

Inhaltsverzeichnis

1.	Einführung.....	3
1.1	Begriffe	3
2.	Datenbanksysteme und Datenbankanwendungen	4
2.1	Datenbankanwendungen: Komponenten und Strukturmodelle..	4
2.1.1	Komponente einer Datenbankanwendungen	4
2.1.2	Strukturmodelle für Datenbankanwendungen	4
2.2	Funktionen von DBMS	6
2.3	Middleware	7
2.4	Organisationsebenen in einem DBMS	7
2.5	Logische Datenmodelle.....	8
2.5.1	Hierarchisches Modell.....	8
2.5.2	Netzwerkmodell.....	9
2.5.3	Relationales Modell.....	9
2.5.4	Entity-Relationship-Modell und -Diagramme	9
2.5.5	Objektorientiertes Modell, NOSQL.....	10
2.5.6	NOSQL: Key-Value Modell	11
3.	Das relationale Datenbankmodell	12
4.	Datenbanknormalisierung.....	13
4.1	Warum „normalisiert“ man Tabellen?	13
4.2	1. Normalform	13
4.3	2. Normalform	13
4.4	3. Normalform	13
4.5	Beispielaufgabe.....	13
4.5.1	Hat die Tabelle die erste Normalform? Wenn nicht, bringen Sie sie in die 1NF.	14
4.5.2	Zeichnen Sie ein Abhängigkeitsdiagramm, das Primärschlüssel sowie funktionale, vollständig funktionale und transitive Abhängigkeiten vom Primärschlüssel enthält.....	14
4.5.3	Hat die Tabelle die zweite Normalform? Wenn nicht, bringen Sie sie in die 2NF.	14
4.5.4	Hat (bzw. haben) die Tabelle(n) jetzt die dritte Normalform? Wenn nicht, bringen Sie sie in die 3NF.....	14
5.	Die Relationale Algebra	16

5.1	Darstellung von Tabellen als Relationen.....	16
5.2	Anfragen auf Relationen - Grundoperatoren	16
5.2.1	Die Selektion σ	16
5.2.2	Die Projektion	17
5.2.3	Die Komposition von relationalen Operatoren	17
5.2.4	Die Vereinigung \cup	18
5.2.5	Die (Mengen-)Differenz.....	18
5.2.6	Das kartesische Produkt \times	19
5.2.7	Die Umbenennung ρ	19
5.3	Anfragen auf Relationen – Zusätzliche Operatoren.....	20
5.3.1	Die Schnittmenge \cap	20
5.3.2	Verbund-Operatoren (Join-Operatoren)	21
5.3.3	Aggregatsfunktionen.....	23
5.3.4	Aggregatsfunktionen und Gruppierung.....	23

1. Einführung

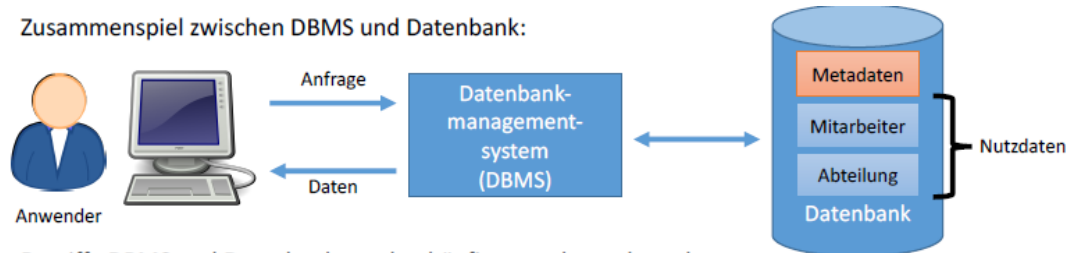
1.1 Begriffe

- **Datenbankmanagementsystem (DBMS)**

= übernimmt die Verwaltung der Daten in einer Datenbank

- Datenorganisation
- Zugriffsregelung

Zusammenspiel zwischen DBMS und Datenbank:



Begriffe DBMS und Datenbank werden häufig unsauber gebraucht

- Z.B. ist Oracle keine Datenbank, sondern ein DBMS, das Datenbanken verwaltet

- **Arten von Anomalien**

Änderungs-Anomalie (Update-Anomalie)

- Daten werden nicht an jeder Stelle geändert, an der sie gespeichert werden (Bsp. siehe oben)

Einfüge-Anomalie (Insert-Anomalie)

- Daten können nicht erfasst werden, ohne andere Daten gleichzeitig erfassen zu müssen

Beispiel?

Lösch-Anomalie (Delete-Anomalie)

- Löschen bestimmter Daten löscht auch andere Daten, die eigentlich erhalten bleiben sollten

- **Ablage von Daten – Dateisystem**

= Hierarchisch organisiert (Schränk für Rechnungen, Schriftverkehr, etc.)

- **Flache Datei**

= Besitzt nur ein Minimum an Struktur (csv-Datei – Jeder Datensatz in neuer Zeile, keine Informationen über Bedeutung der Felder, Datentypen)

- **Problem**

= Gute Entscheidungen basieren auf guten, zeitnah vorliegenden Informationen. Die Datenablage im Dateisystem und in flachen Dateien erfüllt diese Anforderungen nicht!

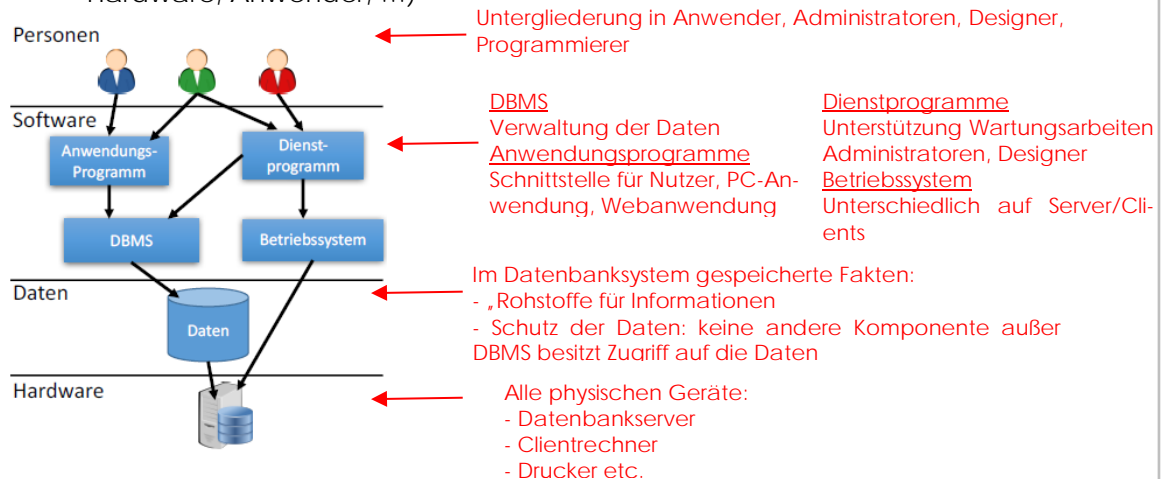
➔ Datenbanksysteme helfen, Daten und Zusammenhänge zu verwalten

