Database Research Group

Datenbanksysteme

Bernhard Seeger

http://www.mathematik.uni-marburg.de/~seeger

Organisation (Vorlesung)

- **Vorlesung**
  - Fr. 10-14h Hörsaal C im Hörsaalgebäude Lahnberge
    - 10:15 – 11:15 Vorlesung
    - 11:30 – 12:30: Vorlesung
    - 13:00 – 14:00: Vorlesung
  - Letzte Vorlesung am 12.7.2019
  - Brückentage: 30. Mai, 20. Juni
    - Was sollen wir machen?

!! Neuer Vorlesungsort !!

!! Neue Vorlesungszeiten !!

- **Am Samstag, den 18 Mai: SQL-Workshop**
  - PC-Saal (Ebene D3) neben dem Treppenhaus D im Mehrzweckgebäude Lahnberge

Organisation (Übung)

- **Übungsleiter:**
  - Jana Holznigenkemper, Andreas Morgen
- **Übungsblätter**
  - Ausgabe und Abgabe des Übungsblatts am Freitag (10h)
  - Übungstermine: Mo 14-16, Di 14-16 und Di 16-18
  - Statt einem Übungsblatt wird es am 24.5 einen 30-minütigen Test über den Inhalt des SQL-Workshops geben.
    - Uhrzeit: 10:15 - 10:45.
    - Vorlesungsräume für den Test werden noch angekündigt.

Organisation (Prüfung)

- **Studienleistung**
  - Mindestens 50% der Übungsaufgaben
  - Maximal zwei Übungszettel mit 0 Punkten
  - Sonderreglung Lehramt
    - Mindestens 40% der Übungsaufgaben
    - Eine fachdidaktische Zusatzleistung

- **Prüfungsleistung**
  - Abschlussklausur am Freitag, den 19.7.2019, von 12-14h im 00/0030 der Biegenstraße 14, Hörsaalgebäude.
  - Nachholklausur: wird noch bekannt gegeben

Organisation (Ilias)

- **Nutzung von Ilias**
  - Alle Materialien werden in Ilias zur Verfügung gestellt.
    - Übungsblätter
    - Folien
  - Abgabe der Übungen

  - Anmeldung zwingend erforderlich (ab sofort)
  - Anmeldung für die Übungen (ab Freitag 18h)

Literatur

Deutsche Bücher

- A. Kemper, A. Eikler: “Datenbanksysteme. Eine Einführung”, De Gruyter Studium, 2015.
  - Frühere Auflagen sind im Oldenbourg-Verlag erschienen.
- G. Saake, K.-U. Sattler, A. Heuer: “Datenbanken Konzepte und Sprachen”, mitp. 2018.
- G. Vossen: “Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme”, Oldenbourg, 2008.

Englische Bücher

- Jeffrey D. Ullman, Jennifer D. Widom: A First Course in Database Systems, Prentice Hall, 2007.
- Raghu Ramakrishnan, Johannes Gehrke: Database Management Systems, Mcgraw-Hill Professional

WEB-Quellen

- **Videokanal**
  - www.datenbankenlernen.de
- **Relationale Algebra**
  - https://dbis-uibk.github.io/relax/calc.htm
  - Table API Apache Flink

Inhaltsverzeichnis

- Einführung (26.4)
- Relationales Modell (26.4, 3.5)
  - Relationale Algebra, Tupelkalkül, Erweiterte Relationale Algebra
- SQL: Die relationale Datenbanksprache (10.5, 17.5, 18.5)
- Konzeptioneller Datenbankentwurf (24.5)
- Entwurfstheorie (7.6)
- Transaktionskonzepte u. Fehlerbehandlung (14.6, 21.6)
- Anwendungsprogrammierung (21.6, 28.6)
- Physische Datenorganisation & Implementierung der relationalen Algebra (5.7)
- NotOnly-SQL Systeme (12.7)

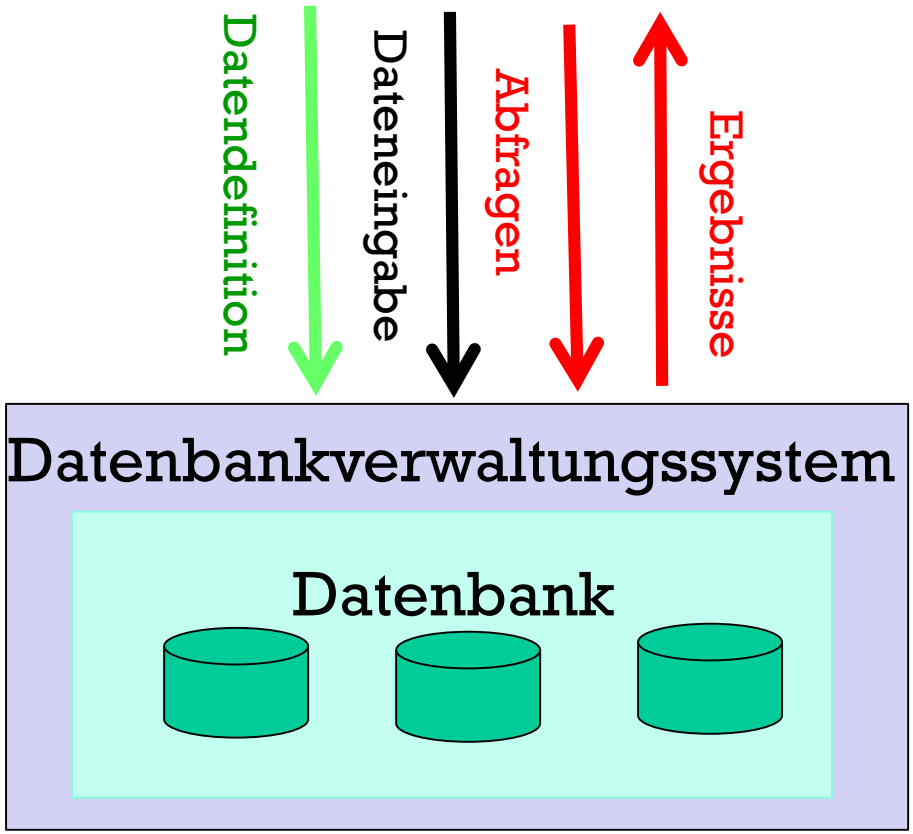
Datenbanken und DBMS

- **Datenbanksysteme (DBS)**
  - **dienen zur rechnergestützten Verwaltung großer, persistent zu verwaltenden Datenbestände.**
    - Lebensdauer der Daten >> Lebensdauer der Programme

- **Datenbanksysteme bestehen aus**
  - **Datenbankverwaltungssystem** (engl. database management system, **DBMS**)
    - Software zur Verwaltung von Daten.
  - **Datenbank** repräsentiert eine logisch zusammenhängende Datenmenge bestehend aus
    - den zu verwaltenden Daten,
    - Hilfsdaten (z. B. Indexe, Logdaten, Metadaten).

Benutzerinteraktion

Benutzer



Klassische Anwendungen

- **Bankinformationssystem**
  - Verwaltung der Kunden, ihre Konten, …
- **Versicherungsinformationssystem**
  - Verwaltung der Kunden, ihre Verträge, …
- **Telekommunikation**
  - Abrechnung
- **Logistik**
  - Paketversand

Beispiele

- **Große Datenbanken heute**
  - Max-Planck-Institut für Meteorologie (2015)
    - Größe
      - 330 TByte (Online-Speicher) + 6PByte (Offline-Speicher)
  - AT&T (2015)
    - Anzahl von Datensätzen
      - 1.9 Billionen Datensätze

- **Technologischer Fortschritt im Bereich DBMS**
  - ➔ Nahezu problemlose Unterstützung aller klassischer Anwendungen (in naher Zukunft)!

Gibt es noch etwas zu tun?

- **Aufzeichnung aller persönlichen Daten in Datenbanken**
  - Banktransaktionen, Email, Bilder, Vitaldaten, Aufzeichnung aller Gespräche
  - Vision
    - Vannevar Bush: „As we may think“ (1945)
- **Technische Realisierung heute möglich**
  - Billiger Magnetplattenspeicher
    - Kapazität: 10 TB
  - SSD
    - Kapazität bis zu 2 TB
  - Sehr große Hauptspeicher
    - In Kombination mit NVRAM

Gibt es noch etwas zu tun? (2)

- **Neue Formen der Datenerfassung**
  - Sensoren
    - Verkehrssensor, Finanzticker

- **Anforderungen**
  - Verwaltung aller Sensordaten in einer Datenbank
  - Neue Arten von Abfragen
    - Unterstützung von historischen Abfragen
      - Wo war ich am 7.7.2009 um 12h?
    - Kontinuierliche Abfragen
      - Informiere mich über Änderungen beim Börsenkurs von AT&T?

