



DATENBANKEN I+II

Sebastian Augspurger



KURZE VORSTELLUNG

SEBASTIAN AUGSPURGER

- WWI11AMC
- B.Sc. Wirtschaftsinformatik
- M.Sc. Innovations- und Technologiemanagement
- 3 Jahre Datenbankadministrator & Prozessmanager in Forschung und Entwicklung
- Seit 3 Jahren SAP-Support in global Supply Chain Logistik
- **Spielregeln:**
 - Jederzeit Fragen stellen
 - Teilnahme nach eigenem Ermessen

NUTZEN FÜR SIE

- Datenbanken sind essenzieller Bestandteil jeder IT-Umgebung
- Datenbankadministration/-Support sind klassische WI-Stellen
- Datenbank-Know How ist Basiswissen für WI/IT
- Big-Data als Buzzword seit 2011
- Entwicklungen in KI basieren in der Regel auf Datenbanken

BRAINSTORMING

- Was sind Datenbanken?
- Wo werden Datenbanken angewandt?
- Was sind Vorteile/Chancen von Datenbanken?
- Was sind Nachteile/Risiken von Datenbanken?
- Was sind Begriffe, die Sie interessieren?

AGENDA

1. Datenbanken im Allgemeinen
2. Relationen
3. Entity-Relation-Modell
4. Datenbankenkonzeption
5. SQL-Abfragen



1. DATENBANKEN IM ALLGEMEINEN

KAPITEL-AGENDA:

1. DATENBANKEN IM ALLGEMEINEN

- a. Basics zu Datenbanken
- b. Historie
- c. DBS & DBMS
- d. Zugriffsmöglichkeiten
- e. Übungen

1.A BASICS ZU DATENBANKEN

Definition: „Selbstständige, auf Dauer und flexiblen und sicheren Gebrauch ausgelegte Datenorganisation, die sowohl eine Datenbasis als auch eine zugehörige Datenverwaltung (DBMS) umfasst. Eine Datenbank dient dazu, eine große Menge von Daten strukturiert zu speichern und zu verwalten.“¹

¹ Gabler Wirtschaftslexikon

1.A BASICS ZU DATENBANKEN

Anforderungen an moderne Datenbanken:

- I. Skalierbarkeit
- II. Performanz auch bei großen Datenmengen
- III. Sicherstellung der Datenintegrität
- IV. Ermöglichung konkurrierender Zugriffe
- V. Strukturelle Änderungen müssen einfach zu realisieren sein
- VI. Datenschutz vor unberechtigten Zugriffen

1.A.I SKALIERBARKEIT

Zwei Arten von Skalierbarkeit:

1. Software:

Die Datenbank und das DBMS müssen in ihrer Größe variable sein (so lange es die Hardware erlaubt).

2. Hardware

Die Hardware der Datenbank muss anpassbar (erweiterbar) sein, ohne dass die Software darunter leidet.

1.A.II PERFORMANCE

Kennzahlen:

1. Verfügbarkeit (i/n oder in Zeit)
2. Lesegeschwindigkeit
3. Schreibgeschwindigkeit

Wichtig: Bottleneckanalyse!

1.A.III DATENINTEGRITÄT

Definition: „Korrektheit (Unversehrtheit) von Daten und der korrekten Funktionsweise von Systemen“¹

1. Korrekter Inhalt
2. Unmodifizierter Zustand
3. Erkennung von Modifikation
4. Temporale Korrektheit

¹ Bundesamt für Sicherheit in der Informationstechnik

1.A.IV KONKURRIERENDE ZUGRIFFE

Schutz vor gleichzeitiger Bearbeitung des gleichen Datums durch mehrere Nutzer.

Lösungen:

1. Locks auf Dateiebene
2. Sperrbit auf Datenbankebene
3. Verwaltung von 1&2 durch DBMS

1.A.V STRUKTURELLE ÄNDERUNGEN

Grundbegriff der relationalen Datenbank ist die Datenbankstruktur. Diese muss auch während des Betriebs noch änderbar sein.

CRUD-Operationen:

1. Erzeugen (Create)
2. Lesen (Read)
3. Verändern (Update)
4. Löschen (Delete)

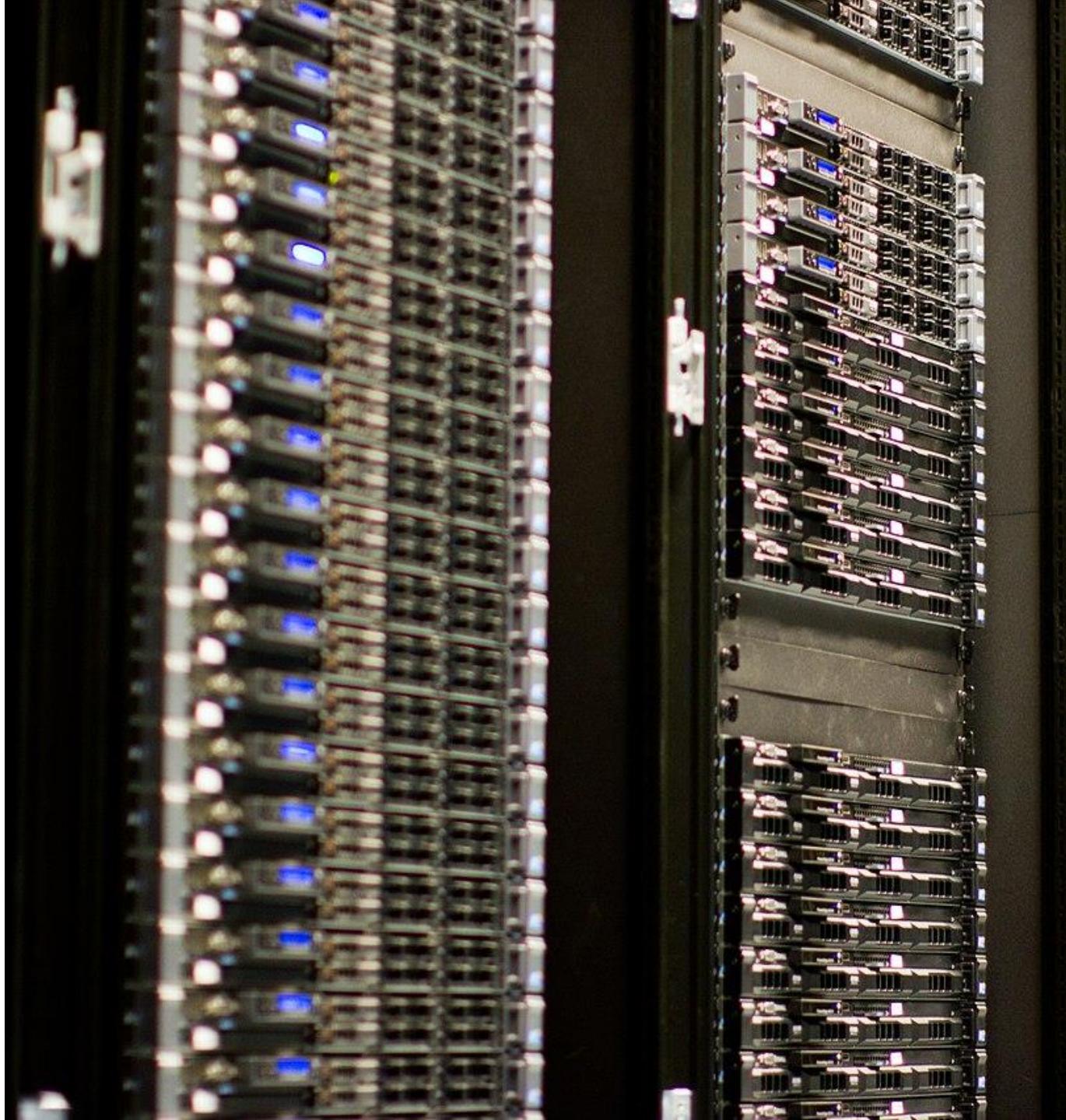
1.A.VI DATENSCHUTZ

Zwei Arten von Schutz:

1. Schutz vor Verlust, Veränderung oder Beschädigung (korreliert mit Datenintegrität)
2. Schutz vor uneingeschränkter Datenerhebung durch unautorisierten Personen

BASICS ZU DATENBANKEN: FRAGEN

1. Was sind die 4 CRUD-Operationen?
2. Was ist Skalierbarkeit?
3. Was sind die 4 Zustände für Datenintegrität?
4. Was sind Kennzahlen für DB-Performanz?
5. Wie kann man konkurrierende Zugriffe vermeiden?



1.B HISTORIE

Hardware 50er Jahre:

- Zugriff nur sequenziell möglich
- Verschleißerscheinungen
- Lange Zugriffszeiten
- Beschränkter Speicherplatz
 - Lochkarte: 60 – 100 Bytes

L	A	B	C	A	B	C	L	C	N	G	A	C	C	S	M	I	H	M	W	A	C	E	F	a	d	
Cr	D	E	F	D	E	F	Lu	Cin	S	Gr	Ag	Cf	Ct	S	M	I	H	M	W	A	C	E	F	a	d	
Li	G	H	I	G	H	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cir	K	L	M	K	L	M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
Cs	N	O	P	N	O	P	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
LS	Q	R	S	Q	R	S	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
Ke	*	*	*	*	*	*	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
RN	*	*	*	*	*	*	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
QC	g	r	i	g	r	i	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	
AV	x	i	m	x	i	m	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	
SH	n	s	p	n	s	p	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
Bo	*	*	*	*	*	*	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	

3994



1.B HISTORIE

Hardware **60er & 70er Jahre**:

- Zugriff direkt möglich
- Erstmaliger Nutzen von „Dateien“
- Schneller Zugriff
- V.a. „Write“ und „Update“ deutlich einfacher
- „Großes“ Datenvolumen
 - Erste Festplatte: 3,75MB
- Fragmentierung als Problem
- Bei Schaden potenziell großer Datenverlust



1.B HISTORIE

Hardware **90er Jahre bis heute:**

- Zugriff direkt möglich
- Schnellster Zugriff am Markt
- Erschütterungsfest
- Nahezu grenzenloser Speicherplatz
- Generieren kaum Hitze + Staubresistent
- Immer noch hohe Kosten
- Relativ neu: **In-Memory!**



1.B HISTORIE

„Software“ pre Codd (bis ca. 1965):

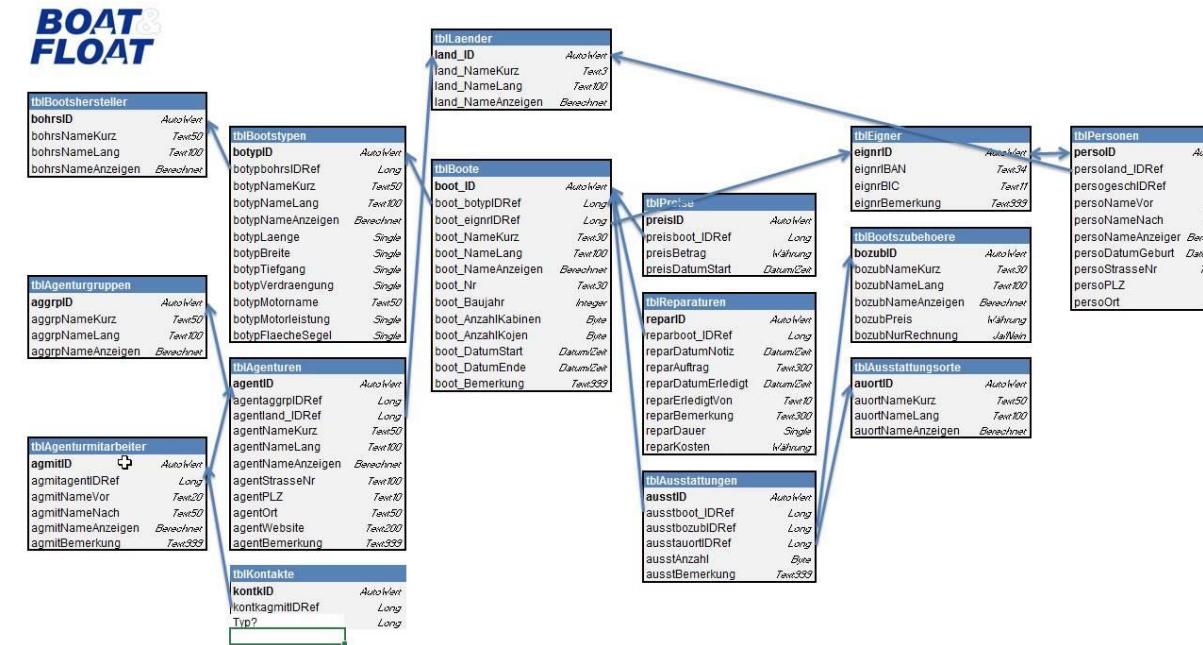
- Hierarchischer Aufbau
- Daten werden sequenziell hinterlegt
- Angepasst an Lochkarten und Magnetbänder
- Langsame Zugriffszeiten
- Sehr aufwändiger „Write“ und „Update“-Vorgang

1	121
	Meier
1.1	100
	abc
1.1.1	1.1.2005
1.2	102
	def
1.2.1	10.3.2005
1.2.2	1.2.2006
2	134
	Steffen
2.1	100
	abc
2.1.1	3.4.2007
3	155
	Huber
3.1	105
	xyz
3.1.1	4.4.2008
3.1.2	1.8.2008

1.B HISTORIE

„Software“ post Codd (ab ca. 1965):

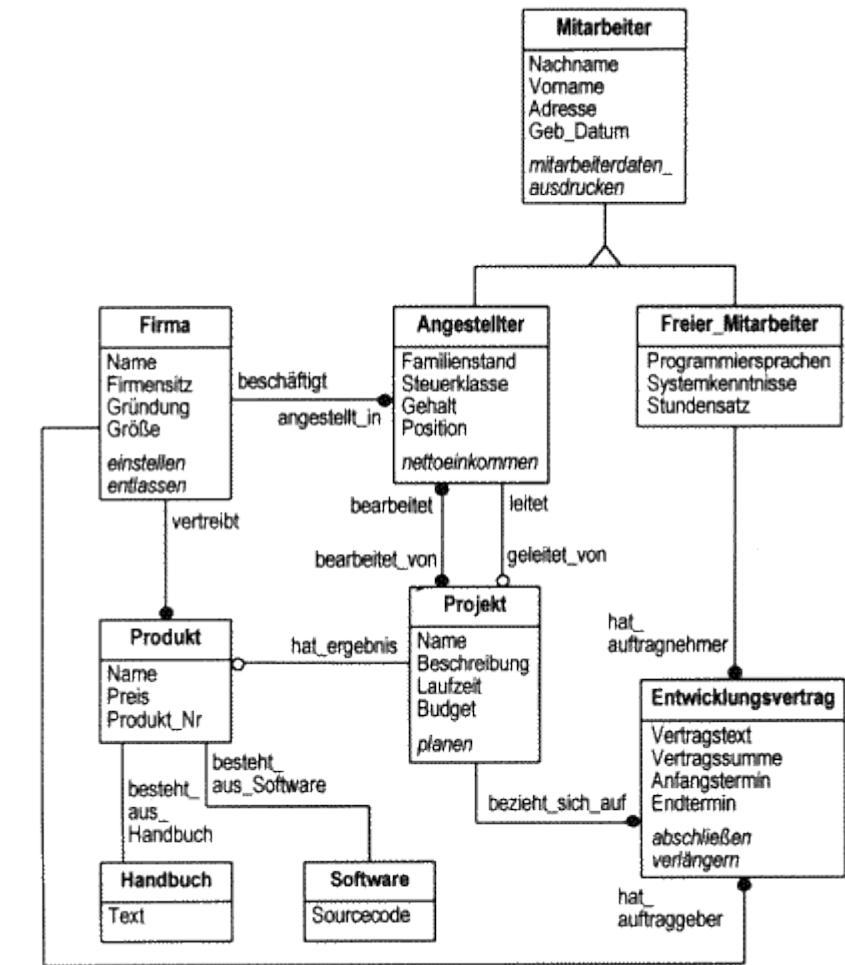
- Relationaler Aufbau
- Trennung in verschiedene Tabellen
- Aufwändiger Handhabung
- Nur bedingt schnellere Zugriffszeiten
- Deutlich einfachere „Write“ und „Update“-Vorgänge
- Marktstandard



