



# **Datenbanksysteme I**

**Vorlesungsskript**

Mitschrift von Falk-Jonatan Strube

Vorlesung von Dr. Axel Toll

10. Mai 2016

# Inhaltsverzeichnis

<b>1 Datenbank als System und Modell</b>	<b>5</b>
1.1 Daten als Unternehmensressource . . . . .	5
1.1.1 Daten und Informationen . . . . .	5
1.1.2 Klassifikation von Daten . . . . .	6
1.1.3 Datenverschlüsselung . . . . .	7
1.1.4 Speicher- und Zugriffsformen . . . . .	9
1.2 Datenmodelle als Abbild . . . . .	11
1.3 Datenbanksysteme als Grundlage . . . . .	15
<b>2 Datenbanksystem</b>	<b>16</b>
2.1 Konventioneller / Datenbankorientierter Ansatz . . . . .	16
2.2 Architektur von Datenbanksystemen . . . . .	19
2.2.1 Grundlegende Begriffe . . . . .	19
2.2.2 3-Ebenen-Architektur . . . . .	19
2.2.2.1 Konzeptionelle Ebene . . . . .	19
2.2.2.2 Externe Ebene . . . . .	20
2.2.2.3 Interne Ebene . . . . .	20
2.3 Aufbau und Arbeitsweise von DBMS . . . . .	21
2.3.1 Zugriffsvermittlung . . . . .	22
2.3.2 Unterstützung Datenbeschreibung-Entwicklung . . . . .	22
2.3.3 Integritätssicherung . . . . .	23
2.3.4 Zugriffsschutz . . . . .	23
2.3.5 Dienstprogrammfunktionen . . . . .	24
2.4 Datenorganisation . . . . .	24
<b>3 Relationales Datenmodell</b>	<b>26</b>
3.1 Terminologie im Relationenmodell . . . . .	26
3.2 Definition und Manipulation im relationalen Datenmodell . . . . .	28
3.2.1 Datendefinition . . . . .	28
3.2.2 Datenmanipulation / Relationenalgebra . . . . .	29
3.2.2.1 Mengenoperationen . . . . .	29
3.2.2.2 Relationale Operationen . . . . .	33
3.3 Normalformenlehre . . . . .	36
3.3.1 1. Normalform . . . . .	37
3.3.2 2. Normalform . . . . .	38
3.3.3 3. Normalform . . . . .	39
3.4 Vergleich relationaler DBMS . . . . .	43
<b>4 Datenbanksprachen für relationale DBMS</b>	<b>45</b>
4.1 Benutzergruppen und Datenbanksprachen . . . . .	45
4.2 SQL als Standardsprache für relationale DBMS . . . . .	47
4.2.1 Überblick . . . . .	47
4.2.1.1 SQL-Datentypen (Auswahl) . . . . .	49
4.2.1.2 Symbole der Syntaxbeschreibung: . . . . .	49

4.2.2	Anweisungen zur Definition (DDL-Befehle) . . . . .	50
4.2.2.1	Tabellendefinition . . . . .	50
4.2.2.2	Arbeiten mit Index . . . . .	52
4.2.2.3	View-Definition . . . . .	52
4.2.3	Anweisungen zur Abfrage (DML-Befehle) . . . . .	53
4.2.3.1	Standardabfrage . . . . .	53
4.2.3.2	Einfache Abfrage auf eine Relation . . . . .	54
4.2.3.3	Sortierung / Gruppenbildung . . . . .	56
4.2.3.4	Built-in-Funktionen . . . . .	56
4.2.3.5	Abfrage mit Bereichsgrenzen und Wertaufzählungen . . . . .	57
4.2.3.6	Einfache Abfrageschachtelung . . . . .	57
4.2.3.7	Abfrage über mehrere Relationen . . . . .	58
4.2.3.8	Weitere Abfragemöglichkeiten . . . . .	59
4.2.3.9	Abfrageschachtelungen . . . . .	60

# Prüfungsmodalitäten

**PVL** unbenoteter Beleg als Voraussetzung zur Prüfung

- 1.) Access-Beleg (in Papier-Form abzugeben bis 27.05.2016)
- 2.) Abnahme der SQL-Praktikums-Aufgaben (Abnahme während Praktikumszeit)

**SP** schriftliche Prüfung, 90min

keine eigenen Unterlagen zugelassen. Nur zuvor ausgegeben Referenzen.

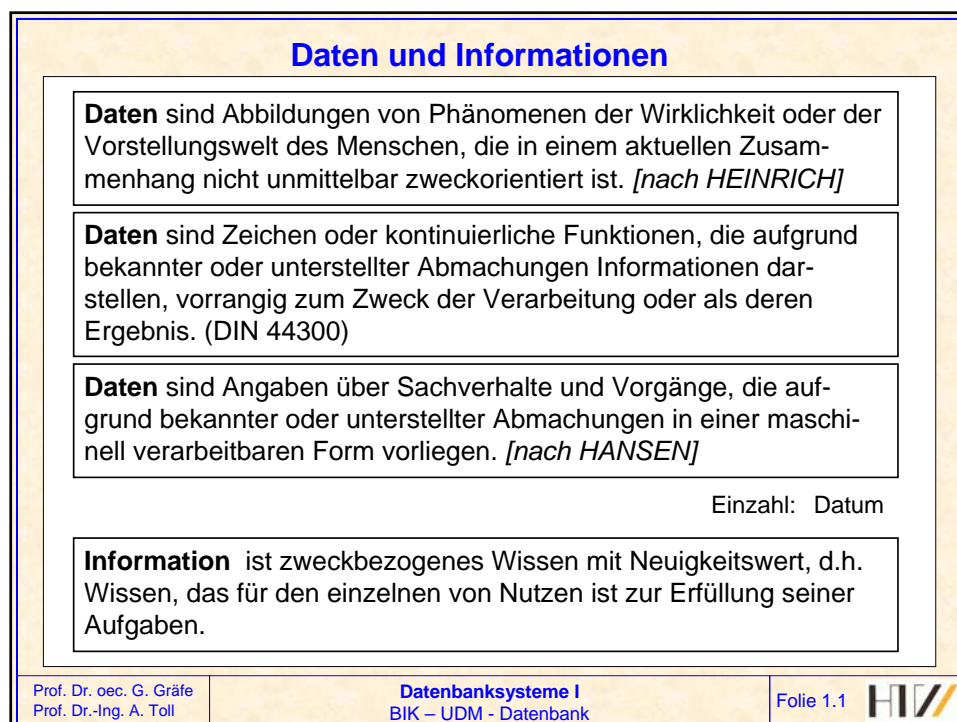
# 1 Betriebliche Informations- und Kommunikationssysteme - Unternehmensmodell - Datenbank

## 1.1 Daten als Unternehmensressource

### 1.1.1 Daten und Informationen

Redundante Daten bergen Gefahr von Inkonsistenz  $\Rightarrow$  Ziel: Schaffen von Datenbank mit folgenden Eigenschaften:

- ohne Inkonsistenzen (redundanzarm)
- Zugriffsschutz
- Mehrfachzugriff
- Backup-Möglichkeiten (mit Widerspruchsfreier Wiederherstellung)



## Betriebliche Produktionsfaktoren

- klassische Faktoren
  - Betriebsmittel
  - Werkstoffe
  - Arbeitskraft
- Daten + Informationen

Probleme des Informatikeinsatzes		
➤	<b>Softwarekrise trotz CASE Tools</b>	mögliche Ursache: ungenügende Anforderungsanalyse und -definition
➤	<b>Zu hoher Kostenaufwand in den letzten Entwicklungsphasen und in der Systemwartung</b>	mögliche Ursache: unzureichende methodische Unterstützung der Anfangsphase
➤	<b>Überschreitung von Lieferterminen</b>	mögliche Ursache: mangelndes Projektmanagement
➤	<b>Jahrhundertproblem der Informatik (Datenchaos)</b>	mögliche Ursache: fehlende unternehmensweite Datenmodellierung
➤	<b>Unzulängliche Anwendungssysteme, fehlende Nutzung</b>	mögliche Ursache: unzureichende Einbeziehung der Nutzer in die Systementwicklung
<b>Das Jahrhundertproblem der Informatik besteht in:</b>		
➤	Der Bewältigung des Datenchaos, das infolge unkontrolliert gewachsener Datenbestände fast überall entstanden ist.	
➤	Der Schaffung einer einheitlichen, zentrale und dezentrale Datenbestände umfassenden Datenbasis, die für die effiziente Nutzung zukunftssträchtiger Möglichkeiten der Informatik - gemeint sind benutzerfreundliche, auch Nichtinformatikern zumutbare Anwendungs-generatoren und höhere Datenbanksprachen - unerlässlich ist. <i>[nach Vetter]</i>	
<div style="display: flex; justify-content: space-between; font-size: small;"> <span>Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll</span> <span><b>Datenbanksysteme I</b> BIK – UDM - Datenbank</span> <span>Folie 1.2 </span> </div>		

Große Datenbestände ⇒ Maßnahmen zur Datenorganisation

Eine mögliche Organisationsform (logisches Konzept): Ablage in Relationen (=Tabelle)

Eine Zeile in dieser Tabelle nennt man *Datensatz* (Tupel, Record, ...).

Eine Spalte nennt man *Datenfeld*.

## 1.1.2 Klassifikation von Daten

### Mögliche Kriterien für Datenfeld

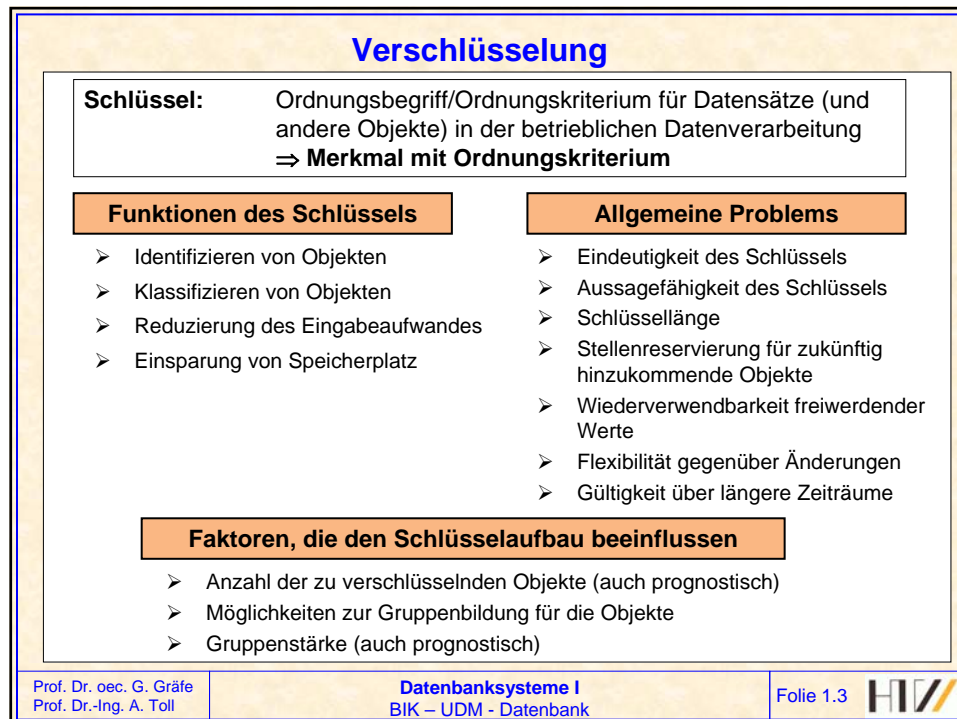
- Zeichenart
  - ganze Zahl ⇒ für Aufzählungen
  - reelle Zahl ⇒ numerische Berechnungen
  - Währung ⇒ finanztechnische Berechnungen
  - Datum ⇒ kalendarische Berechnungen/Werte
  - Text ⇒ Beschreibung
  - Bitmuster ⇒ Video, Bilder, ...
- Erscheinungsform

- sprachlich
- bildlich
- schriftlich
- Stellung im Verarbeitungsprozess (E - V - A)
  - Eingabe
  - Verarbeitung
  - Ausgabe
- Verarbeitbarkeit mittels IT  
(Umwandlung in digitale Daten: analog → diskret → digital)
- Verwendungszweck

	Charakterisierung	Beispiel
Stammdaten	selten zu verändern (über längeren Zeitraum in Struktur und Inhalt konstant)	Personalstammdaten (Name, Adresse)
Änderungsdaten	Aktualisierung der Stammdaten	Änderung der Adresse
Bestandsdaten	Periodische Änderung des Wertes (Inhalt) von Feldern, Datenstruktur besteht über längeren Zeitraum konstant	Lagerbestände, Kassenbestände
Bewegungsdaten	Daten zur Aktualisierung des Wertes von Bestandsdaten	Lagerzugänge und -abgänge
Archivdaten	vergangenheitsbezogene Daten die über längeren Zeitraum aufbewahrt werden	Rechnungen, Buchungen der vergangenen 5 Jahre
Transferdaten	Daten, die von einem anderen Programm erzeugt wurden und an ein anderes transferiert werden	Verkauf von Kundenadressen
Vormerkdaten	Daten, die solange existieren, bis ein genau definiertes Ereignis eintritt	Reservierung einer Materialmenge im Lager

### 1.1.3 Datenverschlüsselung

Gemeint ist nicht die Codierung und Decodierung von Daten, sondern das Zuweisen von Schlüsseln zu Datensätzen.



### Identifizierender Schlüssel

kennzeichnet Objekteindeutig

Bsp.:

- Personal-Nr.
- Material-Nr.

### Klassifizierender Schlüssel

ordnet Objekt einer Klasse zu

Bsp.:

- Länderkennung: D, C, CH, ...
- Geschlecht: M, W

### Hierarchischer Verbundschlüssel

identifizierender Teil hängt vom klassifizierenden Teil ab

Bsp.:

- Autokennzeichen:  $\underbrace{DD}_{\text{klass.}} \underbrace{XY 715}_{\text{ident.}}$

### Parallelschlüssel

zwei unabhängige Schlüsselteile

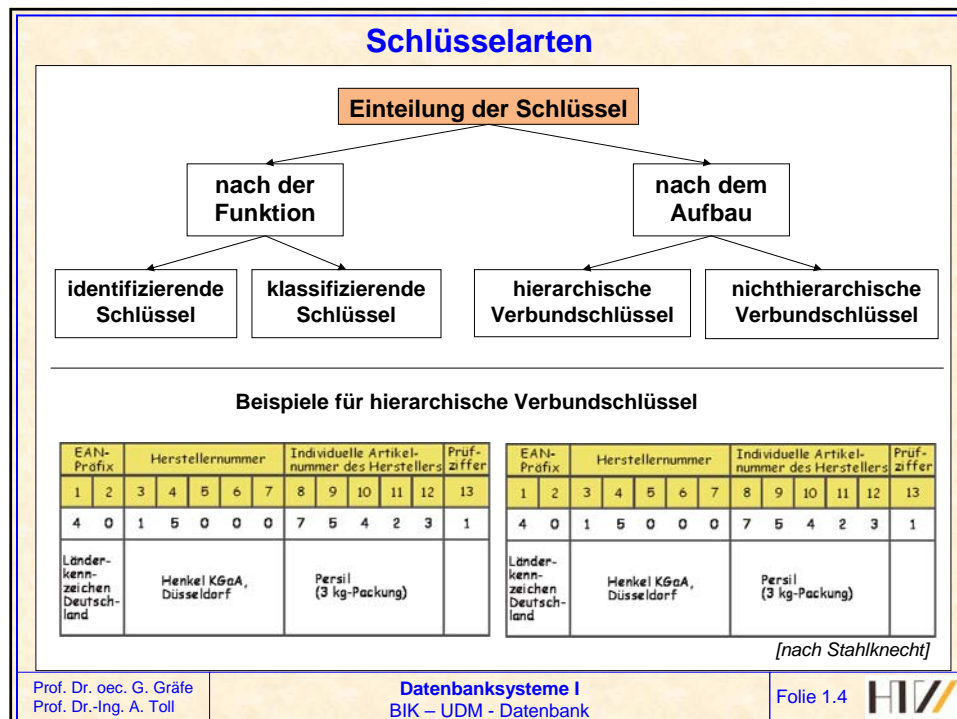
Bsp.:

- Flugnummer  $\underbrace{LH 283}_{\text{Flugnr.}} \underbrace{AB3}_{\text{Flugzeug}}$



## spezielle Schlüssel in Datenbanksystemen

- **Primärschlüssel** (primary key PK): Datenfeld oder die Kombination aus Datenfeldern, die den Datensatz in der Tabelle eindeutig identifizieren.  
Bsp. Vereinsdatenbank:  
Primärschlüssel als einzelnes Datenfeld (Mitgliedertabelle): Mitglieds-ID  
Primärschlüssel als eine Kombination von Datenfeldern (Beitragstabelle): ID mit Jahr (für Vereinsbeitrag abhängig von Jahr)
- **Fremdschlüssel** (foreign key FK): Datenfeld, oder Kombination aus Datenfeldern, der (die) auf den PK einer anderen Tabelle zeigt.  
Bsp.: Mitglieds-ID in Tabelle mit Datenfelder-Primärschlüssel kommt aus der ersten Tabelle
- **Referentielle Integrität**: Jeder Wert eines FK muss gleich dem Wert des PK sein, auf den der FK zeigt.  
Bsp.: Neuer Eintrag in Beitragstabelle kann nur neue Einträge bekommen, die Mitglieder aus Mitgliedertabelle enthält. Anders herum kann aus der Mitgliedertabelle kein Mitglied gelöscht werden, das noch in der Beitragstabelle genutzt wird.