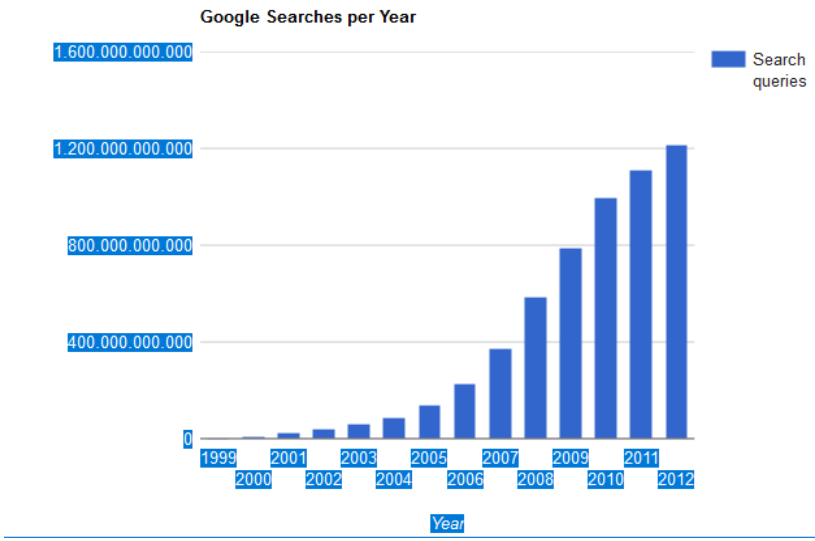
Die Vision wird zu Wahrheit!

Soziale Netzwerke

- **Facebook**
  - 3000 neue Bilder pro Sekunde = 250 Millionen pro Tag
  - > 1 Milliarde Nutzer

Web

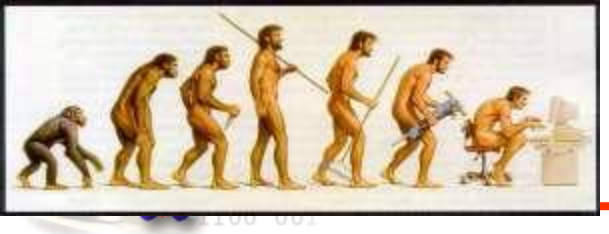
- Google
  - Anzahl von Suchoperationen/Tag
    - 5.5 Milliarden
- Internet Archive
  - 18,5 PByte (2014)
  - 50 Pbyte (2017)



Neue Architekturen für DBMS

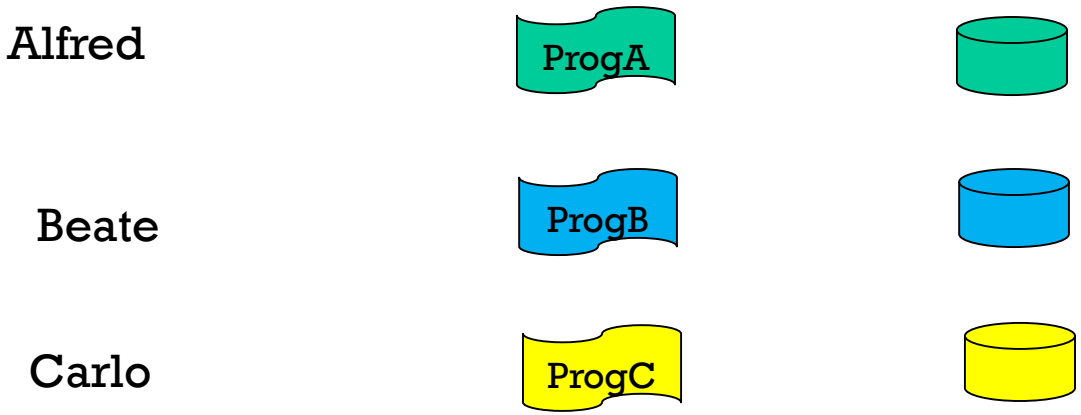
- **DBMS als Ecosystem der Informationsverarbeitung**
  - Entwicklung neuer Architekturen
    - NoSQL-Datenbanken
    - Verteilte Datenbanken
    - Hauptspeicher-Datenbanken
    - Datenstromsysteme
  - Es gibt noch viel zu tun!

- **Diese Entwicklungen werden derzeit unter dem Synonym BIG DATA zusammengefasst.**
  - Informatiktechnik mit hoher gesellschaftlicher Relevanz!
    - Internet of Things
    - Industrie 4.0
    - ….



Als es noch keine Datenbanksysteme gab, ...

- Entwicklung von DBS begann vor etwa 50 Jahren.
  - Zuvor wurden vornehmlich einfache Dateisysteme benutzt.
- Beispiel für die Datenverarbeitung in einer Versicherung:
  - Drei Kundenberater **Alfred**, **Beate** und **Carlo**, die je nach Art des Versicherungstyps Kunden betreuen.
  - Jeder der Kundenberater benutzt für den Zugriff auf die Kundendaten ein selbstentwickeltes Programm
  - Jeder Berater hat seine eigene Kundendatei.



Anwendungsprogramme

- **Anwendungsprogramm (AWP)**
  - Ein Programm, das direkt durch den Benutzer oder eine spezifische Anwendungskomponente aufgerufen wird.
  - Java-Beispiel (ohne Datenbanksystem)

C#-Beispiel (mit Datenbanksystem)

```
static void Main () {

        // Verbindung zur Datenbank

        SQLConnection db= new SQLConnection("Data Source = (local); Initial
                        Catalog = MyDatabase; Integrated Security=SSPI");

        // Anfrage
        var simpleQuery =  from tabelle in db
                           where tabelle.feld > limit
                           select feld;

        // Zugriff auf die Ergebnisse

        foreach (var f in simpleQuery) { Console.WriteLine(f.feld); }

}
```

Probleme

- **Direkte Erzeugung und Verarbeitung der Daten erfolgte im AWP unter Verwendung von Dateien**
  - kein standardisiertes Speicherungsformat
    - ➔ hoher Aufwand beim Datenaustausch
  - mehrfache und unkoordinierte Verwaltung der Daten
    - ➔ häufige Inkonsistenzen im Datenbestand
  - hoher Aufwand bei der Verknüpfung von Daten aus mehreren Dateien
  - Zugriff auf Daten erfolgt explizit im AWP
    - ➔hoher Aufwand bei der AWP-Entwicklung
    - ➔Optimierung des Programmcode durch Entwickler
  - Mehrbenutzerbetrieb nahezu unmöglich
    - ➔ Dateninkonsistenzen und Datenverluste
  - Unzureichende Möglichkeiten beim Datenschutz

Anforderungen an Datenbanksysteme (1)

- **Gemeinsame Datenbasis mehrerer Benutzer**
  - Gemeinsam genutzte, persistente Datenbasis auf schnellem Speicher
    - Direkter Zugriff durch Benutzer
    - Indirekter Zugriff über AWP
  - Kontrollierte Datenredundanz
    - Vermeidung von Kopien der gleichen Daten durch integrierte Verwaltung aller Daten.
- **Mehrbenutzerbetrieb**
  - Gleichzeitiger Zugriff mehrerer Benutzer auf gemeinsame Datenbank
  - Virtuelles Einbenutzersystem
    - Keine Beeinflussung durch andere Benutzer

Anforderungen an Datenbanksysteme (2)

- **Sicherstellung der Datenqualität**
  - **Datenintegrität und Datenkonsistenz**
    - Unterstützung von Integritätsbedingungen
      - ➔Gewährleistung der Korrektheit und Vollständigkeit der Daten
      - Automatische Überprüfung der Bedingungen beim Einfügen, Ändern und Löschen der Daten
  - **Datenschutz**
    - Zugriffskontrolle durch Authentifizierung und Verschlüsselung
      - ➔ Schutz der Datenbank vor nicht-autorisierten Zugriff
  - **Schutz der Daten im Falle eines Systemfehlers**
    - Log-Dateien und Sicherungskopien
      - ➔ Wiederanlauf des Systems und automatisches Wiederherstellen der aktuellen Datenbank

Anforderungen an Datenbanksysteme (3)

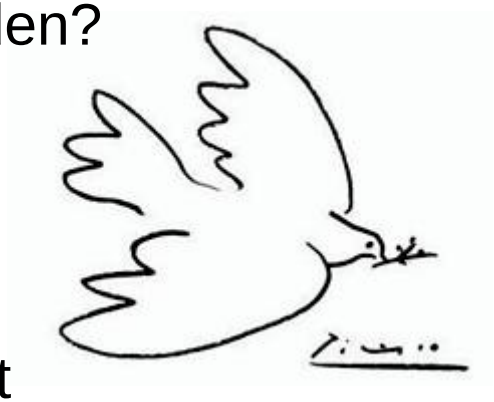
- **Bereitstellung von unterschiedlichen Benutzerschnittstellen**
  - Ad-hoc Anfragesprachen für interaktive Benutzer
  - Menügesteuerte, einfach zu benutzende Schnittstellen
  - Spezieller Zugang für Administrator
- **Unterstützung der Softwareentwicklung mit DBMS**
  - Programmierschnittstellen für die Softwareerstellung
  - Schnelle Entwicklung von Software unter Ausnutzung einer mächtigen Infrastruktur
  - Flexible und schnelle Anpassung der Software bei Änderungen in der Datenbank wie z. B.
    - Verteilung der Daten über mehrere Festplatten
    - Änderung der Speicherorganisation
    - Änderung des Typs der Daten