2. Datenbanksysteme und Datenbankanwendungen

2.1 Datenbankanwendungen: Komponenten und Strukturmodelle

2.1.1 Komponente einer Datenbankanwendungen

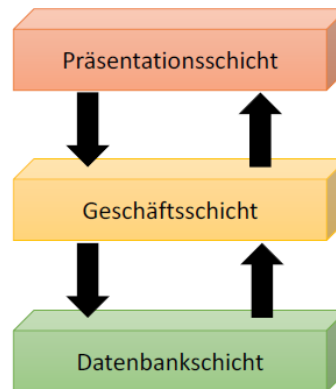
- Datenbankanwendung (Datenbanksystem oder Anwendungssystem mit Datenbank)
 - Komplettes, zur Datenverarbeitung genutztes System
 - Besteht aus DBMS und zahlreichen anderen Komponenten (weiter Software, Hardware, Anwender, ...)



2.1.2 Strukturmodelle für Datenbankanwendungen

Datenbankanwendung kann in drei logische Schichten aufgeteilt werden

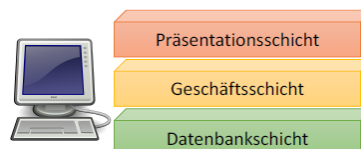
- **Präsentationsschicht**
 - Darstellung und Eingabe von Daten
 - Z.B. Benutzer-Interfaces, Berichte
- **Geschäftsschicht**
 - Auch als **Business-Logik** bezeichnet
 - Regeln für Datenbank, z.B. Überprüfung von Wertebereichen, Gültigkeiten
- **Datenbankschicht**
 - Speicherung, Suche, Integrität
 - Wird von DBMS vollständig implementiert



- Einsichtige Datenbankanwendungen

Alle Schichten auf einem Computer realisiert

- Direkter Zugriff auf Datendateien auf lokaler Festplatte
- Häufig verwendet mit Desktop-Datenbanken



- Zweischichtige Datenbank Anwendungen

Client-Server-Architektur

- Zentraler **Server**
- Zugriff von **Clients** über Netzwerk
- Server koordiniert Zugriffe mehrerer Clients und aktualisiert Daten

Ausprägungen nach Verortung der Geschäftsschicht:

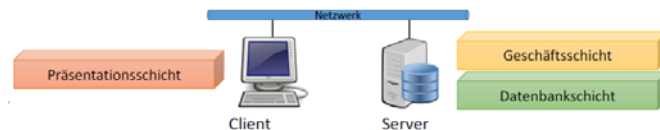
- **Intelligenter Server / Thin Client**
- **Intelligenter Client / Fat Client**

Intelligenter Server / Thin Client



Server implementiert die Geschäftsschicht

- Umfangreiche Berechnungen direkt auf dem Server möglich
 - Nachteil: Server kann zum Engpass werden
- Daten müssen für jeden Verarbeitungsschritt zwischen Client und Server übertragen werden



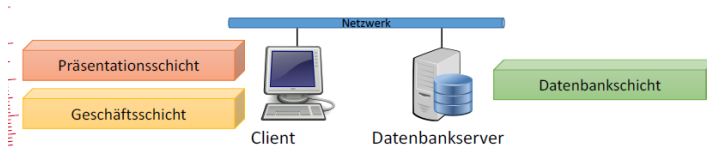
Intelligenter Client / Fat Client



Client implementiert die Geschäftsschicht

- Umfangreiche Berechnungen direkt auf Client
- Ggf. Konsistenzprobleme

In Praxis meist keine strikte Trennung der Modelle



- N-Schichtige Datenbank Anwendung

Schichten (meist drei) strikt getrennt auf verschiedenen Rechnern ausgeführt

- Client: Präsentationsschicht
- Anwendungsserver: Geschäftsschicht
- Datenbankserver: Datenbankschicht

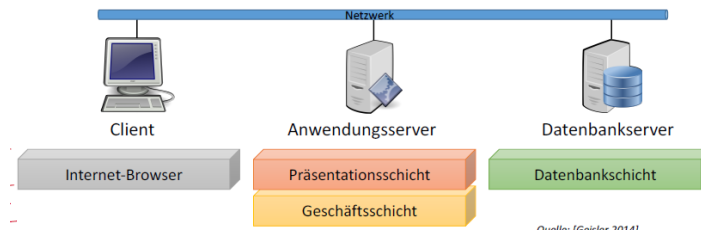


Internet-Datenbankanwendung



Spezialfall der n-schichtigen Datenbankanwendung

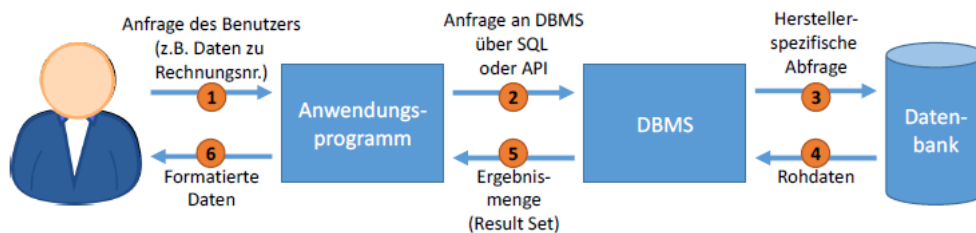
- Lediglich Internet-Browser läuft auf Client
- Keine Software-Installation notwendig



2.2 Funktionen von DBMS

• Datenbankzugriff

- Datenmanipulationssprachen (DML) ermöglicht einfachen Weg für Zugriff auf gespeicherte Daten
 - Als Standardsprache hat sich ANSI-Standard SQL („Structured Query Language“) etabliert
 - Alle Kommunikation mit DBMS findet i.d.R. mit SQL statt
 - Einige Hersteller bieten auch zusätzlich Application Programming Interfaces (APIs)



• Datenumwandlung/Präsentation

- Trennt zwischen Eingabe- und Speicherformat von Daten
- Z.B. wird Datum logisch als TT.MM.JJJJ eingegeben, aber physisch als Julianisches Datum (in Tagen seit dem 1. Januar -4712 12:00 Uhr) gespeichert
- Trennung von logischen und physischen Datentypen gewährleistet Datenunabhängigkeit
 - Anwendungsprogramm nur von logischen Daten abhängig

• Wahrung der Datenintegrität

- Definition von Regeln, die Datenintegrität gewährleisten (z.B. referentielle Integrität durch Prüfung von Verweisen zwischen Daten)
- Regeln werden in Katalog (Data Dictionary) gespeichert, in dem das DBMS Metainformationen ablegt

• Metadatenverwaltung

- Beschreibung der gespeicherten Daten durch Metadaten
- I.d.R. im Katalog (Data Dictionary), auch in Tabellen gespeichert
- Können gelesen, aber nicht direkt sondern nur über spezielle Funktionen verändert werden