1.B HISTORIE

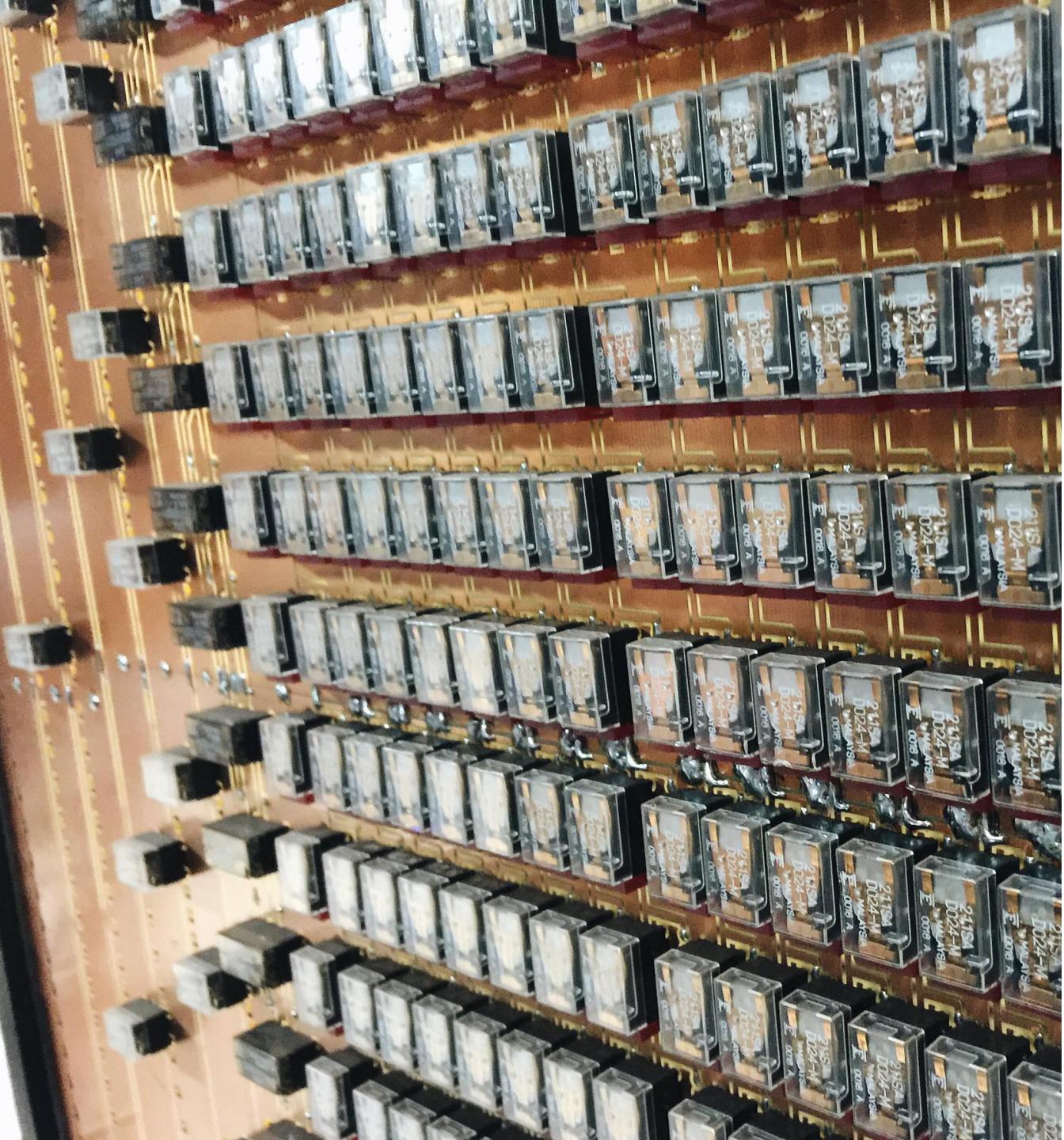
„Software“ von Codd (ab ca. 1988):

- Objektorientierter Aufbau
- Basierung von Tabellen auf Objekten
- Einfachere Anbindung an OO-Anwendungen
- Tabellenverknüpfungen nicht mehr über Join
- Keine Standardisierung
- Nischenprodukt



HISTORIE: FRAGEN

1. Was sind Nachteile von Magnetbändern?
2. Werden Magnetbänder heute noch verwendet?
3. Was ist der Unterschied zwischen einer HDD und einer SSD?
4. Was sind Vorteile von hierarchischen Strukturen?
5. Wieso haben sich objektorientierte Strukturen nicht durchgesetzt?



1.C DBS & DBMS

Grundidee:

Daten werden anhand eines vorher festgelegten Schemas gespeichert.

Es werden Daten und Schema in der Datenbank gespeichert.

Die Integrität wird erzwungen.

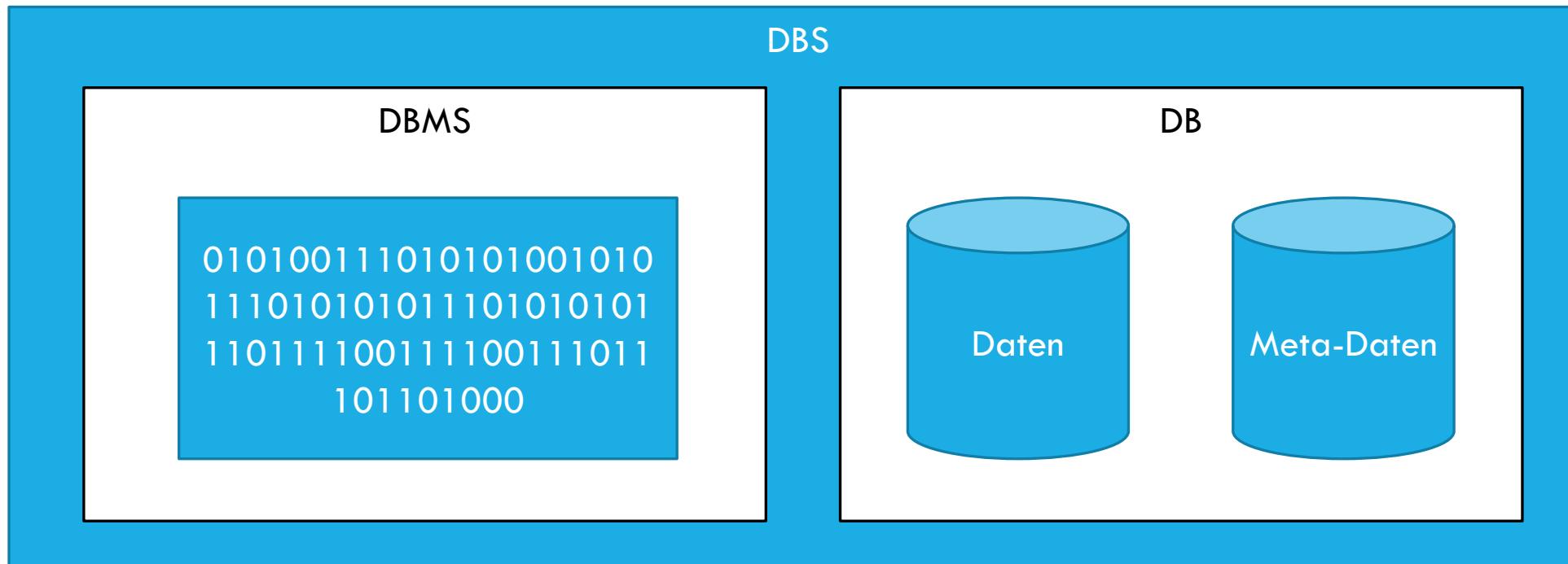
Daten werden von Anwendungen isoliert.

Unterschiedlichen Benutzern können Sichten auf die Daten bereitgestellt werden.

Konkurrierende Zugriffe von zentraler Instanz koordiniert.

1.C DBS & DBMS

Aufbau:



1.C DBS & DBMS

Begriffserklärung:

DBS: Datenbanksystem, die gesamte Systemumgebung der Datenbank

DBMS: Datenbankmanagementsystem, die programmierte Logik und Funktion der Datenbank

DB: Datenbank oder Datenbasis, der Speicherort der Daten und Metadaten

Daten: Einzelne Datensätze

Metadaten: Daten über Daten, Indizes und Hashs

1.C.I DB

DB:

Definition: Eine Datenbank ist eine physische Sammlung von Daten, die einen Ausschnitt der realen Welt beschreiben. Unter Daten verstehen wir bekannte Tatsachen, die aufgezeichnet werden können und eine implizite Bedeutung haben.

Eigenschaften von Datenbanken:

- Eine Datenbank stellt Aspekte der realen Welt dar. Änderungen in der Realität spiegeln sich in der Datenbank.
- Eine Datenbank ist eine logisch zusammenhängende Sammlung von Daten mit einer inhärenten Bedeutung.
- Eine Datenbank wird für einen bestimmten Zweck entworfen, entwickelt und mit Daten gefüllt.

1.C.II DBMS

DBMS:

Definition: Ein Datenbankmanagementsystem ist ein Softwaresystem, welches dem Benutzer das Erstellen und die Pflege einer Datenbank ermöglicht.

Verantwortlichkeiten:

1. Definition der Daten: Spezifikationen von Datentypen, Strukturen und Einschränkungen
2. Konstruktion der Daten: Speichern der Daten auf ein physisches Medium (wird hier nicht weiter betrachtet: ⇒ Vorlesung Datenbanktechnik)
3. Manipulation der Daten: CRUD-Operationen Applikationen greifen auf die Datenbank zu, indem sie Anfragen an das DBMS schicken.

1.C.II DBMS

9 Datenbankfunktionen nach Codd, die ein DBMS erfüllen muss:

- I. Integration: Einheitliche, nicht redundante Verwaltung der Informationen
- II. Operationen: Create, Read, Update, Delete (CRUD-Operationen)
- III. Katalog (Data Dictionary / Metadaten): Speichert Daten über Daten
- IV. Benutzersichten: Verschiedene Benutzer haben unterschiedliche Sichten auf die Daten
- V. Konsistenzüberwachung: Die Datenhaltung stellt die Datenintegrität sicher
- VI. Datenschutz: Vermeidung unautorisierter Zugriffe auf die Datenbasis
- VII. Transaktionen: Zusammenfassung mehrerer Datenbankoperationen zu einer atomaren Operation
- VIII. Synchronisierung: Konkurrierende Zugriffe der Benutzer sind möglich
- IX. Datensicherung: Wiederherstellung nach Systemausfall (Crash Recovery)

1.C.II DBMS

Physische Datenunabhängigkeit:

- Physische Speicherstrukturen werden verborgen
- Änderungen an der Struktur haben keinen Einfluss mehr auf die Anwendung

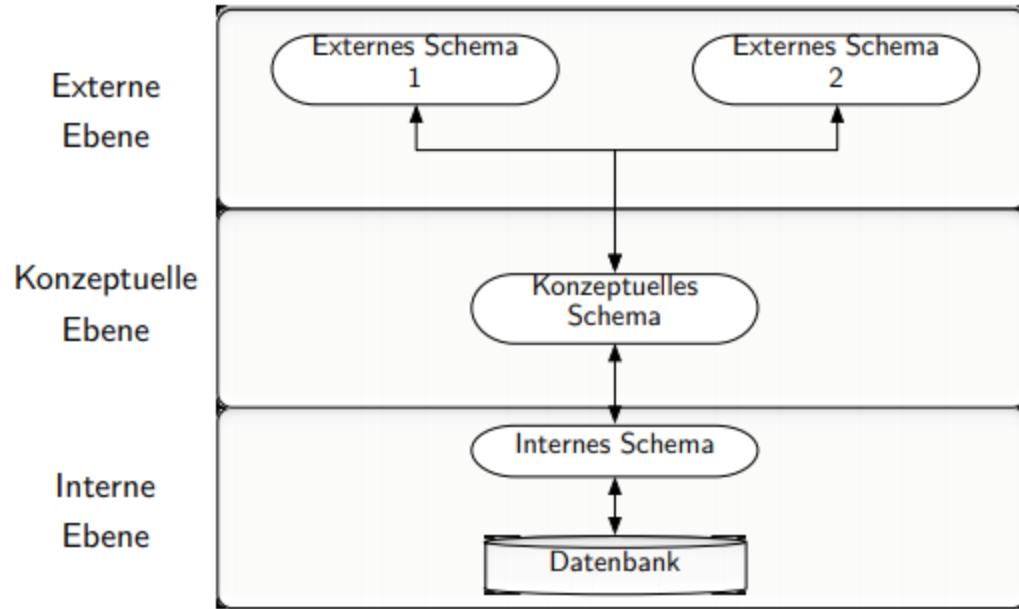
Logische Datenunabhängigkeit:

- Zugriff auf Daten über Sichten
- Änderungen haben keinen Einfluss mehr auf die Anwendung