Anforderungen an Datenbanksysteme (4)

- **Hohe Leistungsfähigkeit**
  - Ziele
    - **Niedrige Antwortzeiten** bei einer Anfrage
    - **Hoher Durchsatz**
      - Maximierung der Anzahl der Anfragen pro Sekunde
- **Lösungen in einem DBMS**
  - Werkzeuge zur effizienten Speicherung und Anfrageverarbeitung
    - Indexstrukturen für große Datenmengen
      - ➔ Logarithmische Zugriffskosten
    - Effiziente Implementierung der Algorithmen
      - ➔ Z. B. zum Sortieren großer Datenmengen
  - Effektive Anfragebearbeitung
    - Automatische Optimierung von Anfragen

Wichtige Konzepte
nicht nur in Datenbanken

- **Datenabstraktion**
  - Welcher Aspekt einer Anwendung ist relevant und soll in der Datenbank abgebildet werden?
  - Einführung von Abstraktionsebenen



- **Datenmodell**
  - Infrastruktur zur Abbildung der realen Welt

- **Datenunabhängigkeit**

Datenabstraktion

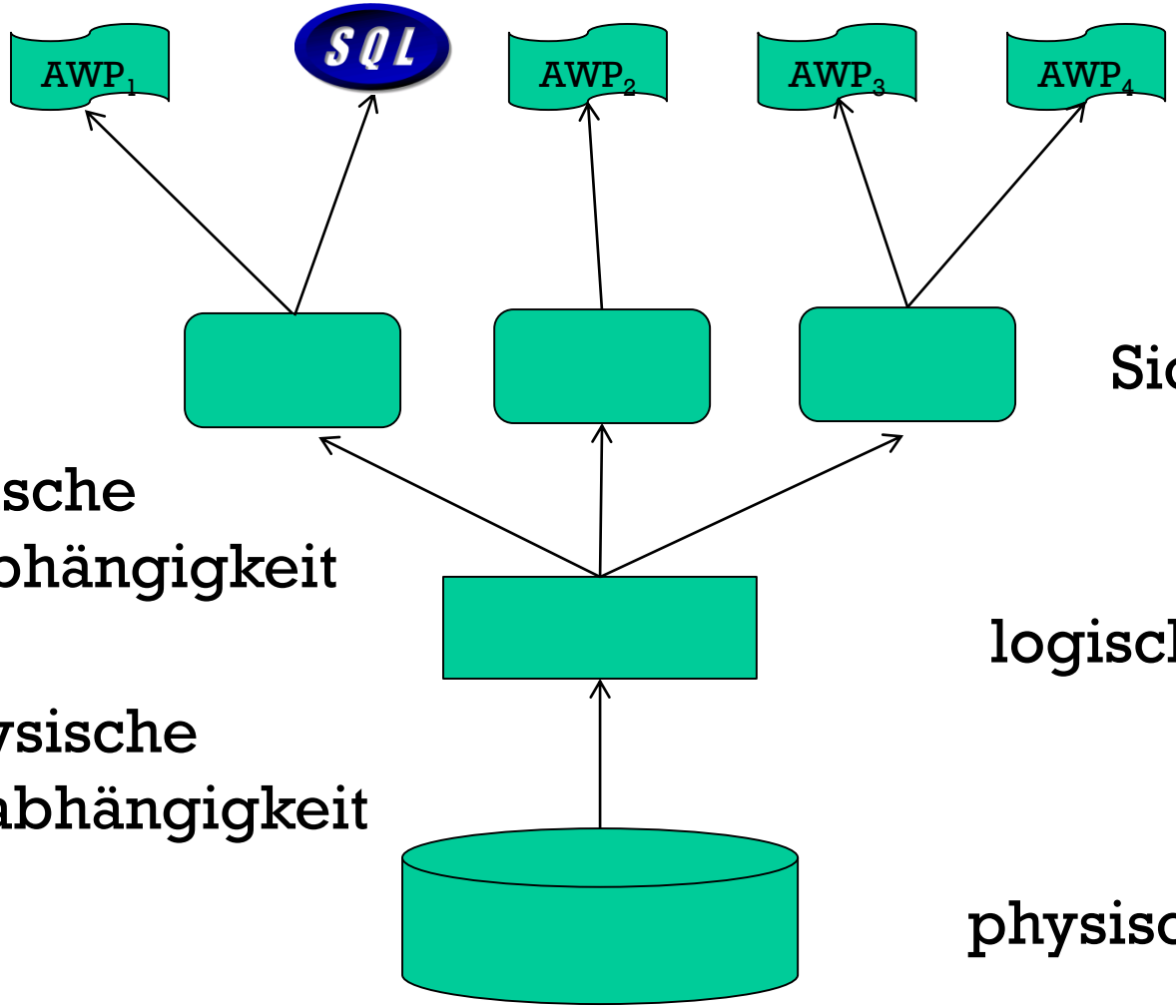
- DBS verfügt über mehrere Abstraktionsstufen
  - **Externe Ebenen** (= **Sichten**)
    - Beschränkung der Datenbank, der für ein Endbenutzer oder Endbenutzergruppe relevant ist.

    ➔ z. B. Datenschutz
  - **Logische Ebene**
    - Beschreibung aller Daten und deren Beziehungen in der Datenbank

    ➔ z. B. Gemeinsame Datenbasis
  - **Physische Ebene**
    - Festlegung der Speicherstrukturen

    ➔ z. B. Leistungsfähigkeit

Datenabstraktion



Sichten

logische Datenunabhängigkeit

logische Ebene

physische Datenunabhängigkeit

physische Ebene

Datenunabhängigkeit

- **Änderung einer Ebene beeinflusst nicht die darüber liegenden Ebenen.**
- **Logische Datenunabhängigkeit**
  - Änderungen der logischen Ebene haben keinen Einfluss auf die Sichten und damit nicht auf die AWPs.
    - Beispiel:

      Kundenkonto soll um das Attribut “Uhrzeit” erweitert werden.
- **Physische Datenunabhängigkeit**
  - Änderungen der physischen Ebene haben keinen Einfluss auf die logische Ebene und damit auch nicht auf die Sichten und die AWPs.
    - Beispiel:

      Anlegen eines Suchbaums, um schneller zu suchen.

Datenmodell

- **Ein Datenmodell bietet eine Infrastruktur zur**
  - Strukturbeschreibung von Daten
    - → Datendefinitionssprache (DDL)
  - Definition der Syntax und Semantik von Operationen.
    - → Datenmanipulationssprache (DML)
      - Einfügen, Ändern und Löschen von Daten
      - Suche nach Daten

- **Unterscheidung**
  - Datenbankschema
    - Menge aller Strukturbeschreibungen
  - Datenbankinstanz
    - Gültiger Zustand in der Datenbank

Logische Datenmodelle

- **DBS besitzen zumindest zwei Datenmodelle:**
  - physisches Datenmodell:
    zur speicherorientierten Repräsentation der Daten
  - logisches Datenmodell:
    zur benutzerorientierten Repräsentation der Daten
- **Logische Datenmodelle**
  - **relationales Datenmodell**
  - objektorientiertes Modell
  - XML
  - JSON
  - HDF
  - …

Beispiel (JSON Schema)