## 1.1.4 Speicher- und Zugriffsformen

- **sequentielle Speicherung** (fortlaufend)  
Bsp.: Bandlaufwerk  

101	102	103	...
-----	-----	-----	-----
- **verkettete Speicherung**  
Bsp.: verkettete Listen (vgl. Programmierung I)
- **indexverkettete Speicherung**  
Trennung: Datenspeicherung und „Weg“ zu den Daten
  - Indexdatei (sortiert nach entsprechendem Index)

- ♦ Primärindex zeigt auf physische Adresse
- ♦ Sekundärindex zeigt auf Primärindex
- Hauptdatei

Grundprinzip der Indizierung				
	Mitnr	Name	Ort	Alter
ADR1	101	Hase	Dresden	37
ADR2	102	Igel	Dresden	19
ADR3	103	Fuchs	Dresden	23
ADR4	104	Elster	Freiberg	26
ADR5	105	Uhu	Berlin	22
ADR6	106	Rabe	Radebeul	68

Primärindex Mitnr		Sekundärindex Ort		Sekundärindex Alter	
Feldinhalt	Adresse	Feldinhalt	Schlüssel	Feldinhalt	Schlüssel
101	ADR1	Berlin	105	19	102
102	ADR2	Dresden	101	22	105
103	ADR3	Dresden	102	23	103
104	ADR4	Dresden	103	26	104
105	ADR5	Freiberg	104	37	101
106	ADR6	Radebeul	106	68	106

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

Datenbanksysteme I  
BIK – UDM - Datenbank

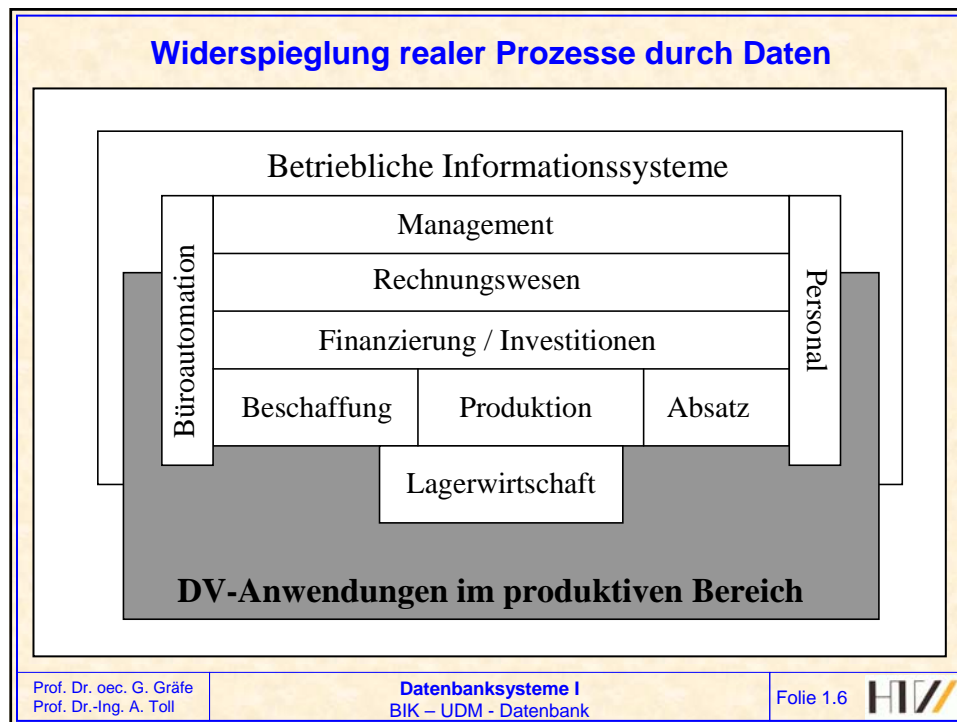
Folie 1.5



Unterschied Primärschlüssel-Primärindex:

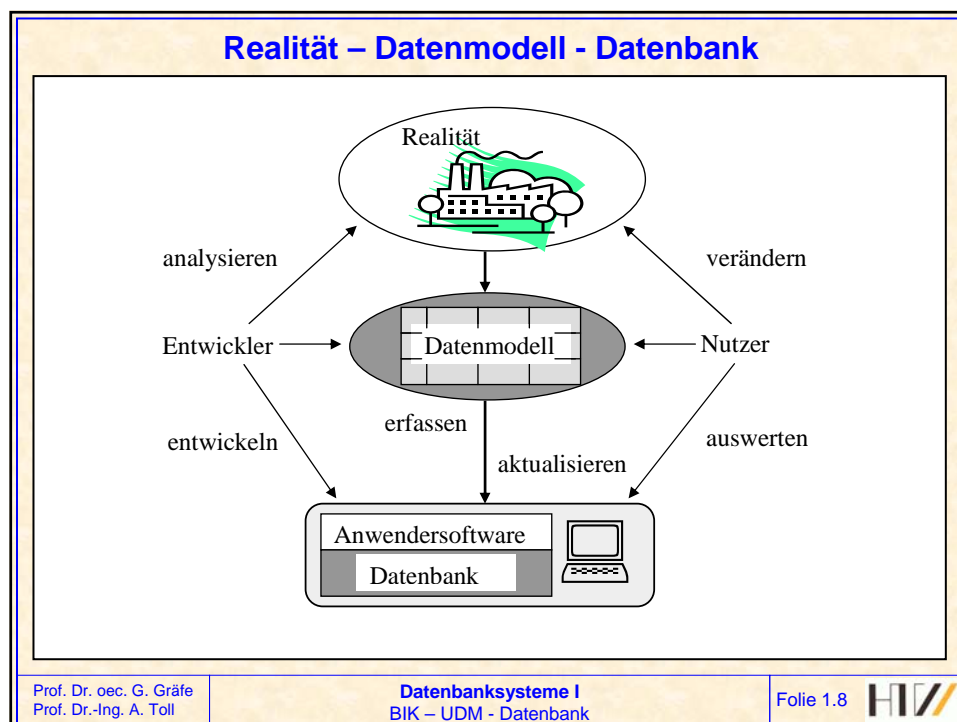
- Primärschlüssel dient dem Identifizieren
- Primärindex zum schnellen Suchen

## 1.2 Datenmodelle als informationelles Abbild der Unternehmensrealität



### Informationssystem

- *Funktionsmodell* (was soll das System leisten: Produktion, Lager, Beschaffung, ...) ⇒ Kernfrage: „Was will ich machen“  
Strukturen, Abläufe  
Technik: Programm-Ablauf-Plan (PAP), Ereignisorientierte Prozessketten (EPK), ...
- *Datenmodell*  
Daten und deren logische Struktur  
Technik: Entity-Relationship-Modell (ERM)



## Datenmodellierung

**Datenmodellierung:**

Aufbau einer einheitlichen, anwendungsübergreifenden Sicht der Datenressourcen des Unternehmens und  
Abbildung von Informationsprozessen und -beziehungen in die konkrete Struktur eines Datenverwaltungs- oder Datenbanksystems

Die **Datenmodellierung** schafft die **Voraussetzung** für systematische Integration der Daten und damit für eine **unternehmensweite Nutzung** der Ressource Information.

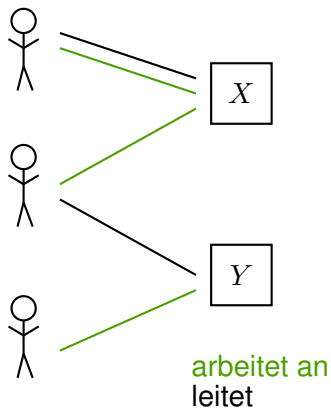
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	Datenbanksysteme I BIK – UDM - Datenbank	Folie 1.9
---	---	-----------

### Bsp.:

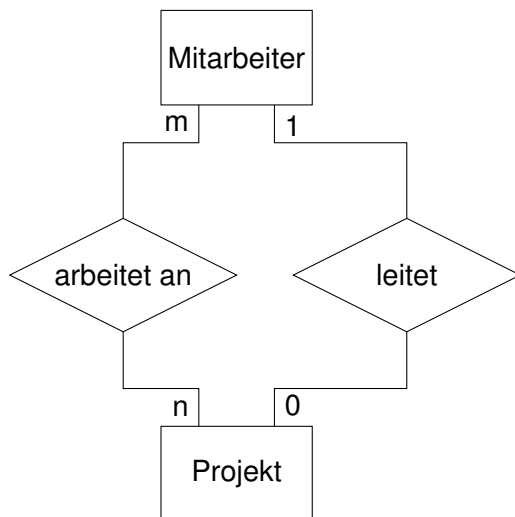
Reale Welt:

Mitarbeiter

Projekt



ERM (semantisches Modell):



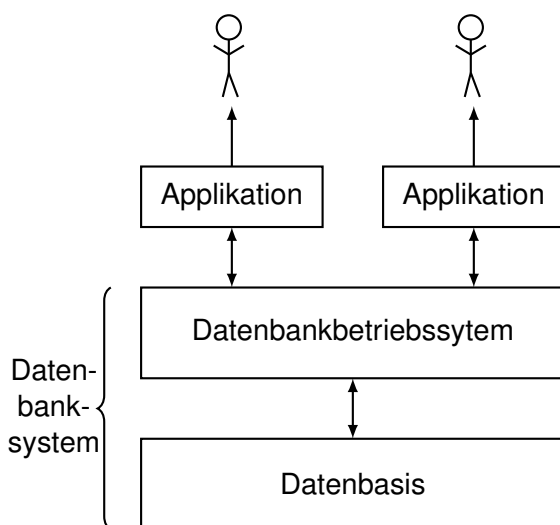
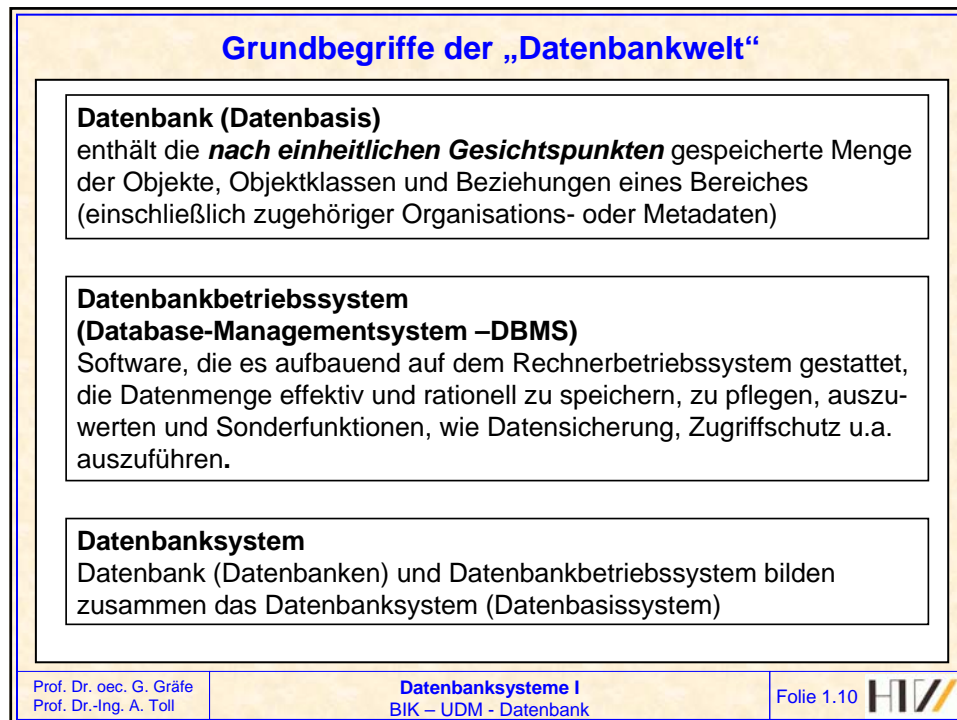
RM (relationales/logisches Modell):

Tabelle Mitarbeiter


Tabelle Bearbeitung


Tabelle Projekt


## 1.3 Datenbanksysteme als technologische Grundlage der Datenverwaltung



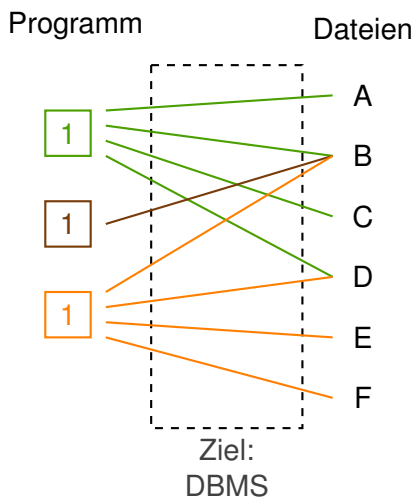
Datenbasis: Tabellen mit Metadaten

Datenbankbetriebssystem (DBMS): Software, die mit Datenbasis kommuniziert

## 2 Grundlagen und Architektur eines Datenbanksystems (DBS)

### 2.1 Defekte des konventionellen Ansatzes der Datenverwaltung / Zielstellung des datenbankorientierten Ansatzes

**konventionell**



#### **konventionelle Datenorganisation**

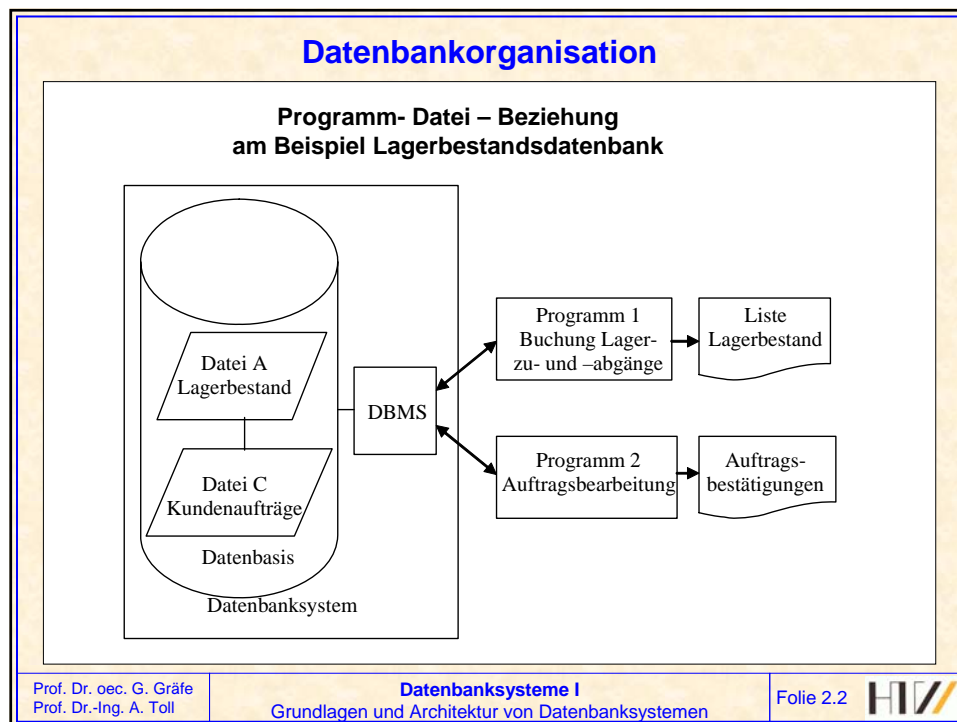
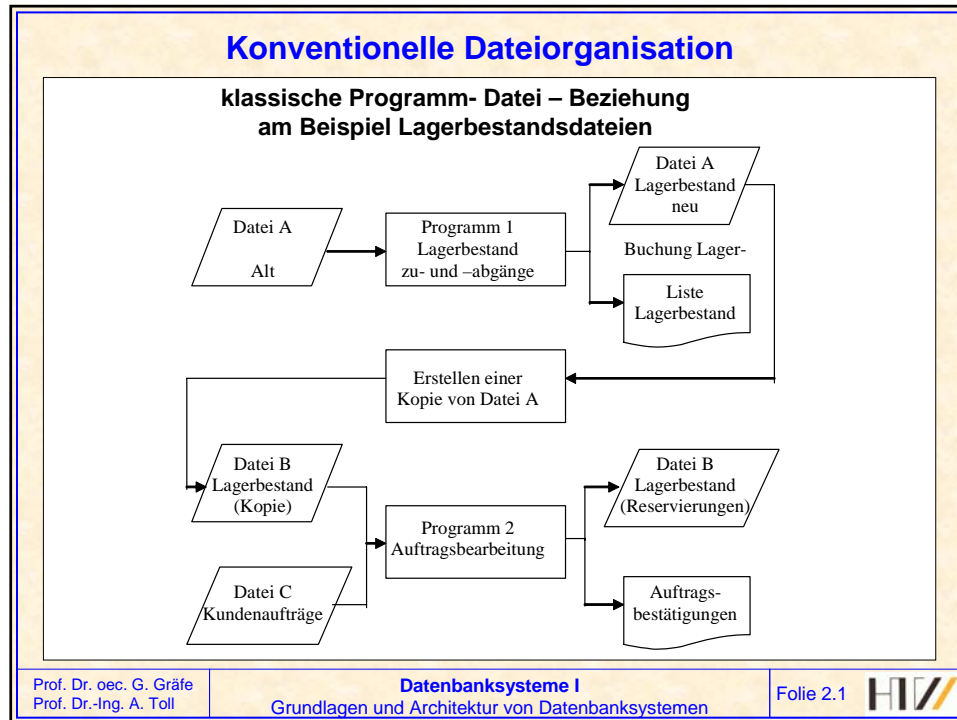
##### *Merkmale*