• Bereitstellung von Kommunikationswegen

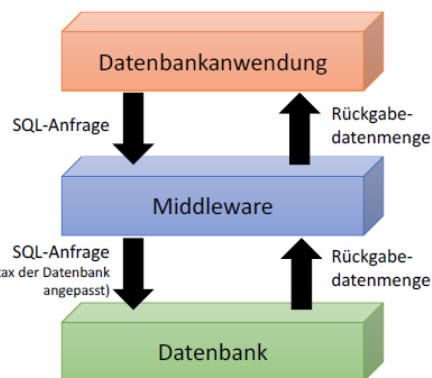
- Möglichkeiten für Client-Rechner zur Kommunikation mit dem Datenbank-Server

- Häufig Kommunikationsprotokolle, die auf Netzwerk-Protokolle (z.B. TCP/IP) aufsetzen
- Middleware für Abstraktion des Anwendungszugriff auf Datenbank (Unabhängigkeit von speziellem Datenbankmanagementsystem, s.u.)
- **Datenträgerverwaltung**
 - Effiziente und performante Speicherung von Daten durch Aufbau von logischen Strukturen
 - Anwender muss sich nicht um Datenablage kümmern
- **Verwaltung der Sicherheit**
 - Beschränkung des Zugriffs auf Datenobjekte für bestimmte Benutzergruppen (z.B. für eine Tabelle nur Lesen vs. Einfügen, Verändern, Löschen)
- **Verwaltung mehrerer Anwender**
 - Verwaltung von konkurrierenden Zugriffen auf die selben Datensätze
 - Vermeidung von Race-Conditions („Wer zuletzt speichert, gewinnt“) durch sperren von Datensätzen, z.B. beim ersten Zugriff
- **Datensicherung/Wiederherstellung**
 - Mögliche Gefahren sind Hardware-Ausfälle und Benutzerfehler (z.B. unbeabsichtigtes löschen)
 - Unterstützt durch Dienstprogramme für Administratoren

2.3 Middleware

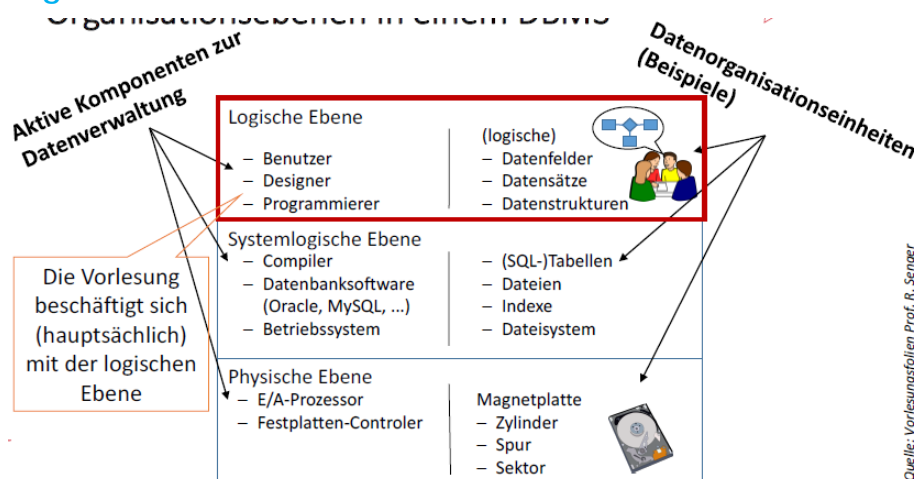
Zwischenschicht für Kopplung zwischen Anwendungsprogramm und Datenbank

- Datenbank- und DBMS-unabhängige Schnittstelle für Programmierung
- Änderungen (anderes DBMS, anderer Speicherort für Daten, ...) für Programmierer transparent (ggf. an Syntax der Datenbank angepasst)
- Beispiele:
 - Open Database Connectivity (ODBC)
 - Java Database Connectivity (JDBC)
 - ...



Quelle: [Geisler 2014]

2.4 Organisationsebenen in einem DBMS

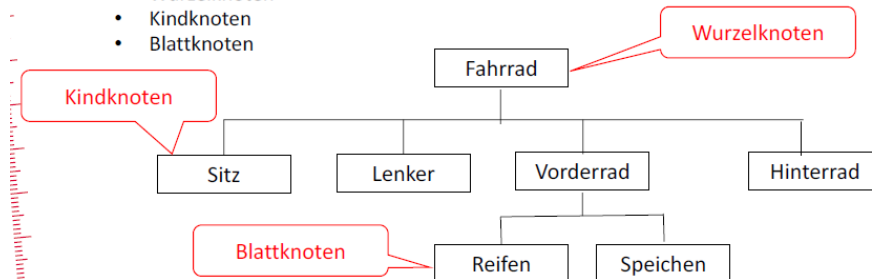


Quelle: Vorlesungsfolien Prof. Dr. Senoer

2.5 Logische Datenmodelle

2.5.1 Hierarchisches Modell

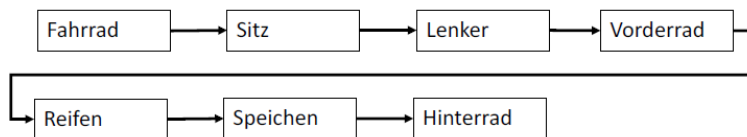
- Idee aus Apollo-Programm: Größere Fertigungseinheiten durch kleinere aufbauen
- Strukturiert hierarchisch mit unterschiedlichen Knoten
 - Wurzelknoten
 - Kindknoten
 - Blattknoten



Wie kann hier der Reifen gefunden werden?

Speicherung einer hierarchischen Struktur nur sequenziell möglich, da der Computer einen linearen Arbeitsraum besitzt

- Traversieren eines Baumes in Pre-Order: zuerst links dann rechts, wenn es nicht weiter geht dann wird eine Stufe höher gesprungen
- Baum in einer linearen Form dargestellt:



Vor-/Nachteile hierarchisches Modell



Vorteile

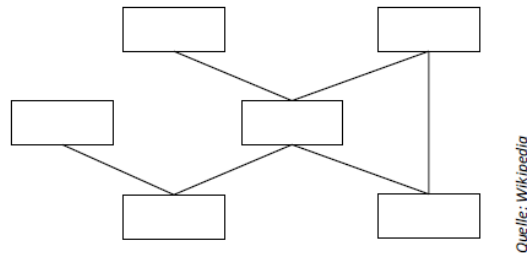
- Datenintegrität wird erzwungen – jedes Kind hat genau einen Elternknoten
- Eignet sich gut für 1:N Beziehungen
 - Bsp.: Ein Fahrrad besteht aus vier Teilen (Sitz, Lenker, Vorderrad, Hinterrad)
- Einfache Suche, wenn Struktur bekannt (man muss durch die Daten navigieren)

Nachteile

- Keine strukturelle Unabhängigkeit → Struktur darf sich nicht verändern
- Ineffiziente Suche, wenn Struktur unbekannt (Traversieren des Baumes)
- Begünstigt Delete-Anomalie bei löschen eines Unterbaums
- Keine Darstellung von N:M Beziehungen
- Komplexe Implementierung
- Fehlende Standards

2.5.2 Netzwerkmodell

- Adressiert Nachteile des hierarchischen Modells. (N-zu-M Beziehung nicht darstellbar)
- Ähnlich zu hierarchischen DB – Unterschied ein Kindknoten kann mehrere Elternknoten haben



- Hat die Vorteile des hierarchischen Modells übernommen
- Integrität wird verbessert
- Nachteil: einige Schwächen der hierarchischen DB (auch hier muss man durch die Daten navigieren, ...)

