z, 2)$ ersetzt wird, sodass $B \rightarrow C$ nicht mehr erfüllt ist?

Integrierte Übung 6.5

a. Überprüfe, ob für diese Instanz von R Verlustlosigkeit gilt.

b. Ist die Zerlegung Z für R mit F für alle Instanzen verlustlos?

Integrierte Übung 6.5

c. Was passiert mit der Zerlegung, wenn das Tupel $(\alpha, y, 2)$ durch $(\alpha, z, 2)$ ersetzt wird, sodass $B \rightarrow C$ nicht mehr erfüllt ist?

Einschränkung

- Gegeben eine Zerlegung $Z = \{R_1, R_2, \ldots, R_n\}$ von R mit FDs F_R.
- Eine Einschränkung F_{R_i} von F_R auf R_i ist eine Menge von FDs, sodass F_{R_i} äquivalent ist zur Menge aller *FDs* $X \rightarrow Y \in F_R^+$ mit $(X \cup Y) \subseteq R_i$.

- Beispiel: $R[A, B, C, D]$, $F_R = \{A \rightarrow C, C \rightarrow B\}$
  Zerlegung: $R_1[A, B]$, $R_2[C, D]$

  Einschränkungen: $F_{R_1} = \{A \rightarrow B\}, F_{R_2} = \emptyset$

  $\rightarrow$ Es reicht also *nicht* die FDs von F_R zu übernehmen, in denen nur Attribute von R_i vorkommen
  - $A \rightarrow B$ gehört zu F_{R_1} obwohl es in F_R nicht vorkommt
  - $A \rightarrow B$ kommt jedoch in F_R^+ vor

  $\rightarrow$ F_{R2} enthält nur triviale FDs.

Berechnen der Einschränkung

- gegeben: Relation R mit FDs F_R, Zerlegung $Z = \{R_1, R_2, \ldots, R_n\}$
- gesucht: Einschränkung F_{R_i}
- Algorithmus:
  $F_{R_i} = \emptyset$
  für jede echte Teilmenge $X \subset sch(R_i)$:
  $Y = \mathcal{H}(F_R, X) \setminus \{X\}$ // triviale FDs entfernen
  $Y = Y \cap sch(R_i)$ // nur Attribute aus R_i betrachten
  falls $Y \neq \emptyset$:
  $F_{R_i} = F_{R_i} \cup \{X \rightarrow Y\}$
  return F_{R_i}

Abhängigkeitsbewahrung/2

- Abhängigkeitsbewahrung: Eine Zerlegung $Z = \{R_1, R_2, \ldots, R_n\}$ von R mit FDs F_R ist abhängigkeitsbewahrend genau dann wenn für die entsprechenden Einschränkungen F_{R_i} gilt:

$$F_R^+ = (F_{R_1} \cup F_{R_2} \cup \ldots \cup F_{R_n})^+$$

- Intuition: Bei abhängigkeitsbewahrender Zerlegung kann jede FD lokal auf einer Relation R_i geprüft werden.
- Praktische Bedeutung der Abhängigkeitsbewahrung:
  - FDs müssen bei jeder Änderung der Datenbank geprüft werden.
  - Wenn FDs nicht auf einzelnen Relation R_i geprüft werden können, muss ein Join $R_i \bowtie R_j$ zur Prüfung durchgeführt werden.
  - Das ist in der Praxis viel zu teuer.

Beispiel: Einschränkung u. Abhängigkeitsbewahrung

Gegeben $R[A, C, D]$, $F_R = \{A \to D, D \to C, C \to D\}$, sowie die Zerlegung von R in $R1[A, C]$, $R2[C, D]$.

a. Berechnen Sie die Einschränkungen F_{R1} und F_{R2} von F_R.

b. Ist diese Zerlegung abhängigkeitsbewahrend?

a. Einschränkungen:
$\mathcal{H}(F, A) \setminus \{A\} = \{D, C\}$
$\mathcal{H}(F, C) \setminus \{C\} = \{D\}$
$\mathcal{H}(F, D) \setminus \{D\} = \{C\}$
$F_{R_1} = \{A \to C\}$
$F_{R_2} = \{C \to D, D \to C\}$

b. Wir prüfen: $F_R^+ = (F_{R_1} \cup F_{R_2})^+$
$\{A \to D, D \to C, C \to D\}^+ =$
$\{A \to C, C \to D, D \to C\}^+$

$F_R^+ \supseteq (F_{R_1} \cup F_{R_2})^+$, da wir F_{R_1} und F_{R_2} aus F_R hergeleitet haben.