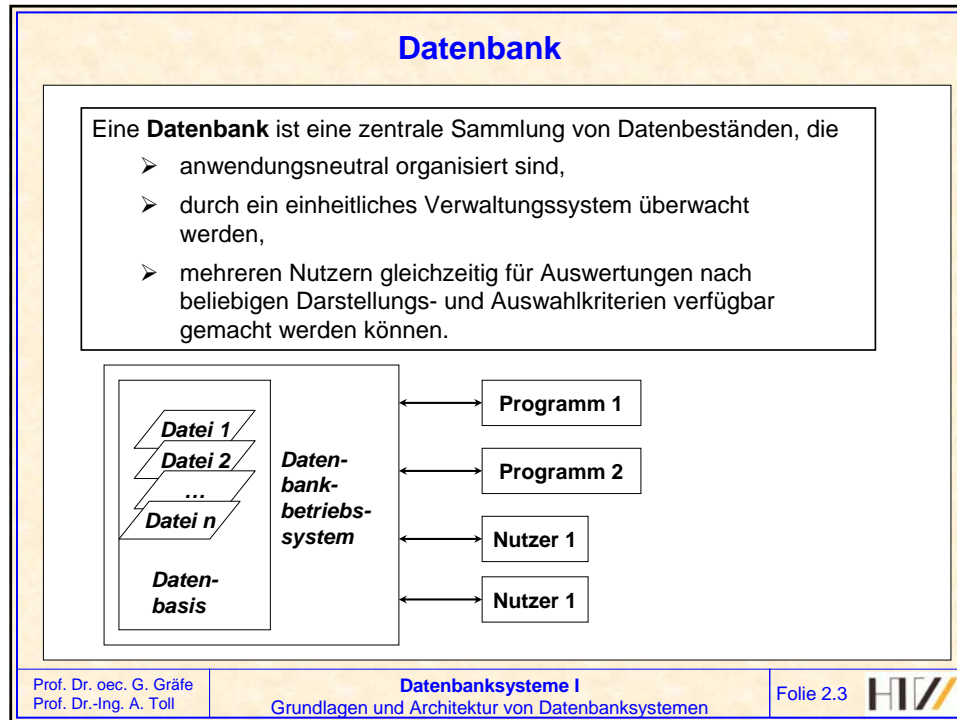
- Datenspeicherung je Anwendung
- Datenspeicherung auf physischem Niveau

##### *Nachteile*

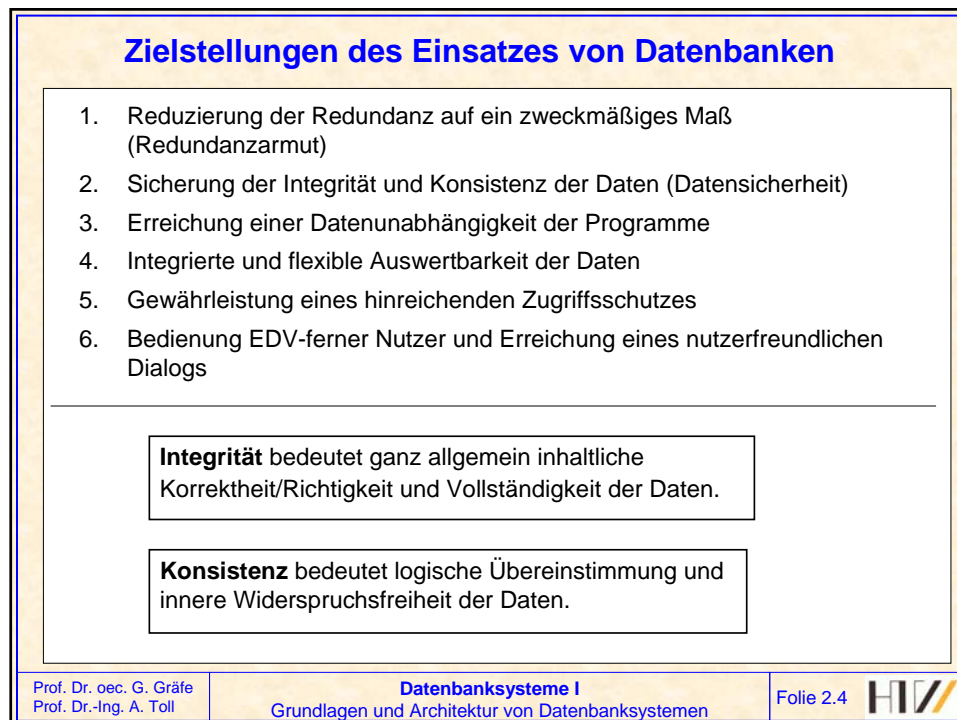
- mangelnde Passfähigkeit (Zugriffskonflikte usw.)
- Redundanz
- Konsistenzprobleme
- mangelnde Flexibilität
- Daten-Programm-Abhängigkeit (kurz: Datenabhängigkeit)







### Zielsetzung des Datenbankeinsatzes



1.) Bsp. für gewollte Redundanz: Sekundärindex

2.) Datensicherheit:

- physisch, falls bspw. der Server abbrennt
- logisch, dass bspw. alle Daten den richtigen Typ haben

## 2.2 Architektur von Datenbanksystemen

### 2.2.1 Grundlegende Begriffe

Am Beispiel der Objekte der Datenmodellierung mittels ERM

Begriff	Erklärung	Beispiel
Entity	Objekt der realen Welt	Max Meier, Arbeitsaufgabe Reportgenerator
Entity-Typ	Objektklasse (-Menge), enthält Elemente mit struktureller Ähnlichkeit	Mitarbeiter, Arbeitsaufgabe, Abteilung
Merkmale / Attribut / Prädikat	Beschreibungen eines Entity-Typs	Name, Vorname, Gehalt
Wert	Ausprägung des Merkmals je Entity, aus einem bestimmten Wertevorrat (Domain)	„Meier“, „Max“, 3800,-
Beziehung, Set	Logischer Zusammenhang zwischen Entity-Typen	Mitarbeiter – <u>arbeitet an</u> – Arbeitsaufgabe
Beziehungstyp, Settyp	Art der Beziehung (mögliche Anzahl an Entitäten, die in Beziehung treten)	$n : 1$ Mitarbeiter – <u>gehört zu</u> – Abteilung ABB50

### 2.2.2 3-Ebenen-Architektur

gemäß ANSI x3/SPARC (1975)

- Architekturebene
  - externe Ebene
  - konzeptionelle Ebene
  - interne Ebene
- Modell
  - externes Modell
  - konzeptionelles Modell
  - internes Modell
- Schema (konkrete Ausprägung des Modells)
  - externes Schema
  - konzeptionelles Schema
  - internes Schema

#### 2.2.2.1 Konzeptionelle Ebene

**Gegenstand:** logisches Modell des gesamten Systems

### Beschreibungselemente:

- Entity-Typen
- Beziehungen
- Attribute
- Wertevorräte (bspw. Einschränkung von Alter: nur Zahlen zwischen 1 und 100)
- Integritätsbedingung (bspw. NOT NULL, vgl. Wertevorrat)

#### 2.2.2.2 Externe Ebene

**Gegenstand:** Beschreibung *ausgewählter* Elemente der konzeptionellen Ebene aus Sicht des jeweiligen Endbenutzers



**Element:** Sicht (View)

#### 2.2.2.3 Interne Ebene

**Gegenstand:** Form/Art der Ablage der Elemente der konzeptionellen Ebene im physischen Speicher

**Element:** Index

Charakterisierung der Ebenen eines Datenbanksystems		
Ebene/Modell/Schema	Beschreibung	Verantwortlichkeit
extern	Enthält verschiedene Sichten (views) auf die Daten eines Bereichs der objektiven Realität (externe Objekte mit von speziellen Nutzern vorgegebenen Beziehungen)	Anwendungsadministrator (application administrator)
konzeptuell	Enthält die Gesamtschau der Daten eines Bereichs. Beschreibt die Daten des Bereichs auf einer logischen Ebene unabhängig von den Gesichtspunkten der EDV. Es werden Typen von Objekten und die bestehenden Beziehungen zwischen den Objekten definiert sowie die Attribute (von Objekten und Beziehungen) und deren Wertevorrat spezifiziert.	Unternehmensadministrator (enterprise administrator)
intern	Enthält die Form der Ablage der logisch beschriebenen Daten im Speicher und die Zugriffsmöglichkeiten zu diesen Daten (physische Datenorganisation mit Angaben zu Aufbau, Speicherungsform und Zugriffspfaden).	Datenbankadministrator (database administrator)

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

Datenbanksysteme I  
Grundlagen und Architektur von Datenbanksystemen

Folie 2.7 


## 2.3 Aufbau und Arbeitsweise von DBMS

### 5 Grundfunktionen eines DBMS


Funktionen eines DBMS (1)	
Übersicht	
1.	Speichern und Wiederauffinden von Daten und Metadaten/ Zugriffsvermittlung <ul style="list-style-type: none"> <li>- Auswahl/Anzeigen</li> <li>- Anzeigen/Hinzufügen</li> <li>- Ändern/Löschen</li> </ul>
2.	Erstellen/Ändern/Löschen von Datenbeschreibungen
3.	Datensicherung/Integritätssicherung <ul style="list-style-type: none"> <li>- Sicherung der semantischen Integrität</li> <li>- Sicherung der Ablaufintegrität/operationalen Integrität</li> <li>- Sicherung der physischen Integrität</li> </ul>
4.	Zugriffsschutz <ul style="list-style-type: none"> <li>- Eingangskontrolle</li> <li>- Rechtevergabe und Zugriffskontrolle</li> <li>- Verschlüsselung</li> </ul>
5.	Dienstprogrammfunktionen <ul style="list-style-type: none"> <li>- Monitoring und Tuning</li> <li>- Unterstützung der Aufgaben des DBA</li> <li>- Export und Import von Daten</li> <li>- Unterstützung der Erstellung von Masken, Reports und AP's</li> </ul>

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll


Datenbanksysteme I  
Grundlagen und Architektur von Datenbanksystemen

Folie 2.8 

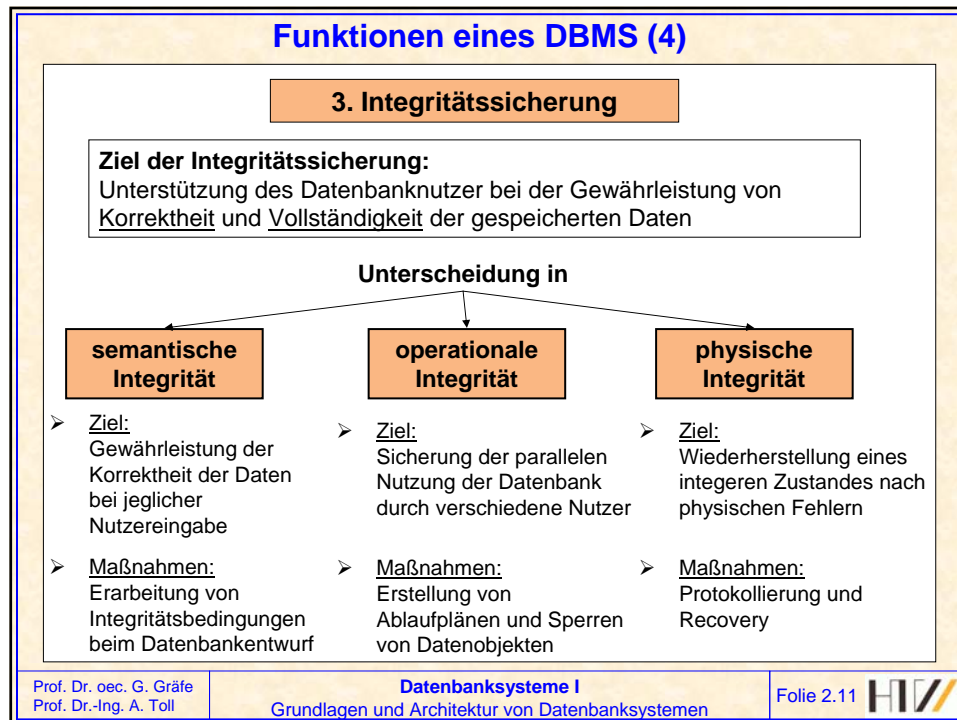
## 2.3.1 Zugriffsvermittlung

<b>Funktionen eines DBMS (2)</b>		
<b>1. Zugriffsvermittlung</b>		
<ul style="list-style-type: none"> <li>➤ Basisfunktion eines DBMS</li> <li>➤ Die Zugriffsvermittlung beeinflusst darüber hinaus die Notwendigkeit und die konkreten Realisierungsbedingungen anderer Funktionen</li> <li>➤ Die Zugriffsvermittlung des DBMS erfolgt in Abhängigkeit von der Architektur des DBS mehrstufig (für 3-Ebenen-Modell):             <ol style="list-style-type: none"> <li>1. Abbildung von Zugriffsforderungen, die in den Begriffen eines externen Modells formuliert sind, auf die konzeptionelle Ebene</li> <li>2. Transformation der konzeptuellen Entities und ihrer Beziehungen auf das interne Modell</li> <li>3. physische Bereitstellung von gespeicherten Informationen über das Betriebssystem</li> </ol> </li> <li>➤ Allgemeine Zugriffsfunktionen:             <ul style="list-style-type: none"> <li>• <u>Auswahl</u> von Entities und Beziehungen</li> <li>• <u>Hinzufügen</u> neuer Entities und Beziehungen</li> <li>• <u>Ändern</u> existierender Entities und Beziehungen</li> <li>• <u>Löschen</u> existierender Entities und Beziehungen</li> </ul> </li> </ul>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Grundlagen und Architektur von Datenbanksystemen	Folie 2.9 

## 2.3.2 Unterstützung Datenbeschreibung-Entwicklung

<b>Funktionen eines DBMS (3)</b>		
<b>2. Unterstützung der Entwicklung von Datenbeschreibungen</b>		
<p>Beschreibungssprachen bei Mehr-Ebenen-Architekturen:</p> <ul style="list-style-type: none"> <li>➤ DDL zur Beschreibung der Externen Schemas</li> <li>➤ DDL zur Beschreibung des Konzeptuellen Schemas einschließlich der Beschreibung der Abbildungen extern/konzeptuell;</li> <li>➤ SDDL zur Beschreibung des Internen Schemas, einschließlich der Beschreibung der Abbildungen konzeptuell/intern</li> </ul>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Grundlagen und Architektur von Datenbanksystemen	Folie 2.10 