2.5.3 Relationales Modell

1970 von E.F. Codd vorgestellt

- Basiert auf dem mathematischen Konstrukt **Relation**
- Damals wegen der fehlenden Leistung der Computer als nicht umsetzbar eingestuft

Relationale DBMS (RDBMS) stellt Relation als **Tabellen** dar

- Matrix, die aus verschiedenen Zeilen/Spalten besteht

Beziehungen zwischen Relationen / Tabellen über **Schlüssel**

- Tabellen enthalten **Primärschlüssel** (Personalnummer, Abteilungsnummer)
- Verweis auf andere Tabelle durch Verwendung von Primärschlüssel als **Fremdschlüssel**
- Beispiel: Beziehung zwischen PERSON und ABTEILUNG.

Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

Personalnr	Nachname	Vorname	Funktion	Abteilung
1	Geldberg	Günter	Geschäftsführer	1
2	Buchhalter	Benno	Abteilungsleiter	4
3	Fehler	Fritz	Programmierer	3
4	Hacker	Hans	Chefprogrammierer	3
5	Nieda	Frieda	Chefsekretärin	1
6	Gründlich	Gerda	Sachbearbeiterin	4
7	Klugscher	Karl	Azubi	2
8	Lueger	Ludwig	Abteilungsleiter	2
801	Tutnix	Toni	Praktikant	3
...

2.5.4 Entity-Relationship-Modell und -Diagramme

Entität (Entity)

- Kann fast alles sein: eine Person, ein Ort, ein Ding, ein Event – alles worüber Daten gesammelt werden können
- Stellt ein Objekt in der realen Welt dar
- Bsp.: KUNDEN, PRODUKTE oder abstraktere Objekte wie ROUTEN

Attribut (Attribute)

- Charakterisiert eine Entität
- Beispiel: Ein Produkt hat einen Namen, Preis, etc.
- Attribute werden als Datenfelder umgesetzt

Beziehung (Relationship)

Beschreibt eine Assoziation verschiedener Entitäten

- Bsp.: Die Beziehung zwischen einem Buch und der Bibliothek

Beziehungstypen:

1-zu-N Beziehung

- Bsp.: Ein AUTOR schreibt mehrere GEDICHTe, wobei ein GEDICHT immer nur von einem AUTOR geschrieben wird.

N-zu-M Beziehung

- Bsp.: Ein STUDENT kann mehrere MODULen belegen, wobei ein MODUL von mehreren STUDENTen besucht werden kann

1-zu-1 Beziehung

- Bsp.: Eine PERSON hat einen einzigen PERSONALAUSWEIS, wobei ein PERSONALAUSWEIS immer nur von einer PERSON besessen wird.

- Eine Beziehung ist bidirektional

1-zu-N Beziehung:



N-zu-M Beziehung:



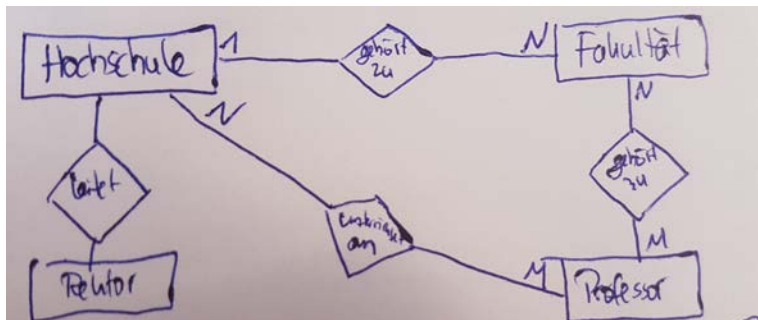
1-zu-1 Beziehung:



- Ziel: Grafische Darstellung für Übersicht, Reduktion der Komplexität
- Stellt Entitäten (Entities) und Ihre Beziehungen (Relationships) dar
- Wurde 1976 von Peter Chen eingeführt
- Folgende Beziehungen werden verwendet: 1-zu-1, 1-zu-N, M-zu-N
- Verschiedene Notationen, z.B:
 - Chen (Beispiel rechts)
 - Crow's Foot (Krähenfuß),
 - UML (Unified Modeling Language)
- Details in Kapitel 9

Beispiel:

Modellieren Sie die Hochschule Karlsruhe mit Ihren Fakultäten und Ihren Professoren als ER-Diagramm



2.5.5 Objektorientiertes Modell, NOSQL

Anwendungsprogramm wird häufig in objektorientierte Programmiersprache geschrieben

- Nutzt Objekte mit Eigenschaften, Vererbung, Methoden

Relationale Datenbank arbeitet im Hintergrund

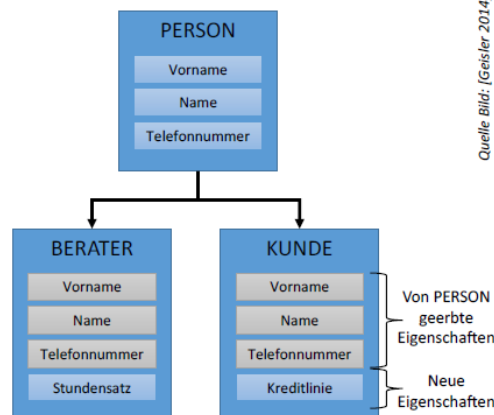
- Basiert Tabellen, Felder, Beziehungen, ...

Tabellen werden objektorientiert abgebildet

- Viel Code notwendig
- Methoden existieren nur in der objektorientierten Welt

Object Oriented Data Modelling (OODM)
– Daten und Beziehungen sind in einer Struktur – dem Objekt – enthalten.