$F_R^+ \subseteq (F_{R_1} \cup F_{R_2})^+$?

$A \to C,\ C \to D \overset{T}{\models} A \to D$ ✓

Inhalt

Normalisierung/1

Übersicht über die Normalformen:

- 1NF: Attributwerte müssen atomar sein.
- 2NF, 3NF, BCNF: basieren auf Schlüsseln und FDs einer Relation
- 4NF: basieren auf Schlüsseln und mehrwertigen Abhängigkeiten (multi-valued dependencies, MVDs)
- 5NF: basieren auf Schlüsseln und Join Dependencies (JDs)

Weiters müssen beim relationalen Entwurf berücksichtigt werden:

- Verlustlosigkeit der entsprechenden Joins (sehr wichtig, darf niemals geopfert werden)
- Abhängigkeitsbewahrung der funktionalen Abhängigkeiten (kann unter Umständen aufgegeben werden)

Normalisierung/2

- Normalisierung: Die Attribute eines (schlechten) Schemas einer Relation werden auf kleinere (gute) Schemas aufteilen, welche den Normalformen genügen.
- Die Normalisierung wurde von Codd im Jahr 1972 eingeführt.
- Während des Normalisierungsprozesses werden eine Reihe von Tests auf einem Schema durchgeführt um zu überprüfen, ob sich das Schema in einer bestimmten Normalform befindet.
- Eine normalisierte Datenbank besteht aus *guten* Schemas.

- Praxistipp: Datenbank-Designer brauchen nicht bis zur höchsten Normalform normalisieren:
  - es gibt einen Trade-off zwischen NF und Abfrage-Effizienz
  - normalerweise wird 3NF, BCNF oder 4NF ausgewählt

Normalisierung/3

- Kontrollierte Redundanz:
  - Redundanz, welche dem System bekannt ist
  - kontrollierte Redundanz ist gut
  - Beispiele: Fremdschlüssel, Indices
- Denormalisierung:
  - Der Join mehrerer Relationen in einer höheren Normalform wird als Relation gespeichert.
  - Die Ergebnis-Relation befindet sich in einer niedrigeren Normalform, da Joins Normalformen zerstören.

Erste Normalform (1NF)/1

- Verbietet:
  - zusammengesetzte Attribute
  - mehrwertige Attribute
  - verschachtelte Relationen: Attribute, deren Wert für jedes Tupel eine Relation ist
- 1NF wird oft als Teil der Definition einer Relation gesehen.
- Beispiel: Folgende Instanz der Relation *Fachbereiche* ist nicht in 1NF (mehrwertiges Attribut):

Fachbereiche

FName	FNum	LeiterSVN	Standorte
Research	5	334455	{Salzburg, Wien, Linz }
Administration	4	987654	{ Innsbruck }
Headquarters	1	888666	{ Linz }

Erste Normalform (1NF)/2

- Abhilfe um 1NF zu erhalten:
  - zusammengesetzte Attribute: jeder Teil wird ein eigenes Attribut
  - mehrwertige Attribute: neues Tupel für jeden Wert des mehrwertigen Attributs erzeugen
  - geschachtelte Relationen: neues Tupel für jedes Tupel der geschachtelten Relation erzeugen

- Beispiel: *Fachbereiche* in 1NF gebracht.