## 2.3.3 Integritätssicherung

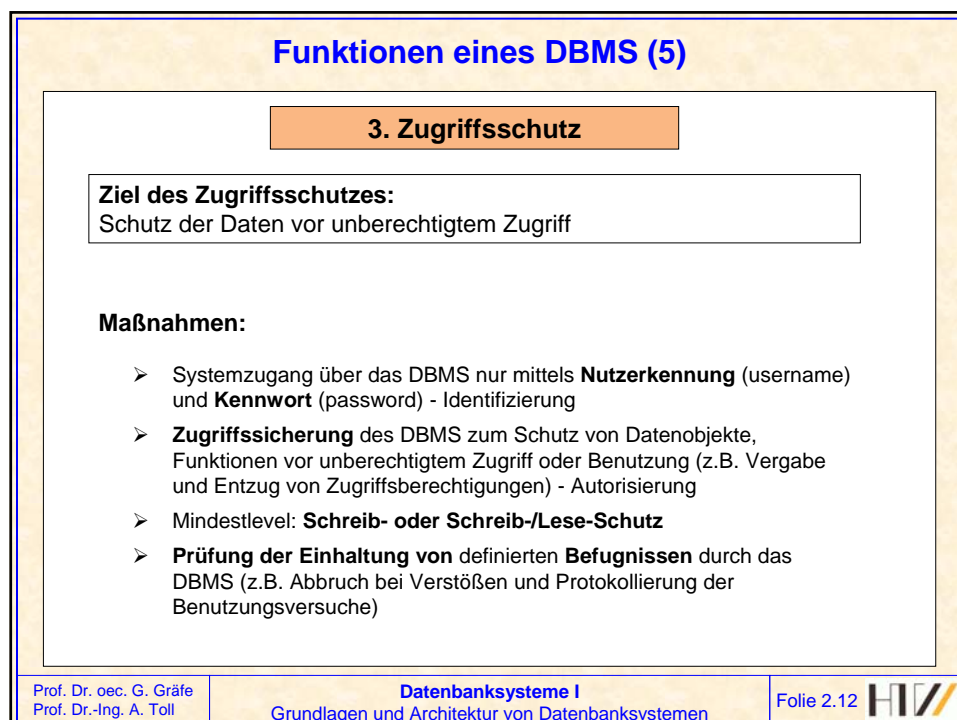


Bsp. operationale Integrität:

Gehaltserhöhungen sowohl für Organisatoren (O) und Programmierer (P) um €50,-.

Gehaltserhöhung darf nicht doppelt erfolgen ⇒ Sperren von Gehalt, solange ein Nutzer das Gehalt ändert (bei Gefahr bezgl. Deadlock, muss das System das Problem erkennen und entsprechend auflösen).

## 2.3.4 Zugriffsschutz



## 2.3.5 Dienstprogrammfunktionen

**Funktionen eines DBMS (6)**

**4. Dienstprogrammfunktion**

**Ziel:**  
Rationalisierung des Entwurfs, des Aufbaus und eines effektiven Betriebes eines DBS

**Beispiele für Dienstprogrammfunktionen:**

- Registrieren und Auswerten von Zugriffshäufigkeiten zu den Dateneinheiten
- Unterstützung der Arbeit des Betriebs- und Datenbankadministrators
- Unterstützung des Entwurfs der Datenstrukturen
- Gewährleistung der Datensicherheit
- Export/Import von Datenbanken
- Bestimmen optimaler Reorganisationszeitpunkte (wenn Reorganisation überhaupt notwendig)

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
Grundlagen und Architektur von Datenbanksystemen

Folie 2.13

## 2.4 Datenorganisation

- logische Datenorganisation (DO)
  - externe Ebene
  - konzeptionelle Ebene
- physische DO
  - interne Ebene

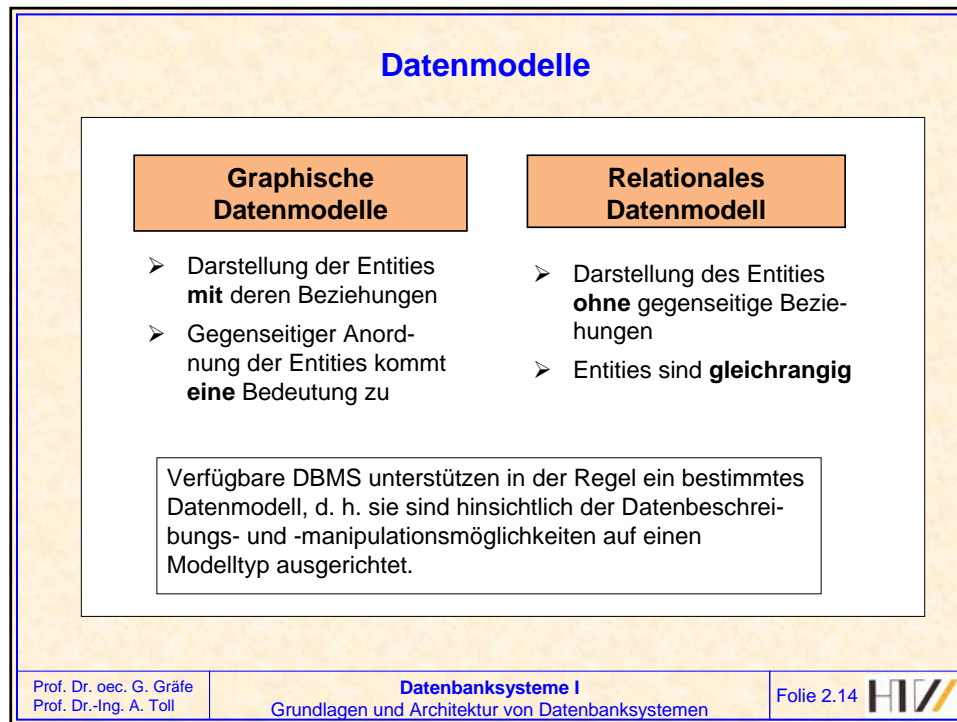
### klassische Datermodelle (logisch)

- hierarchisch DM (graphisches DM)
- Netzwerk DM (graphisches DM)
- relationales DM (behandelt in DBS I+II)

### weitere DM

- objektorientiertes DM (DBS II)
- objektrelationales DM (DBS II)
- XML-DM / NoSQL DM ... (DBS III)





	Hierarchisches DM ABB 51	Netzwerk DM ABB 52	relationales DM ABB 53
Einstiegspunkt	ein Entity-Typ	mehrere Entity	beliebig
strukturelle Beschränkung	Hierarchie	keine	keine
Zeitpunkt des Aufbau der Beziehung	zur Entwicklungszeit	zur Entwicklungszeit	zur Laufzeit
Performance	+	+	–
Flexibilität bzgl. Änderung	–	–	+

# 3 Relationales Datenmodell

## 3.1 Terminologie im Relationenmodell

Relationenbegriff

Mathematische Definition

Sind  $W_1, W_2, \dots, W_n$  nichtleere Mengen, dann ist jede nichtleere Teilmenge der Produktmenge

$$PM = W_1 * W_2 * \dots * W_n \text{ eine } n\text{-stellige}$$

**Relation**  $R \subseteq W_1 \times W_2 \times \dots \times W_n$

$W_1, W_2, \dots, W_n$  sind die Wertebereiche (Menge aller Werte = Domäne) der Attribute  $A_1, A_2, \dots, A_n$  von Entities.  
 $n$  ist der Grad (degree) der Relation.

Darstellung als Tabelle

Mitarbeiter	MITNR	VORNAME	NAME	ANSCHRIFT	ALTER	← Attributnamen
	101	Peter	Silie	Dresden	5	
	102	Mario	Nette	Dresden	6	
	103	Klaus	Uhr	Radebeul	20	
	104	Otto	Graffie	Freital	17	
	105	Kurt	Isane	Friedersdorf	35	
	106	Paul	Aner	Merseburg	25	

↑

Primär-schlüssel

Nichtschlüsselspalten  
Datenspalten

Domäne

Attributwerte

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

Datenbanksysteme I  
Relationale Datenmodell

Folie 3.1

**Bsp.:**

Entitytyp:

- Zeugnis

Attribute:

- $A_1$  Fach
- $A_2$  Note

Wertebereiche:

- $W_1 \{Ma, Ph\}$
- $W_2 \{1, 2, 3, 4, 5\}$

$n = 2$ , d.h. 2-stellige Relation ableitbar (Grad = degree = 2)

$$PM = W_1 * W_2 = W_1 \times W_2$$

Fach	Note
Ma	1
Ma	2
Ma	3
Ma	4
Ma	5
Pd	1
Pd	2
Pd	3
Pd	4
Pd	5

Teilmenge 1 = Relation 1:

Fach	Note
Ma	1
Ph	2

Teilmenge 2 = Relation 2:

Fach	Note
Ma	1
Ph	1
Ph	4

### Charakteristika des relationalen Modells

- K1: Es gibt eine Menge von Relationen unterschiedlichen Grades über den Attributwerten.
- K2: Die Relationen sind untereinander gleichberechtigt.
- K3: Die Relationen sind zeitlich veränderlich (Einfügen, Löschen, Ändern von Tupeln).
- K4: Jede Relation hat dabei charakteristische Eigenschaften.
- K41: Jedes Tupel der Relation kommt nur einmal vor.
- K42: Die Reihenfolge der Tupel ist beliebig.
- K43: Die Reihenfolge der Spalten ist auch beliebig, da die Bezugnahme auf die Spalten über die eindeutigen Attributnamen und nicht über eine Spaltennummer erfolgt. Attributnamen müssen in einer Relation unterschiedlich sein. Die Reihenfolge wird einmal vorgegeben, bleibt dann bestehen.
- K44: Es gibt genau einen Primärschlüssel, der die Tupel eindeutig identifiziert.

### Weitere Kernaussagen zum relationalen Modell:

- Darstellung der Relation als Tabelle
- Identifikation der Relation über Namen

- Anzahl an Attributen (Spalten) ist fest (degree)
- Anzahl der Tupel (Zeilen) ist variabel (Mächtigkeit)
- Wertebereiche der Attribute = Domain
- Im Kreuzungspunkt von Attribut und Tupel stehen *atomare* Werte

## 3.2 Definition und Manipulation im relationalen Datenmodell

### 3.2.1 Datendefinition

⇒ Definition von Relationen


Datendefinition im relationalen Datenmodell


Eine im relationalen Datenbanksystem agierende Datenbeschreibungssprache (DDL) muß die im relationalen Modell vorhandenen Komponenten definieren:

- Name der Relation,
- Attributnamen,
- Wertebereiche,
- Primärschlüssel,
- ggf. Integritätsbedingungen

Prof. Dr. oec. G. Gräfe  
 Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
 Relationale Datenmodell

Folie 3.3 

Datendefinition im relationalen Datenmodell - Beispiel		
<b>Relation:</b>	<b>Mitarbeiter</b>	
	Attribute	Mitarbnr; INT Name; CHAR(20) Geburtsdatum; DATE Gehalt; NUMERIC(8,2)
<b>Relation:</b>	<b>Abteilung</b>	
	Attribute	Abteilnr; INT Bezeichnung; CHAR(15) Raum; CHAR(5) Leiter; INT
	Integritätsbedingung	Leiter → Mitarbeiter.Mitarbnr 100 ≤ Raum < 451
<b>Relation:</b>	<b>Mitabt</b>	
	Attribute	Mitarbnr; INT Abteilnr; INT Anteil; NUMERIC(3,1)
	Integritätsbedingung	(Mitarbnr, Abteilnr) ist Primärschlüssel; Mitarbnr → Mitarbeiter.Mitarbnr; Abteilnr → Abteilung.Abteilnr; 0,1 ≤ Anteil ≤ 1,0
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll		Datenbanksysteme I Relationale Datenmodell
		Folie 3.4 

### 3.2.2 Datenmanipulation / Relationenalgebra

Relationenalgebra nach: Codd

Grundidee:

Operationen auf Relationen

⇒ Ergebnis ist wieder eine *Relation*

D.h. mengenweise Arbeit *nicht* satzweise.

#### 3.2.2.1 Mengenoperationen

$\cup \cap \setminus \times$

ABB57

## RELATIONALE ALGEBRA

Die relationale Algebra ist die grundlegende Datenmanipulationssprache zum ursprünglichen Relationenmodell und wurde gleichfalls von E.F. Codd beschrieben. Sie ist eine formale Sprache, die im Wesentlichen auf der Mengenalgebra basiert und von Codd um relationentypische Operationen ergänzt wurde.

Gegenstand der relationalen Algebra ist, dass sich auf eine oder mehrere Relationen spezielle Operationen definieren lassen, die als Ergebnis eine neue Relation liefern. Gleiches gilt auch, wenn diese Operationen in beliebiger Reihenfolge verknüpft und verschachtelt werden.

### Beispielrelationen:

KUNDEN (Kunden aller Vertriebsabteilungen)

Kundnr	Name	Ort	Region
112	Schmidt	München	Süd
115	Richter	Bremen	Nord
123	Meier	Dresden	Ost
222	Schulze	Berlin	Mitte
333	Müller	Berlin	Mitte
345	Kunze	Bonn	West

KUNDEN1 (Kunden der Vertriebsabteilung A)

Kundnr	Name	Ort	Region
123	Meier	Dresden	Ost
222	Schulze	Berlin	Mitte
345	Kunze	Bonn	West

KUNDEN2 (Kunden der Vertriebsabteilung B)

Kundnr	Name	Ort	Region
112	Schmidt	München	Süd
333	Müller	Berlin	Mitte
345	Kunze	Bonn	West

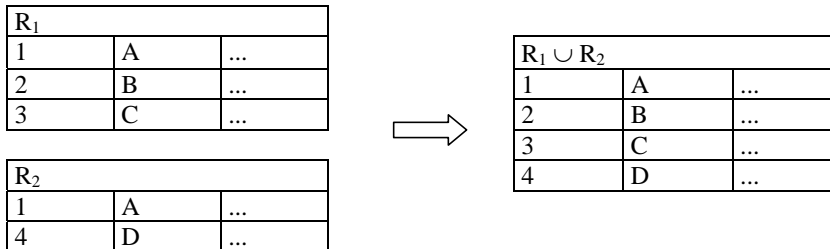
AUFTRAG

Auftrnr	Kundnr	Auftragdat	Betrag
99001	123	07.08.1999	125,50
99003	345	14.08.1999	1.500,00

**Vereinigung**  $\cup$   
 ABB58 orange  
 UNION

## VEREINIGUNG

Bei der **Vereinigung** (union)  $R_1 \cup R_2$  wird die Menge der Tupel der Relation  $R_1$  um die Menge der Tupel der Relation  $R_2$  erweitert (oder umgekehrt). Die Ergebnisrelation enthält gleiche Tupel der ersten und zweiten Relation nur einmal.



Prinzipiskizze der Vereinigung zweier Relationen

Beispiel:

Die Vereinigung  $KUNDEN1 \cup KUNDEN2$  ergibt die Relation der Kunden, die durch die Vertriebsabteilung A oder die Vertriebsabteilung B betreut werden.

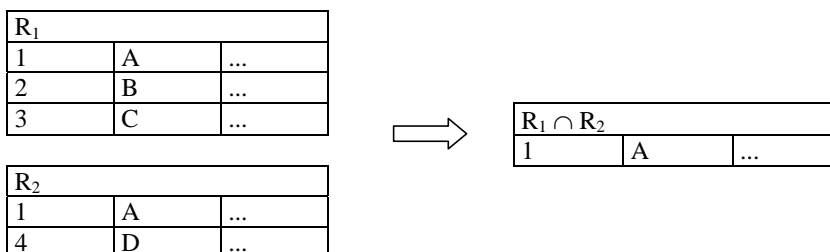
UNION (KUNDEN1, KUNDEN2)

Kundnr	Name	Ort	Region
112	Schmidt	München	Süd
123	Meier	Dresden	Ost
222	Schulze	Berlin	Mitte
333	Müller	Berlin	Mitte
345	Kunze	Bonn	West

## Durchschnitt $\cap$ ABB58 grün INTERSECTION

### DURCHSCHNITT

Der **Durchschnitt** (intersection)  $R_1 \cap R_2$  ermittelt gleiche Tupel aus zwei Relationen und enthält jedes identische Tupel nur einmal (siehe Abb.).