- Unterschied zwischen Entität und Objekt: Entität enthält keine Info zu Beziehungen; Ein Objekt enthält Beziehungen zwischen seinen Daten
- Komponenten des OODM:
 - **Objekt** – Abstraktion einer Entität aus der realen Welt. Wird manchmal als äquivalent zu einer Entität angesehen
 - **Attribute** sind Eigenschaften des Objektes
 - Objekte mit gleichen Eigenschaften sind in **Klassen** gruppiert
 - **Methoden** sind Teil der Klasse und werden zum bearbeiten der Daten eines Objektes benötigt
 - **Vererbung und Generalisierung**



Quelle Bild: [Geisler 2014]

2.5.6 NOSQL: Key-Value Modell

NOSQL-Datenbanken adressieren Anforderungen von Big Data

- Verarbeiten von großen nicht-strukturierten Daten
- Nutzen kein relationales Modell
- Unterstützen verteilte Architekturen
- Hohe Skalierbarkeit
- Hohe Verfügbarkeit
- Fehlertoleranz
- Nachteil: Es gibt kein Standard, nur viele Ansätze

Struktur besteht aus einem **Schlüssel (key)** und einem **Wert (value)**

- Schlüssel kann beliebig sein (ggf. mit Einschränkungen auf Text-Zeichen)
- Wert kann beliebig sein (ggf. mit Größenbeschränkungen): Text, Bild, Video, ...

Color	Red
Age	18
Size	Large
Name	Smith
Title	The Brown Dog

- Jeder Zeileneintrag kann als ein Attribut einer Entität mit seinem Wert aufgefasst werden

```

user1923_color Red
user1923_age 18
user3371_color Blue
user4344_color Brackish
user1923_height 6' 0"
user3371_age 34
  
```

- Im relationalen Modell wird beim Hinzufügen eines Attributes Tabelle geändert, hier wird einfach Attribut als Zeile hinzugefügt
- Es werden kein Relationen zwischen Entitäten gespeichert! Dies ist die Aufgabe des Programmierers
- Indexieren und Suchen ist schwierig, wenn nicht nach Key gesucht wird

3. Das relationale Datenbankmodell

Relationale Datenbanken speichern Daten in Tabellen

- Tabellenstruktur repräsentiert Entitätstypen
- Tabellenspalten repräsentieren Attribute mit Datentyp und Domäne
- Tabellenzeilen repräsentieren Entitäten

4. Datenbanknormalisierung

4.1 Warum „normalisiert“ man Tabellen?

Relationale Datenbanken enthalten häufig Redundanzen

- Gleiche Information wird mehrfach gespeichert
- Folgen:
 - Erhöhter Speicherplatzbedarf
 - Gefahr von Inkonsistenzen, wenn Daten nur teilweise oder unvollständig geändert werden (Anomalien)

Die Normalisierung ist ein Vorgehen zur Vermeidung von Redundanzen

- Ergebnis ist eine Menge von Tabellen in Normalform(en)
- Gleichen Semantik wie Ursprungstabelle(n)
- Keine vermeidbaren Redundanzen
- Die Menge der Tabellen nimmt durch den Normalisierungsprozess zu

4.2 1. Normalform

Eine Tabelle R ist in 1. Normalform (1NF), wenn die Wertebereiche aller Attribute von R atomar sind.

Atomar bedeutet:

- Nur einfache, unstrukturierte Attribute sind erlaubt
- Listenartige, mengenwertige oder aufzählungsartige Attribute sind nicht erlaubt
- Erlaubte Datentypen: integer, real, string (CHAR(), VARCHAR()), enum
- Nicht erlaubte Datentypen: array, record, list

4.3 2. Normalform

Ein Tabelle R ist in 2. Normalform (2NF), wenn

- R in 1. Normalform ist und
- jedes Nichtschlüsselattribut vollständig funktional von jedem Schlüssel abhängt (und nicht nur von einem Teil des Schlüssels)

4.4 3. Normalform

Ein Tabelle R ist in 3. Normalform (3NF), wenn

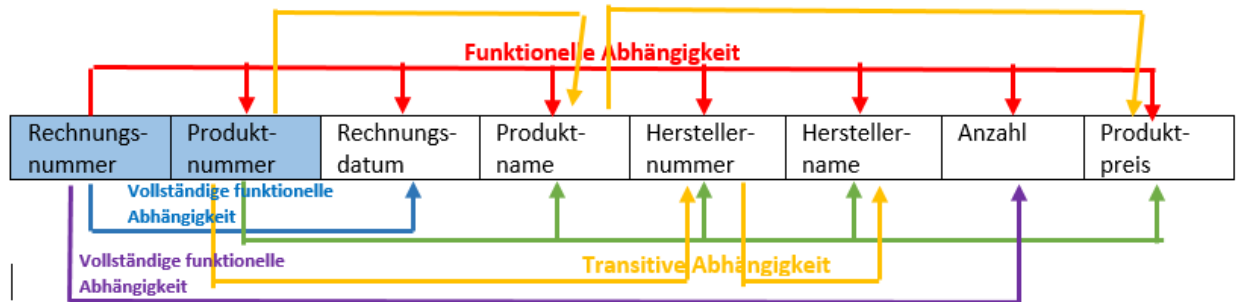
- R in 2. Normalform ist (und somit auch in 1. Normalform) und
- wenn kein Nichtschlüsselattribut transitiv über ein anderes Nichtschlüsselattribut von einem Schlüsselkandidaten abhängt

4.5 Beispielaufgabe

Rechnungsnummer	Produktnummer	Rechnungsdatum	Produktname	Herstellernummer	Herstellername	Anzahl	Produktpreis
1203	56493	11.01.2017	Tisch	321	ikea	1	299,00
1203	56283	11.01.2017	Stuhl	321	ikea	4	87,00
1204	56123	12.01.2017	Sofa	310	BoConcept	1	1.320,00
1204	56012	12.01.2017	Kissen	301	Joop	3	99,99
1204	56283	12.01.2017	Stuhl	321	ikea	2	87,00

4.5.1 Hat die Tabelle die erste Normalform? Wenn nicht, bringen Sie sie in die 1NF.

Die Tabelle ist in der 1. NF, da alle Attribute von der Tabelle atomar sind. Ausnahme wäre das Rechnungsdatum, da hier 3 Daten in einer Spalte zusammengefasst werden. Hier könnte man eine Unterteilung in „Tag“, „Monat“ und „Jahr“ vornehmen.

4.5.2 Zeichnen Sie ein Abhängigkeitsdiagramm, das Primärschlüssel sowie funktionale, vollständig funktionale und transitive Abhängigkeiten vom Primärschlüssel enthält.**4.5.3 Hat die Tabelle die zweite Normalform? Wenn nicht, bringen Sie sie in die 2NF.**

Das Attribut „Rechnungsdatum“ hängt vollständig funktional vom Teil-Schlüssel „Rechnungsnummer“ ab.

Das Attribut „Anzahl“ hängt vollständig funktional vom Teil-Schlüssel „Rechnungsnummer“ ab.

Lösung: Attribute, die von Teilschlüssel abhängen in eigene Tabelle auslagern.

Tabelle Rechnung	
Rechnungsnummer	Rechnungsdatum
1203	11.01.2017

Tabelle Produkt		
Rechnungsnummer	Produktnummer	Anzahl
1203	56493	1
1203	56283	4

Tabelle Produkt				
Produktnummer	Produktname	Herstellernummer	Herstellername	Produktpreis
56493	Tisch	321	Ikea	299,00
56283	Stuhl	321	Ikea	87,00

4.5.4 Hat (bzw. haben) die Tabelle(n) jetzt die dritte Normalform? Wenn nicht, bringen Sie sie in die 3NF.

Das Attribut „Herstellername“ hängt funktional vom Teil-Schlüssel „Herstellernummer“ ab. Und dieses wiederum hängt vom Attribut „Produktnummer“ ab.