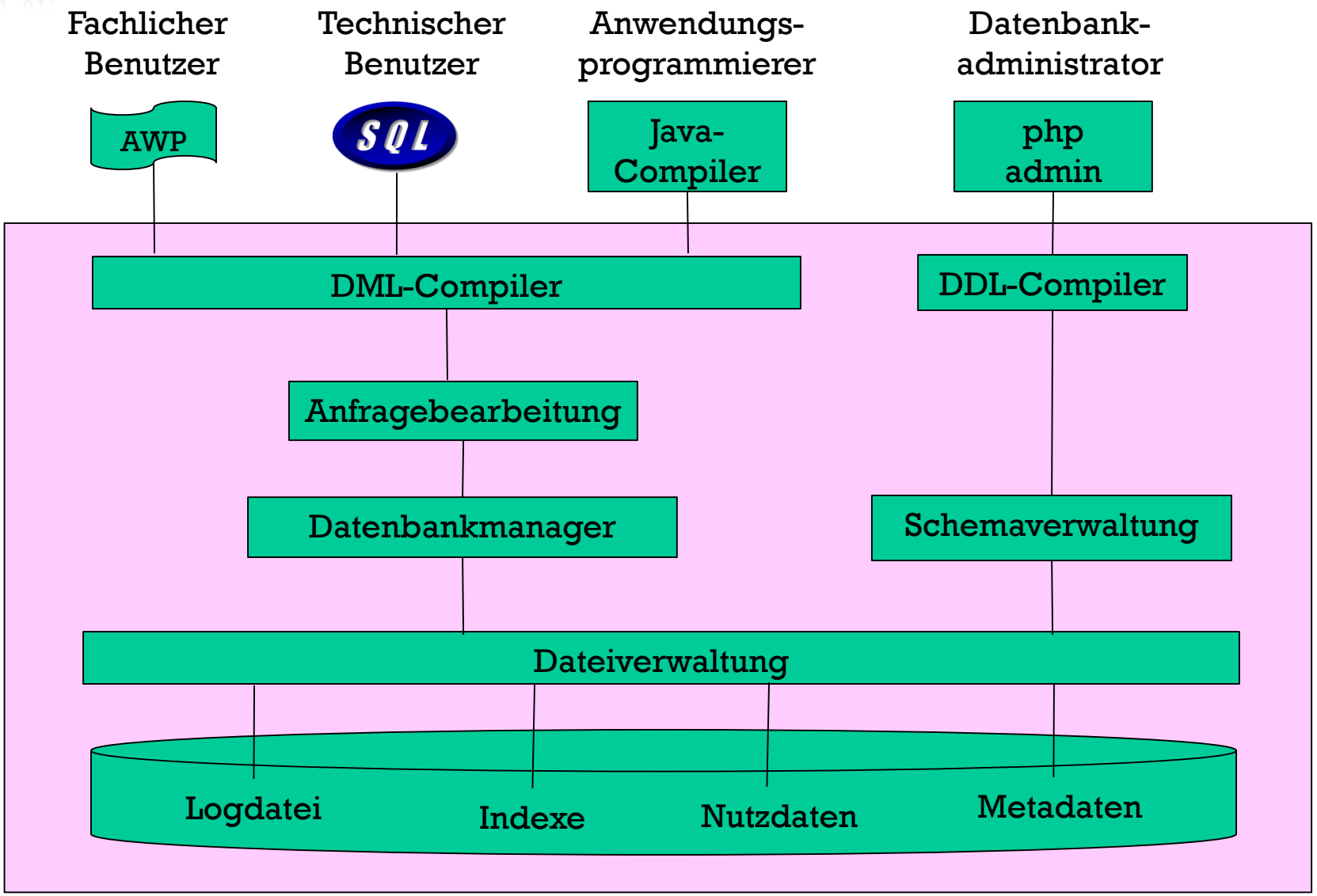
```
{    "name":"Product",
     "properties": {
          "id": { "type":"number",
                  "description":"Product identifier",
                  "required":true },
          "name": { "type":"string",
                  "description":"Name of the product",
                  "required":true },
          "price": { "type":"number",
                  "minimum":0,
                  "required":true },
          "tags": { "type":"array",
                  "items": { "type":"string" } }
     }
}
```

Typ der Eigenschaft

Name der Eigenschaft

Semantische Bedingungen

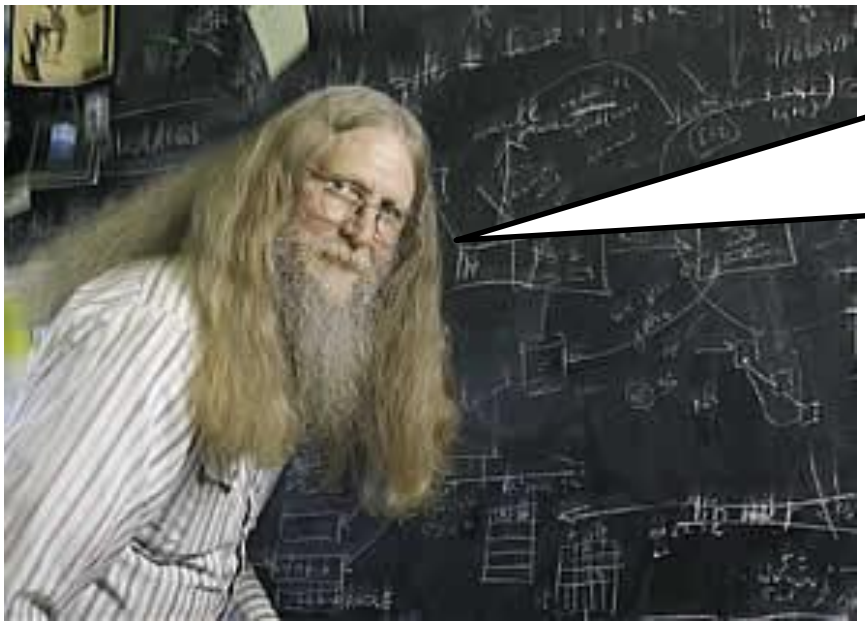
Grobarchitektur eines DBMS



Wichtiges zusammengefasst

- **Informelle Definition eines Datenbanksystems**
- **Gründe, weshalb Dateisysteme sich nicht immer für die Verwaltung von Daten eignen.**
- **Datenunabhängigkeit**
  - logische und physische
- **Datenmodell**
- **Performance**
  - Antwortzeit und Durchsatz

2. Das Relationale Datenmodell



Relational databases are the foundation of the western civilization.

Bruce Lindsay,
IBM Fellow @ IBM Almaden Research Center

Stimmt diese Behauptung?

- **Fast alle kommerziellen DBMS wie z. B.**
  - Oracle, SQL Server, IBM DB2, …

  und nicht-kommerzielle Systeme wie z. B.
  - mySQL, PostgreSQL, SQLite H2, …

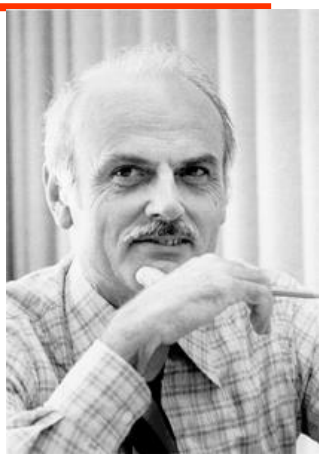
  sind relationale Datenbanksysteme und basieren auf dem relationalen Datenmodell.

- **Nahe zu alle wichtigen Daten unserer „zivilisierten“ Welt werden in relationalen DBMS verwaltet.**
  - Banken
  - Versicherungen
  - Unternehmens- und Wirtschaftsdaten

Auslöser für die Entwicklung relationaler DBMS

- **Mitte der sechziger Jahre**
  - IBM entwickelt für American Airlines (AA) das Reservierungssystem SABRE
    - Riesiges Prestigeprojekt
    - 200 Datenbankprogrammierer über 4 Jahre
    - Ständige Verzögerungen auf Grund immer wieder neuer Anforderungen (neue Datenfelder) durch AA
      - ➔ Teures Umschreiben und Anpassen der Datenbank mit einer halben Million Programmzeilen
- **IBM damaliges Datenbanksystem IMS hatte substantiell diese Probleme verursacht.**
  - Insbesondere deshalb hat sich dann IBM entschlossen ein grundlegend neues System zu entwickeln.

Grundlage der relationalen DBMS



- **Edgar Codd schrieb zwei Artikel (1969,1970) über das relationale Modell.**
  - E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Comm. of the ACM 13(6): 377-387(1970)*
- **Durch den Einsatz der neuen Technologie konnte das SABRE-Projekt erfolgreich umgesetzt werden.**
  - ➔ Heute würde man von einer „disruptive technology“ sprechen.
- **Codd erhielt für diese bahnbrechenden Arbeiten 1981 den Touring Award („Nobelpreis der Informatik“)**

* Gutachter haben empfohlen diesen Artikel nicht für eine Publikation anzunehmen.

Gründe für den Erfolg des relationalen Modells

"Make things as simple as possible, but not simpler."

- **Einfachheit**
  - Relation (entspricht einer Tabelle) als grundlegende Datenstruktur
  - Wenige, aber ausdrucksstarke Grundoperationen zur Verarbeitung von Relationen
    - klare Semantik
- **Mengenorientierte Verarbeitung der Daten**
- **Formale Grundlagen**
  - Datenmodell
  - Anfragebearbeitung

2.1 Relationen

- **Informelle Beschreibung**
  - Eine Relation ist eine Tabelle!
- **Beispiel:**
  - Tabelle **Studierende**

Tabellenname

Attribut

Tabellenschema

MatNr	Name	Wohnort
7	Bond	London
42	Adams	Cambridge
1527	Philipp	Marburg

Tupel

  - Implizit gehört zu jedem Attribut ein Wertebereich
    - Wertebereich von MatNr = ganze Zahlen
    - Wertebereich von Wohnort = String

Formale Definition: 1. Versuch

- Sei U eine nicht-leere Menge, das **Universum** der Attribute.
  - Zu jedem **Attribut** A ∈ U gibt es einen **Wertbereich** *dom(A)*.
  - Der Wertebereich dom(A) eines Attributs A ist endlich und besteht aus atomaren Elementen, die keine weitere Struktur besitzen.
    - Beispiele hierfür sind int oder String.