Prinzipiskizze des Durchschnitts zweier Relationen

Beispiel:

Der Durchschnitt  $KUNDEN1 \cap KUNDEN2$  ergibt die Relation der Kunden, die von der Vertriebsabteilung A und der Vertriebsabteilung B betreut werden.

INTERSECTION (KUNDEN1, KUNDEN2)

Kundnr	Name	Ort	Region
345	Kunze	Bonn	West

## Differenz \

 $R_1 \setminus R_2$  ABB 58 lila

Bedingung für  $\cup, \cap, \setminus$  (Vereinigungsverträglichkeit):

- Anzahl an Attributen ist gleich
- unzugeordnete Attribute besitzen gleiche Domain (Domainverträglichkeit)

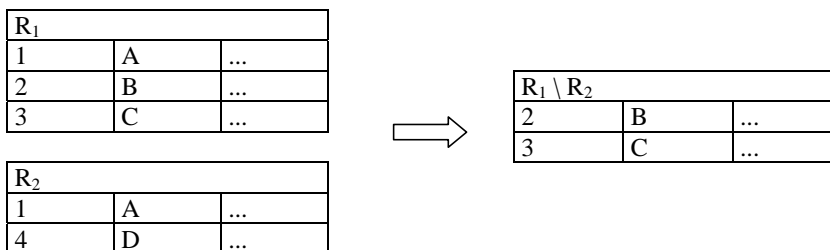
$$R_1 \cup R_2 = R_2 \cup R_1$$

$$R_1 \cap R_2 = R_2 \cap R_1$$

$$R_1 \setminus R_2 \neq R_2 \setminus R_1$$

DIFFERENCE

### DIFFERENZ

Die **Differenz** (difference)  $R_1 \setminus R_2$  bildet eine Relation, die alle Tupel der Relation  $R_1$  abzüglich der Tupel der Relation  $R_2$  enthält.


Prinzipskizze der Differenz zweier Relationen

Beispiel:

Die Differenz  $KUNDEN \setminus KUNDEN1$  ergibt die Relation der Kunden, die nicht von der Vertriebsabteilung A betreut werden.

DIFFERENCE (KUNDEN, KUNDEN1)

Kundnr	Name	Ort	Region
112	Schmidt	München	Süd
115	Richter	Bremen	Nord
333	Müller	Berlin	Mitte

## Kartesisches Produkt $\times$

 $R_1 \times R_2$ 

Ergebnisrelation enthält:

- alle Attribute aus  $R_1$  und  $R_2$ .
- alle Kombinationen an Tupeln aus  $R_1$  und  $R_2$ .

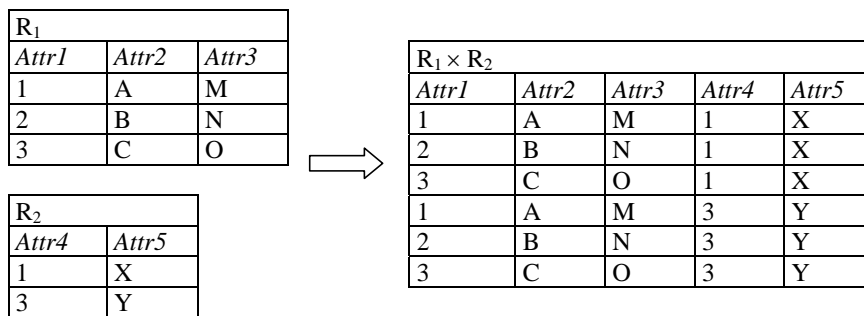
ABB 59



### KARTESISCHES PRODUKT

Die Operation des kartesischen Produktes wurde bereits bei der Definition des relationalen Datenmodells eingeführt. Dabei wurde eine Relation als Teilmenge des kartesischen Produktes von mehreren Mengen gebildet.

Das **kartesische Produkt** (product) aus zwei Relationen wird gebildet, indem jedes Tupel der ersten Relation mit jedem Tupel der zweiten Relation kombiniert wird. Alle Attribute der beteiligten Relationen werden vollständig in die Ergebnisrelation übernommen. Dabei wird jede Kombinationsmöglichkeit der beiden Relationen gebildet.



Prinzipskizze des kartesischen Produktes

In der Abbildung entsteht bei der Bildung des kartesischen Produktes aus einer Relation mit 3 Tupel und einer Relation mit 2 Tupel eine Ergebnisrelation, die 6 Tupel enthält ( $3 \times 2$ ). Analog würde das kartesische Produkt von zwei Relationen mit beispielsweise 100 Tupel in der einen und 50 Tupel in der anderen Ausgangsrelation zu 5000 Tupel in der Ergebnisrelation führen. Diese Vervielfachung der Tupel in der Ergebnisrelation des kartesischen Produktes ist ein typisches Merkmal dieser Operation.

Beispiel:

Es soll das kartesische Produkt zwischen der Relation KUNDEN1 und AUFTRAG gebildet werden. Die Ergebnisrelation enthält alle Attribute der beiden Relationen und eine Kombination der Tupel der Relation KUNDEN1 mit jedem Tupel der Relation AUFTRAG.

PRODUCT (KUNDEN1, AUFTRAG)

Kundnr	Name	Ort	Region	Auftrnr	Kundnr	Auftragdat	Betrag
123	Meier	Dresden	Ost	99001	123	07.08.1999	125,50
222	Schulze	Berlin	Mitte	99001	123	07.08.1999	125,50
345	Kunze	Bonn	West	99001	123	07.08.1999	125,50
123	Meier	Dresden	Ost	99003	345	14.08.1999	1.500,00
222	Schulze	Berlin	Mitte	99003	345	14.08.1999	1.500,00
345	Kunze	Bonn	West	99003	345	14.08.1999	1.500,00

### 3.2.2.2 Relationale Operationen

**Projektion** Spaltenauswahl


PROJ

ABB 60 grün

### PROJEKTION

Die folgenden Operationen bringen jene Erweiterungen, die die Relationenalgebra von der gewöhnlichen Algebra unterscheiden. Zunächst sollen die zwei Operationen Projektion und Selektion erläutert werden, die in ihrer Grundform auf eine Relation anzuwenden sind.

Durch eine **Projektion** werden bestimmte Attribute einer Relation ausgewählt. Bei der Darstellung in Tabellenform entspricht dies der Auswahl von Spalten. Das Ergebnis der Projektion ist selbst wieder eine Relation.

R					PROJ(R, <Attr1, Attr3>)	
Attr1	Attr2	Attr3	Attr4		Attr1	Attr3
1	A	A	W		1	a
2	B	B	X		2	b
3	A	C	Y		3	c
4	C	D	Z		4	d

Prinzipskizze der Projektion

Wird die Projektion auf Nichtschlüsselattribute geführt, müssen gleichzeitig alle jetzt mehrfach vorhandenen gleichen Tupel bis auf je eines ebenfalls gestrichen werden.

Beispiel:

Auf die Relation der Kunden aller Vertriebsabteilungen soll eine Projektion ausgeführt werden, die als Ergebnis die Namen und Wohnorte aller Kunden liefert:

PROJ (KUNDEN, <Name, Ort>)	
Name	Ort
Schmidt	München
Richter	Bremen
Meier	Dresden
Schulze	Berlin
Müller	Berlin
Kunze	Bonn

**Selektion** Tupelauswahl (laut Bedingung)

REST

ABB 60 orange

### SELEKTION

Die **Selektion** (restriction) wählt alle Tupel in einer Relation aus, die einer bestimmten Bedingung genügen. In der Tabellendarstellung führt die Selektion zu einer Auswahl von Zeilen. Die relationale Operation Selektion darf nicht mit dem SQL-Befehl SELECT verwechselt werden.

R			
Attr1	Attr2	Attr3	Attr4
1	A	A	W
2	B	B	X
3	A	c	Y
4	C	d	Z

→

REST(R, Attr2='A')			
Attr1	Attr2	Attr3	Attr4
1	A	a	W
3	A	c	Y

Prinzipskizze der Selektion

Eine Bedingung kann sich auf ein oder mehrere Attribute beziehen. Bei der Selektion werden die entsprechenden Merkmalsausprägungen überprüft und jedem Tupel der Relation die Aussage „wahr“ oder „falsch“ zugeordnet. Als Vergleichsoperator innerhalb der Bedingung kommen dabei =, <, <=, >, >= sowie „ungleich“ in Frage. Mehrere Bedingungen sind untereinander mit den logischen Operatoren „UND“ (AND), „ODER“ (OR) und „NICHT“ (NOT) verknüpfbar.

Beispiel:

Für die Relation der Kunden aller Vertriebsabteilungen sollen die zwei Bedingungen Region=„Ost“ und Region=„Mitte“ im Rahmen einer Selektion verknüpft werden. Dann ergibt das Ergebnis der Selektion alle Kunden, die entweder in der Region Ost oder der Region Mitte wohnen.

REST (KUNDEN, Region=„Ost“ OR Region=„Mitte“)

Kundnr	Name	Ort	Region
123	Meier	Dresden	Ost
222	Schulze	Berlin	Mitte
333	Müller	Berlin	Mitte

**Verbund** Verbindung zwischen zwei Relationen bezüglich der Gleichheit der Attributwerte in einer Verbindungsspalte

JOIN

intern:

- 1.) Kartesisches Produkt der Relation
- 2.) auf Ergebnisrelation Selektion nach Gleichheit der Werte in der/den Verbindungsspalten

Merkmale des JOIN:

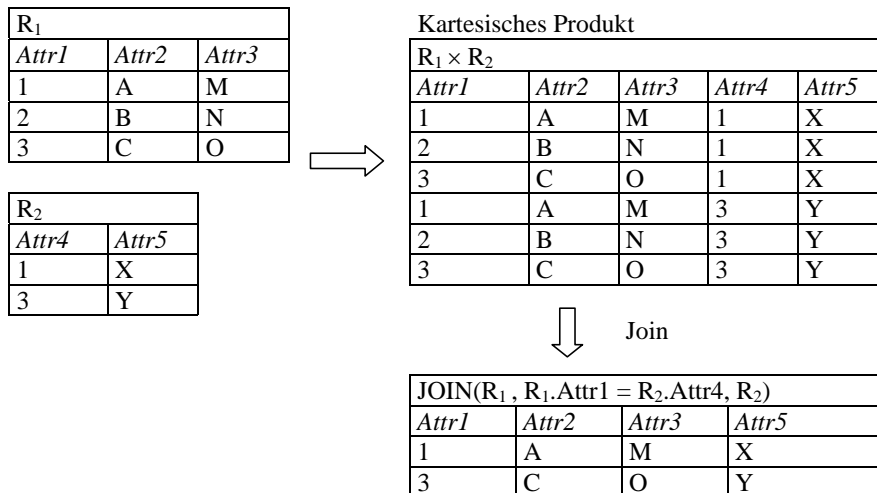
- Attribute über die den JOIN ausgeführt wird, müssen
  - keine Schlüsselspalten sein
  - gleiche Domain besitzen
  - nicht die gleichen Namen besitzen

Jede Relation ist mit jeder Relation via JOIN verbindbar (auch mit sich selbst).

### VERBUND (JOIN)

Mit dem Verbund (Join) werden Relationen miteinander verknüpft. Dabei werden zwei Relationen ähnlich wie beim kartesischen Produkt zusammengefügt, allerdings nur für solche Tupel, in der zwei bestimmte Attributwerte in einer gewissen Beziehung zueinander stehen.

Der sogenannte Natürliche Join (Natural Join) wird genau wie der Equi-Join gebildet. Der Unterschied besteht jedoch darin, daß die Ergebnisrelation gleiche Attributspalten nur einmal beinhaltet. Die in der Abbildung noch doppelt vorhandenen (gleiche) Attribute Attr1 und Attr4 werden in die Ergebnisrelation des Natural-Join nur einmal aufgenommen.



Prinzipiskizze des Natural-Join

Beispiel:

Die Operation des Natural-Join soll genutzt werden, um die Relation KUNDEN1 mit der Relation AUFTRAG zu verbinden. Als Join-Attribut kommt nur die Kundnr in Frage, da beide Attribute die gleiche Domäne haben. Eine Verbindung der Tupel wird nur möglich, wenn der Wert der Kundnr der Relation KUNDEN1 gleich dem Wert der Kundnr in der Relation AUFTRAG ist. Die Ergebnisrelation enthält alle Attribute der beiden Relationen, wobei das gleiche Attribut Kundnr nur einmal erscheint.

JOIN (KUNDEN1, KUNDEN1.Kundnr=AUFTRAG.Kundnr, AUFTRAG)						
Kundnr	Name	Ort	Region	Auftrnr	Auftragdat	Betrag
123	Meier	Dresden	Ost	99001	07.08.1999	125,50
345	Kunze	Bonn	West	99003	14.08.1999	1.500,00

## 3.3 Normalformenlehre

Ziele der Normalisierung:

- Vermeidung unerwünschter Abhängigkeiten beim Ändern, Löschen und Einfügen
- Reduzierung der Umbildung von Relationen bei Einführung neuer Attribute
- Erhöhung der Transparenz und Aussagekraft für den Nutzer (Trennung der unterschiedlichen Konzepte der realen Welt)
- Gewährung der Korrektheit der Datenbank (zu jedem Zeitpunkt)

Vorteile der Normalisierung:

- Sicherung von relativ einfachen, überschaubaren und einfach handhabbaren Relationen
- Beseitigung von Update-/Insert- und Delete-Anomalien
- Einfachere Überprüfung von Konsistenzbedingungen

Nachteile:

- größere Redundanz (Schlüsselredundanz)
- höherer Aufwand bei komplexen Auswertungen

**Codd** (1970)

Normalform (NF):  $1. NF \Rightarrow 2. NF \Rightarrow 3. NF \Rightarrow 4. NF \Rightarrow 5. NF$   
⏟  
praktisch relevant

### 3.3.1 1. Normalform

Normalformen (1 bis 3) nach Codd

Erste Normalform

Eine Relation ist in der **ersten Normalform** (1. NF), wenn alle Attribute nur atomare Werte enthalten. Das bedeutet, dass in der Relation keine Wiederholgruppen vorhanden sein dürfen, die selbst Relationen sein können.

Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Relationale Datenmodell	Folie 3.5
---	--	-----------

⇒ Relation

- atomare Werte
- PS erweitern

### 3.3.2 2. Normalform

**Normalformen (1 bis 3) nach Codd**

**Zweite Normalform**

Eine Relation ist in der **zweite Normalform** (2. NF), wenn sie sich in der ersten Normalform befindet und zusätzlich jedes Nichtsschlüsselattribut voll funktional vom Gesamtschlüssel abhängig ist, nicht aber von einzelnen Schlüsselteilen.

**Funktionale Abhängigkeit**

In einer Relation  $R(A, B)$  ist das Attribut (bzw. die Attributkombination)  $B$  von dem Attribut (bzw. der Attributkombination)  $A$  **funktional abhängig**, falls zu jedem Wert des Attributs  $A$  genau ein Wert des Attributs  $B$  gehört.

**Darstellung:  $R.A \rightarrow R.B$**


**Volle funktionale Abhängigkeit**

In einer Relation  $R(S1, S2, B)$  ist das Attribut (bzw. die Attributkombination)  $B$  von den Attributen  $S1, S2$  **voll funktional abhängig**, wenn  $B$  von den zusammen-gesetzten Attributen  $(S1, S2)$  funktional abhängig ist, aber nicht von einem einzelnen Attribut  $S1$  oder  $S2$ .