Tabelle Rechnung	
Rechnungsnummer	Rechnungsdatum
1203	11.01.2017

Tabelle Produkt		
Rechnungsnummer	Produktnummer	Anzahl
1203	56493	1
1203	56283	4

Tabelle Produkt explizit		
Produktnummer	Produktname	Herstellernummer
56493	Tisch	321
56283	Stuhl	321

Tabelle Produkt - Hersteller	
Herstellernummer	Herstellername
321	Ikea
321	Ikea

Tabelle Produkt - Preis	
Produktname	Produktpreis
Tisch	299,00
Stuhl	87,00

5. Die Relationale Algebra

5.1 Darstellung von Tabellen als Relationen

- Darstellung als Tabelle



- Bsp. als Relation

Attribut i	Wertebereich Mi
Abteilungsnummer	{1, 2, 3, 4}
Abteilungsname	{'Geschäftsführung', 'Vertrieb', 'Programmierung', 'Verwaltung'}
Abteilungsleiter	{1, 2, 4, 8}

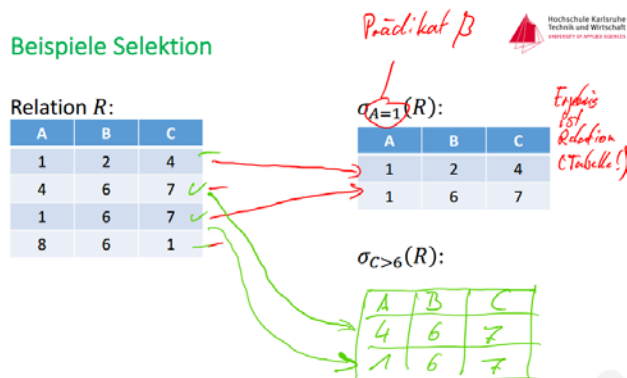
5.2 Anfragen auf Relationen - Grundoperatoren

Es gibt verschiedene Operatoren mit denen Anfragen auf Relationen gestellt werden können.

Unäre Operatoren (=Eingabe ist eine Relation)	Binäre Operatoren (=Eingabe sind zwei Relationen)
- Selektion	- Vereinigung ($E_1 \cup E_2$)
- Projektion	- Komplement
- Umbenennung	- Kartesisches Produkt ($E_1 \times E_2$)
- Zuweisung	- Schnittmenge
	- Verbund (Join)

5.2.1 Die Selektion σ

- Anschaulich
Die Menge aller Tupel aus R , die das Prädikat β erfüllen
- Beispiel



- Beispiel SQL
 - Allgemein:
 $\sigma_{\beta}(R)$ lässt sich in SQL darstellen als

SELECT * FROM R WHERE β ;

- o Spezifisch:

Beispiel $\sigma_{C>6}(R)$:

*SELECT * FROM R WHERE C > 6;*

- Anmerkung zum Prädikat
 - o Kann Vergleiche mit $=, \neq, <, \leq, >, \geq$ enthalten
 - o Verkettung von Vergleichen mit \wedge, \vee, \neg möglich
 - o Auch Vergleiche zwischen Attributen möglich

5.2.2 Die Projektion

- Anschaulich
Die Menge aller Tupel aus R, beschränkt auf Attribute aus γ
- Beispiel

Relation R:

A	B	C
1	2	3
4	5	6
1	3	8

$\pi_{A,B}(R)$:

A	B
1	2
4	5
1	3

$\pi_A(R)$:

A
1
4

'1' kommt nur 1x vor!

Achtung: Relationen sind Mengen von Tupeln, d.h. es gibt keine doppelten Tupel!

→ anders als in SQL!

- Beispiel SQL
 - o Allgemein:
 $\pi_{\gamma}(R)$ lässt sich in SQL darstellen als
SELECT γ FROM R;
 - o Spezifisch:
Beispiel $\pi_{A,B}(R)$:
SELECT A,B FROM R;

5.2.3 Die Komposition von relationalen Operatoren

- Anschaulich
Die Verwendung einer Selektion und Projektion
- Beispiel

$\pi_{Name}(\sigma_{Abteilung='Physik'}(Dozent))$

ID	Name	Abteilung	Gehalt
...	Gold	'Physik'	...
...	Einstein	'Physik'	...

Name
Gold
Einstein

- Beispiel SQL

Beispiel $\pi_{Name}(\sigma_{Abteilung='Physik'}(Dozent))$:

*SELECT Name FROM Dozent
WHERE Abteilung = 'Physik';*

5.2.4 Die Vereinigung \cup

- Anschaulich
Die Menge aller Tupel, die in R oder in S (oder in beiden) sind
- Beispiel

Relation R :

A	B	C
1	2	3
4	5	6

Relation S :

A	B	C
7	8	9
4	5	6

$R \cup S$:

A	B	C
1	2	3
4	5	6
7	8	9

Achtung: Keine Doppelten, weil Relationen Mengen sind!

- Beispiel Aufgabenformulierung
= KursID aller Kurse, die im Herbst 2009 oder Frühling 2010 stattfanden.
- Beispiel in SQL

- o Allgemein:

$R \cup S$ lässt sich in SQL darstellen als

SELECT * FROM R UNION SELECT * FROM S;

- o Spezifisch:

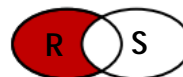
Beispiel Übung Folie 21:

SELECT KursID FROM Kurs
WHERE Jahr = '2009' AND Semester = 'Herbst'
Union
SELECT KursID FROM Kurs
WHERE Jahr = '2010' AND Semester = 'Frühling';

10. Praktische Arbeit: SQL und Datenbanken

5.2.5 Die (Mengen-)Differenz

- Anschaulich
Die Menge aller Tupel, die in R und nicht in S sind



- Beispiel

Relation R :

A	B	C
1	2	3
4	5	6

Relation S :

A	B	C
1	8	9
4	5	6

← Nicht
Sum!

Es geht immer um den gesamten Tupel, dieser muss komplett übereinstimmen (siehe Bsp. rot)

$R - S$:

A	B	C
1	2	3

A	B	C
1	2	3

- Beispiel Aufgabenformulierung
= KursID aller Kurse, aber nicht die, die im Frühling 2010 stattfanden.
- Beispiel SQL

- o Allgemein:

$R - S$ lässt sich in SQL darstellen als

SELECT * FROM R EXCEPT SELECT * FROM S;

bzw. in Oracle als

SELECT * FROM R (MINUS) SELECT * FROM S;

- o Spezifisch:

Beispiel Übung Folie 21:
`SELECT KursID FROM Kurs
 WHERE Jahr='2009' AND Semester='Herbst'
 MINUS SELECT KursID FROM Kurs
 WHERE Jahr='2010' AND Semester='Frühling';`

VL Datenbanken und Informationssysteme 1 Sommersemester 2017 5 - Relationale Algebra

5.2.6 Das kartesische Produkt x

- Anschaulich
 - o Die Menge aller möglichen Kombinationen von je einem Tupel aus R mit je einem Tupel aus S
 - o Im Allgemeinen $n*m$ Tupel!