1.C.II DBMS

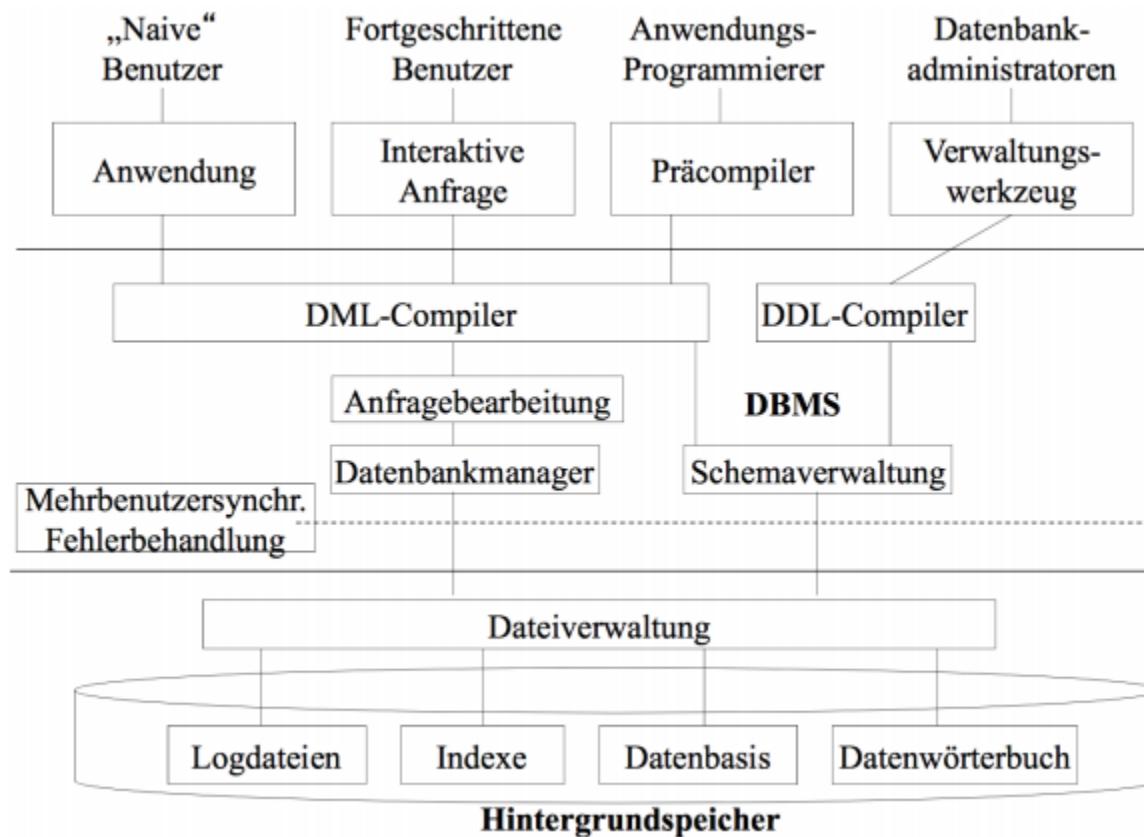
ANSI/SPARC 3-Schicht-Architektur:

- Externe Ebene beschreibt Sichten auf die Daten ([Formulare, Layouts, APIs](#))
- Konzeptuelle Ebene beschreibt die system- und anwendungsunabhängige Struktur ([Datenbankschema](#))
- Interne Ebene beschreibt [physikalische Speicherstrukturen](#) einer Datenbank



Gewährleistung der logischen und physischen Datenunabhängigkeit

1.C.II DBMS



1.C.II DBMS

Vorteile bei der Nutzung eines DBMS:

1. Kontrolle über Redundanzen
2. Zugriffskontrolle nicht-authentifizierter Benutzer
3. Persistenter Datenspeicher für Programmobjekte
4. Realisierung von Speicherstrukturen und effizienten Suchtechniken
5. Backup und Recovery
6. Ermöglichen einer Vielzahl von Benutzerschnittstellen
7. Repräsentieren komplexer Beziehungen entlang der Daten
8. Sicherstellung der referentiellen Integrität

DBS & DBMS: FRAGEN

1. Wo ist der Unterschied zwischen einer DB und einem DBMS?
2. Was sind die drei Ebenen des ANSI/SPARC-Modells? Wofür sind sie jeweils da?
3. Was sind die Hauptaufgaben eines DBMS? Wer hat diese definiert?
4. Auf welcher Ebene im ANSI/SPARC-Modell ist eine Datenreport zu sehen?

1.D ZUGRIFFSMÖGLICHKEITEN

- I. ACID
- II. APIs
 - a. Generisch – SOAP und REST
 - b. Angepasst
 - c. Vor-/Nachteile
- III. DML und DDL
- IV. SQL und ABAP

1.D.I ACID

Eigenschaften von Transaktionen:

a. Atomarität

Eine Transaktion ist in der kleinsten Einheit, d.h. sie wird entweder ganz oder gar nicht ausgeführt.

b. Konsistenz

Hinterlässt einen konsistenten und integren Datenbestand.

c. Isolation

Keine gegenseitige Beeinflussung von Transaktionen.

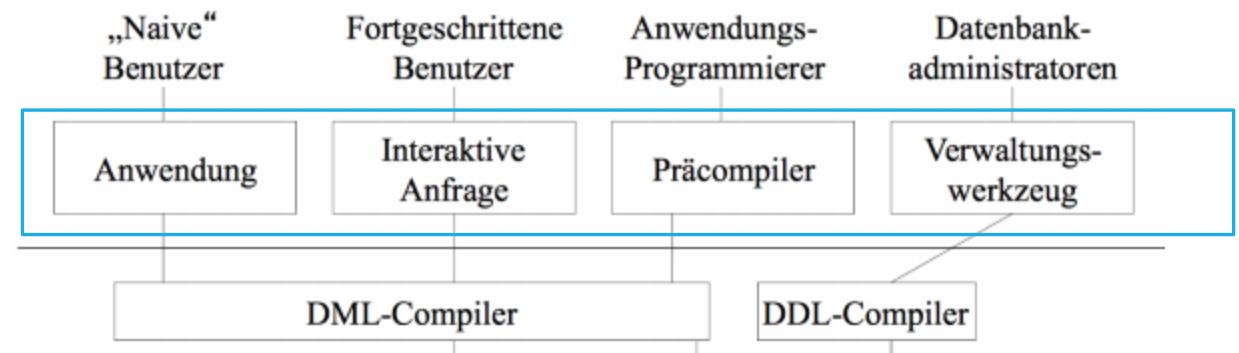
d. Dauerhaftigkeit

Dauerhafte Speicherung von Daten nach Transaktion.

1.D.II API

Application Programming Interface

- Schnittstelle zwischen Systemen
- Request – Response – Prinzip
- Dient der Standardisierung, Datenintegrität und Datensicherheit
- Teil des DBMS



1.D.II.A GENERIC API / SOAP

- Simple Object Access Protocol
- Benutzt das HTT- und TC-Protokoll
- Kommunikation i.d.R. in XML
- Beispiele: Ebay, Amazon
- ACID-Standard

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns="urn:enterprise.soap.sforce.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <loginResponse>
      <result>
        <metadataServerUrl>https://na30.salesforce.com/services/Soap/m/36.0/00D36000000wCdk</metadataServerUrl>
        <passwordExpired>false</passwordExpired>
        <sandbox>false</sandbox>
        <serverUrl>https://na30.salesforce.com/services/Soap/c/36.0/00D36000000wCdk</serverUrl>
        <sessionId>00D36000000wCdk!AOEAKz3h7Ak_SyGTu_WrzClz2KTGnZacSOWiR4jc8Mp2Jt7_f4t8XkJ.g64Wlp7_JfweOHn.Ak.SwUEkQR0XGcKAxKq4gtU</sessionId>
        <userId>00536000002BU42AA</userId>
        <userInfo>
          <accessibilityMode>false</accessibilityMode>
          <currencySymbol>$</currencySymbol>
          <orgAttachmentFileSizeLimit>5242880</orgAttachmentFileSizeLimit>
          <orgDefaultCurrencyIsoCode>USD</orgDefaultCurrencyIsoCode>
          <orgDisallowHtmlAttachments>false</orgDisallowHtmlAttachments>
          <orgHasPersonAccounts>false</orgHasPersonAccounts>
          <organizationId>00D36000000wCdEEE</organizationId>
          <organizationMultiCurrency>false</organizationMultiCurrency>
          <organizationName>Pirates, Inc.</organizationName>
          <profileId>00e3600001JVJdAAO</profileId>
          <roleId xsi:nil="true"/>
          <sessionSecondsValid>7200</sessionSecondsValid>
          <userDefaultCurrencyIsoCode>xsi:nil="true"/>
          <userEmail>pirate_jack@trailhead-pirates.org</userEmail>
          <userFullName>Pirate Jack</userFullName>
          <userId>00536000002BU42AA</userId>
          <userLanguage>en_US</userLanguage>
          <userLocale>en_US</userLocale>
          <userName>pirate_jack@trailhead-pirates.org</userName>
          <userTimeZone>America/Los_Angeles</userTimeZone>
          <userType>Standard</userType>
          <userUiSkin>Theme3</userUiSkin>
        </userInfo>
      </result>
    </loginResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

1.D.II.A GENERIC API / REST

- Representational State Transfer
- Benutzt das HTTP-Protokoll
- Kommunikation i.d.R. in JSON
- Authentication über Access-Token
- Kein ACID-Standard

```
[  
  {  
    "description": "quarter",  
    "mode": "REQUIRED",  
    "name": "qtr",  
    "type": "STRING"  
  },  
  {  
    "description": "sales representative",  
    "mode": "NULLABLE",  
    "name": "rep",  
    "type": "STRING"  
  },  
  {  
    "description": "total sales",  
    "mode": "NULLABLE",  
    "name": "sales",  
    "type": "INTEGER"  
  }  
]
```

1.D.II.B ANGEPASSTE API

- Muss aktiv programmiert werden
- Bessere Kontrolle über Zugriffe
- Kann als Zwischenebene für Programmierung genutzt werden (Factory-Schema)
- Liegt (in Windows) i.d.R. als Dynamic Link Library (DLL) vor

1.D.II.C VOR- NACHTEILE

	SOAP	REST	Angepasst
Funktion	Funktionen (Remote-Call)	Daten	Abhängig
Datenformat	XML	JSON uvm.	Abhängig
Sicherheit	SSL	SSL/HTTPS	Individuell einstellbar
Daten	Ressourcenintensiv	Ressourcenarm	Abhängig
ACID	Inhärent	Nicht inhärent	Muss individuell gewährleistet werden
Nachteile	Höherer Aufwand zur Implementation	Schnell zu implementieren	Muss komplett erstellt werden

1.D.III DML UND DDL

DML: Data Manipulation Language

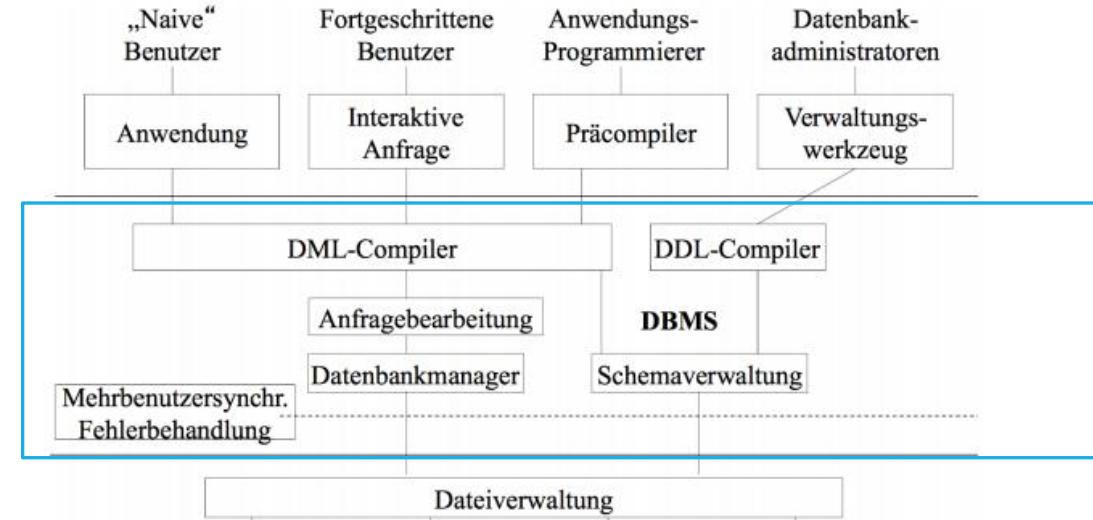
Liest, ändert und schreibt Datensätze

DDL: Data Definition Language

Liest, ändert und schreibt Datenstrukturen

DCL: Data Control Language

Liest, ändert und schreibt Datenberechtigungen



1.D.IV SQL UND ABAP

SQL: Structured Query Language

- Sprache zum Bearbeiten von Datenstrukturen und Datenbeständen
- Deskriptive Sprache (nur „Was“, nicht „Wie“)
- Muss in programmierte Umgebung eingebunden werden

ABAP: Advanced Business Application Programming

- Sprache zum Bearbeiten von Datenstrukturen und Datenbeständen, sowie zur Programmierung von komplexen Transaktionen
- 4GL Programmiersprache (nur noch bedingt für Reports)
- SAP – Standard und dient zur Programmierung von Systemen

ZUGRIFFS-MÖGLICHKEITEN: FRAGEN

1. Was ist eine API?
2. Was sind die Begriffe von ACID und wofür stehen sie?
3. Wo ist der Unterschied zwischen DDL und DML?
4. Was sind Vorteile einer angepassten API?
5. Wenn Sie eine dynamische Abfrage mit einem Formular realisieren wollen, wie würden Sie vorgehen?



1.E ÜBUNGEN

1. Zeichnen Sie die Komponenten, die an einer Datenbank beteiligt sind (oder sein können), und deren Verbindungen auf.
2. Entwerfen Sie eine Datenbank für folgendes Fallbeispiel:
 - Mitarbeiterdatenbank: Jeder Mitarbeiter arbeitet in einer Abteilung. Jede Abteilung ist in einem Bereich zugeordnet. Jeder Bereich hat einen zugeordneten Kundenkreis. Jede Abteilung und jeder Bereich hat einen Vorgesetzten, der auch ein Mitarbeiter ist. Kunden kaufen Dienstleistungen bei einer Abteilung ein, die dann durch einen Mitarbeiter ausgeführt werden. Eine Dienstleistung kann durch fast jede Abteilung ausgeführt werden, nicht jede Abteilung kann jede Dienstleistung durchführen.
3. Zeigen Sie den Weg der Daten (v.a. zwischen Google Maps und Google Search) auf, wenn der Name eines Restaurants in der Google-Suche eingegeben wird.



2. RELATIONEN

KAPITEL-AGENDA:

2. RELATIONEN

- a. Basics zu Relationen
- b. Tupel & Schlüssel
- c. Relationensoperationen
- d. Arten von Joins
- e. Übungen

2.A BASICS ZU RELATIONEN

Eine Relation ist (umgangssprachlich) eine Tabelle bestehend aus

- einem eindeutigen **Namen**,
- **Attributen** (Spalten) über Wertebereichen (Domänen), sowie
- **Tupeln** (Zeilen).

Tupel

Attribute		Kunde	Name	
KNR	Vorname	Nachname	PLZ	Ort
1	Elsa	Musterfrau	68165	Mannheim
2	Max	Mustermann	57234	Wilnsdorf
...

2.A BASICS ZU RELATIONEN

Anders gesagt: Attribute besitzen **Wertebereiche**, woraus sich ein **Schema** bilden lässt. Eine konkrete Ausprägung des Schemas wird **Relation** genannt und ist eine Menge von **Tupeln**, wobei jeder Wert der entsprechenden **Attribut-Domäne** entstammt.

Relationenschema:

Kunde (KNR:int, Vorname:string, Nachname:string, PLZ:string,
Ort:string)

Schreibweise ohne Angabe der Domänen:

Kunde (KNR, Vorname, Nachname, PLZ, Ort)

Kunde				
KNR	Vorname	Nachname	PLZ	Ort
1	Elsa	Musterfrau	68165	Mannheim
2	Max	Mustermann	57234	Wilnsdorf
...