**Darstellung:  $R.S1, R.S2 \rightarrow R.B$**

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

Datenbanksysteme I  
Relationale Datenmodell

Folie 3.6 


Abhängigkeiten:

PS	Nichtschlüssel-Attribute	2. NF
<u>Mitnr</u> , <u>Projnr</u>	Anteil	MiPro
<u>Mitnr</u>	Name, Beruf, Gehalt, Abtnr, Abtbez	Mitarbeiter
<u>Projnr</u>	Projbez	Projekt

⇒ Zerlegung

- volle funktionale Abhängigkeit

### 3.3.3 3. Normalform

<p style="text-align: center;"><b>Normalformen (1 bis 3) nach Codd</b></p> <p style="text-align: center;"><b>Dritte Normalform</b></p> <p>Eine Relation ist in der <b>dritten Normalform</b> (3. NF), wenn sie sich in der zweiten Normalform befindet und zusätzlich jedes Nichtsschlüsselattribut nicht transitiv von einem Schlüsselattribut abhängig ist.</p> <p><b>Transitive Abhängigkeit</b></p> <p>In einer Relation <math>R(S, A, B)</math> ist das Attribut B vom Attribut S (Schlüssel), der auch ein zusammengesetzter Schlüssel sein kann, <b>transitiv abhängig</b>, wenn A von S funktional abhängig ist, S jedoch nicht von A und B von A funktional abhängig ist.</p> <p style="text-align: center;"><b>Darstellung: <math>R.S \rightarrow R.A \rightarrow R.B</math> (<math>R.A \not\rightarrow R.S</math>)</b></p> <p>Transitive Abhängigkeit ist immer eine mehrfache Abhängigkeit über mehrere Attribute.</p>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Relationale Datenmodell	Folie 3.7 

für Mitarbeiter (M): ( $x \rightarrow y$ : von x kann man auf y schließen)

$M.Mitnr \rightarrow M.Abtnr$

$M.Abtnr \not\rightarrow M.Mitnr$

$M.Abtnr \rightarrow M.Abtbez$

Also:

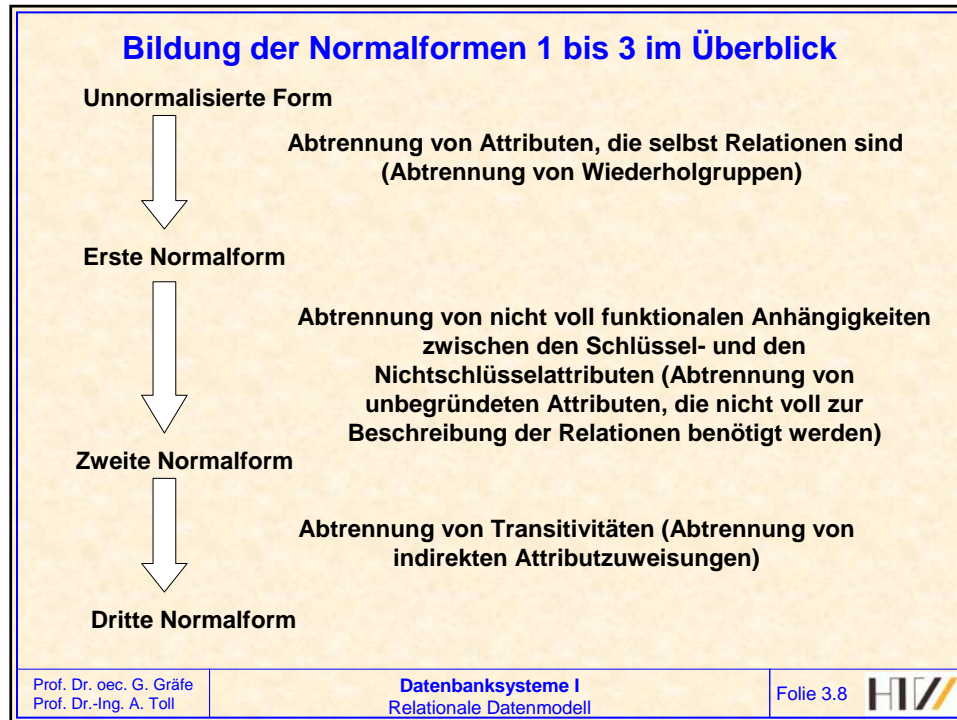
$M.Mitnr \rightarrow M.Abtnr \rightarrow M.Abtbez$

aber:

$M.Abtnr \not\rightarrow M.Mitnr$

$\Rightarrow$  weitere Zerlegung

Abtnr  $\rightarrow$  weitere Tabelle Abteilung mit PS=Abtnr.





## BEISPIEL NORMALISIERUNG

Die Datenbasis eines Handelsunternehmens soll in einer Datenbank zentralisiert werden. Eine Analyse ergibt die folgenden Feststellungen:

Für einen Mitarbeiter ist die Personalnummer (Pnr), sein Name und Familienstand (Fst) erfasst. Er ist in einer Abteilung tätig, für die eine Abteilungsnummer (Anr) und ein Abteilungsname (Aname) geführt werden. Der Mitarbeiter verkauft eine bestimmte Anzahl (Menge) unterschiedlicher Artikel. Jeder Artikel besitzt eine Artikelnummer (Artnr), eine Artikelbezeichnung (Artbez) und einen Verkaufspreis (Preis). Jeder Mitarbeiter des Unternehmens kann jeden Artikel verkaufen.

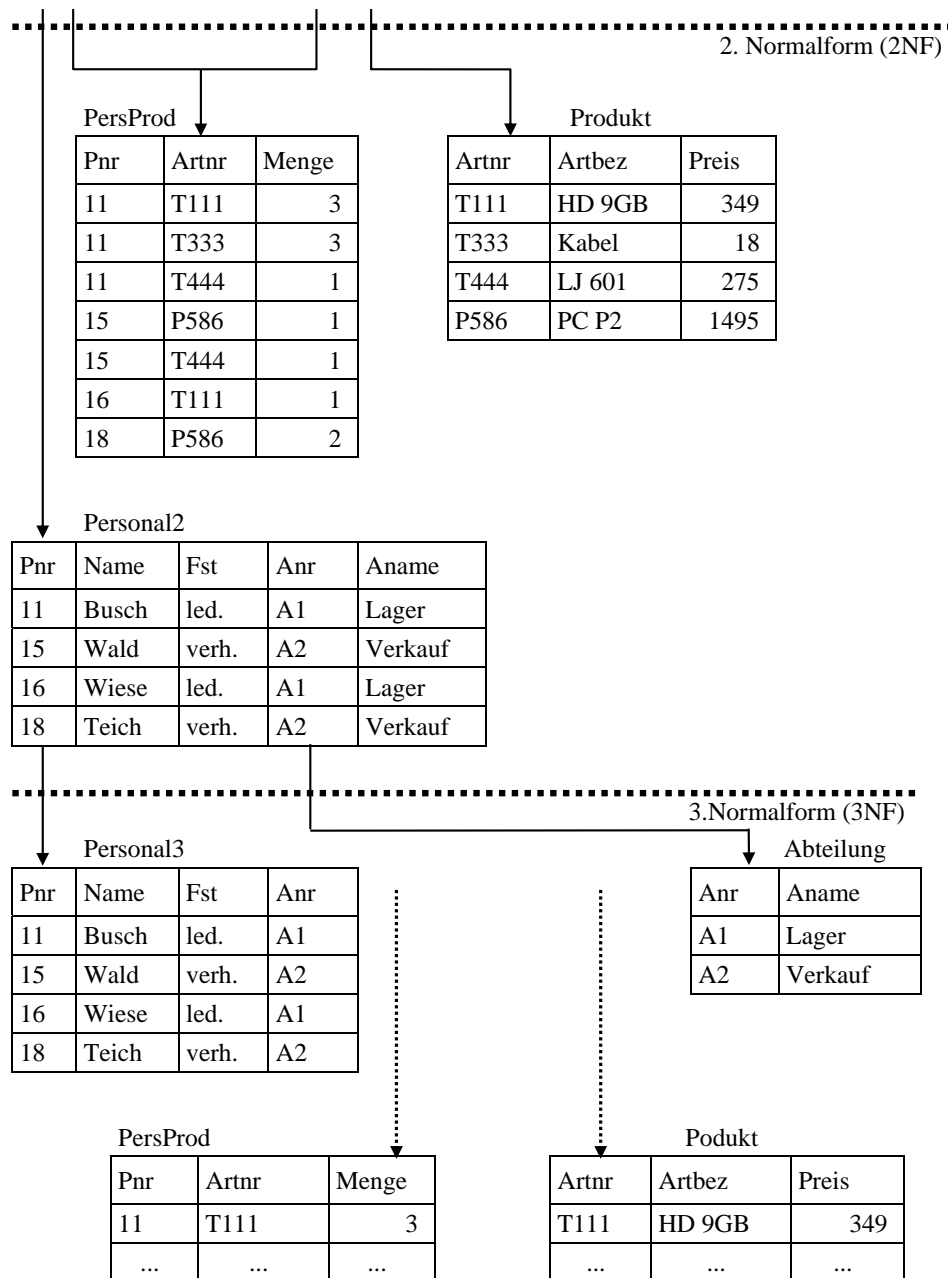
Pnr	Name	Fst	Artnr	Artbez	Preis	Menge	Anr	Aname
11	Busch	led.	T111	HD 9GB	349	3	A1	Lager
			T333	Kabel	18	3		
			T444	LJ 601	275	1		
15	Wald	verh.	P586	PC P2	1495	1	A2	Verkauf
			T444	LJ 601	275	1		
16	Wiese	led.	T111	HD 9GB	349	1	A1	Lager
18	Teich	verh.	P586	PC P2	1495	2	A2	Verkauf

Sie sehen vorstehend den ersten, völlig unstrukturierten Lösungsansatz zum Aufbau der Datensätze. Entwickeln Sie eine bessere Lösung, indem Sie schrittweise den Prozess der Normalisierung von der ersten bis zur dritten Normalform (3NF) durchführen.

Personall

1. Normalform (1NF)

Pnr	Name	Fst	Artnr	Artbez	Preis	Menge	Anr	Aname
11	Busch	led.	T111	HD 9GB	349	3	A1	Lager
11	Busch	led.	T333	Kabel	18	3	A1	Lager
11	Busch	led.	T444	LJ 601	275	1	A1	Lager
15	Wald	verh.	P586	PC P2	1495	1	A2	Verkauf
15	Wald	verh.	T444	LJ 601	275	1	A2	Verkauf
16	Wiese	led.	T111	HD 9GB	349	1	A1	Lager
18	Teich	verh.	P586	PC P2	1495	2	A2	Verkauf




## 3.4 Vergleich relationaler DBMS

<b>Codd'sche Regeln I</b>		
<p>Auf dem relationalen Datenmodell aufbauende DBMS müssen nach Codd folgende Regeln genügen:</p> <ol style="list-style-type: none"> <li><b>1. Informationsregel</b> Alle Informationen in einer relationalen Datenbasis sind auf genau eine Weise dargestellt, durch Werte in Tabellen.</li> <li><b>2. Identifizierung</b> Jedes Objekt einer relationalen Datenbasis ist durch die Werte seiner Primärschlüsselattribute eindeutig identifiziert. Der Primärschlüssel, der beim Kreieren der Tabelle deklariert wird, ist eine Spalte oder eine Kombination von Spalten.</li> <li><b>3. Nullwerte</b> In einer relationalen Datenbasis wird jedes Datenelement mit unbekanntem Wert durch denselben Nullwert repräsentiert. Dieser Wert ist unabhängig vom Domänen- oder Datentyp. Es muss möglich sein, Nullwerte für bestimmte Attribute zu verbieten.</li> <li><b>4. Data-Dictionary</b> Die Meta-Daten werden auf der logischen Ebene wie gewöhnliche Daten behandelt, so dass dieselbe DML für Abfragen verwendet werden kann.</li> </ol>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Relationale Datenmodell	Folie 3.9 

NULL = missing value (kein Wert)

≠ ''

≠ ∅

<b>Codd'sche Regeln II</b>		
<ol style="list-style-type: none"> <li><b>5. Umfassende Abfragensprache</b> Ein relationales System unterstützt mehrere Sprachen (z.B. SQL, QBE). Es muss jedoch eine Sprache geben, deren Anweisungen in einer exakt definierten Syntax verfügbar sind, und die alle folgenden Einrichtungen unterstützt: <ul style="list-style-type: none"> <li>- Tabellen-Definition</li> <li>- View-Definition</li> <li>- Datenmanipulation (Unterstützg der Operatoren der Relationenalgebra)</li> <li>- Integritätsregeln</li> <li>- Autorisierung</li> <li>- Transaktionen-Verwaltung (Commit, Rollback)</li> </ul> </li> <li><b>6. View-Update</b> Alle Views, die theoretisch änderbar sind, müssen mit der DML änderbar sein.</li> <li><b>7. Update-Level</b> Update-, Insert-, Delete-Operationen müssen auf einem Niveau verfügbar sein, das dem System die Möglichkeit der Optimierung lässt.</li> <li><b>8. Physische Datenunabhängigkeit</b></li> <li><b>9. logische Datenunabhängigkeit</b></li> </ol>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Relationale Datenmodell	Folie 3.10 

### Codd'sche Regeln III

10. Integritätsbedingungen
  - Entity-Integrität:  
Keine Komponente des Primärschlüssels darf einen Nullwert enthalten.
  - Referentielle Integrität:  
Fremdschlüsselwerte müssen mit einem Primärschlüssel derselben Domäne korrespondieren, wobei die Domäne ein unterlegter „Pool“ von typbehafteten Werten ist, aus dem eine oder mehrere Spalten ihre Wertebereiche beziehen.
  - Definition zusätzlicher Integritätsregeln mit Hilfe einer Dialogsprache
  - Speicherung der Integritätsregeln im Data-Dictionary
11. Verteilungstransparenz  
Die Terminalaktivitäten und Programme sind unabhängig von der Verteilung der Daten.
12. Nicht-Unterlaufbarkeit  
Wenn ein relationales System über eine Eintupelschnittstelle verfügt, so dürfen die Integritätsregeln damit nicht unterlaufen werden. Dies gilt auch für alle anderen Nutzerschnittstellen des Systems.

# 4 Datenbanksprachen für relationale DBMS

## 4.1 Benutzergruppen und Datenbanksprachen


Benutzergruppen:  
ABB 77

**Einordnung der Datenbanksprache SQL** Einteilung der Programmiersprachen:

- klassisch nach Gen. 1-4
- weitere Einteilung

<b>Einteilung von Programmiersprachen (1)</b>		
<b>1. Generation: Maschinensprachen</b>		
<ul style="list-style-type: none"><li>➤ interne Sprache eines bestimmten Rechnertyps</li><li>➤ Formulierung des Befehlscodes und Angabe von Speicheradressen in dualer, oktaler oder hexadezimaler Form</li></ul>		
<b>2. Generation: Maschinenorientierte Sprachen (Assemblersprachen)</b>		
<ul style="list-style-type: none"><li>➤ Sprache eines bestimmten Rechnertyps</li><li>➤ Abkürzung der Befehle durch mnemotechnische Bezeichnungen (z.B. ADD, SUB, MUL, DIV) und Symbolisierung von Speicheradressen</li></ul>		
<b>3. Generation: höhere Programmiersprachen</b>		
<ul style="list-style-type: none"><li>➤ Verwendung von Anweisungen oder Folgen von Anweisungen (an Stelle der Befehle von Maschinen- und Assemblerprachen)</li><li>➤ weitgehende Rechnerunabhängigkeit und Portierbarkeit</li><li>➤ breiter Einsatz bei der Anwendungsentwicklung</li></ul>		
<b>4. Generation: nichtprozedurale Programmiersprachen</b>		

Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.1 
---	---	---

<b>Einteilung von Programmiersprachen (2)</b>		
<p><b>4. Generation: nichtprozedurale Programmiersprachen</b></p> <ul style="list-style-type: none"> <li>➤ Beschreibung der <u>auszuführenden Aktionen</u> und nicht, wie bei den prozeduralen Sprachen (=1. bis 3. Generation) , der Folge an auszuführenden Befehlen</li> <li>➤ Formulierung, <u>Was</u> ist zu tun ist, nicht Wie</li> <li>➤ weitgehende Rechnerunabhängigkeit und Portierbarkeit</li> <li>➤ weitere Bezeichnungen: deskriptive Sprache, deklarative Sprache, 4 GL = Fourth Generation Language</li> <li>➤ breiter Einsatz bei der Anwendungsentwicklung im Bereich von Datenbanken und deren Kopplung mit Programmiersprachen</li> <li>➤ <u>Bsp.:</u> <ul style="list-style-type: none"> <li>• Abfragesprache SQL (<u>S</u>tructure <u>Q</u>uery <u>L</u>anguage)</li> </ul> </li> </ul>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.2 

### 3. Generation of Language (GL) (prozedural)

```

1 opne (buecher);
2 while (not EOF(buecker)){
3   read (buch);
4   if (buch.leihfrist > 21)
5     print (buch.titel);
6 }
7 close (buecher);
  