- Ein **Relationenschema** RS ist eine Liste von unterschiedlichen Attributen $(A_1,\ldots,A_k)$ aus dem Universum U.
  - Der Parameter k wird **Grad** oder **Stelligkeit** des Schemas genannt.
  - Der **Wertebereich** dom(RS) des Schema ergibt sich aus:

$$dom(RS) = dom(A_1) \times dom(A_2) \times \ldots \times dom(A_k)$$

- Zu einem Relationenschema RS bezeichnet r eine **Relation** falls $r \subseteq dom(RS)$.

- Für ein **Tupel** t einer Relation r gilt t ∈ r.

Was ist bei diesen Definitionen das grundlegende Problem?

Formale Definition: 2. Versuch

- Sei U eine nicht-leere Menge, das **Universum** der Attribute.
  - Zu jedem **Attribut** A ∈ U gibt es einen **Wertbereich** dom(A).
  - Der Wertebereich dom(A) eines Attributs A ist endlich und besteht aus atomaren Elementen, die keine weitere Struktur besitzen.
    - Beispiele hierfür sind int oder String.

- Ein **Relationenschema** RS ist eine Teilmenge des Universums U.
  - Die Anzahl der Elemente in RS wird **Grad** oder **Stelligkeit** des Schemas genannt.

- Zu einem Relationenschema RS ist die **Relation** r = r(RS) eine **endliche Menge von totalen Abbildungen t** mit .

$$t : RS \rightarrow \bigcup_{A \in RS} dom(A)$$

und t(A) ∈ dom(A) für alle A ∈ RS. t wird auch als **Tupel** bezeichnet.

Was ist der Vorteil dieser Definitionen?

Beispiel

- **Beispiel:** Tabelle **Studierende**

MatNr	Name	Wohnort
7	Bond	London
42	Adams	Cambridge
1527	Philipp	Marburg

  - **Attribute:** MatNr, Name, Wohnort
    - Wertebereiche
      - dom(MatNr) = Menge der ganzen Zahlen
  - **Relationenschema: {MatNr, Name, Wohnort}**
  - **Tupel t_1**
    - t_1(MatNr) = 7
    - t_1(Name) = "Bond"
    - t_1(Wohnort) = "London"

Gleichheit von zwei Relationen

- **Zwei Relationen r und s sind gleich, falls**
  - ihre Relationenschemata gleich sind und
  - ihre Mengen der totalen Abbildungen gleich sind.

- **Die Reihenfolge der Attribute hat keine Bedeutung!**

Weitere Begriffe

- **Relationen können über einen eindeutigen Namen angesprochen werden.**
  - Wenn dies nicht der Fall ist, sprechen wir von einer **temporären Relation**.
  - Eine Relation r hat somit zwei wichtige Bestandteile
    - Relationenschema RS_r
    - Relationenname $Name_r$

- Sei r eine Relation und $t \in r$ ein Tupel. Sei $X \subseteq RS_r$.
  - Dann bezeichnet **t[X]** das Tupel t eingeschränkt auf X.
  - Ist X = {A}, so schreiben wir kurz **t[A]** (statt t[{A}]).

Integritätsbedingungen

- Zu einem Schema RS können wir nun die Menge der möglichen Relationen betrachten

  REL(RS) = {r | r(RS)}

  - Wenn wir die Tabelle Studierende betrachten, sind weitere Einschränkungen dieser Menge sinnvoll.

    Warum?

- Das Attribut **MatNr** der Tabelle Studierende hat eine besondere Bedeutung.
  - MatNr ist eindeutig!

    ➔ Es gibt keine zwei unterschiedlichen Tupel mit einem gleichen Wert für das Attribut MatNr.
  - Diese Eigenschaft wird als **Integritätsbedingung** bezeichnet. Wir verkleinern damit die Menge der möglichen Relationen REL(RS).

Schlüssel

- Für ein Relationenschema RS wird $X \subseteq RS$ als **Schlüssel** bezeichnet, falls folgende Bedingungen für alle möglichen Relationen r(RS) gelten:
  - **Eindeutigkeit:**
    - Für zwei beliebige Tupel $t_1, t_2 \in r$ gilt:

$$t_1[X] = t_2[X] \Rightarrow t_1 = t_2$$

  - **Minimalität:**
    - Es gibt keine echte Teilmenge $Y \subset X$, so dass die Eindeutigkeit erfüllt ist.

- Es gibt mindestens einen Schlüssel in einem Relationenschema
  - Es kann aber auch mehrere Schlüssel geben.

Primärschlüssel

- Der **Primärschlüssel** ist ein ausgezeichneter Schlüssel des Relationenschemas.
  - Es gibt nur einen Primärschlüssel.
  - Der Primärschlüssel wird in einer Datenbank als Stellvertreter für Datensätze genutzt.
  - Die Attribute des Primärschlüssels werden im Schema durch Unterstreichen hervorgehoben.

Kennst Du Max Mustermann aus Musterhausen, der am Fachbereich 12 Mathe(Bachelor) studiert, derzeit Datenbanksysteme I hört, und im siebten Fachsemester ist?

Meinst Du den Studenten mit Matrikelnr. 124223?

Formale Definition

- Eine **(lokale) Integritätsbedingung** *lib* eines Relationenschemas RS ist eine Boolesche Funktion

  lib: REL(RS) → {true, false}

- Zu einem Relationenschema RS bezeichnet $\boldsymbol{LIB_{RS}}$ die **Menge aller lokalen Integritätsbedingungen**.

- Beispiel (Relationschema Studierende)
  - MatNr ist ein Schlüssel

- Definition (**Konsistente Relation**)
  - Eine Relation r ist in einem konsistenten Zustand, wenn alle dem Schema zugeordneten lokalen Integritätsbedingungen erfüllt sind.

Datenbank

- **Bisher haben wir nur eine einzige Relation/Tabelle betrachtet.**
- **Eine relationale Datenbank besteht aber i. A. aus sehr vielen Relationen.**
  - Beispiele
    - Universitätsinformationssystem
      - Relationen: Studierende, Vorlesung, Belegung
    - Online-Shop
      - Relationen: Kunde, Produkt, Bestellung
    - ERP (Enterprise Resource Management)
      - Relationen: Personal, Abteilung, Maschine,

Beispiel (ERP-Datenbank)

PMZuteilung

pnr	mnr	Note
67	84	3
67	93	2
67	101	3
73	84	5
114	93	5
114	101	3
51	93	2
69	101	2
333	84	3
701	84	2
701	101	2
82	101	2

Personal

pnr	PName	VName	Abt	Lohn
67	Meier	Helmut	B10	65000
73	Müller	Margot	B10	51000
114	Bayer	Martin	A63	60000
51	Daum	Birgit	A64	72000
69	Störmer	Willi	A64	60000
333	Haar	Hans	A63	75000
701	Reiner	Willi	A64	42500
82	Just	Michael	A64	65000

Maschine

mnr	MName
84	Presse
93	Füllanlage
101	Säge

ALeitung

abtnr	pnr
B10	67
A63	333
A64	333

Abteilung

abtnr	AName
B10	Spielzeug
A63	Computer
A64	Suppen

ERP-Datenbankschema

- **Relation Personal**
  - Angestellte in einem Unternehmen
  - Schema: 5 Attribute, Primärschlüssel: pnr
- **Relation Maschine**
  - Maschinen in einem Unternehmen
  - Schema: 2 Attribute, Primärschlüssel: mnr
- **Relation PMZuteilung**
  - Welche Angestellte können welche Maschine, wie gut bedienen?
  - Schema: 3 Attribute, Primärschlüssel: {mnr, pnr}
- **…**

Fremdschlüssel

Beobachtung

- Relation PMZuteilung besitzt Attribute, die in einer anderen Relationen Primärschlüssel sind.
  - Diese Attribute werden als **Fremdschlüssel** bezeichnet.

Fremdschlüsselbedingung

- Jeder Wert eines Fremdschlüssel muss auch in der Relation vorhanden sein, in der das Attribut Primärschlüssel ist.