2.B TUPEL & SCHLÜSSEL

Innerhalb einer Relation müssen Tupel eindeutig zu unterscheiden sein:

Kunde			
Vorname	Nachname	PLZ	Ort
Elsa	Musterfrau	68165	Mannheim
Max	Mustermann	57234	Wilnsdorf
...
Elsa	Musterfrau	68165	Mannheim

2.B TUPEL & SCHLÜSSEL

Schlüssel dienen der eindeutigen Identifikation von Tupeln und benötigen eine **funktionale Abhängigkeit**:

Eine funktionale Abhängigkeit existiert dann, wenn für alle Tupel einzelne Attribute in der Relation die anderen Werte der Relation festlegen. Heißt: aus einem oder mehreren Werten ergeben sich die anderen Werte.

Person_Auto					
<u>PNr</u>	Vorname	Nachname	<u>Kennzeichen</u>	Typ	Baujahr
1	Max	Mustermann	MA-BC 111	VW	2013
1	Max	Mustermann	MA-BC 112	Audi	2011
3	Erika	Musterfrau	LU-BY 42	Ford	2014

2.B TUPEL & SCHLÜSSEL

Schlüssel dienen der eindeutigen Identifikation von Tupeln:

1. Superschlüssel

- Ein Superschlüssel ist eine Menge von einem oder mehreren Attributen, von denen alle anderen Attribute **funktional abhängig** sind. Ein Superschlüssel garantiert damit die eindeutige Identifizierbarkeit des Tupels.

2. Kandidatenschlüssel

- Ein Schlüsselkandidat ist ein **minimaler** Superschlüssel, d. h. die Attributmenge, von der alle andern Attribute der Relation funktional abhängen, kann nicht weiter verkleinert werden.

3. Primärschlüssel

- **Genau ein** Schlüsselkandidat wird als Primärschlüssel festgelegt. Primärschlüssel werden in der Notation unterstrichen dargestellt.

2.B TUPEL & SCHLÜSSEL

4. Fremdschlüssel

- Ein Fremdschlüssel ist eine Menge von Attributen in einer **referenzierenden** Relation, deren Werte garantiert als Primärschlüssel in der referenzierten Relation vorkommen. Fremdschlüssel sind in der referenzierenden Relation nicht eindeutig!

Kunde		
KNR	Vorname	Nachname
1	Elsa	Musterfrau
2	Max	Mustermann
3	Hans	Fritsche
...

Auftrag		
ANR	KNR	Datum
1001	1	02.01.2013
1002	2	04.05.2013
1003	2	03.01.2014
...

2.B TUPEL & SCHLÜSSEL

Business/Natural Keys vs. technical/surrogate Keys

Business/Natural Keys

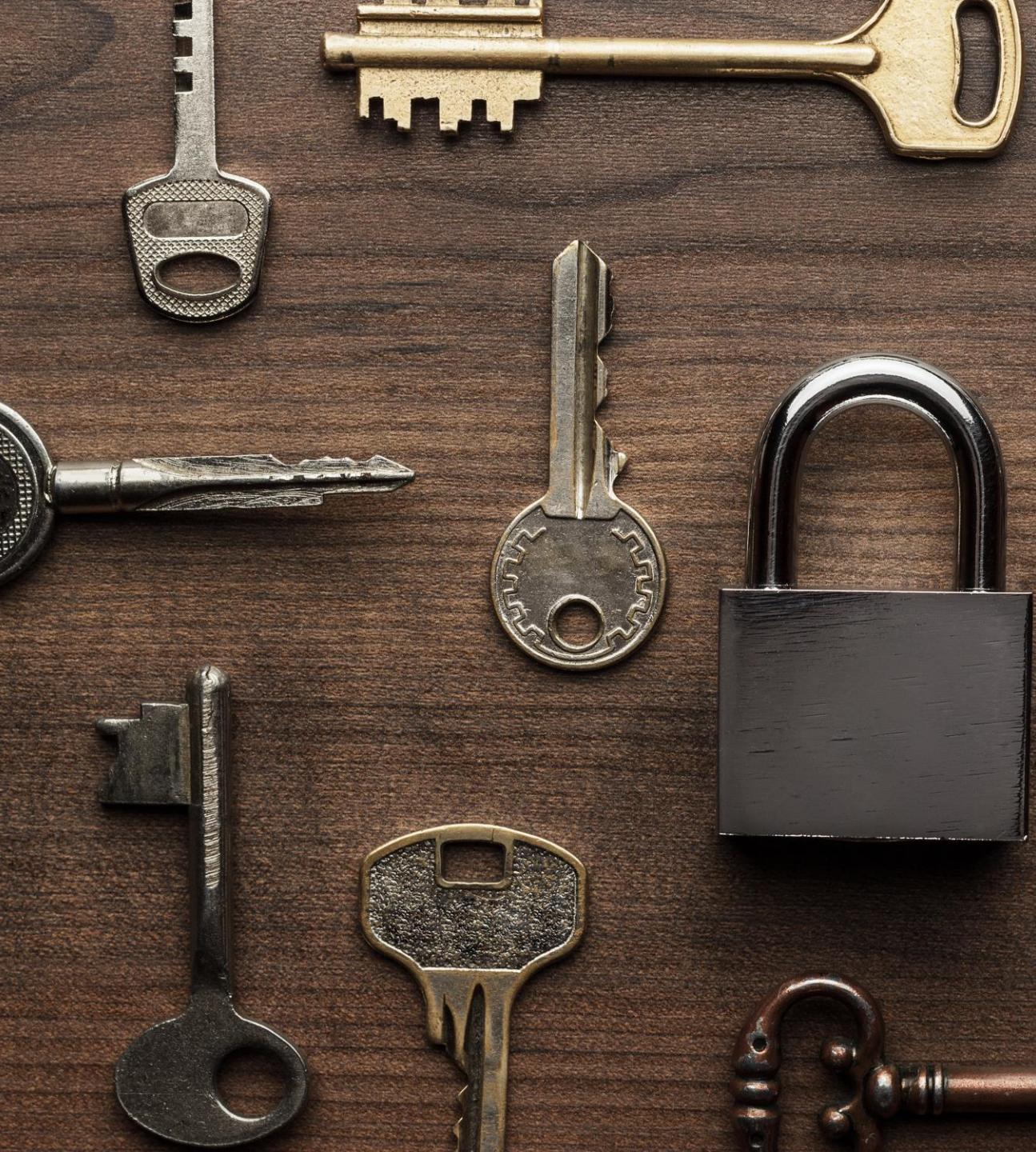
- Schlüsselattribute, die natürlicher Weise einer Entität zugeordnet werden können und diese identifizieren
- Beispiele: Kundennummern, Fahrgestellnummern, Seriennummern

Technical/Surrogate Keys

- Existiert kein natürlicher Schlüssel, muss ein künstliches Schlüsselattribut (z. B. eine laufende Nummer) eingefügt werden.
- Beispiel: Signaturen auf zwei Exemplaren eines Buches in der Bibliothek

BASICS, TUPEL & SCHLÜSSEL: FRAGEN

1. Was ist eine Relation? Worin unterscheidet sie sich von einer Tabelle?
2. Was ist ein Tupel?
3. Kann jedes Feld ein Schlüssel sein?
4. Was darf auf keinen Fall in einer Relation vorkommen?
5. Was sind Domänen? Nennen Sie verschiedene Datentypen.
6. Ist eine Mitarbeiter-ID ein Business- oder ein Technical Key?



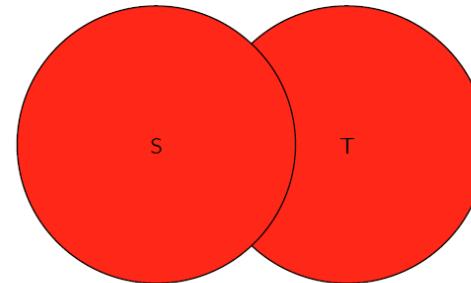
2.C RELATIONSOOPERATIONEN

Mehrere Relationen können zueinander in Verbindung & Abhängigkeit gebracht werden.

Verschiedene Operationen:

- I. Vereinigung
- II. Differenz
- III. Schnitt
- IV. Projektion
- V. Aggregation
- VI. Kartesisches Produkt (incl. Umbenennung und Selektion)

2.C.I VEREINIGUNG



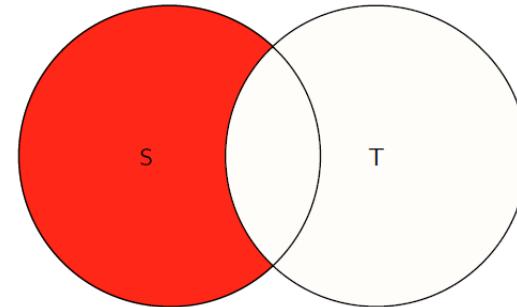
Mit der Vereinigung lassen sich die Tupel von zwei gleichartigen (gleiche Anzahl, Reihenfolge und Domänen der Attribute) Relationen miteinander zusammenfügen. Der Bestand von Relation S wird um diejenigen Tupel aus Relation T erweitert, die in Relation S noch nicht vorhanden waren.

Formel: $R = S \cup T$

S		T		\cup	R	
<u>KNR</u>	Nachname	<u>KNR</u>	Nachname		<u>KNR</u>	Nachname
1	Musterfrau	1	Musterfrau		1	Musterfrau
2	Mustermann	42	Meiser		2	Mustermann
...		42	Meiser
				

Praxisbeispiel: Auswertung mehrerer gleichartiger Niederlassungsdatenbanken.

2.C.II DIFFERENZ



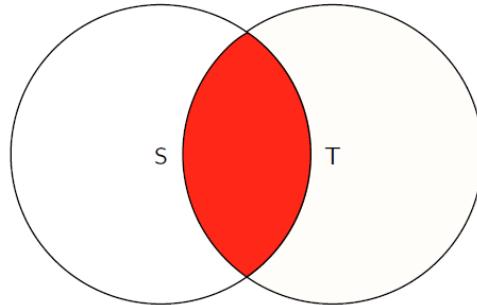
Mit der Differenz lassen sich die Tupel von zwei gleichartigen (gleiche Anzahl, Reihenfolge und Domänen der Attribute) Relationen voneinander abziehen. Der Bestand der Tupel in Relation S wird um diejenigen Tupel verringert, die auch in Relation T vorkommen.

Formel: $R = S - T$

S		T		R	
<u>KNR</u>	Nachname	<u>KNR</u>	Nachname	<u>KNR</u>	Nachname
1	Musterfrau	1	Musterfrau	2	Mustermann
2	Mustermann	42	Meiser
...

Praxisbeispiel: Überprüfung von Datenintegrität nach einem Datenbankreset/-upgrade/-rollback

2.C.III SCHNITT



Mit dem Schnitt lassen sich die Tupel von zwei gleichartigen (gleiche Anzahl, Reihenfolge und Domänen der Attribute) Relationen voneinander abziehen. Der Bestand der Tupel in Relation S wird um diejenigen Tupel verringert, die nicht in Relation T vorkommen.

Formel: $R = S \cap T$ oder $R = S - (S - T)$

S		T		R	
KNR	Nachname	KNR	Nachname	KNR	Nachname
1	Musterfrau	1	Musterfrau	1	Musterfrau
2	Mustermann	42	Meiser
...

Diagram illustrating set intersection:

The intersection of sets S and T is shown as the overlapping region of the two circles. The resulting relation R contains only the tuples that are present in both S and T.

Praxisbeispiel: Überprüfung von Redundanzen

2.C.IV PROJEKTION

Mit der Projektion werden alle Werte aller Tupel einer oder mehrerer Attribute ausgewählt, die bei der Projektion angegeben werden.

Formel: $R = \pi_{\text{Attribut}_1, \dots, \text{Attribut}_N}(\text{Relation})$

Auftrag		
<u>ANR</u>	<u>KNR</u>	<u>Datum</u>
1001	1	02.01.2013
1002	2	04.05.2013

$\pi_{ANR, Datum}(Auftrag)$

Auftrag	
<u>ANR</u>	<u>Datum</u>
1001	02.01.2013
1002	04.05.2013

Praxisbeispiel: Ausblenden von unnötigen Daten

2.C.V AGGREGATION

Mit der Aggregation werden alle Werte einer Relation auf Basis eines angegebenen Attributs auf definierte Weise aggregiert, während nicht-aggregierte Attribute als Gruppierungsattribute genutzt werden.

Formel: $R = \gamma_{KNR, count(*), sum(Wert)}(Auftrag)$

Auftrag		
<u>ANR</u>	<u>KNR</u>	<u>Wert</u>
1001	1	10,12
1002	2	12,13
1003	1	14,23

$\gamma_{KNR, count(*), sum(Wert)}(Auftrag)$		
KNR	count(*)	sum(Wert)
1	2	24,35
2	1	12,13

Praxisbeispiel: Ausblenden von unnötigen Daten

S1T1	S1T2	S1T3
S2T1	S2T2	S2T3
S3T1	S3T2	S3T3

2.C. VI KARTESISCHES PRODUKT

Das kartesische Produkt kombiniert alle Tupel einer Relation S mit allen Tupeln einer Relation T.

Schemata müssen **dijunkt**, also **attributsunderschiedlich** sein.

Formel: $R = S \times T$

Kunde		
KNR	Vorname	Nachname
1	Elsa	Musterfrau
2	Max	Mustermann

Auftrag	
ANR	Datum
1001	02.01.2013
1002	04.05.2013

x

=

R				
KNR	Vorname	Nachname	ANR	Datum
1	Elsa	Musterfrau	1001	02.01.2013
1	Elsa	Musterfrau	1002	04.05.2013
2	Max	Mustermann	1001	02.01.2013
2	Max	Mustermann	1002	04.05.2013

Praxisbeispiel: Verbinden von unterschiedlichen Datenbanken

2.C. VI KARTESISCHES PRODUKT

Problem hierbei: Es werden **unechte Tupel** erzeugt, die nicht der Realität entsprechen.

Abhilfe: Schlüssel. Da aber Tabellen paarweise disjunkt sein müssen, sind Fremdschlüssel nicht möglich, sofern das Attribut den gleichen Namen hat.

Um einer generellen Umbenennung zu entgehen, kann sich der temporären **Umbenennung ρ** bedient werden, die ein Attribut a einer Relation zu b für eine einzelne Operation umbenannt:

Auftrag	
<u>ANR</u>	Datum
1001	02.01.2013
1002	04.05.2013

$\rho_{AngNr \leftarrow ANR}(Auftrag)$

Auftrag	
<u>AngNr</u>	Datum
1001	02.01.2013
1002	04.05.2013

2.C. VI KARTESISCHES PRODUKT

Mit der Umbenennung des Fremdschlüssel-Attributs kann nun ein kartesisches Produkt hergestellt werden.

Bei den Relationen gelten folgende Umbenennungen (kann implizit geschehen):

Formel: $\rho_{\text{Kunde.KNR} \leftarrow \text{KNR}}(\text{Kunde})$

Formel: $\rho_{\text{Auftrag.KNR} \leftarrow \text{KNR}}(\text{Auftrag})$

Kunde			Auftrag			R					
<u>KNR</u>	Vorname	Nachname	<u>ANR</u>	KNR	Datum	Kunde.KNR	Vorname	Nachname	ANR	Auftrag.KNR	Datum
1	Elsa	Musterfrau	1001	1	02.01.2013	1	Elsa	Musterfrau	1001	1	02.01.2013
2	Max	Mustermann	1002	2	04.05.2013	1	Elsa	Musterfrau	1002	2	04.05.2013
						2	Max	Mustermann	1001	1	02.01.2013
						2	Max	Mustermann	1002	2	04.05.2013

2.C. VI KARTESISCHES PRODUKT

Zur Vermeidung von unechten Tupeln muss nun eine Selektion ausgeführt werden.