Fachbereiche_1NF

FName	FNum	LeiterSVN	Standort
Research	5	334455	Salzburg
Research	5	334455	Wien
Research	5	334455	Linz
Administration	4	987654	Innsbruck
Headquarters	1	888666	Linz

Zweite Normalform (2NF)/1

- Zweite Normalform (2NF): Eine Relation R befindet sich in der zweiten Normalform (2NF) genau dann wenn sie sich in 1NF befindet und jedes Nicht-Schlüssel Attribut voll funktional abhängig von allen Kandidatenschlüsseln ist.
- Nicht-Schlüssel Attribut: Attribut, das nicht Teil eines Kandidatenschlüssels (inklusive Primärschlüssel) ist.
- Ein Attribut A ist voll funktional abhängig vom Kandidatenschlüssel K ($K \xrightarrow{\bullet} A$), wenn es keine echte Teilmenge $X \subset K$ gibt, sodass $X \rightarrow A$:

$$K \xrightarrow{\bullet} A \Leftrightarrow K \rightarrow A \wedge \forall X \subset K : X \not\rightarrow A$$

- Intuition: 2NF ist verletzt, wenn mehrere Entitäten in einer einzigen Relation modelliert werden.

Zweite Normalform (2NF)/2

- Beispiel: Folgende Relation ist nicht in 2NF:

AngProj

SVN	PNum	Stunden	AName	PName	POrt
1234	1	32.5	Schmidt	ProductX	Salzburg
1234	2	7.5	Schmidt	ProductY	Wien
6688	3	40.5	Mair	ProductZ	Linz
4567	1	20.0	Huber	ProductX	Salzburg
4567	2	20.0	Huber	ProductY	Wien
3334	2	10.0	Wong	ProductY	Wien
3334	3	10.0	Wong	ProductZ	Linz
3334	10	10.0	Wong	Computerization	Innsbruck
3334	20	10.0	Wong	Reorganization	Linz

- Warum ist *AngProj* nicht in 2NF?
  - Kandidatenschlüssel ist $\{SVN, PNum\}$, von dem aber nur *Stunden* voll funktional abhängig ist.
  - *SVN* ist ein Teilschlüssel der *AName* bestimmt.
  - *PNum* ist ein Teilschlüssel der *PName* und *POrt* bestimmt.

Zweite Normalform (2NF)/3

- Abhilfe um 2NF zu erhalten:
  - neue Relation für jeden Teilschlüssel mit seinen abhängigen Attributen
  - eine Relation mit ursprünglichem Schlüssel und allen voll funktional abhängigen Attributen
- Beispiel: $AngProj[SVN, PNum, Stunden, AName, PName, POrt]$
  - FDs: $\{SVN, PNum\} \rightarrow Stunden, SVN \rightarrow AName, PNum \rightarrow \{PName, POrt\}$
  - $\{SVN, PNum\}$ ist einziger Kandidatenschlüssel.
  - *SVN* und *PNum* sind Teilschlüssel mit abhängigen Attributen.

2NF Normalisierung von *AngProj*:

- AngProj1(SVN, AName)
- AngProj2(PNum, PName, POrt)
- AngProj3(SVN, PNum, Stunden)

Integrierte Übung 6.6

Befinden sich die folgenden Relationen in 2NF?

a. $R[A, B, C]$ mit $F = \{B \rightarrow C\}$

b. $R[A, B, C]$ mit $F = \{A \rightarrow BC, B \rightarrow C\}$

Zweite Normalform reicht nicht aus

- Für Relation *AngFB* gelten folgende funktionale Abhängigkeiten:
  - $fd1 : SVN \rightarrow sch(AngFB)$ (d.h. SVN ist Kandidatenschlüssel)
  - $fd2 : FNum \rightarrow \{FName, LeiterSVN\}$

AngFB

AName	SVN	Jahrgang	Adresse	FNum	FName	LeiterSVN
Schmidt	1234	1965	Linz	5	Research	2345
Schmidt	2345	1965	Linz	5	Research	2345
Wong	6688	1968	Linz	4	Admin	4567
Zelaya	4567	1941	Linz	4	Admin	4567
Borg	3334	1937	Dallas	4	Admin	4567

- *AngFB* ist in 2NF, dennoch gibt es Redundanz:
  - *FName* und *LeiterSVN* für einen Fachbereich werden für jeden Angestellten redundant abgelegt.
- Problem: *FName* und *LeiterSVN* sind transitiv abhängig vom Schlüssel *SVN*:
  - $SVN \rightarrow FNum$, $FNum \rightarrow FName$
  - $SVN \rightarrow FNum$, $FNum \rightarrow LeiterSVN$

Dritte Normalform (3NF)