- Beispiel

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S:

E	F	G
1	2	3
7	8	9

$R \times S$:

A	B	C	D	E	F	G
1	2	3	4	1	2	3
1	2	3	4	7	8	9
4	5	6	7	1	2	3
4	5	6	7	7	8	9
7	8	9	0	1	2	3
7	8	9	0	7	8	9

$\pi_{Name, KursID} (6_{Dozent.ID = Unterrichts.ID} (Unterricht \times Dozent))$

- Beispiel Aufgabenformulierung
 = Namen aller Dozenten zusammen mit den KursIDs der Kurse, die sie jeweils unterrichten
- Beispiel SQL

- o Allgemein:

$R \cup S$ lässt sich in SQL darstellen als

`SELECT * FROM R, S;`

- o Spezifisch:

Beispiel Übung Folie 32:

`SELECT Name, KursID FROM Unterrichts, Dozent
 WHERE Unterrichts.ID = Dozent.ID;`

- Anmerkung zu Attributnamen beim kartesischen Produkt
 = Streng genommen müssen die Namen der Attribute (Spaltenbezeichnungen) in R und S verschieden sein (Berechnung von $R \times S$)
 ➔ Lösung: Voranstellen der Relationsnamen

5.2.7 Die Umbenennung p

- Anschaulich
Umbenennung der Relation
 - $\rho_S(R)$ ist die Umbenennung der Relation R in S
- Beispiel

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

 $\rho_S(R)$:

Relation S:

A	B	C	D
1	2	3	4

 $\rho_{S(E,F,G,D)}(R)$:

Relation S:

E	F	G	D
1	2	3	4
4	5	6	7
7	8	9	0

 VL Datenbanken und Informationssysteme 1
Sommersemester 2017

5 - Relationale Algebra

• Beispiel SQL

- o Allgemein:

$\rho_{S(B_1, B_2, \dots, B_n)}(R)$ lässt sich in SQL darstellen als

```
SELECT A1 AS B1, A2 AS B2, ..., An AS Bn
FROM R AS S;
```

Bzw. in Oracle

```
SELECT A1 AS B1, A2 AS B2, ..., An AS Bn
FROM R S;
```

- o Spezifisch:

Beispiel $\rho_{S(E,F,G,D)}(R)$:

```
SELECT A AS E, B AS F, C AS G, D
FROM R S;
```

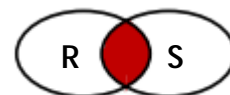
5.3 Anfragen auf Relationen – Zusätzliche Operatoren

Mit den bisher gelernten Grundoperatoren lässt sich jede Anfrage der relationalen Algebra ausdrücken. Manchen Anfragen sind aber etwas „sperrig“. Zusätzliche Operatoren vereinfachen die Darstellung von Anfragen.

5.3.1 Die Schnittmenge \cap

- Anschaulich

Die Menge aller Tupel, die in R und in S sind



- Beispiel

Relation R:

A	B	C
1	2	3
4	5	6

Relation S:

A	B	C
7	8	9
4	5	6

 $R \cap S$:

A	B	C
4	5	6

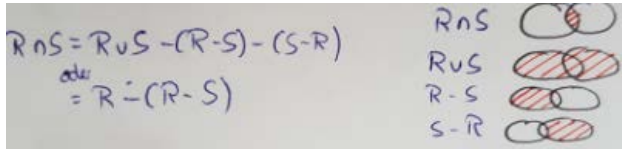
- Beispiel Aufgabenformulierung
= KursID aller Kurse, die im Herbst 2009 und Frühling 2010 stattfanden
- Beispiel SQL
 - o Allgemein:
 $R \cap S$ lässt sich in SQL darstellen als

```
SELECT * FROM R INTERSECT SELECT * FROM S;
```
 - o Spezifisch:

Beispiel Übung Folie 43:

*SELECT KursID FROM Kurs
WHERE Semester = 'Herbst' AND Jahr = '2009'
INTERSECT
SELECT KursID FROM Kurs
WHERE Semester = 'Frühling' AND Jahr = '2010'*

- Wie lässt sich die Schnittmenge mit Grundoperatoren anzeigen?



Relation Kurs:	KursID	GruppenID	Semester	Jahr	Gebäude	Raum	Block
BIO-1001	1		Sommer	2009	Painter	514	B
BIO-301	1		Sommer	2010	Painter	514	A
INF-101	1		Herbst	2009	Packard	101	H
INF-101	1		Frühling	2010	Packard	101	F
CS-190	1		Frühling	2009	Taylor	3128	E
CS-190	2		Frühling	2009	Taylor	3128	A
CS-315	1		Frühling	2010	Watson	120	D
CS-319	1		Frühling	2010	Watson	100	B
CS-319	2		Frühling	2010	Taylor	3128	C
CS-347	1		Herbst	2009	Taylor	3128	A
EE-181	1		Frühling	2009	Taylor	3128	C
FIN-201	1		Frühling	2010	Packard	101	B
HIS-351	1		Frühling	2010	Painter	541	C
MU-199	1		Frühling	2010	Packard	101	D
PHY-101	1		Herbst	2009	Watson	100	A

5.3.2 Verbund-Operatoren (Join-Operatoren)

Anfragen in Zusammenhang mit kartesischem Produkt x erfordern häufig Kombinationen mit Selektion σ . Um dies zu vereinfachen, gibt es die Join-Operatoren.

5.3.2.1 Der natürliche Verbund \bowtie (Natural Join)

- Anschaulich
 - = Kombination aus
 - Kartesischem Produkt von zwei Relationen
 - Selektion der Tupel in Ergebnis, die gleiche Wert ein gemeinsamen Attributen beider Relationen haben
 - Löschung doppelter Attribute
 - = Konvention: Reihenfolge der Attribute
 - Erst Attribute, die nur in R sind
 - Dann Attribute, die nur in R und in S sind
 - Zuletzt Attribute, die nur in S sind

- Beispiel

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

$R \bowtie S$:

B	C	D	A	F	G
2	3	4	1	2	3
8	9	0	7	8	9

Relation S:

A	F	G
1	2	3
7	8	9

- Beispiel Aufgabenformulierung
 - = Namen aller Dozenten mit Kursnamen ihrer Kurse
- Beispiel SQL

- o Allgemein:

$R \bowtie S$ lässt sich in SQL darstellen als

SELECT * FROM R NATURAL INNER JOIN S;
SELECT * FROM R NATURAL JOIN S;

- o Spezifisch:

Beispiel Übung Folie 51:

*SELECT Name, KursID FROM Unterrichtet
NATURAL INNER JOIN Dozent;*

Relation Kursinhalt:

KursID	Titel	Abteilung	Creditpoints
BIO-101	Einführung in die Biologie	Biologie	4
BIO-301	Genetik	Biologie	4
BIO-399	Computational Biology	Biologie	3
CS-101	Einführung in die Informatik	Informatik	4
CS-190	Spieleentwurf	Informatik	4
CS-315	Robotik	Informatik	3
CS-319	Bildverarbeitung	Informatik	3
CS-347	Datenbanken	Informatik	3
EE-181	Digitale Systeme	Elektrotechnik	3
FIN-201	Investment Banking	Wirtschaft	3
HIS-351	Weltgeschichte	Geschichte	3
MU-199	Musikvideoproduktion	Musik	3
PHY-101	Physikalische Prinzipien	Physik	4

5.3.2.2 Der Theta Join (θ -Join)

- Anschaulich
Verallgemeinerung des Natural Joins für beliebige Prädikate θ
- Beispiel

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

$R \bowtie_{R.A \neq S.E} S$:

RA	RB	RC	RD	SE	SF	SG
1	2	3	4	7	8	9
4	5	6	7	1	2	3
4	5	6	7	7	8	9
7	8	9	0	1	2	3

Relation S:

E	F	G
1	2	3
7	8	9

- Beispiel SQL
 - Allgemein:
 $R \bowtie_{\theta} S$ lässt sich in SQL darstellen als
`SELECT * FROM R INNER JOIN S ON θ ;`
`SELECT * FROM R JOIN S ON θ ;`
 - Spezifisch:
`Beispiel $R \bowtie_{R.A \neq S.E} S$:`
`SELECT * FROM R INNER JOIN S`
`ON R.A <> S.E;`

5.3.2.3 Outer Join Operatoren

- Anschaulich
Outer Join Operationen nehmen zusätzliche Tupel in Ergebnisse auf:
 - Left Outer Join** $R \bowtie_{\leftarrow} S$: Tupel aus R, die Join-Bedingung nicht erfüllen
 - Right Outer Join** $R \bowtie_{\rightarrow} S$: Tupel aus S, die Join-Bedingung nicht erfüllen
 - Full Outer Join** $R \bowtie_{\leftrightarrow} S$: Tupel aus R und S, die Join-Bedingung nicht erfüllen
- Outer Join Operationen gibt es sowohl als **Natural Outer Join** (z.B. $R \bowtie_{\leftarrow} S$) als auch als **Theta Outer Join** (z.B. $R \bowtie_{\theta \leftarrow} S$)
- Zur Abgrenzung spricht auch von **Inner Join**, wenn kein Outer Join gemeint ist
- Beispiel

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

$R \bowtie_{\leftarrow} S$: (Natural Left outer Join)

B	C	D	A	F	G
2	3	4	1	2	3
5	6	7	4	8	9
7	8	9	0	Null	Null

Relation S:

A	F	G
1	2	3
7	8	9
4	3	4

$R \bowtie_{\rightarrow} S$: (Natural Right outer Join)

B	C	D	A	F	G
2	3	4	1	2	3
5	6	7	4	8	9
Null	Null	Null	2	3	4

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

$R \bowtie_{\leftrightarrow} S$:

B	C	D	A	F	G
2	3	4	1	2	3
5	6	7	4	8	9
Null	Null	Null	2	3	4

Relation S:

A	F	G
1	2	3
7	8	9
4	3	4

Untersch. kart. Produkt: Auffüllen mit Null statt
13. Datenbanken und Informationssysteme 1 Sommersemester 2017 5 - Relationale Algebra: alle Kombinationen! 6/3

- Beispiel SQL
 - Allgemein:

$$R \bowtie_{\theta} S, R \ltimes_{\theta} S, R \rhd_{\theta} S \text{ lässt sich in SQL darstellen als}$$

$$\text{SELECT } * \text{ FROM } R \text{ LEFT OUTER JOIN } S \text{ ON } \theta;$$

$$\text{SELECT } * \text{ FROM } R \text{ RIGHT OUTER JOIN } S \text{ ON } \theta;$$

$$\text{SELECT } * \text{ FROM } R \text{ FULL OUTER JOIN } S \text{ ON } \theta;$$
 - Spezifisch:

$$\text{SELECT Name, KursID FROM Unterrichtet RIGHT OUTER JOIN Dozent ON Unterrichtet.ID = Dozent.ID};$$

5.3.3 Aggregatsfunktionen

- Anschaulich
 - Aggregatsfunktionen berechnen einen einzelnen Wert aus einer Menge von Werten
 - Bsp.: Berechnung durchschnittliches Gehalt

$$G_{\text{avg}(\text{Gehalt})}(\text{Dozent})$$

Mögliche Aggregatsfunktionen sind *avg*, *min*, *max*, *sum*, *count*, *avg-distinct*, *sum-distinct*, *count-distinct*
 - Achtung: Ergebnis ist eine Relation!
- Beispiel

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

$G_{\text{max}(\text{Gehalt})}(\text{Dozent})$:

max(Gehalt)
95000

- Beispiel Aufgabenformulierung

= Summe der Gehälter aller Dozenten der Hochschule
- Beispiel SQL
 - Allgemein:

$$G_{F(A)}(R) \text{ für Aggregatfunktion } F \text{ lässt sich in SQL darstellen als}$$

$$\text{SELECT } F(A) \text{ FROM } R;$$
 - Spezifisch:

Beispiel $G_{\text{max}(\text{Gehalt})}(\text{Dozent})$:

$$\text{SELECT Max(Gehalt) FROM Dozent};$$

5.3.4 Aggregatsfunktionen und Gruppierung

- Anschaulich
 - Aggregatsfunktionen können auf Gruppen von Tupeln angewendet werden
 - Bsp.: Berechnung durchschnittliches Gehalt pro Abteilung

Abteilung $G_{avg(Gehalt)}(Dozent)$

• Beispiel

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Abteilung $G_{avg(Gehalt)}(Dozent)$:

Abteilung	avg(Gehalt)
Informatik	$\langle \emptyset \text{ blau} \rangle$
Wirtschaft	$\langle \emptyset \text{ grün} \rangle$
Musik	40000
Physik	$\langle \emptyset \text{ rot} \rangle$
:	:

• Beispiel Aufgabenformulierung

= Durchschnittliche Bevölkerung und maximale Höhe aller Städte gruppiert nach Land und Provinz

• Beispiel SQL

◦ Allgemein:

 $G_1, G_1, \dots, G_n G_{F_1(A_1), F_2(A_2), \dots, F_m(A_m)}(R)$ für Aggregatfunktionen F_1, F_2, \dots, F_m lässt sich in SQL darstellen als

```
SELECT F_1(A_1), F_2(A_2), ..., F_m(A_m) FROM R
GROUP BY G_1, G_1, ..., G_n;
```

◦ Spezifisch:

Beispiel Folie 76:

```
SELECT AVG(Population), Max(ELEVATION)
FROM City GROUP BY Country, Province;
```