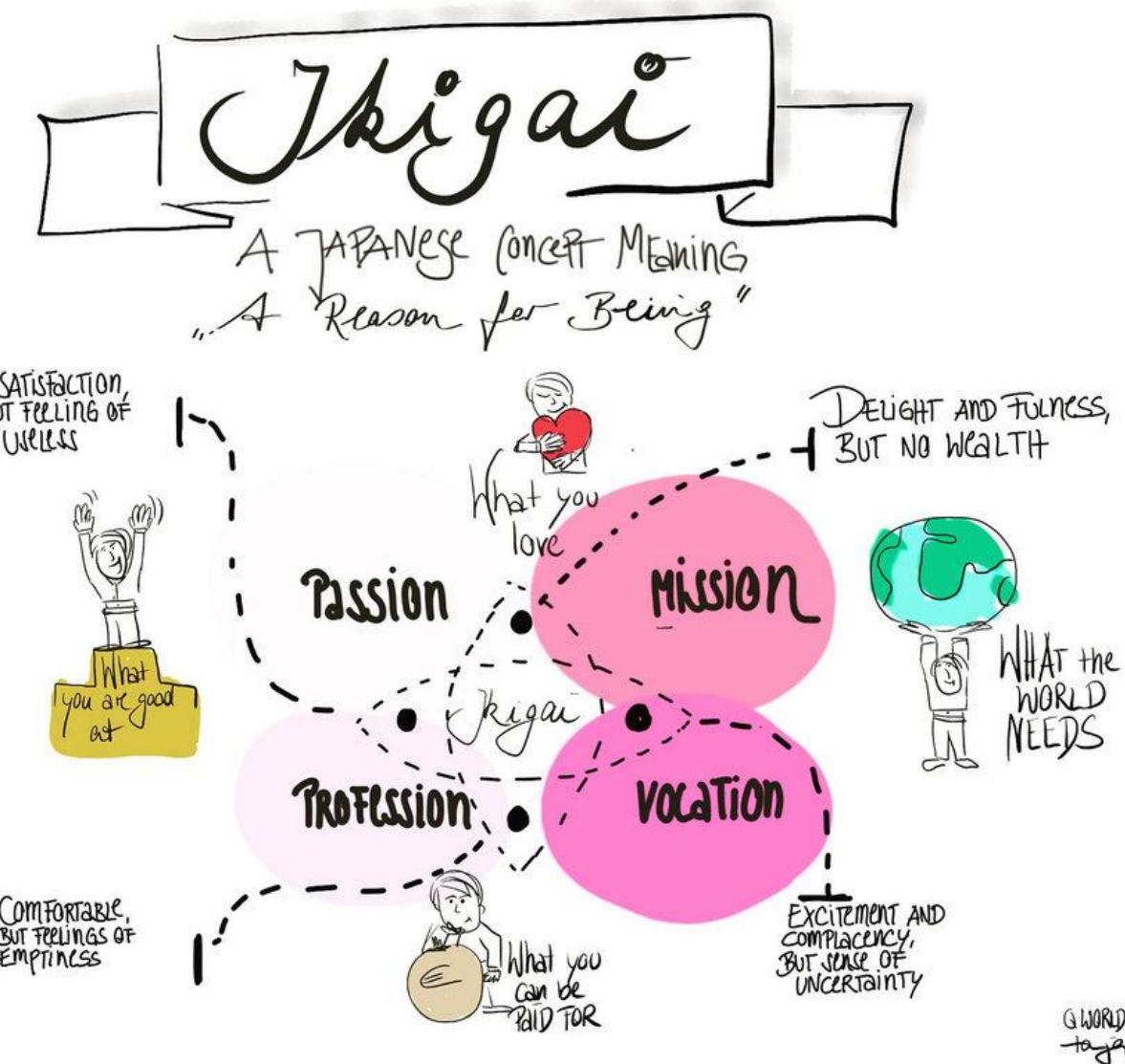
Die Selektion ist eine boolesche Operation und wählt nur Tupel aus, bei denen die Bedingung erfüllt ist.

Formel: $\sigma_{\text{Kunde.KNR}=\text{Auftrag.KNR}}(R)$

R					
Kunde.KNR	Vorname	Nachname	ANR	Auftrag.KNR	Datum
1	Elsa	Musterfrau	1001	1	02.01.2013
1	Elsa	Musterfrau	1002	2	04.05.2013
2	Max	Mustermann	1001	1	02.01.2013
2	Max	Mustermann	1002	2	04.05.2013

RELATIONSOOPERATIONEN: FRAGEN

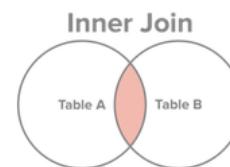
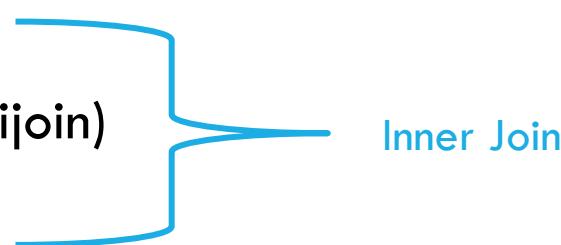
1. Wann verwendet man eine Vereinigung, wann einen Schnitt?
2. Nennen Sie 3 Aggregationsfunktionen.
3. Was ist die Grundvoraussetzung für ein kartesisches Produkt? Wie umgeht man diese?
4. Was kommt bei folgenden Operationen heraus? O sei eine attributsgleiche Relation zu S ohne Tupel.
 1. $S \cup O$
 2. $S - O$
 3. $S \times O$
 4. $S \cap O$



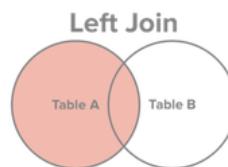
2.D ARTEN VON JOINS

Joins sind die technischen Verbundoperatoren zur Verknüpfung von Relationen in SQL:

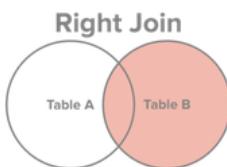
- I. Cross Join
- II. Theta-Join (mit Sonderform Equijoin)
- III. Natural Join
- IV. Left Outer Join
- V. Right Outer Join
- VI. Full Outer Join



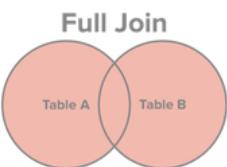
Select all records from Table A and Table B, where the join condition is met.



Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

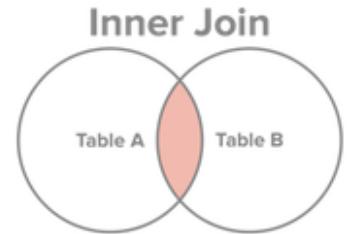


Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

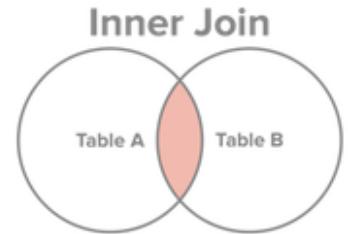
2.D.I CROSS JOIN



Cross Join

- `FROM Mitarbeiter CROSS JOIN Abteilung`
- Kartesisches Produkt
- Alle Tupel mit allen Tupeln kombiniert
- Unechte Tupel

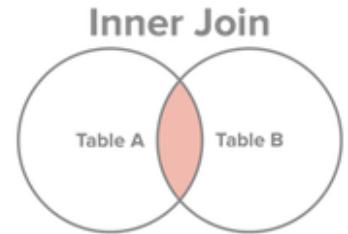
2.D.II THETA-JOIN



Theta-Join (mit Sonderform Equijoin)

- `FROM Mitarbeiter JOIN Abteilung ON Mitarbeiter.AbId = Abteilung.AbId;`
- Kartesisches Produkt mit Selektion anhand von logischen Operatoren
 - Bei Operator = wird der Join auch **Equijoin** genannt
- Benötigt vergleichbare Attribute
- Ggf. Umbenennung bei Fremdschlüsseln nötig

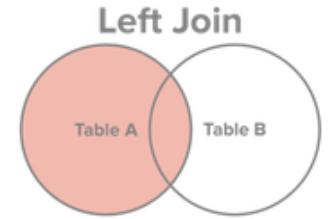
2.D.III NATURAL JOIN



Natural Join

- `FROM Mitarbeiter NATURAL JOIN Abteilung`
- Sonderform des Equijoin unter Umbenennung und Selektion
- Zu vergleichende Attribute werden anhand von gleichen Namen selektiert

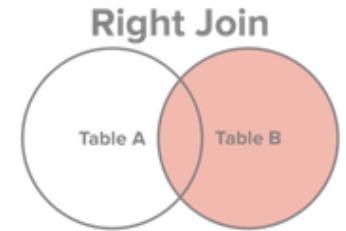
2.D.IV LEFT OUTER JOIN



Left Outer Join

- `FROM Mitarbeiter LEFT OUTER JOIN Abteilung USING (AbtId)`
- Vereinigung auf Basis von Relation Mitarbeiter um Werte aus Relation Abteilung

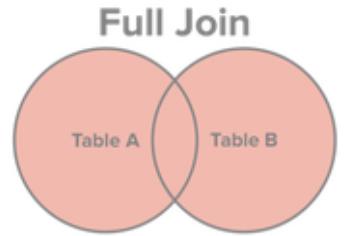
2.D.V RIGHT OUTER JOIN



Right Outer Join

- FROM Mitarbeiter RIGHT OUTER JOIN Abteilung USING (AbtId)
- Vereinigung auf Basis von Relation Abteilung um Werte aus Relation Mitarbeiter

2.D.VI FULL OUTER JOIN



Full Outer Join

- `FROM Mitarbeiter FULL OUTER JOIN Abteilung USING (AbtId)`
- Volle Vereinigung von zwei Relationen

2.D VERGLEICH

m_id	nachname	vorname	a_id	a_id	bezeichnung	standort
1	Schmidt	Udo	3	1	Vertrieb	Frankfurt
2	Müller	Wolfgang	1	2	IT	Bad Homburg
3	Meyer	Günther	5	3	Personal	Offenbach
4	Krause	Helmut	2	4	Forschung	Bad Homburg
5	Schneider	Kevin	NULL			

I. Cross Join

m_id	nachname	vorname	mitarbeiter.a_id	abteilungen.a_id	bezeichnung	standort
1	Schmidt	Udo	3	1	Vertrieb	Frankfurt
1	Schmidt	Udo	3	2	IT	Bad Homburg
1	Schmidt	Udo	3	3	Personal	Offenbach
1	Schmidt	Udo	3	4	Forschung	Bad Homburg
2	Müller	Wolfgang	1	1	Vertrieb	Frankfurt
2	Müller	Wolfgang	1	2	IT	Bad Homburg
2	Müller	Wolfgang	1	3	Personal	Offenbach
2	Müller	Wolfgang	1	4	Forschung	Bad Homburg
...						

II. Theta-Join/Natural Join

m_id	nachname	vorname	a_id	bezeichnung	standort
1	Schmidt	Udo	3	Personal	Offenbach
2	Müller	Wolfgang	1	Vertrieb	Frankfurt
4	Krause	Helmut	2	IT	Bad Homburg

2.D VERGLEICH

m_id	nachname	vorname	a_id	a_id	bezeichnung	standort
1	Schmidt	Udo	3	1	Vertrieb	Frankfurt
2	Müller	Wolfgang	1	2	IT	Bad Homburg
3	Meyer	Günther	5	3	Personal	Offenbach
4	Krause	Helmut	2	4	Forschung	Bad Homburg
5	Schneider	Kevin	NULL			

IV. Left Outer Join

m_id	nachname	vorname	a_id	bezeichnung	standort
1	Schmidt	Udo	3	Personal	Offenbach
2	Müller	Wolfgang	1	Vertrieb	Frankfurt
3	Meyer	Günther	5	NULL	NULL
4	Krause	Helmut	2	IT	Bad Homburg
5	Schneider	Kevin	NULL	NULL	NULL

V. Right Outer Join

m_id	nachname	vorname	a_id	bezeichnung	standort
2	Müller	Wolfgang	1	Vertrieb	Frankfurt
4	Krause	Helmut	2	IT	Bad Homburg
1	Schmidt	Udo	3	Personal	Offenbach
NULL	NULL	NULL	4	Forschung	Bad Homburg

2.D VERGLEICH

m_id	nachname	vorname	a_id	a_id	bezeichnung	standort
1	Schmidt	Udo	3	1	Vertrieb	Frankfurt
2	Müller	Wolfgang	1	2	IT	Bad Homburg
3	Meyer	Günther	5	3	Personal	Offenbach
4	Krause	Helmut	2	4	Forschung	Bad Homburg
5	Schneider	Kevin	NULL			

IV. Full Outer Join

m_id	nachname	vorname	a_id	bezeichnung	standort
1	Schmidt	Udo	3	Personal	Offenbach
2	Müller	Wolfgang	1	Vertrieb	Frankfurt
3	Meyer	Günther	5	NULL	NULL
4	Krause	Helmut	2	IT	Bad Homburg
5	Schneider	Kevin	NULL	NULL	NULL
NULL	NULL	NULL	4	Forschung	Bad Homburg

RELATIONSOOPERATIONEN: FRAGEN

1. Was ist der Unterschied zwischen einem Inner Join und einem Outer Join?
2. Welche Arten von Outer Join gibt es?
3. Was wird für einen Natural Join benötigt?
4. Wie können zwei Relationen miteinander verknüpft werden, die keine übereinstimmenden Attribute haben?
5. Warum wird nicht immer ein Natural Join verwendet?



2.E ÜBUNGEN

Rechnung		
R.Nr.	Datum	K.Nr
1	01.12.2019	1
2	02.12.2019	17
3	02.12.2019	3

Artikel		
A.Nr.	Artikel	Preis
017856	Stift	1,95€
222746	Mappe	4,95€
335264	Stuhl	117,50€
849238	Blatt	0,50€

Rechnungsposition			
R.P.Nr.	R.Nr.	A.Nr.	Anzahl
1	1	017856	3
2	1	222746	5
1	2	849238	2

Kunde			
K.Nr	Name	Adresse	Mail
1	Hans Müller	Xyzstraße 4	hans@müller.de
2	Lena Schulz	Blablaplatz 3	leni@gmail.com
3	Ramona Meier	Rundweg 5	ramram@web.de

Aufgaben:

1. Mit welchen Joins können Sie zu allen Rechnungen ausschließlich alle relevanten Daten ausgeben? Verwenden Sie auch andere Joins als den Natural Join.
2. Welche Joins liefern keine „NULL“-Werte?
3. Welche Verbindung besteht zwischen Artikel und Kunde? Wie kann diese realisiert werden?