- Dritte Normalform (3NF): Eine Relation R befindet sich in 3NF genau dann wenn sie sich in 1NF befindet und für alle FDs $X \rightarrow Y \in F^+$ mindestens eine der folgenden Bedingungen gilt:
  - $X \rightarrow Y$ ist trivial (d.h. $Y \subseteq X$)
  - X ist ein Superschlüssel von R
  - jedes Attribut $A \in Y$ ist in einem Kandidatenschlüssel von R enthalten
- Intuition: 3NF verbietet transitive Abhängigkeiten.
- 3NF $\subset$ 2NF: eine Relation in 3NF ist auch in 2NF

Beispiel: Dritte Normalform (3NF)

- Wir betrachten Relation *AngFB* mit den funktionale Abhängigkeiten:
  - $fd1 : SVN \rightarrow sch(AngFB)$ (d.h. SVN ist Kandidatenschlüssel)
  - $fd2 : FNum \rightarrow \{FName, LeiterSVN\}$

AngFB

AName	SVN	Jahrgang	Adresse	FNum	FName	LeiterSVN
Schmidt	1234	1965	Linz	5	Research	2345
Schmidt	2345	1965	Linz	5	Research	2345
Wong	6688	1968	Linz	4	Admin	4567
Zelaya	4567	1941	Linz	4	Admin	4567
Borg	3334	1937	Dallas	4	Admin	4567

- *fd2* erfüllt keine der drei Bedingungen für 3NF:
  - *fd2* ist nicht trivial
  - *FNum* ist keine Superschlüssel von *AngFB*
  - *FName* ist in keinem Kandidatenschlüssel enthalten

$\Rightarrow$ *AngFB* ist nicht in 3NF.

Synthesealgorithms zur Zerlegung in 3NF/1

- Syntesealgorithms: Zerlegt das Schema einer Relation R mit funktionalen Abhängigkeiten F in die Schemas $R_1, R_2, \ldots, R_n$ mit folgenden Eigenschaften:
  - alle R_i $(1 \leq i \leq n)$ sind in 3NF
  - die Zerlegung ist verlustfrei
  - die Zerlegung ist abhängigkeitsbewahrend

Synthesealgorithms zur Zerlegung in 3NF/2

Relation R mit funktionalen Abhängigkeiten F in 3NF zerlegen:

1. Bestimme kanonische Überdeckung F_c zu F.
2. Für jede funktionale Abhängigkeit $X \to Y \in F_c$:
   a. Kreiere eine Relation R_X mit dem Schema $X \cup Y$.
   b. Ordne R_X die FDs $F_X = \{X' \to Y' \in F_c \mid X' \cup Y' \subseteq sch(R_X)\}$ zu.
3. Falls keine der neuen Relation R_X einen Kandidatenschlüssel von R bezüglich F_c enthält, wähle einen Kandidatenschlüssel $K \subseteq sch(R)$:
   a. Erzeuge eine Relation R_K mit Schema K.
   b. Die FDs von R_K sind $F_K = \emptyset$.[2]
4. Eliminiere R und alle neue erzeugten Relationen R_i die in einer anderen Relation R_j enthalten sind, d.h., $sch(R_i) \subseteq sch(R_j)$.

[2] Würde F_K eine FD $X \to Y \in F_c$ enthalten, dann wäre F_K kein Kandidatenschlüssel.

Beispiel: Synthesealgorithms

Zerlegen Sie $R[A, B, C, D, E, G]$ mit
$F = \{A \to BD, AB \to E, B \to EG, C \to AB\}$ in 3NF.

1. kanonische Überdeckung $F_c = \{A \to BD, B \to EG, C \to A\}$
2. $R_A[A, B, D], F_A = \{A \to BD\}$
   $R_B[B, E, G], F_B = \{B \to EG\}$
   $R_C[C, A], F_C = \{C \to A\}$
3. Nichts zu tun, da Kandidatenschlüssel C in R_3 enthalten.
4. Keine redundanten Teilschemas.