```

⇒ WIE

### 4. GL (descriptiv, *nicht* prozedural)

```

1 SELECT titel
2 FROM buecher
3 WHERE leihfrist > 21
  
```

⇒ WAS


<b>Weitere Einteilung von Programmiersprachen</b>		
<p><b>Objektorientierte Programmiersprachen</b></p> <ul style="list-style-type: none"> <li>➤ Grundprinzip: Zusammenfassung von Daten und den darauf anzuwendenden Funktionen zu Objekten (Kapselung)</li> </ul>		
<p><b>Wissensbasierte Sprachen</b></p> <ul style="list-style-type: none"> <li>➤ weitere Bezeichnung: KI-Sprachen (KI ... Künstliche Intelligenz)</li> <li>➤ Entwicklung Wissensbasierter Systeme (insbesondere Expertensysteme)</li> <li>➤ Bsp. LISP, Prolog</li> </ul>		
<p><b>Scriptsprachen</b></p> <ul style="list-style-type: none"> <li>➤ basieren auf bereits vorhandenen, in beliebiger Programmiersprachen, erstellten Programmkomponenten</li> <li>➤ oft keine strenge Datentypisierung ⇒ flexibel einsetzbar</li> <li>➤ Bsp.: - Unterstützung von WEB-Technologien: jscript, php  - Anwendungs- und Herstellerspezifische Scriptsprachen: ABAP (SAP), SQL*Forms(Oracle), Powerscript (Sybase), VBA (Microsoft)</li> </ul>		
<p><b>Beschreibungssprachen (engl.: Markup Language)</b></p> <ul style="list-style-type: none"> <li>➤ im eigentlichen Sinne keine Programmiersprachen</li> <li>➤ Beschreibung strukturierter Informationen (z.B. Druckseiten, Webseiten, Dokumenten)</li> <li>➤ Bsp.: - SGML (Standard Generalized Markup Language) für Druckerzeugnisse  - HTML (Hypertext ML), XML (Extensible ML)</li> </ul>		
<small>Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll</small>	<b>Datenbanksysteme I</b> <small>Datenbanksprachen für relationale DBMS</small>	<small>Folie 4.3</small>


## 4.2 SQL als Standardsprache für relationale DBMS

### 4.2.1 Überblick

SQL - Structured Query Language

- Structured – „oft etwas übertrieben“
- Query – „bescheiden“
- Language – „unanfechtbar“

Chronologie des SQL-Standards (1)		
<b>SEQUEL</b> <ul style="list-style-type: none"> <li>➤ 1974 durch D. Chamberlin von IBM San Jose Laboratory definiert Anfragesprache <i>Structured English Query Language</i> für den DBMS-Prototypen „System R“</li> <li>➤ 1976 Nachfolgeversion SEQUEL/2 und aus rechtlichen Gründen Umbenennung in SQL</li> </ul>		
<b>SQL86 (SQL1)</b> <ul style="list-style-type: none"> <li>➤ „Data Base Language SQL“</li> <li>➤ weitestgehend Übernahme der IBM-Spezifikationen (z.B. CREATE TABLE, SELECT, UPDATE, INSERT, DELETE, COMMIT, ROLLBACK, ...)</li> </ul>		
<b>SQL89</b> <ul style="list-style-type: none"> <li>➤ „Data Base Language SQL with Integrity Enhancements“</li> <li>➤ Einführung von Integrity Enhancement Features (z.B. DEFAULT, CHECK-Klauseln, PRIMARY KEY, REFERENCES, ...)</li> </ul>		
<b>SQL92 (SQL2)</b> <ul style="list-style-type: none"> <li>➤ umfangreiche Erweiterungen der relationalen Ausdrucksmöglichkeiten</li> <li>➤ Internationalisierung und benutzerdefinierte Zeichensätze</li> <li>➤ drei Versionen: ENTRY SQL, INTERMEDIATE SQL, FULL SQL</li> </ul>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.4 

Chronologie des SQL-Standards (2)		
<b>SQL99 (SQL3)</b> <ul style="list-style-type: none"> <li>➤ ISO/IEC-9075 – 1999, neue Struktur (Einführung von Teilen – parts) und Erweiterung</li> <li>➤ Erweiterung des bisherigen relationalen Modells durch zusätzliche prozedurale und objektorientierte Konzepte</li> <li>➤ Datentypen BOOLEAN, CLOB, BLOB</li> <li>➤ User Defined Types (UDTs)</li> <li>➤ Typ-Konstrukturen: strukturierte und Collection-Typs</li> <li>➤ Typed Tables und Views</li> <li>➤ Ereignissteuerung (trigger)</li> <li>➤ Datenbankprozeduren (stored procedures)</li> <li>➤ Einbettung von SQL in Java (SQLJ)</li> </ul>		
<b>SQL2003</b> <ul style="list-style-type: none"> <li>➤ Neue Konstrukte in SQL/Foundation</li> <li>➤ Neuer Teil SQL/XML</li> <li>➤ Call Level Interface</li> </ul>		
<hr/> <p>Von den Herstellern wurden bis heute der Standard SQL99 nur teilweise implementiert.</p>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.5 

Klassen an SQL-Befehlen:

- DDL – Data Definition Language
- DCL – Data Control Language
- DML – Data Manipulation Language

Dialekte:

z.B.:



- pl SQL  $\Rightarrow$  Oracle (Postgresql)
- t SQL  $\Rightarrow$  MS SQL-Server, Sybase (SAP)

#### 4.2.1.1 SQL-Datentypen (Auswahl)

- exakte Numerische Werte
  - INT ganze Zahl mit VZ
  - SMALLINT ganze Zahl mit VZ (kleinerer Wertebereich als INT)
  - DECIMAL (m,n) Dezimalzahl mit  $m$  Stellen, davon  $n$  exakte Nachkommastellen
- numerische Näherungswerte (Gleitkommazahl)
  - FLOAT (n) Gesamtstellenanzahl  $n$
  - DOUBLE PRECISION i.A. größerer Wertebereich als FLOAT
- Datum/Uhrzeit (abhängig vom System)
  - DATE
  - TIME
  - DATETIME
- Zeichenketten (Zeichenkette der Länge  $n$ )
  - CHAR (n) (Speichert gesamte Zeichenketten)
  - VARCHAR (n) (Speichert Zeichenkette ohne Leerzeichen)
- weitere spezielle Typen
  - BIT (n) Bitkette der Länge  $n$
  - BLOB (n) Binary-Array
  - TEXT Zeichenkette
  - CLOB (n) Zeichenkette
  - IMAGE Bilder
  - BOOLEAN Boolesche Werte
  - MONEY Währung
  - XML
  - ...

#### 4.2.1.2 Symbole der Syntaxbeschreibung:

(EBNF)

- |     |  |
|-----|--|
|     | Alternative (XOR)                      |
| [ ] | optionales Element (ein- oder keinmal) |
| { } | Gruppierungen (ein- oder mehrmals)     |
| ... | Wiederholung                           |
| < > | Sprachbeschreibende Variablen          |
| ::= | Definitionssymbol                      |

## 4.2.2 Anweisungen zur Definition (DDL-Befehle)

### 4.2.2.1 Tabellendefinition

#### Anlegen einer Tabelle

Tabellendefinition

**SQL-Befehl CREATE TABLE**

```
CREATE TABLE tab_name
( sp_name1 <Datentyp> [<column_constraint>],
  sp_name2 <Datentyp> [<column_constraint>],
  ...
  sp_name3 <Datentyp> [<column_constraint>]
  [, <table_constraint>]
)
```

---

**Beispiele für die Definition sprachbeschreibender Variablen:**

<Datentyp> ::= INT | SMALLINT | DECIMAL(m,n) | FLOAT(n) |  
DOUBLE PRECISION | DATE | TIME | CHAR(n) |  
VARCHAR(n)

<column\_constraint> ::= { NULL | NOT NULL | UNIQUE |  
PRIMARY KEY | CHECK(<condition>) }

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
Datenbanksprachen für relationale DBMS

Folie 4.6

```

1 CREATE TABLE Kunde (
2   KuNr      INT PRIMARY KEY,
3   Name      CHAR(10) NOT NULL,
4   Vorname   CHAR(10),
5   Ort       CHAR(20),
6   Gebdatum  DATETIME NULL,
7   Geschlecht CHAR(1) NULL,
8   TelNr     INT,
9   CHECK( 999 < TelNr < 10000)
10 )

```

- PRIMARY KEY: Primärschlüssel
- NULL: missing value (empty)  $\neq 0 \neq ' '$  (typübergreifend)
- NOT NULL: Werteingabe zwingend  
(wenn weder NULL noch NOT NULL da steht: wird vom System bestimmt → wird entweder NULL oder NOT NULL gesetzt. Empfehlung: für jeden Eintrag selber festlegen!)
- UNIQUE: eindeutiger Wert

```

1 CREATE TABLE Artikel (
2   ArtNr SMALLINT PRIMARY KEY,
3   Bezeichnung CHAR(15) NOT NULL,
4   Beschreibung CHAR(30) NULL,
5   EPreis DECIMAL(7,2) NOT NULL
6 )

```

```

1 CREATE TABLE Kauf (
2   Kunr INT,
3   Artnr SMALLINT,
4   Menge INT NOT NULL,
5   VPreis decimal(8,2) NOT NULL,
6   PRIMARY KEY(Kunr, Artnr),
7   FOREIGN KEY(Kunr) REFERENCES Kunde(Kunr),
8   FOREIGN KEY(Artnr) REFERENCES Artikel(Artnr)
9 )

```


Bei zusammengesetzten Primärschlüsseln muss dieser am Ende von *creat table* definiert werden. Bei einem einfachen kann dies *inline* passieren oder auch wie beim zusammengesetzten am Schluss:

```

1 CREATE TABLE Artikel (
2   Artnr SMALLINT,
3   Bezeichnung CHAR(15) NOT NULL,
4   Beschreibung CHAR(30) NULL,
5   EPreis DECIMAL(7,2) NOT NULL,
6   PRIMARY KEY (Artnr)
7 )

```

## Ändern der Tabellendefinition

Ändern der Tabellendefinition		
<b>SQL-Befehl ALTER TABLE</b>		
<ul style="list-style-type: none"> <li>➤ <b>ALTER TABLE</b> verändert die Tabellenstruktur</li> <li>➤ Mit ALTER TABLE muß sorgsam umgegangen werden, da nach ALTER TABLE eventuell VIEW's oder Embedded-SQL-Programme nicht mehr funktionieren.</li> <li>➤ Es gibt Varianten von ALTER TABLE die, <ul style="list-style-type: none"> <li>• nur mit leeren Tabellen funktionieren,</li> <li>• es zulassen, Spalten einzufügen, zu löschen oder Randbedingungen hinzuzufügen oder zu löschen.</li> </ul> </li> <li>➤ Angefügte Spalten werden mit NULL-Werten gefüllt.</li> </ul>		
Spalte anfügen: ALTER TABLE <i>tab_name</i> ADD <i>sp_name</i> <Datentyp> [NULL] [, ...]		
Datentypen ändern: ALTER TABLE <i>tab_name</i> ALTER COLUMN <i>sp_name</i> <Datentyp> [NULL]		
Spalte löschen: ALTER TABLE <i>tab_name</i> DROP <i>sp_name</i>		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.7 

**Bsp.:** Spalte Email an Tabelle Kunde anfügen.

```

1 ALTER TABLE Kunde ADD email CHAR(20) NULL

```

Fall: neue Spalte soll *NOT NULL* sein. Entweder *default*-Wert übergeben oder erst als *NULL* einführen, Inhalte befüllen und dann *NOT NULL* setzen.

## Löschen einer Tabelle

```

1 DROP TABLE tab_name

```

Beispiel:

```
1 DROP TABLE Kauf
```

Befehl wird ohne Nachfrage ausgeführt (wenn Tabelle nicht als Referenz verwendet wird)!

### 4.2.2.2 Arbeiten mit Index

(Primär-)Index dient dem schnellen Suchen, Primärschlüssel dient nur dem Identifizieren.

#### Anlegen des Index

```
1 CREATE INDEX index_name ON tab_name (index)
```

Bsp.: Zusammengesetzter Index (Kunde nach Ort und dann nach Name indexiert)

```
1 CREATE INDEX iOrtName ON Kunde (Ort ,Name)
```

#### Löschen eines (Sekundär-)Index

```
1 DROP INDEX tab_name . index_name
```

```
1 DROP INDEX Kunde . iOrtName
```

### 4.2.2.3 View-Definition

Bemerkungen zu Views

- Abfragen, die zur Laufzeit abgearbeitet werden, keine physischen Tabellen
- aber Ergebnisse können Tabellen entsprechen und so verwendet werden (z.B. SELECT view\_name ...)
- fast alle SQL-Befehle benutzbar, Ausnahmen:
  - kein UNION
  - kein ORDER BY
- gut für Realisierung des Zugriffsschutzes
- Änderungen in den Basistabellen => Änderung der Ergebnisse der View, aber Umkehrung gilt nur teilweise!
- Viewupdate => Änderung in den zugrundeliegenden Basistabellen (problematisch)
 

Einschränkungen des Viewupdates

  - Bezug nur auf eine Tabelle
  - Einbeziehung des Primärschlüssels
  - keine statistischen Funktionen
  - keine DISTINCT, GROUP BY, HAVING-Klausel

Prof. Dr. oec. G. Gräfe  
 Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
 Datenbanksprachen für relationale DBMS

Folie 4.8

Eine View kann auch Daten aus mehreren Tabellen nehmen.

## Anlegen einer View

```
1 CREATE VIEW view_name [ ( <Spaltenliste> ) ]
2 AS <select_anweisung>
```

Bsp.:

```
1 CREATE VIEW Beruf_Inf
2 AS (
3     SELECT Kunr, Name, Vorname, Ort
4     FROM Kunde
5     WHERE Beruf = 'Informatiker'
6 )
```

## Löschen einer View

```
1 DROP VIEW view_name
```

Bsp.:

```
1 DROP VIEW Beruf_Inf
```

## Zugriff auf eine View

```
1 SELECT Kunr, Name
2 FROM Beruf_Inf — (View Name)
```

## 4.2.3 Anweisungen zur Abfrage (DML-Befehle)

### 4.2.3.1 Standardabfrage

### Query Languages

Bei abbildungsorientierten Sprachen wie SQL werden die Daten des Definitionsbereiches (from) mit Hilfe eines Auswahlkriteriums (where) auf einen Bildbereich (select) abgebildet.

---

**Vergleichsoperatoren:**

=	gleich
!=, <>	nicht gleich
<, >	größer, kleiner als
>=, <=	größer gleich, kleiner gleich
!>, !<	nicht größer, nicht kleiner als

**Wildcards zur Patternsuche:**

% ...	beliebiges Zeichen	[ ] ...	im Bereich enthalten
_ ...	genau ein Zeichen	[^] ...	nicht im Bereich enthalten

**Logische Operatoren:**

NOT
AND
OR

↓ abnehmende Priorität

Prof. Dr. oec. G. Gräfe  
Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
Datenbanksprachen für relationale DBMS

Folie 4.9

1	<b>SELECT</b> Bildbereich	<b>&lt;Attributliste&gt;</b>	Was?
2	<b>FROM</b> Definitionsbereich	<b>&lt;Relation&gt;</b>	Wovon?
3	<b>WHERE</b> Auswahlkriterien	<b>&lt;Bedingungen&gt;</b>	Bedingung?