3. ENTITY-RELATION-MODELL

KAPITEL-AGENDA:

3. ENTITY-RELATION-MODELL

- a. Datenbankentwurf
- b. Basics zum Entity-Relation-Modell
- c. Beziehungen
- d. Kardinalitäten
- e. Übungen

3.A DATENBANKENTWURF

Datenbanken bilden einen Teil der realen Welt ab ([Universe of Discourse](#)) und sind logisch zusammenhängend. Sie beinhalten Daten, die einem vordefinierten Schema entsprechen:

- Objekte der realen Welt
- Beziehungen zwischen den Objekten
- Nebenbedingungen

[Der Datenbank-Entwurfsprozess](#)

- Identifizierung der notwendigen Ausschnitte der realen Welt, die für die Abbildung eines Themengebietes notwendig sind
- Überführung dieses Ausschnittes in ein adäquates Datenbankschema.

3.A DATENBANKENTWURF

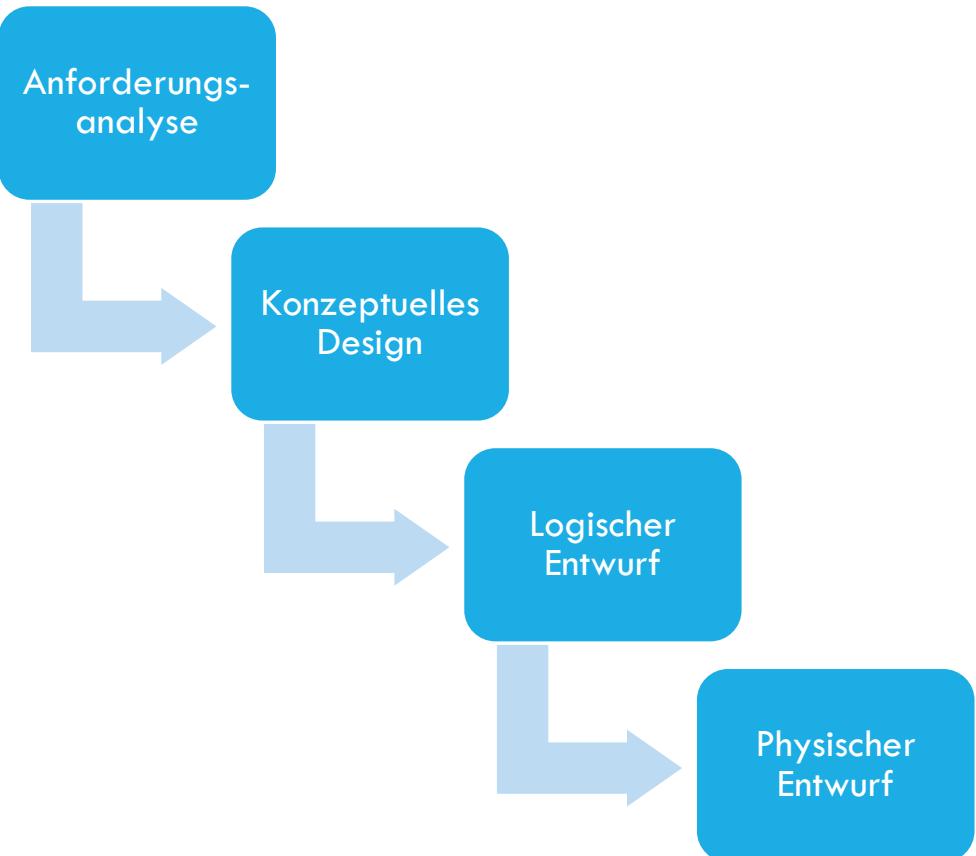
Im Datenbank-Entwurfsprozess müssen zwei Dinge strikt beachtet werden:

1. Vermeidung von Redundanzen im Datenbestand. Mehrfache Speicherung einer Information führt zu Inkonsistenzen, fehlerhaften Datenbeständen
2. Vollständigkeit: Ein nicht vollständiges Datenmodell, welches Aspekte der realen Welt nur unzureichend bzw. zu stark vereinfacht abbildet, führt zu starken Einschränkungen in der späteren Verwendung der Anwendung.

Das Design einer Datenbank

Es gibt nicht das eine Datenbankdesign - vielmehr können unterschiedliche korrekte Datenbank-Designs existieren. Viele Wege führen nach Rom – aber eben nicht alle!

3.A DATENBANKENTWURF



Anforderungsanalyse

- Gespräche mit Experten der Fachdomäne
- Dokumentation der Anforderungen (Textdokumente, UML usw.)

Konzeptueller Entwurf

- Übersetzung der Anforderungen in ein konzeptuelles Modell
- Output: ER-Diagramme

Logischer Entwurf

- Übersetzung des ER-Modells in ein implementierendes Datenmodell
- Hier: das relationale Datenmodell

Physischer Entwurf

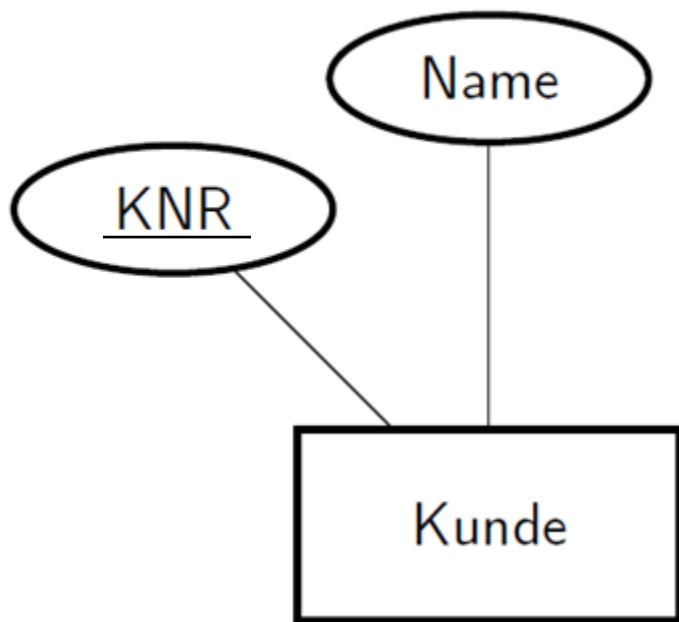
- Festlegung von Datei- und Index-Strukturen
- Vorlesung *Datenbanktechnik*

3.B BASICS ZUM ENTITY-RELATION-MODELL

ER-Modelle sind Datenmodelle, die während der konzeptionellen Entwurfsphase von Datenbanken eingesetzt werden.

- **Entitäten** stellen Objekte der realen Welt dar.
 - Physisch existent: Autos, Personen, Gebäude etc.
 - Konzeptuell existent: Arbeitsstellen, Firmen etc.
- **Attribute** sind Eigenschaften
 - So hat z.B. eine Person die Attribute Vorname, Nachname, Geburtsdatum
 - Jede spezifische Entität hat für jedes Attribut einen Wert
- **Beziehungen** zwischen Entitäten existieren
- Entitäten mit den gleichen Attributen nennt man **Entitätstyp**. Sie beschreiben das Schema einer Entität.
- Die Sammlung von Entitäten eines Entitätstyps zu einem bestimmten Zeitpunkt nennt man **Entitätsmenge**.

3.B BASICS ZUM ENTITY-RELATION-MODELL



Entitätstyp

Kunde:
KNR 4711
Name Mustermann

Entität

Kunde	
KNR	Name
1	Mustermann
2	Musterfrau
...	...

Entitätsmenge

3.B BASICS ZUM ENTITY-RELATION-MODELL

Mehrere Attributtypen kommen in ER-Diagrammen vor:

- Atomare Attribute, die nicht geteilt werden können (z.B. Alter, Postleitzahl, Nachname)
- Zusammengesetzte Attribute (z.B. das Attribut *Anschrift*, welches sich in PLZ, Ort und Straße zerlegen lässt)
- Einwertige Attribute, bei denen zu einem bestimmten Zeitpunkt genau ein Wert existiert
- Mehrwertige Attribute, bei denen zu einem bestimmten Zeitpunkt mehrere Werte existieren können (z.B. wenn eine Person zwei akademische Titel hat)
- Abgeleitete Attribute, deren Wert sich aus anderen Attributen herleiten lässt (z.B. kann ein Attribut *Alter* aus dem Geburtsdatum errechnet werden)
- null-Attribute, die bei manchen Entitäten nicht belegt sind.

Achtung: null-Attribute bergen Interpretationsspielraum!

3.B BASICS ZUM ENTITY-RELATION-MODELL

- In ER-Diagrammen liegt der Fokus auf der Darstellung des Schemas, nicht der **Instanzen**
- Die Namen von Entitätstypen, Beziehungstypen usw. sollte nach Möglichkeit die **Semantik** ausdrücken
- Entitätstypen werden mit Substantiven im Singular benannt
- ER-Diagramme werden meist stufenweise verfeinert:
 - Erstes Konzept mit grundlegenden Entitätstypen und Attributen
 - Einfügen von Beziehungstypen und Kardinalitäten
 - Attribute, die in mehreren Entitätstypen vorkommen, können auf einen eigenen Entitätstyp verfeinert werden
 - Zusammenlegung von Entitätstypen (z.B. Entitätstypen mit nur einem Attribut)

3.B BASICS ZUM ENTITY-RELATION-MODELL

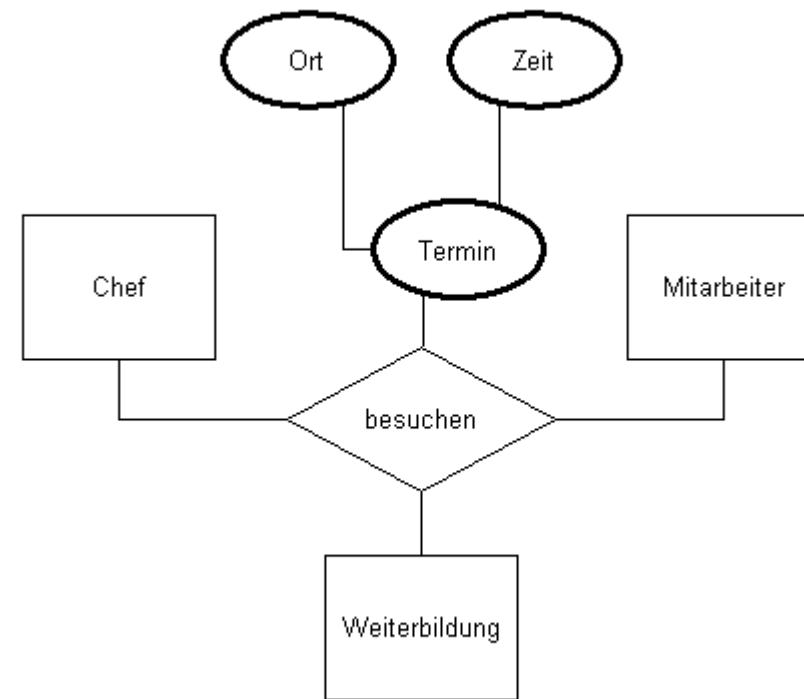
Es existieren viele unterschiedliche Notationsformen von ER-Modellen:

- Chen-Notation (wie in der Vorlesung vorgestellt, nach Peter Chen, 1976)
- UML-Notation
 - Aussehen ähnlich wie Klassendiagramme
 - Vererbung etc. möglich
 - Keine Beziehungen vom Grad $n > 2$ erlaubt
- Crow's Feet Notation
 - Ähnlich wie UML-Notation
 - Stellt Kardinalitäten durch Krähenfüsse dar

Im Laufe der Vorlesung wird die Chen-Notation weiter verwendet.