3NF-Zerlegung: R_A mit F_A, R_B mit F_B, R_C mit F_C

Boyce-Codd NormalForm/1

- Boyce-Codd Normal Form (BCNF): Eine Relation R ist in BCNF genau dann wenn sie in 1NF ist und für alle $X \to Y \in F^+$ mindestens eine der folgenden Bedingungen gilt:
  - $X \to Y$ ist trivial (d.h. $Y \subseteq X$)
  - X ist ein Superschlüssel von R
- BCNF $\subset$ 3NF: Eine Relation in BCNF ist auch in 3NF.
- Beispiel: Folgende Relation ist in 3NF aber nicht in BCNF:

Lernen

Stud	Kurs	Buch
Schmidt	Data Structures	Bertram
Schmidt	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz
Gale	Data Structures	Horowitz

Boyce-Codd Normal Form/2

Lernen

Stud	Kurs	Buch
Schmidt	Data Structures	Bertram
Schmidt	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz
Gale	Data Structures	Horowitz

- FDs in der Relation *Lernen*:
  - fd1: $\{Stud, Kurs\} \to Buch$
  - fd2: $Buch \to Kurs$
- *Lernen* ist in 3NF:
  - fd1: $\{Stud, Kurs\}$ ist Kandidatenschlüssel (d.h. auch Superschlüssel)
  - fd2: *Kurs* ist im Kandidatenschlüssel enthalten
- Lernen ist nicht in BCNF:
  - fd2 ist nicht trivial und *Buch* ist kein Superschlüssel

Boyce-Codd Normal Form/3

- Zerlegung von *Lernen* in $KB[Kurs, \underline{Buch}]$ und $SB[\underline{Stud}, \underline{Buch}]$:

KB

Kurs	Buch
Data Structures	Bertram
Data Management	Martin
Compilers	Hoffman
Data Structures	Horowitz

SB

Stud	Buch
Schmidt	Bertram
Schmidt	Martin
Hall	Hoffman
Brown	Horowitz
Gale	Horowitz

- FDs in den Relationen KB und SB:
  - KB : $Buch \rightarrow Kurs$
  - SB : nur triviale Anhängigkeiten
  - $\{Kurs, Stud\} \rightarrow Buch$ von *Lernen* ist verloren gegangen

Dekompositionsalgorithmus zur Zerlegung in BCNF/1

- Dekompositionsalgorithms: Zerlegt das Schema einer Relation R mit funktionalen Abhängigkeiten F in die Schemas $R_1, R_2, \ldots, R_n$ mit folgende Eigenschaften:
  - alle R_i $(1 \leq i \leq n)$ sind in BCNF
  - die Zerlegung ist verlustfrei
- Die Zerlegung in BCNF ist in manchen Fällen nicht abhängigkeitsbewahrend.

Dekompositionsalgorithmus zur Zerlegung in BCNF/2

Relation R mit funktionalen Abhängigkeiten F_R in BCNF zerlegen:

1. Z=$\{R\}$
2. Solange es ein $R_i \in Z$ gibt, sodass R_i nicht in BCNF ist:
   a. Finde eine FD $X \rightarrow Y \in F^+_{R_i}$, sodass:
      - $X \cap Y = \emptyset$, d.h. keine (teilweise) triviale Anhängigkeit und
      - $X \not\rightarrow sch(R_i)$, d.h. X ist kein Superschlüssel von R_i

   b. Erweitere Y zu Y', das möglichst viele abhängige Attribute enthält:

$$X \rightarrow Y' \quad \text{wobei } Y' = \mathcal{H}(F_{R_i}, X) - X$$

   c. Zerlege R_i in zwei Relationen R_{i1} und R_{i2} mit den Schemas
      - $sch(R_{i1}) = X \cup Y'$
        $F_{R_{i1}}$ ist die Einschränkung von R_i auf R_{i1}
      - $sch(R_{i2}) = R_i - Y'$
        $F_{R_{i2}}$ ist die Einschränkung von R_i auf R_{i2}

   d. Ersetze R_i durch R_{i1} und R_{i2} in Z.

Beispiel: Dekompositionsalgorithmus/1

Gegeben $R[A, B, C, D]$ mit $F_R = \{A \rightarrow C, C \rightarrow D, B \rightarrow D\}$. Zerlege das Schema verlustlos in BCNF. Ist die Zerlegung abhängigkeitsbewahrend?