Formale Definition (Datenbank)

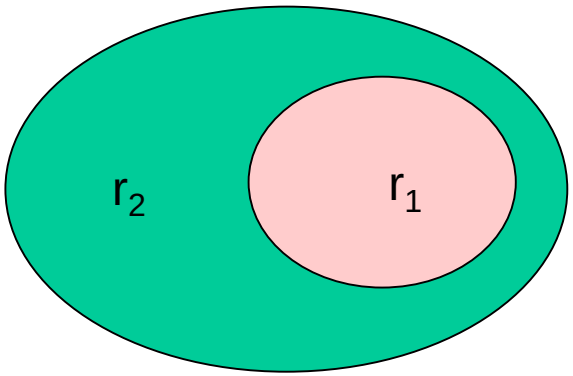
- Ein **Datenbankschema** DS = {RS_1, …, RS_m} besteht aus einer endlichen Menge von Relationenschemata.
  - Jedem Schema RS_j ist eine Menge von lokalen Integritätsbedindungen $LIB_j = LIB_{RS_j}$ zugeordnet.

- Zu einem Datenbankschema DS ist d = {r_1, …, r_m} eine **Datenbank**, falls $r_i \in REL(RS_i)$.
  - DB(DS) bezeichnet die **Menge aller Datenbanken** über dem Schema DS.

Formale Definition (Fremdschlüssel)

- Sei DS ein Datenbankschema und $RS_1, RS_2 \in DS$ zwei Relationenschemata, wobei X Primärschlüssel von RS_2 ist. Dann ist $Y \subseteq RS_1$ ein **Fremdschlüssel**, wenn für alle Relationen $r_1 \in REL(RS_1)$ und $r_2 \in REL(RS_2)$ gilt:

$$\{t[Y] \mid t \in r_1\} \subseteq \{t[X] \mid t \in r_2\}$$



Formale Definition

- Eine globale **Integritätsbedingung** *ib* eines Datenbankschemas DS ist eine Boolesche Funktion

  ib: DB(DS) → {true, false}

- Zu einem Datenbankschema DS bezeichnet $\boldsymbol{IB_{DS}}$ die **Menge aller Integritätsbedingungen**.

- Beispiel (Relationschema PMZuteilung)
  - mnr ist ein Fremdschlüssel

Definition (**Konsistente Datenbank**)

- Eine Datenbank d ist in einem **konsistenten Zustand**, wenn alle dem Schema zugeordneten Integritätsbedingungen erfüllt sind.

2.2 Die relationale Algebra

- Motivation
  - Definition einer Sprache für die Verarbeitung von Relationen → **SQL**
    - Sprache wird genutzt, um Daten
      1. aus Tabelle(n) zu lesen,
      2. diese zu verarbeiten,
      3. und das Ergebnis wieder in Form einer (temporären) Tabelle zur Verfügung zu stellen.

PMZuteilung

pnr	mnr	Note
67	84	3
67	93	2
67	101	3
73	84	5
…	…	…

Eingabe

SQL

select pnr
from **PMZuteilung**
where Note = 5

Ausgabe

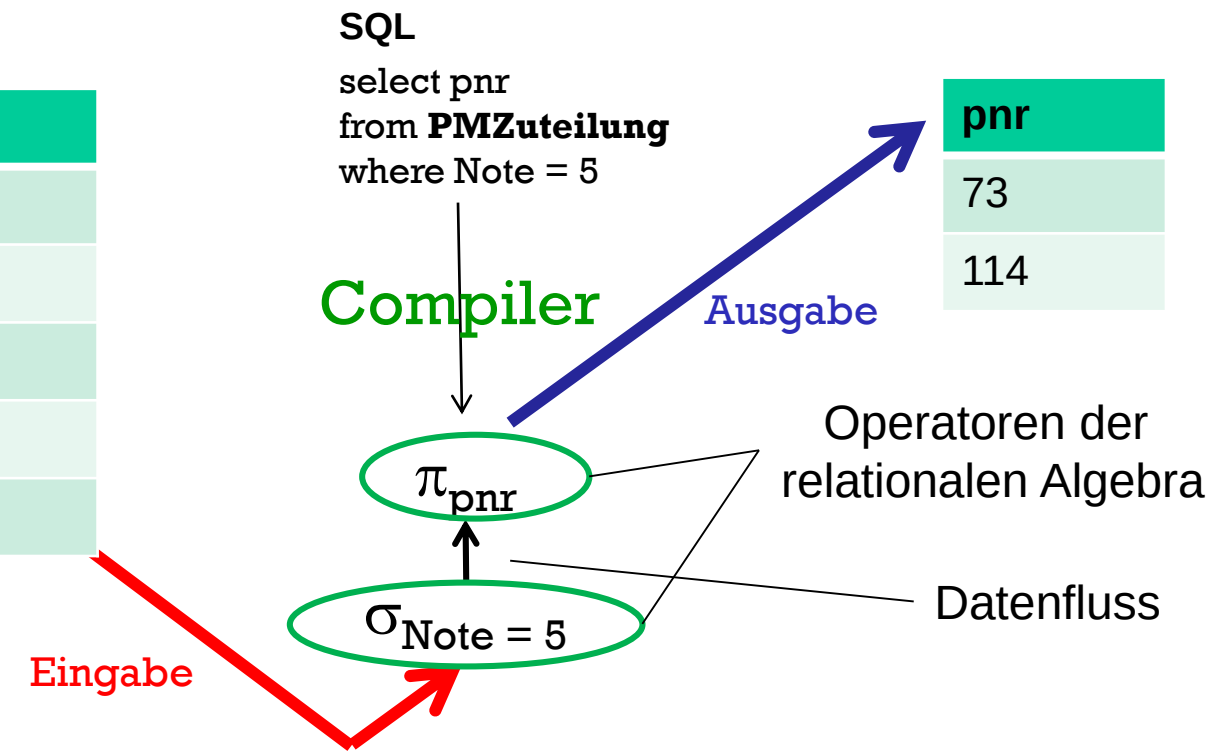
pnr
73
114

SQL und relationale Algebra

- **Übersetzung einer SQL-Anfrage**
  - Ausdruck in SQL-Ausdruck wird übersetzt in einen Ausdruck einer **Algebra von Operatoren**.
    - Jeder Operator hat als Eingabe eine Tabelle und erzeugt als Ausgabe ebenfalls eine Tabelle.

PMZuteilung

pnr	mnr	Note
67	84	3
67	93	2
67	101	3
73	84	5
…	…	…



Anforderungen an die Algebra

- **Ausdrucksstärke**
  - Was ist mit SQL bzw. der relationalen Algebra alles berechenbar?

    → siehe Theoretische Informatik: Turing-berechenbar
- **Effizienz**
  - Können die Operatoren der Algebra effizient implementiert werden?
    - In welcher Komplexitätsklasse liegen die Operatoren?

      $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$

    → siehe Praktische Informatik II
- **Einfachheit**
  - Was ist die minimale Anzahl von Operatoren für die gewünschte Ausdrucksstärke?

Relationale Algebra

- Algebra
  - Gegeben eine Menge N (“Anker der Algebra”)
  - Menge von Operatoren OP = {op_1, …,op_n} der Form

    op_j: $N^k \rightarrow N$
  - Beispiel (Boolesche Algebra)
    - N = {true, false}
    - OP = {$\neg$, $\wedge$, $\vee$, 0, 1} mit $\wedge$, $\vee$: N x N → N und $\neg$: N → N

- Definition (Relationale Algebra)
  - Anker ist die Menge aller Relationen
  - Menge von 6 Operatoren: σ, π, $\cup$, $\times$, -, ρ
    - Diese Bezeichnungen sind historisch bedingt und werden heute noch in allen Büchern zu Datenbanken genutzt.

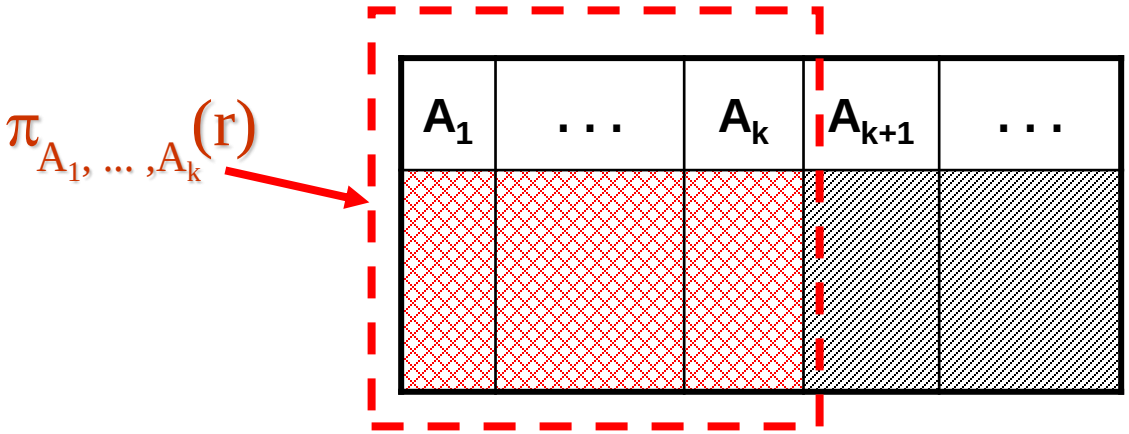
2.2.1 Basisoperatoren

Die Relationale Algebra besitzt 6 Basisoperatoren:

- **Projektion**
- **Selektion**
- **Umbenennung**
- **Vereinigung**
- **Differenz**
- **Kartesisches Produkt**

Projektion π

- Eingabe
  - Eine Relation und ein oder mehrere Attribute
- Ausgabe
  - Filtern von Spalten aus einer Relation
    - Alle nicht genannten Attribute werden eliminiert.