### 4.2.3.2 Einfache Abfrage auf eine Relation

```

1 SELECT < attributliste >
2 FROM < tab_name > | < view_name >
3 [ WHERE < auswahlbedingung > ]

```

einfachste Abfrage:

```

1 SELECT * — * ... alle Attribute (Spalten)
2 FROM Kunde

```

Beim Programmieren am besten nicht \* verwenden. Nur für's Abfragen. \* ist Fehleranfällig bei Veränderungen!

```

1 SELECT Name, Vorname
2 FROM im15s12345.dbo.Kunde — (Datenbank.Eigentümer.Tabellenname)

```

```

1 SELECT Name, Vorname
2 FROM Kunde
3 WHERE Ort= 'Dresden'

```

### Abfrage mit Vergleichs- und logischen Operatoren

```

1 SELECT * FROM Kunde
2 WHERE Geschl = 'M'
3
4 SELECT * FROM Kunde
5 WHERE (Ort= 'Pirna' OR Ort= 'Dresden') AND Beruf = 'Projektant'
6
7 SELECT * FROM Artikel
8 WHERE NOT Artnr = '1234' — Alternativ: Artnr != '1234'

```

### Abfrage mit Patternsuche

```

1 ... WHERE <spaltenname> LIKE 'muster'

```

```

1 SELECT * FROM Kunde
2 WHERE Name LIKE '%GmbH'
3
4 SELECT * FROM Kunde
5 WHERE Name LIKE 'M__er' — sucht bspw. Maier, Meyer oder Mayer

```

### Funktion

```

1 SUBSTR ( <spaltenname>, <startpos>, <länge> )

```

```

1 SELECT * FROM Kunde
2 WHERE SUBSTR(Name, 2,2)= 'ei' — ACHTUNG: substring fängt bei 1 an zu zählen: 1. Zeichen
   ist nicht an der Stelle 0, sondern tatsächlich 1
3 — das gleiche wie:
4 WHERE Name LIKE '_ei%'

```

### Bereich

```

1 SELECT * FROM Kunde
2 WHERE Name LIKE '[E-N]%' — alle Namen die mit Buchstaben zwischen E und N beginnen
3 WHERE Name LIKE '[^E-N]%' — alle, die nicht mit Buchstaben zwischen E und N beginnen

```

## Attributzuweisung in Abfrage

```
1 AS <spaltenname>
```

```
1 SELECT Kunr, Name, AS Kundenname
2 FROM Kunde
3
4 SELECT Kunr, Vorname | ', ' | Vorname AS Kundenname
5 FROM Kunde
```

weiteres würde liefern:

Kunr	Kundenname
123	Maier, Uwe
234	Adler, Sabine
...	

```
1 SELECT Kunr, Artnr, Menge*vPreis AS Umsatz
2 FROM Kauf
```

liefert:

Kunr	Artnr	Umsatz
123	1234	3600
234	5555	500
...		

## Datumsfunktionen

1.)

```
1 DAY(), MONTH(), YEAR(), HOUR(), ...
```

```
1 SELECT YEAR(GebDat)
2 FROM Kunde
```

2.)

```
1 DATEDIFF ( <datepart>, <begin>, <end> ) — <datepart>: dd, mm, yy
```

```
1 SELECT DATEDIFF (dd, GETDATE(), '24/12/2016') — Achtung Formatierung des Datums ist
von DBMS abhängig (ob / oder . und Reihenfolge)
```

3.)

```
1 DATEADD ( <datepart>, <number>, <date> )
```

```
1 DATEADD ( dd, 50, GETDATE() )
```

### 4.2.3.3 Sortierung / Gruppenbildung

#### Sortieren/Gruppieren

**Sortierung:**

```

SELECT      <attributliste>
FROM        tab_name      [WHERE ...]
ORDER BY    sp_name1 [, sp_name2 ...] [DESC]

DESC ... absteigend (Standard ist ASC aufsteigend)

```

---

**Gruppierung**

```

SELECT      <attributliste>
FROM        tab_name      [WHERE ...]
GROUP BY    sp_name1 [, sp_name_2, ...]

```

☐ Gruppierung ist die Zusammenfassung von Tupeln mit gleichen Attributwerten in Verbindung mit statistischen Funktionen.  
 Ergebnisse dürfen nur einen Wert pro auszugebender Gruppe haben, oder anders ausgedrückt:  
 Attribute der SELECT-Komponente müssen auch in der GROUP BY Komponente stehen, außer sie kommen nur innerhalb von Statistikfunktionen vor!

Prof. Dr. oec. G. Gräfe  
 Prof. Dr.-Ing. A. Toll

**Datenbanksysteme I**  
 Datenbanksprachen für relationale DBMS

Folie 4.10

#### Sortierung

```

1 SELECT * FROM Kunde
2 ORDER BY Name, Vorname
3
4 SELECT * FROM Kunde
5 ORDER BY GebDat DESC — jüngste zuerst

```

#### Gruppierung

```

1 SELECT Ort, SUM( Kredit ) AS Kreditsumme
2 — Achtung! Auswahl kann nicht auf bspw. Name erweitert werden, weil dann die Tabelle für
  einen Ort mehrere Namen hätte
3 — Ähnlich: Nicht mit bspw. Kredit erweiterbar. ABER: Möglich ist SUM(Kredit), was dann
  alle Werte für den Ort aufsummieren würde
4 FROM Kunde
5 GROUP BY Ort — Jeder Ort taucht nur einmal auf

```

### 4.2.3.4 Built-in-Funktionen

(Aggregatfunktionen, Statistikfunktionen)

```

1 SUM ( )
2 AVG ( )
3 MAX ( )
4 MIN ( )
5 COUNT ( )

```

```

1 SELECT SUM( Kredit )
2 FROM Kunde — Summe aller Kredite von allen Kunden (eine Tabelle mit nur einem Eintrag
  : der Summe)

```



```

3
4 — weiteres Bsp. siehe Gruppierung
5
6 SELECT Artnr, SUM( Menge*vPreis ) AS UmSum
7 FROM Kauf
8 GROUP BY Artnr

```

Count:

```

1 COUNT (<spaltenname>) — zählt belegte Werte einer Spalte. Einträge mit NULL werden nicht
   gezählt.
2 COUNT (*) — zählt auch NULL-Werte, also einfach alle Zeilen
3 DISTINCT — zählt keinen Wert mehrfach

```

```

1 SELECT COUNT(*) — Anzahl der Kunden
2 FROM Kunde
3
4 SELECT COUNT(DISTINCT Ort) — Anzahl der Orte ohne Dopplungen
5 FROM Kunde
6
7 SELECT DISTINCT Ort — Zählt Orte ohne Dopplungen auf
8 FROM Kunde

```

#### 4.2.3.5 Abfrage mit Bereichsgrenzen und Wertaufzählungen

**Bereich**

```

1 BETWEEN x AND y

```

```

1 SELECT * FROM Kauf
2 WHERE vPreis BETWEEN 400 AND 1000

```

**Aufzählung**

```

1 IN (<liste >)

```

```

1 SELECT * FROM Kunde
2 WHERE Beruf IN ( 'Maler', 'Schlosser', 'Tischler' )

```

#### 4.2.3.6 Einfache Abfrageschachtelung

(hinter dem WHERE steht eine Unterabfrage)

```

1 SELECT * FROM <tab_name>
2 WHERE <spaltenname> [ = | IN ] ( — = benutzen, falls Unterabfrage einzelnen Wert zurück
   gibt, IN-Operator benutzen, falls eine Wertliste durch Unterabfrage erzeugt wird
3     SELECT <sp_name>
4     FROM <tab_name>
5     [WHERE]
6 )

```

Abfrage von Name, Ort, GebDat des ältesten Dresdner Kunden:

```

1 SELECT Name, Ort, GebDat FROM Kunde
2 WHERE GebDat = (
3     SELECT MIN(GebDat)
4     FROM Kunde
5     WHERE Ort='Dresden' — genau ein Wert

```

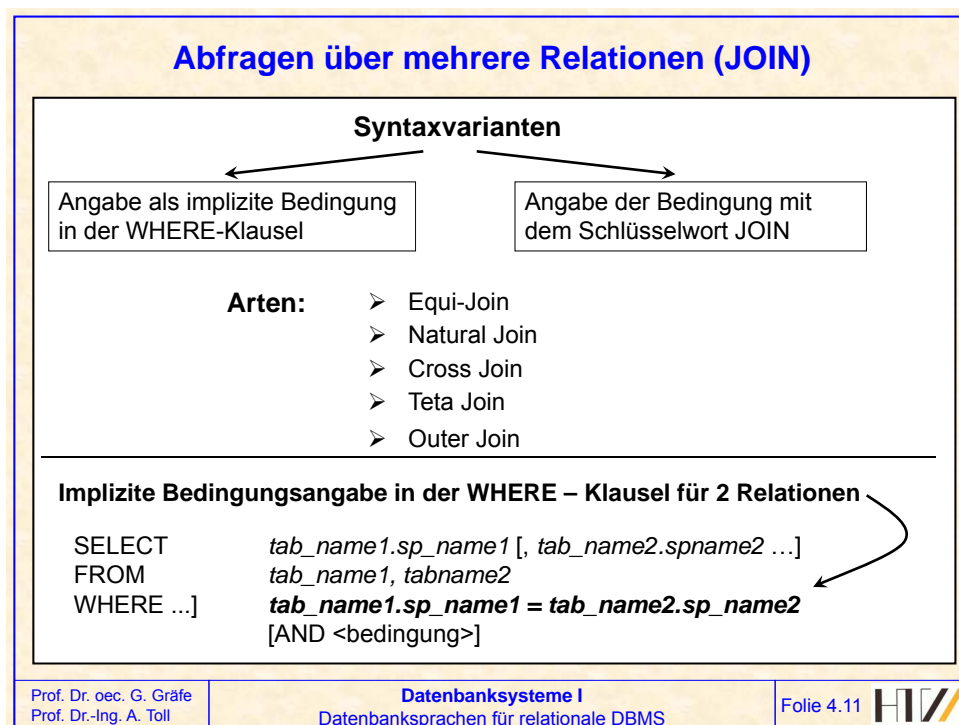
6 ) — Achtung: hätten zwei älteste am gleichen Tag Geburtstag und einer würde nicht aus  
Dresden kommen, dann würde diese mit angegeben. Also:  
7 AND Ort= 'Dresden'

Alle Kunden, die etwas gekauft haben (bzw. in Kauf eingetragen sind) aus Dresden:

```
1 SELECT * FROM Kauf
2 WHERE Kunr IN (
3   SELECT KuNr FROM Kunde
4   WHERE Ort= 'Dresden'
5 )
```

### 4.2.3.7 Abfrage über mehrere Relationen

⇒ JOIN



**Bsp.:** Anzeige der Artikeldaten und deren Käufe Mehrere Varianten:

1.) WHERE:

```
1 SELECT Artikel.Artnr , Bezeichnung , Menge , VPreis
2 FROM Artikel , Kauf
3 WHERE Artikel.Artnr = Kauf.Artnr
```

2.) mit Alias:

```
1 SELECT a.Artnr , Bezeichnung , Menge , VPreis
2 FROM Artikel a, Kauf k
3 WHERE a.Artnr = k.Artnr
```

3.) JOIN:

```
1 SELECT a.Artnr , Bezeichnung , Menge , VPreis
2 FROM Artikel a
3 JOIN Kauf k ON a.Artnr = k.Artnr
```

Zusätzlich: Name des Kunden anzeigen und nur Kunden aus Dresden:

### 1.) WHERE:

```

1 SELECT A.Artnr , Bezeichnung , Menge , U.Kunr , Name
2 FROM Artikel A , Kauf K , Kunde U
3 WHERE A.Artnr = K.Artnr      — implizite Bedingung(en)
4    AND K.Kunr = U.Kunr      — implizite Bedingung(en)
5    AND Ort = 'Dresden'      — explizite Bedingung(en)

```

### 2.) JOIN:

```

1 SELECT A.Artnr , Bezeichnung , Menge , U.Kunr , Name
2 FROM Artikel A
3 JOIN Kauf K ON A.Artnr = K.Artnr  — JOIN: implizite Bedingung(en)
4 JOIN Kunde U ON K.Kunr = U.Kunr  — JOIN: implizite Bedingung(en)
5 WHERE Ort = 'Dresden'            — WHERE: explizite Bedingung(en)

```

⇒ JOIN ist der ersten Variante zu bevorzugen, da die impliziten und expliziten Bedingungen klar und auf den ersten Blick erkennbar sind.

## 4.2.3.8 Weitere Abfragemöglichkeiten

Verknüpfung von Relationen über JOIN		
<b>Equi-Join (Inner Join):</b> Es wird nur die Gleichheit der Attributswerte zweier Tupel überprüft und dabei ausschließlich die UND-Verknüpfung verwendet. Die doppelten Attributwerte beider Tupel werden ausgegeben.		
<b>Natural Join (Natürlicher Join):</b> Analog Equijoin, zusätzlich werden doppelt auftretende Attribute eliminiert.		
<b>Cross Join (Kartesisches Produkt):</b> Alle Tupel der ersten Relation werden mit jedem einzelnen Tupel der zweiten Relation verknüpft.		
<b>Theta Join</b> Beim Theta Join werden die Join-Spalten in der WHERE-Klausel mit einem der Vergleichsoperatoren verglichen.		
<b>Outer Join</b> Auswahl auch von Tupeln, die die Bedingung mit den Join-Spalten nicht erfüllen.		
Prof. Dr. oec. G. Gräfe Prof. Dr.-Ing. A. Toll	<b>Datenbanksysteme I</b> Datenbanksprachen für relationale DBMS	Folie 4.12

### • Equi-JOIN

```

1 SELECT Kunde.* , Kauf.*
2 FROM Kunde , Kauf
3 WHERE Kunde.Kunr = Kauf.Kunr

```

### • Natural JOIN

```

1 SELECT Kunde.* , Artnr , Menge , VPreis — Kauf.*—Kunr oder vergleichbares nicht mö
   glich! Es müssen alle Spalten manuell angegeben werden
2 FROM Kunde , Kauf
3 WHERE Kunde.Kunr = Kauf.Kunr

```

```

4
5 — alternativ in JOIN-Schreibweise:
6 SELECT Kunde.*, Artnr, Menge, VPreis
7 FROM Kunde
8 JOIN Kauf ON Kunde.Kunr = Kauf.Kunr

```

#### • Kartesisches Produkt

```

1 SELECT *
2 FROM Kunde, Kauf

```

#### • Theta JOIN

```

1 SELECT *
2 FROM Kunde, Kauf
3 WHERE Kunde.Kunr != Kauf.Kunr

```

Entspricht (Kartesisches Produkt) – (Equi-Join)

#### • Outer JOIN

- Right
- Left
- Full

```

1 SELECT *
2 FROM Kunde K
3 FULL OUTER JOIN Kauf F — Schaut in beide Tabellen (Kauf und Kunde) und fügt die
4   hinzu, die keinen Zusammenhang haben
5 — bei LEFT OUTER JOIN: Schaut in die "linke" Tabelle (Kunde) und prüft, welche im
6   Zusammenhang mit Kauf nicht auftauchen (sprich: welcher Kunde nichts gekauft hat
   )
7 — bei RIGHT OUTER JOIN: Schaut in die "rechte" Tabelle (Kauf) (sprich: welcher Kauf
   keinen Kunden hat: nicht möglich)
8 ON K.Kunr = F.Kunr

```

### 4.2.3.9 Abfrageschachtelungen

Arten von Unterabfragen(UA):

- einfach/unkorreliert
  - UA wird einmal ausgeführt und in die Hauptabfrage (HA) eingesetzt
- abhängig/korreliert
  - UA wird für jeden Tupel der HA einmal ausgeführt
  - UA (innere Abfrage) bezieht sich auf eine (oder mehrere) Tabellen, die *nur* in der FROM-Klausel der HA aufgeführt sind

**Bsp.:** unkorrelierte UA

- Alle Käufe in Dresden (Kunden).

```

1 SELECT *
2 FROM Kauf
3 WHERE Kunr IN (
4   SELECT Kunr
5   FROM Kunde
6   WHERE Ort= 'Dresden'
7 )

```

- Anzeige Artnr, Bezeichnung und Einkaufspreis, sowie maximal erzielter VPreis je Artikel.

```

1 SELECT UA.ArtNr, VPreisMax, Bezeichnung, EPreis
2 FROM (SELECT ArtNr, MAX(VPreis) AS VPreisMax — Alias sehr wichtig, damit Spalte in
      HA wieder verwendet werden kann!
3       FROM Kauf
4       GROUP BY ArtNr
5 ) UA — Unterabfrage für Tabelle von max. VPreis
6 JOIN Artikel ON Artikel.ArtNr = UA.ArtNr

```

Diese Schachtelung kann beliebig weiter fortgesetzt werden.

- Anzeige aller Kunden, deren Umsatz (= Menge \* VPreis) größer als deren Kreditlimit ist.

```

1 SELECT K.*
2 FROM (SELECT Kunr, SUM(Menge*VPreis) AS Umsatzsumme
3       FROM Kauf
4       GROUP BY Kunr
5 ) UA
6 JOIN Kunde K ON UA.Kunr = K.Kunr
7 WHERE Umsatzsumme > Kredit

```

### Bsp.: korrelierte UA

- Gleiches Beispiel, wie letztes unkorreliertes.

```

1 SELECT * — hier HA
2 FROM Kunde
3 WHERE Kredit < (
4     SELECT SUM(Menge*VPreis) — ab hier UA
5     FROM Kauf
6     WHERE Kauf.Kunr = Kunde.Kunr
7 )

```

Ergebnis ist das gleiche wie in der unkorrelierten Abfrage.