Lösung:

Nebenrechnung: Attributhülle der linken Seiten der FDs in F_R
$\mathcal{H}(F_R, A) = \{A, C, D\}$ $(A \rightarrow CD)$
$\mathcal{H}(F_R, B) = \{B, D\}$ $(B \rightarrow D)$
$\mathcal{H}(F_R, C) = \{C, D\}$ $(C \rightarrow D)$

Einziger Kandidatenschlüssel: AB

Beispiel: Dekompositionsalgorithmus/2

1. $Z = \{R\}$
2. Loop 1: $Z = \{R\}$, R ist nicht in BCNF
   a. teste $A \to C$
      $X = \{A\}$, $Y = \{C\}$
      $\{A\} \cap \{C\} = \emptyset$ ✓
      $A \not\to sch(R)$: $\mathcal{H}(F_R, A) = \{A, C, D\} \neq sch(R)$ ✓
   b. $X = \{A\}$, $Y = \{C\}$, $Y' = \mathcal{H}(F_R, A) - \{A\} = \{C, D\}$
   c. $R_1[A, C, D]$, $F_{R_1} = \{A \to CD, C \to D\}$
      $R_2[A, B]$, $F_{R_2} = \emptyset$
   d. $Z = \{R_1, R_2\}$

Beispiel: Dekompositionsalgorithmus/3

2. Loop 2: $Z = \{R_1, R_2\}$
   R_2 ist in BCNF, da $F_{R_2} = \emptyset$
   R_1 ist wegen $C \to D$ nicht in BCNF:
   - $C \to D$ ist nicht trivial
   - C ist kein Superschlüssel

   a. teste $C \to D$:
      $X = \{C\}$, $Y = \{D\}$
      $\{C\} \cap \{D\} = \emptyset$ ✓
      $C \not\to sch(R_1)$: $\mathcal{H}(F_R, C) = \{C, D\} \neq sch(R_1)$ ✓
   b. $X = \{C\}$, $Y = \{D\}$, $Y' = \mathcal{H}(F_{R_1}, C) - \{C\} = \{D\}$
   c. $R_{11}[C, D]$, $F_{R_{11}} = \{C \to D\}$
      $R_{12}[A, C]$, $F_{R_{12}} = \{A \to C\}$
   d. $Z = \{R_{11}, R_{12}, R_2\}$
      $F_{R_{11}} = \{C \to D\}$, $F_{R_{12}} = \{A \to C\}$, $F_{R_2} = \emptyset$

   Loop 3: alle Relationen in Z sind in BCNF $\Rightarrow$ Ende

Beispiel: Dekompositionsalgorithmus/4

Ist die Zerlegung $Z = \{R_{11}, R_{12}, R_2\}$ mit $F_{R_{11}} = \{C \to D\}$, $F_{R_{12}} = \{A \to C\}$, $F_{R_2} = \emptyset$ abhängigkeitsbewahrend?

$F_Z = F_{R_{11}} \cup F_{R_{22}} \cup F_{R_2} = \{C \to D, A \to C\}$

$F_R^+ \supseteq F_Z^+$ ist erfüllt, da $F_{R_{11}}, F_{R_{22}}, F_{R_2}$ aus F_R abgeleitet wurden

$F_R^+ \subseteq F_Z^+$ ist nicht erfüllt, da $B \to D$ verloren geht:
$B \to D \in F_R^+$ aber $B \to D \notin F_Z^+$: $\mathcal{H}(F_Z, B) = \{B\}$

Die Zerlegung ist nicht abhängigkeitsbewahrend.

Zusammenfassung Normalformen

- Die gebräuchlichsten Normalformen sind:
  - 1NF: atomare Attribute
  - 2NF: jede Relation entpricht einer eigenen Entität
  - 3NF: keine transitiven Abhängigkeiten
  - BCNF: nur Schlüsselabhängigkeiten erlaubt
- Wenn eine Relation in BCNF ist, gibt es keine Redundanz aufgrund funktionaler Abhängigkeiten mehr.
- Für die Normalformen gilt:
  $$\text{BCNF} \subset \text{3NF} \subset \text{2NF} \subset \text{1NF}$$
- Jede Relation lässt sich verlustlos bis zu BCNF zerlegen.
- Jede Relation lässt sich abhängigkeitsbewahrend bis zu 3NF zerlegen.
- Bei der Zerlegung in BCNF können FDs verloren gehen.