Formale Definition der Projektion

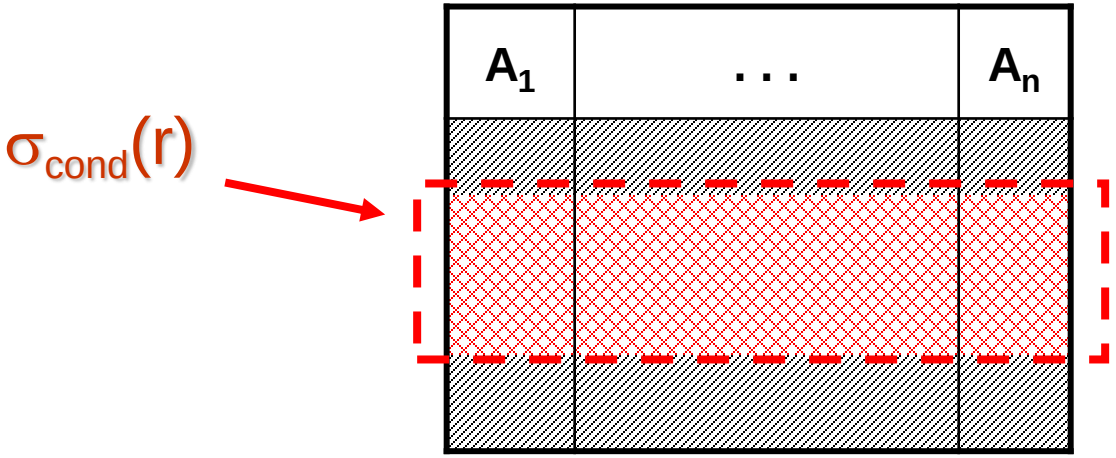
- Sei r $\in$ REL(RS) eine Relation und X $\subseteq$ RS. Die Ausgabe von $\pi_X(r)$ ist eine (temporäre) Relation s mit
  - $RS_s = X$
  - $s = \{t[X] \mid t \in r\}$

- Kurzschreibweise: **$s = \pi_X(r)$**

- **Man beachte die Mengensemantik!**
  - **Mögliche Duplikate in der Ausgaberelation werden eliminiert.**

Selektion σ

- **Eingabe**
  - Eine Relation r und eine Bedingung B
- **Ausgabe**
  - Eine Relation mit allen Zeilen von r, die B erfüllen.



Formale Definition der Selektion

- **Sei r ∈ REL(RS) eine Relation und F: RS → {true, false} eine Boolesche Funktion. Die Ausgabe von $\sigma_F(r)$ ist eine Relation s mit**
  - $RS_s = RS$
  - $s = \{t \mid F(t) \text{ und } t \in r\}$

- **Kurzschreibweise: $s = \sigma_F(r)$**

- **Boolesche Funktion F besteht aus Attributen von RS, Konstanten, Vergleichsoperatoren (=, ≠ , < , ≤ , > , ≥) und Booleschen Operatoren (∧ , ∨ , ¬).**
  - Lohn > 50000
  - Lohn > 50000 ∧ VName = "Willi" !

Umbenennung ρ

- **Eingabe**
  - Relation r und ein Attribut A aus dem Schema und ein Attribut B, das nicht im Schema von r ist.
- **Ausgabe**
  - Eine Relation mit exakt den gleichen Datensätzen wie r, aber statt dem Attribut A mit dem Attribut B.

$\rho_{B_k \leftarrow A_k}(r)$

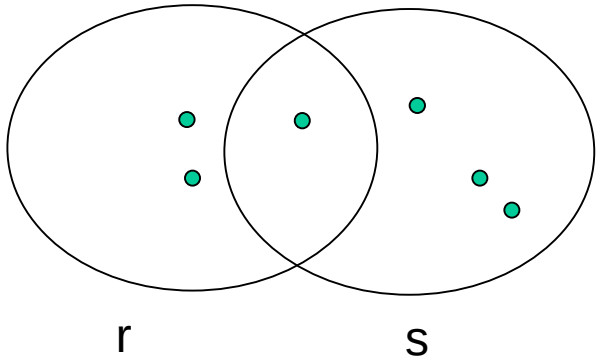
A_1	…	A_{k-1}	B_k	A_{k+1}	…	A_n

Formale Definition (Umbennenung)

- **Umbenennung eines Attributs**
  - Sei $r \in REL(RS)$ eine Relation und sei $A \in RS$ und $X \notin RS$ mit $dom(X) = dom(A)$. Die Ausgabe von $\rho_{X \leftarrow A}(r)$ ist dann eine (temporäre) Relation s mit
    - $RS_s = RS \setminus \{A\} \cup \{X\}$
    - $s = \{t \mid t \in r\}$
- **Umbenennung von zwei (und mehreren) Attributen**
  - $\rho_{X \leftarrow A,\, Y \leftarrow B}(r)$
- **Zusätzlich kann auch die Umbenennung einer Relation definiert werden.**
  - Sei $r \in REL(RS)$ eine Relation und *Name* ein eindeutiger Bezeichner. Dann erzeugt $s= \rho_{Name}(r)$ eine neue Relation s mit $Name_s$ = *Name*.

Vereinigung ∪

- **Eingabe**
  - Zwei Relationen r und s mit gleichem Schema ($RS_r = RS_s$).
- **Ausgabe**
  - Eine Relation mit allen Datensätzen aus r und s.



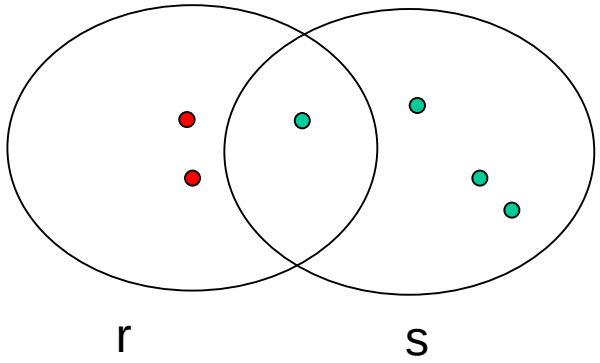
Formale Definition (Vereinigung)

- Seien $r_1, r_2 \in REL(RS)$ zwei Relationen über dem gleichen Schema RS. Die Ausgabe von $r_1 \cup r_2$ ist dann eine (temporäre) Relation s mit
  - $REL_s = RS$
  - $s = \{t \mid t \in r_1 \text{ oder } t \in r_2\}$

- Man beachte dabei, dass die Mengensemantik gilt.
  - Ein Tupel, das in beiden Eingaberelationen vorkommt, wird in der Ausgaberelation nur einmal vorkommen.

Differenz -

- **Eingabe**
  - Zwei Relationen r und s mit gleichem Schema.
- **Ausgabe**
  - Eine Relation mit allen Datensätzen aus r die nicht in s sind.



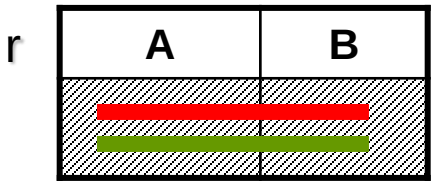
Formale Definition (Differenz)

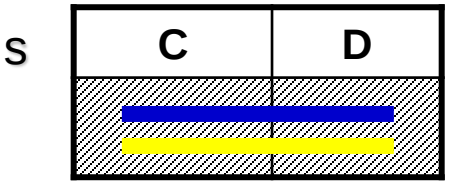
- Seien $r_1, r_2 \in REL(RS)$ zwei Relationen über dem gleichen Schema RS. Die Ausgabe von $r_1 - r_2$ ist dann eine (temporäre) Relation s mit
  - $REL_s = RS$
  - $s = \{t \mid t \in r_1 \text{ und } t \notin r_2\}$

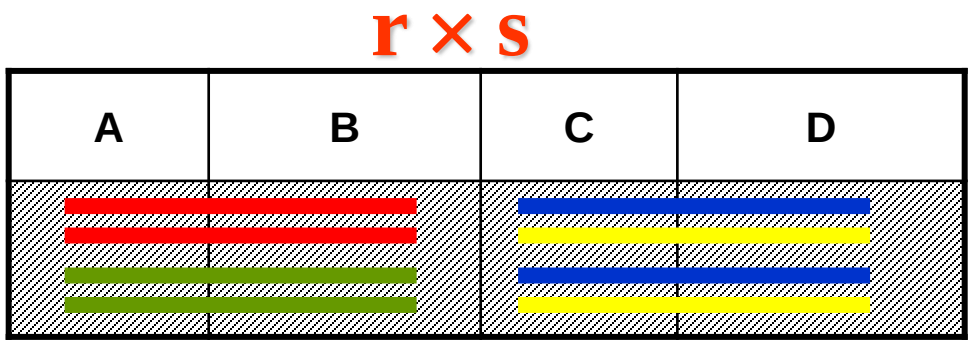
- Mengensemantik!

Kartesisches Produkt ×

- **Eingabe**
  - Zwei Relationen r und s mit disjunkten Schemata.
- **Ausgabe**
  - Eine Relation, in der jedes Tupel aus r mit jedem Tupel aus s verknüpft ist.







Formale Definition (Kartesisches Produkt)

- Seien $r_1 \in REL(RS_1)$, $r_2 \in REL(RS_2)$ zwei Relationen mit $RS_1 \cap RS_2 = \emptyset$. Die Ausgabe von r_1 x r_2 ist dann eine (temporäre) Relation s mit
  - $REL_s = RS_1 \cup RS_2$
  - $s = \{t \mid t[RS_1] \in r_1 \text{ und } t[RS_2] \in r_2\}$

- **Anmerkungen**
  - Bei Gleichheit von Attributen in r_1 und r_2 kann man über Umbenennung dafür sorgen, dass die Schemata disjunkt werden.

Beispiele für Anfragen (1)

Datenbankschema siehe Folie 49

- *Bestimme alle Angestellte und deren Lohn, die mehr als 60000 verdienen.*

Beispiele für Anfragen (2)

Datenbankschema siehe Folie 49

In welcher Abteilung arbeiten die Angestellten mit Nachnamen Müller?

Beispiele für Anfragen (3)

Datenbankschema siehe Folie 49

- *Finde die Namen aller Angestellten aus der Abteilung Computer.*

Beispiele für Anfragen (4)

Datenbankschema siehe Folie 49

- *Bestimme die Angestellten mit gleichem Vornamen.*

Beispiele für Anfragen (5)

Datenbankschema siehe Folie 49

- *Finde alle Angestellten, die nur an der Maschine mit Nummer 84 ausgebildet sind.*

Implementierung einer Algebra als Klasse

- **Definition einer Klasse Table mit folgenden Methoden:**

Relationale Algebra	Methoden
Filter	Table filter (?)
Projektion	Table select (?)
Umbenennung	Table as (?)
Vereinigung	Table union (Table right)
Differenz	Table minus (Table right)
Kartesisches Produkt	Table cross (Table right)

  - Bei den ersten drei Operatoren müssen die Eingabeparameter noch angegeben werden.
  - Es fehlen noch Methoden, um Table-Objekte zu erstellen.

Verfeinerung des Entwurfs

Relationale Algebra	Methoden
Filter	Table filter(String pred)
Projektion	Table select (String plist)
Umbenennung	Table as (String list)
Vereinigung	Table union(Table right)
Differenz	Table minus(Table right)
Kartesisches Produkt	Table cross(Table right)
Initialisierung	Table scan(String relName)

- **Die Parameter für die Methoden filter, select und as werden als Zeichenkette übergeben.**
  - Hierfür müssen wir noch überlegen, wie die Zeichenkette interpretiert werden soll.
- **Zusätzlich gibt es eine Methode scan, um eine Relation aus der Datenbank zu lesen.**

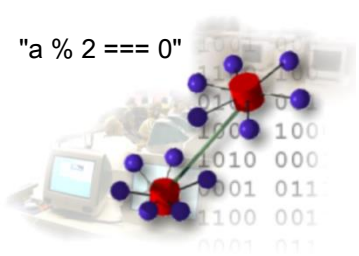
Umbenennung

- **Anwendung**
  - Schritt für Schritt:
    - Table t = new Table();
      t = t.scan("Maschinen");
      t = t.as("m, name");
  - In einem Ausdruck:
    - Table t = (new Table()).scan(("Maschinen").as ("m, name");
- **Wichtige Eigenschaften**
  - Jedes Attribut des Table-Objekts bekommt einen neuen Namen.
    - Die Namen werden in eine durch Komma separierte Liste angegeben.
    - Jeder Name in der Liste ist eindeutig.

Projektion

- **Anwendung**
  - Schritt für Schritt:
    Table t = new Table();
    t = t.scan("Personal");
    t = t.select("pnr, Lohn");
  - In einem Befehl:
    Table t = (new Table()).scan(("Personal").select("pnr, Lohn");

- **Wichtige Eigenschaften**
  - Es werden Attribute desTable-Objekts ausgewählt.
  - Die Namen müssen in dem Table-Objekt wirklich vorhanden sein.



Filter

- **Anwendung**
  - Schritt für Schritt:
    Table t = new Table();
    t = t.scan("Personal");
    t = filter("Lohn === 60000");
  - In einem Befehl:
    Table t = (new Table()).scan(("Personal").filter("Lohn === 60000");
- **Vereinfachende Annahmen**
  - Für unsere Vorlesung beschränken wir uns zunächst auf Gleichheitsprädikate mit einem Attribut und einem Wert.
    - Wir verwenden === für den Test auf Gleichheit.
    - Der Wert und der Typ des Attributs müssen zusammenpassen.

Apache Flink

- **Diese Funktionalität existiert bereits in dem Open-Source Framework Apache Flink.**
  - Flink bietet sowohl Funktionalität für die Verarbeitung von Datenströmen als auch Tabellen.
  - Für die Verarbeitung von Tabellen gibt es die Table-API.

- **Diese API bietet noch viel mehr an Funktionalität.**
  - Prädikate lassen sich noch viel flexibler angeben als wir es bisher in der Vorlesung kennengelernt haben.
    - Auf einige dieser Möglichkeiten werden wir in der Vorlesung noch eingehen.
  - Es gibt weitere Operationen, die wir z. T. noch in der Vorlesung vorstellen werden.

2.2.2 Abgeleitete Operatoren

- **Motivation**
  - Definition von höherwertigen Operatoren, um Anfragen einfacher zu formulieren.
- **Operatoren**
  - Verbundoperationen
    - Theta-Verbund (theta join)
    - Natürlicher Verbund (natural join)
    - Semi-Verbund (semi join)
    - Anti-Join (anti join)
  - Schnitt
  - Division
    - Allquantifizierte Anfragen

Schnitt

- **Eingabe**
  - Zwei Relation r_1 und r_2 mit gleichem Schema.
- **Ausgabe**
  - Ergebnisrelation mit Tupeln, die in r_1 **und** in r_2 vorkommen.

- **Formale Definition (Schnitt)**
  - $r_1 \cap r_2 := r_1 - (r_1 - r_2)$

- **Wir können also das Ergebnis auf die bisherigen Operationen zurückführen.**
  - Eine Angabe des Relationenschemas ist nicht erforderlich.

Theta Join

- **Informell**
  - Verknüpfung von zwei Relationen bezgl. einem Prädikat θ, dass sich auf Attribute der Relation r_1 und r_2 bezieht.

- **Beispiel**
  - Gegeben eine Menge von POI (Point of Interests) mit Attributen X und Y. Bestimme die Paare von Punkten, die höchstens 100m voneinander entfernt sind.

- **Formale Definition**
  - Seien $r_1 \in REL(RS_1)$, $r_2 \in REL(RS_2)$ mit $RS_1 \cap RS_2 = \emptyset$.

$$r_1 \bowtie_\theta r_2 := \sigma_\theta(r_1 \times r_2).$$

Theta Join - Eigenschaften

Beispiel für Theta Join:

r_1

A	D
1	a
3	b
2	a

$\bowtie_{A \leq C \wedge B > 0}$

r_2

B	C
5	2
0	1
1	1

$=$

A	D	B	C
1	a	5	2
1	a	1	1
2	a	5	2

- Joinbedingungen Θ sind aufgebaut wie Selektionsbedingungen.
- Theta Join und natürlicher Verbund sind kommutativ und assoziativ, d.h. die Klammerungsreihenfolge bei Mehrfach-Joins ist im Prinzip unwesentlich.

Umsetzung in Flink

- **In Flink gibt es zusätzlich die Methode**
  - Table join(Table right)
- **Dieser Methode muss noch ein Aufruf der Methode where mit der Joinbedingung folgen.**
  - Es werden nur Bedingungen mit Gleichheit unterstützt.

- **Beispiel**
  - Table left = new Table().scan(Rel1, "a, b, c");
  - Table right = new Table().scan(Rel2, "d, e, f");

Table result = left.join(right).where("a = d");