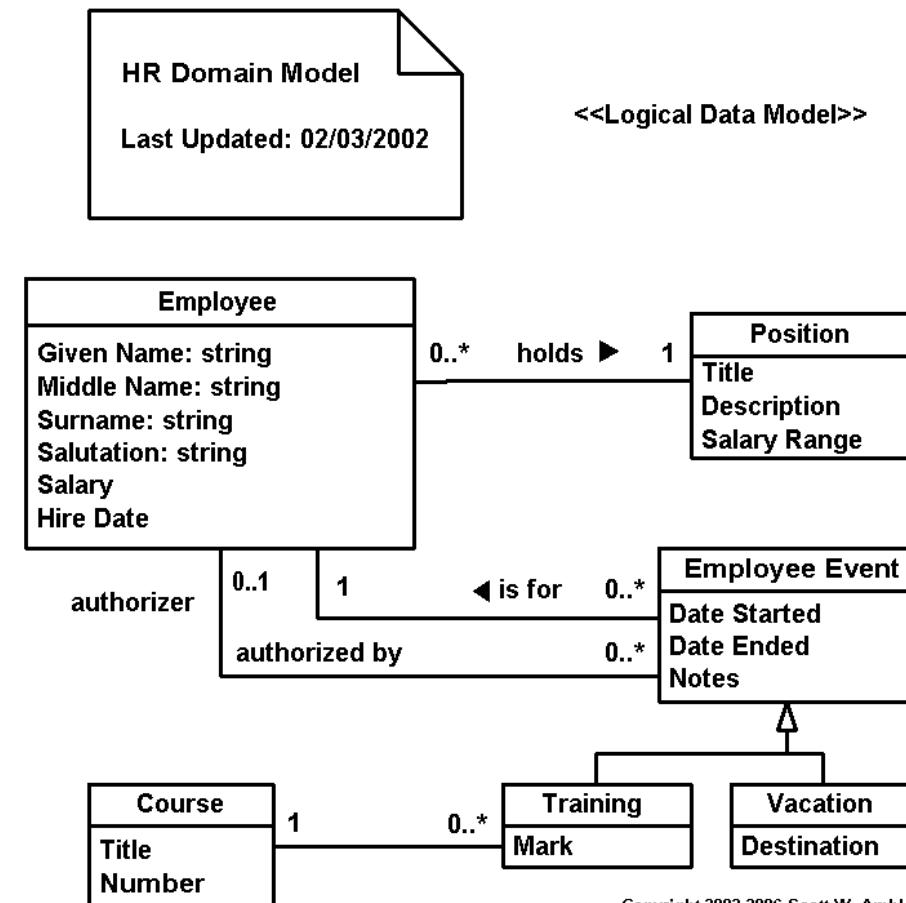
3.B BASICS ZUM ENTITY-RELATION-MODELL

Chen-Notation



3.B BASICS ZUM ENTITY-RELATION-MODELL

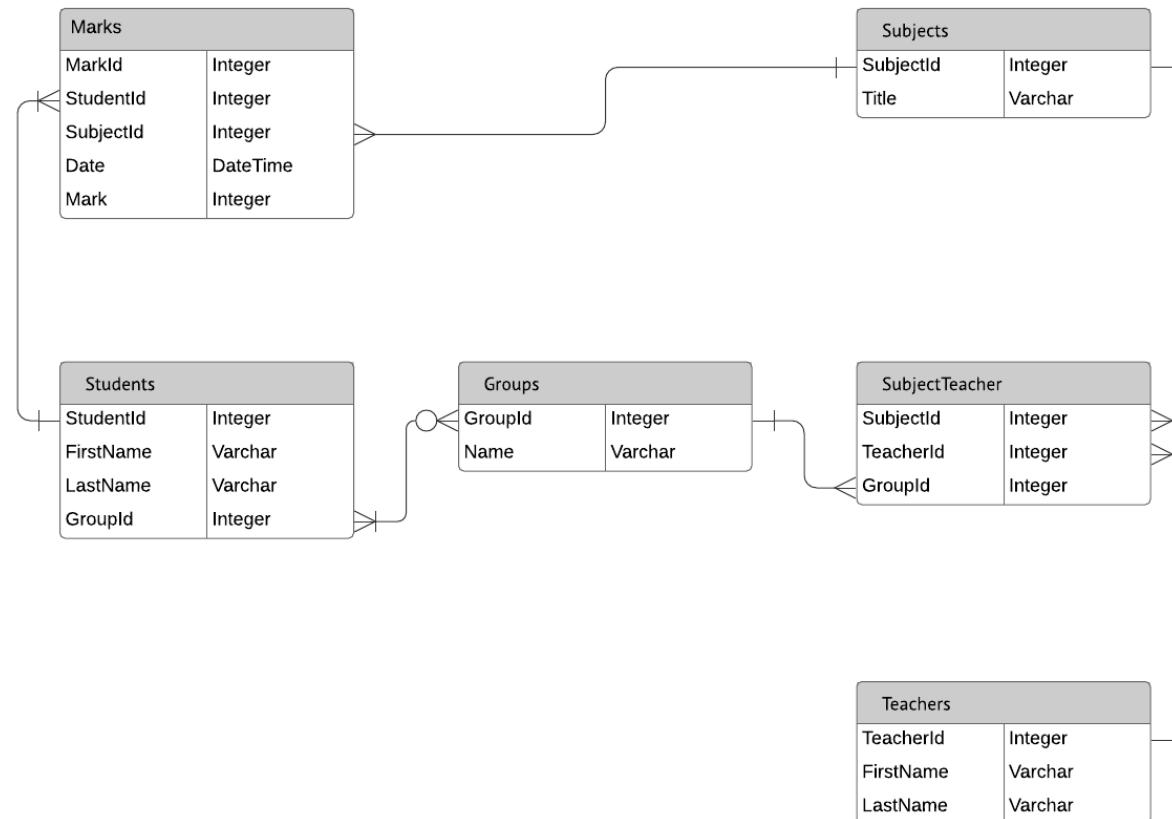
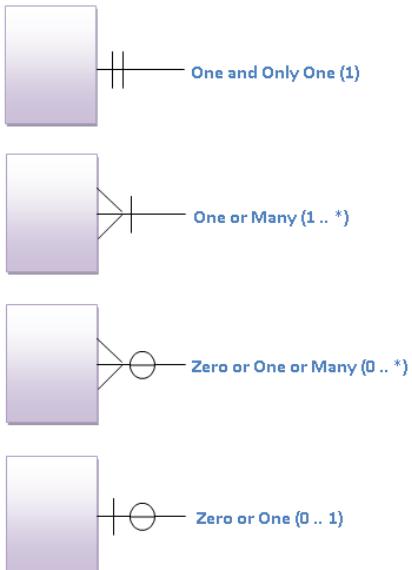
UML-Notation



3.B BASICS ZUM ENTITY-RELATION-MODELL

Crow's Feet Notation

Crow Foot Notation Symbols



BASICS: FRAGEN

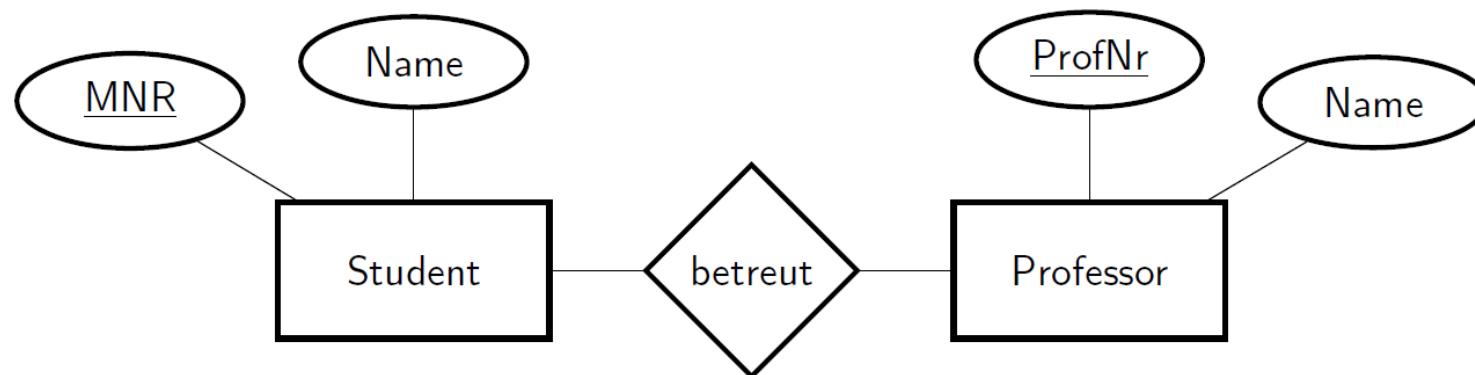
1. Was ist der Unterschied zwischen einer Entität, einem Entitätstyp und einer Entitätsmenge?
 2. Was stellen Entitäten dar?
 3. Was ist der Nachteil an der UML-Notation?
 4. Was sollte bei der Datenbankkonzeption dringend vermieden werden?
 5. Was sind die 4 Schritte einer Datenbankkonzeption?



3.C BEZIEHUNGEN

Beziehungen zwischen Entitätstypen sind Beziehungstypen und werden als Raute abgebildet.

Achtung: Es gibt keine vorgegebene Leserichtung!



3.C BEZIEHUNGEN

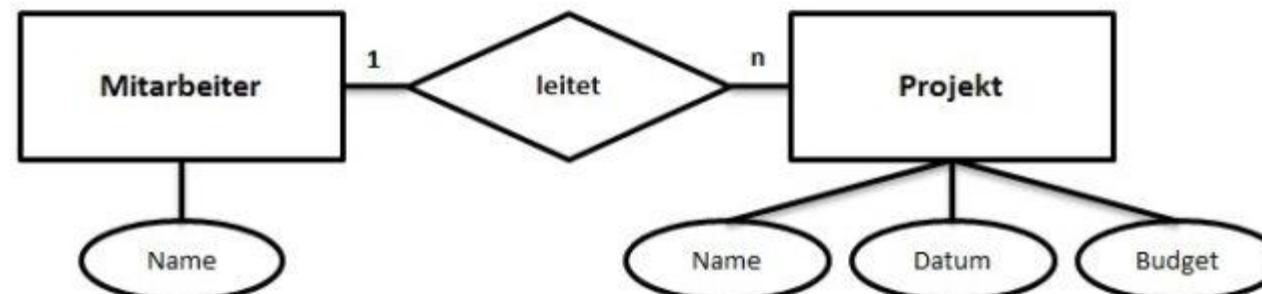
Arten von Beziehungen:

- I. Binäre Beziehungen
- II. Ternäre Beziehungen
- III. Rekursive Beziehungen
- IV. Spezielle Beziehungen

3.C.I BINÄRE BEZIEHUNGEN

Der **Grad** einer Beziehung ist die Anzahl der teilnehmenden Entitätstypen.

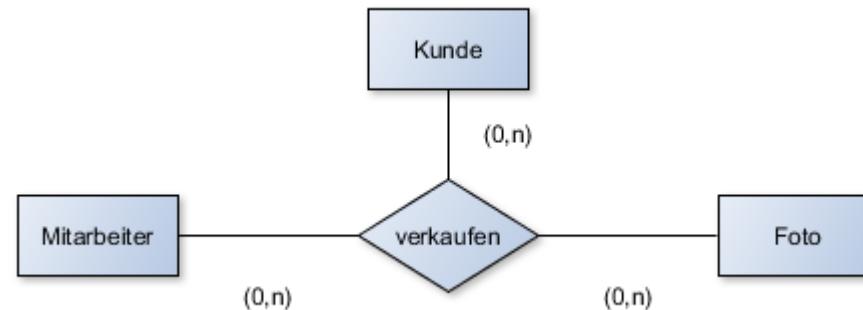
Beziehungen vom Grad 2 nennt man auch **binäre** Beziehungen



3.C.II TERNÄRE BEZIEHUNGEN

Der **Grad** einer Beziehung ist die Anzahl der teilnehmenden Entitätstypen.

Beziehungen vom Grad 3 nennt man auch **ternäre** Beziehungen.

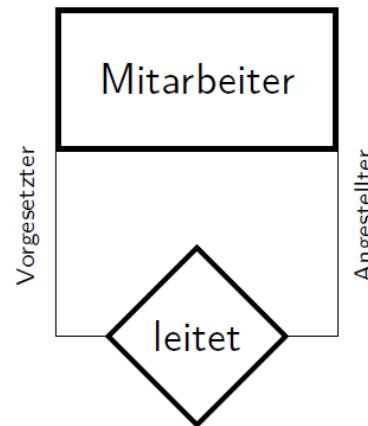


3.C.III REKURSIVE BEZIEHUNGEN

Eine rekursive Beziehung bedeutet, dass ein Entitätstyp in einer Beziehung zu sich selbst steht.

Rekursive Beziehungen sind erlaubt, wenn sie der Realität entsprechen!

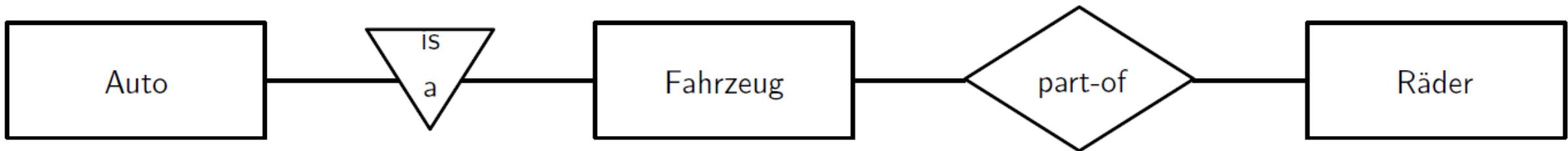
Jeder Entitätstyp, der an einer Beziehung teilnimmt, spielt eine doppelte Rolle.



3.C.IV SPEZIELLE BEZIEHUNGEN

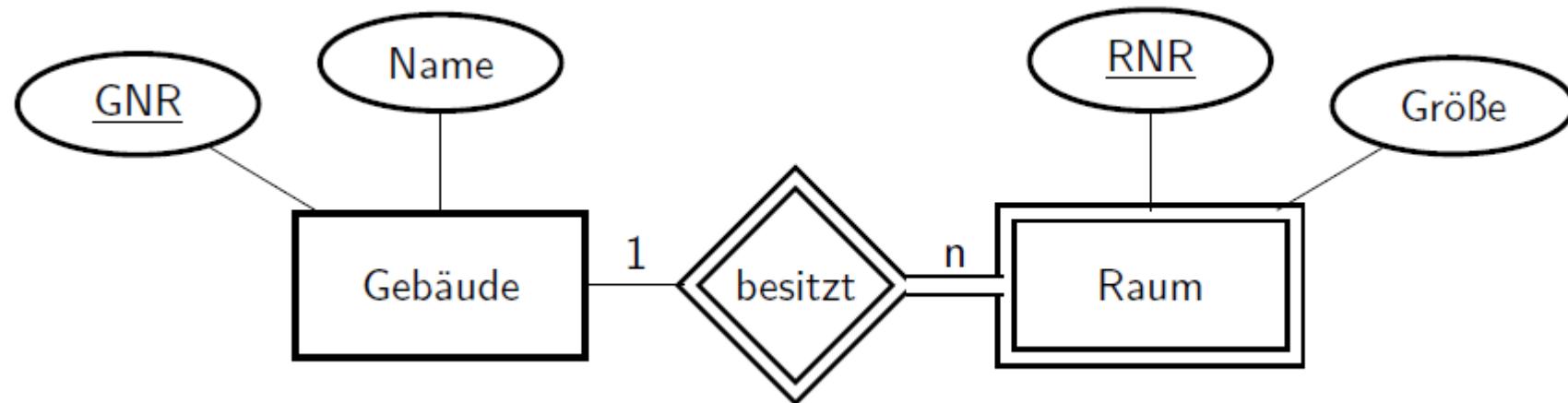
Für die Abbildung von Vererbungen oder Aggregationen können spezielle Beziehungstypen eingesetzt werden:

- **is-a:** Diese Beziehung verkörpert eine Generalisierung (z.B. Auto is-a Fahrzeug). Sie werden durch ein Dreieck dargestellt und verdienen besondere Beachtung bei der Überführung in ein Relationenschema (später in der Vorlesung).
- **part-of:** Diese Beziehung verkörpert eine Aggregation und wird mit einem normalen Beziehungssymbol versehen.



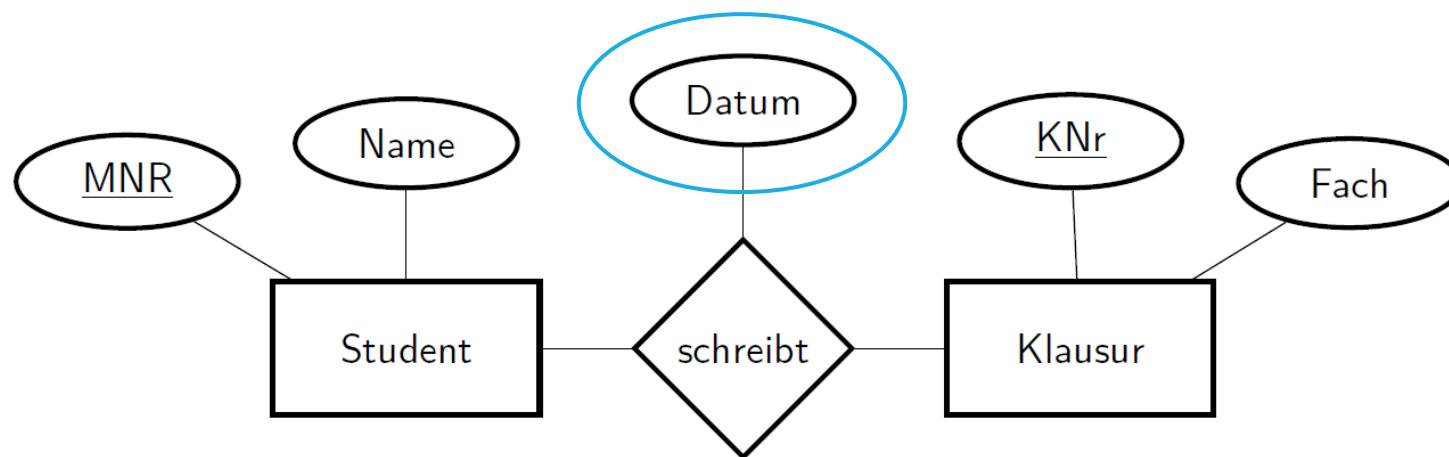
3.C.IV SPEZIELLE BEZIEHUNGEN

Schwache Entitäten sind Entitäten, welche nicht alleine durch deren Attribute identifiziert werden können, sondern die Entitäten von anderen Entitätstypen benötigen. Diese Beziehungen markiert man mit Doppelstrichen:



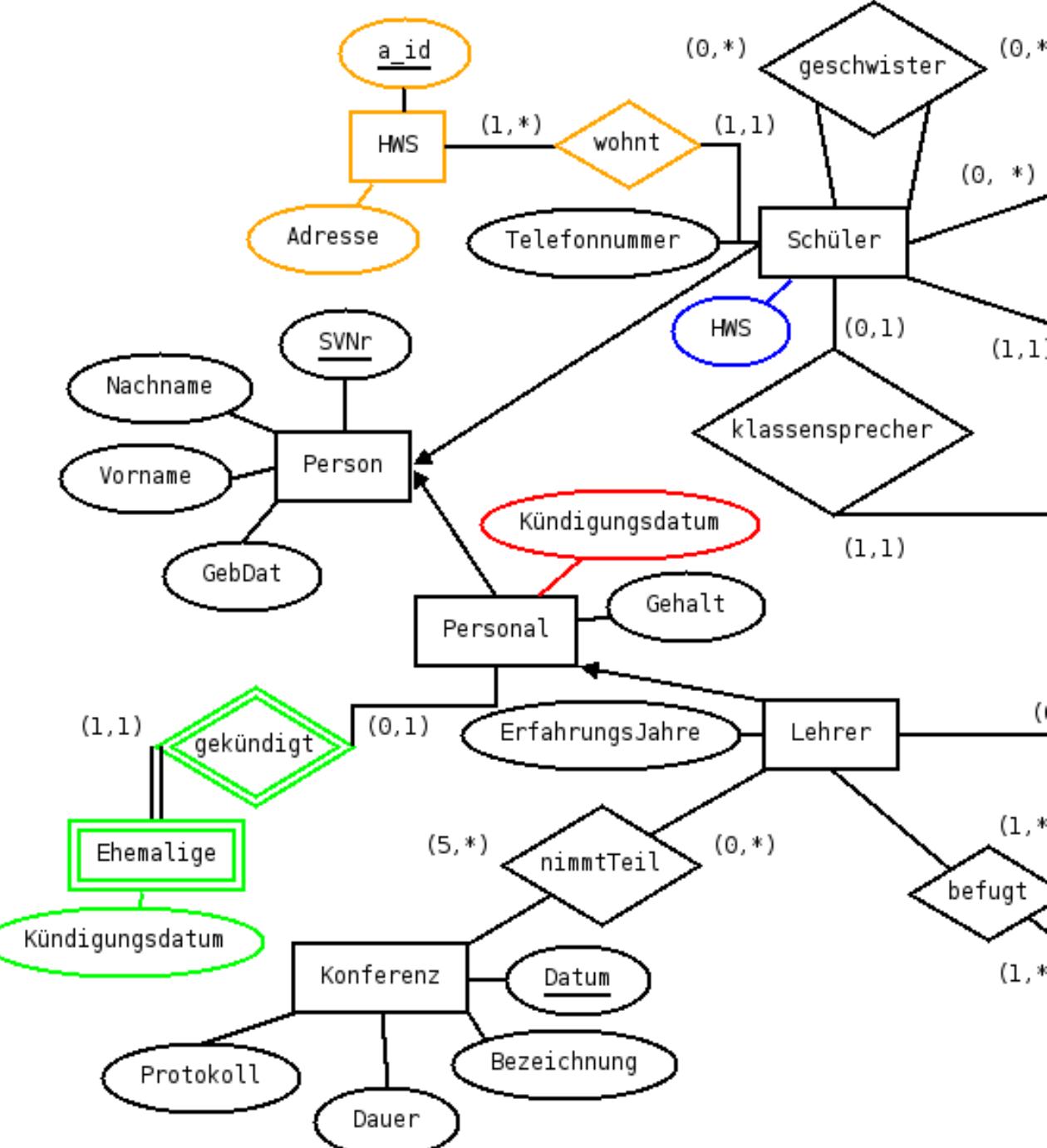
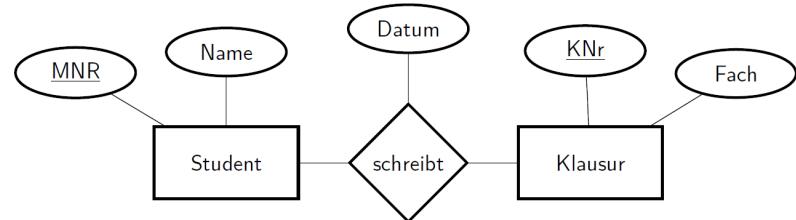
3.C BEZIEHUNGEN

Zusätzlich können Beziehungen mit Attributen versehen werden:



BEZIEHUNGEN: FRAGEN

1. Welche Arten von Beziehungen gibt es?
2. Wie werden Beziehungen im Entity-Relation-Diagramm dargestellt?
3. Was kann ein Grund für eine rekursive Beziehung sein?
4. Warum ist bei folgender Beziehung das Attribut „Datum“ sinnvoller bei der Beziehung, als bei den Entitätstypen aufgehängt?



3.D KARDINALITÄTEN

Die Anzahl von potentiellen Beziehungsinstanzen, an denen eine Entität teilnehmen kann, ist die Kardinalität.

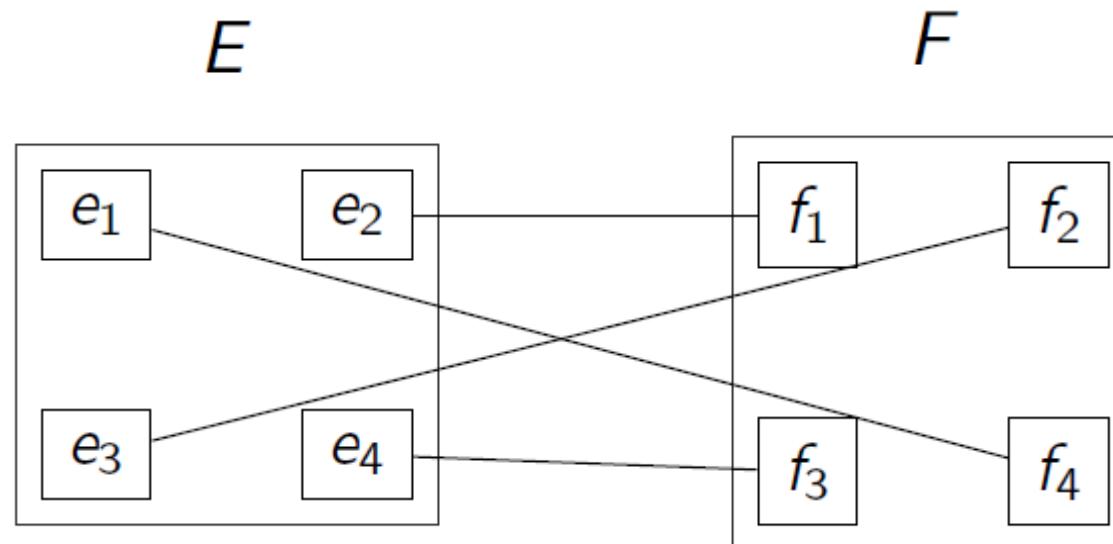
Mögliche Kardinalitätsverhältnisse an Beispielen

- 1:1 Eine Person besitzt genau einen Ausweis. Zwischen den Entitätstypen *Person* und *Ausweis* besteht ein 1:1 - Beziehungstyp.
- 1:n Eine Person besitzt mehrere Autos, jedes Auto wird aber nur von einer Person besessen.
- n:m Eine Person arbeitet an mehreren Projekten, jedes Projekt wird durch mehrere Personen bearbeitet.

3.D KARDINALITÄTEN

1:1 Kardinalität:

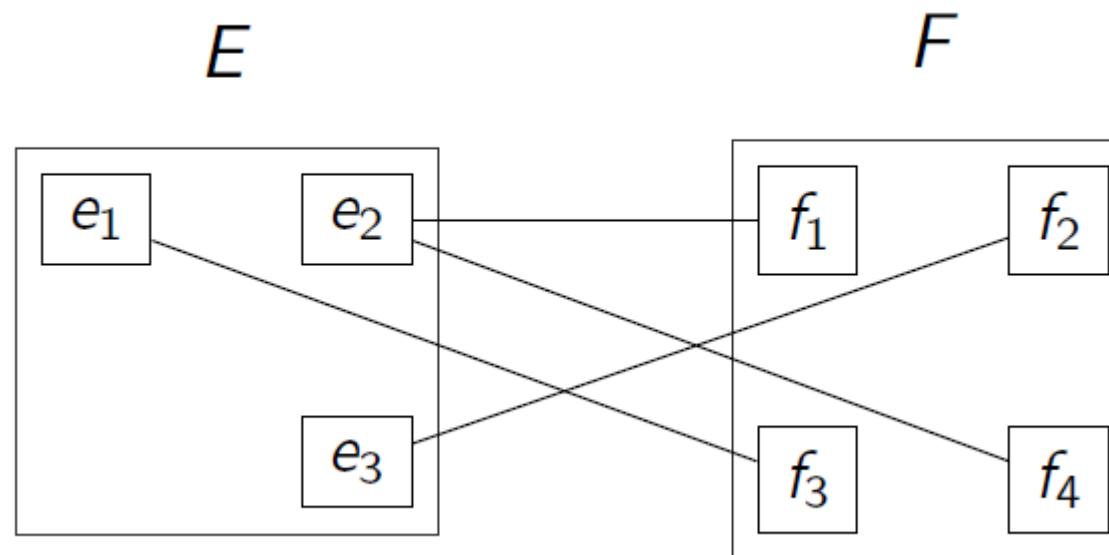
Ordnet eine Entität e_n aus einer Entitätenmenge E genau einer Entität f_m aus der Entitätenmenge F zu. Eine Entität f_m ist genau einer Entität e_n zugeordnet.



3.D KARDINALITÄTEN

1:n Kardinalität:

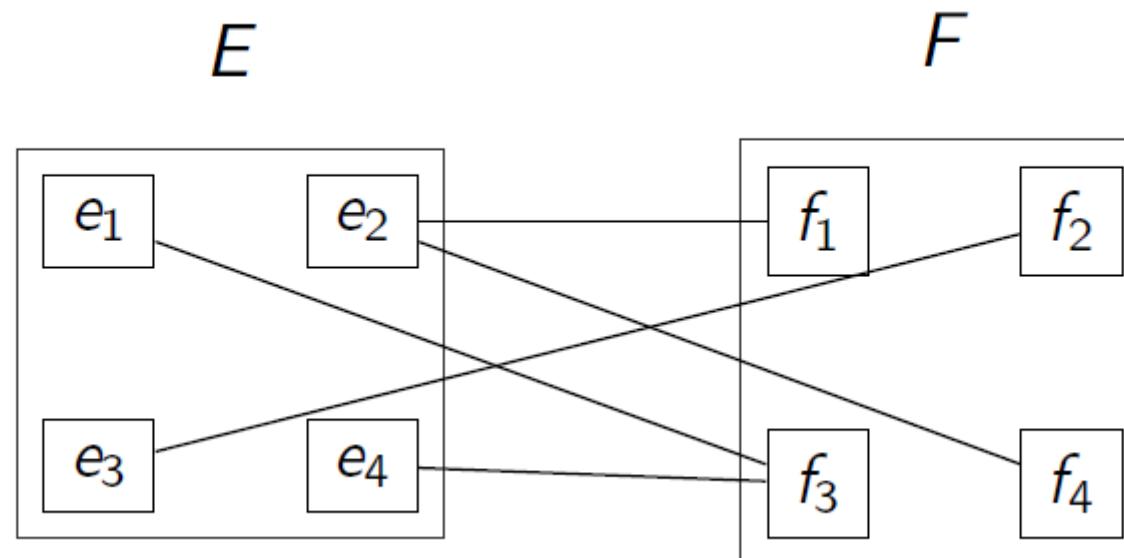
Ordnet einer Entität e_n aus einer Entitätenmenge E mehrere Entitäten f_m aus der Entitätenmenge F zu.



3.D KARDINALITÄTEN

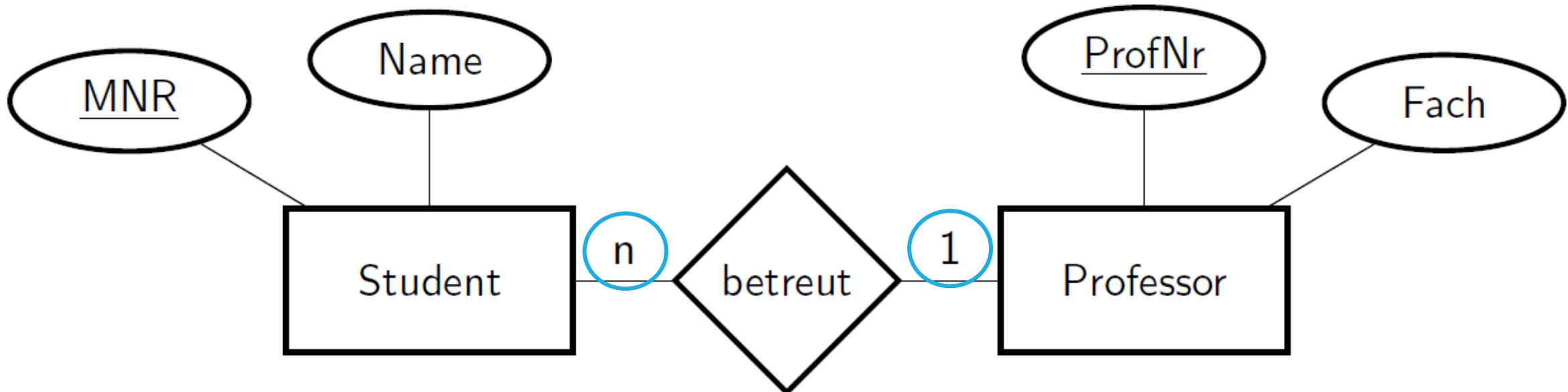
n:m Kardinalität:

Ordnet einer Entität e_n aus einer Entitätenmenge E mehrere Entitäten f_m aus der Entitätenmenge F zu. Allerdings können jeder Entität f_m mehrere Entitäten e_n zugeordnet werden.



3.D KARDINALITÄTEN

Kardinalitäten werden im ER-Diagramm direkt an die Verbindung zwischen Entitätstyp und Beziehungstyp geschrieben!



KARDINALITÄTEN: FRAGEN

1. Welche Arten von Kardinalitäten gibt es?
2. Was ist das Problem einer n:m-Beziehung?
3. Wie werden Kardinalitäten notiert?
4. Was bedeutet eine 0 als Kardinalität?



3.E ÜBUNGEN

1. Erstellen Sie ein Anforderungsprofil für eine Kalender-App mit zugehöriger Datenbank. Leiten Sie ein rudimentäres ER-Diagramm mit Kardinalitäten daraus ab.
2. Bringen Sie Beispiele aus einem der folgenden Umfelder für je eine 1:1, eine 1:n und eine n:m-Beziehung:
 - Sport
 - Film
 - Programmierung
3. Erstellen Sie eine sinnvolle ternäre Beziehung, bei der die Beziehung mit Attributen versehen ist.



4. DATENBANK-KONZEPTION

KAPITEL-AGENDA:

4. DATENBANKKONZEPTION

- a. Überführung ER-Modell zu Relationen
- b. Probleme & Normalisierung
- c. Übungen

4.A ÜBERFÜHRUNG ER-MODELL ZU RELATIONEN

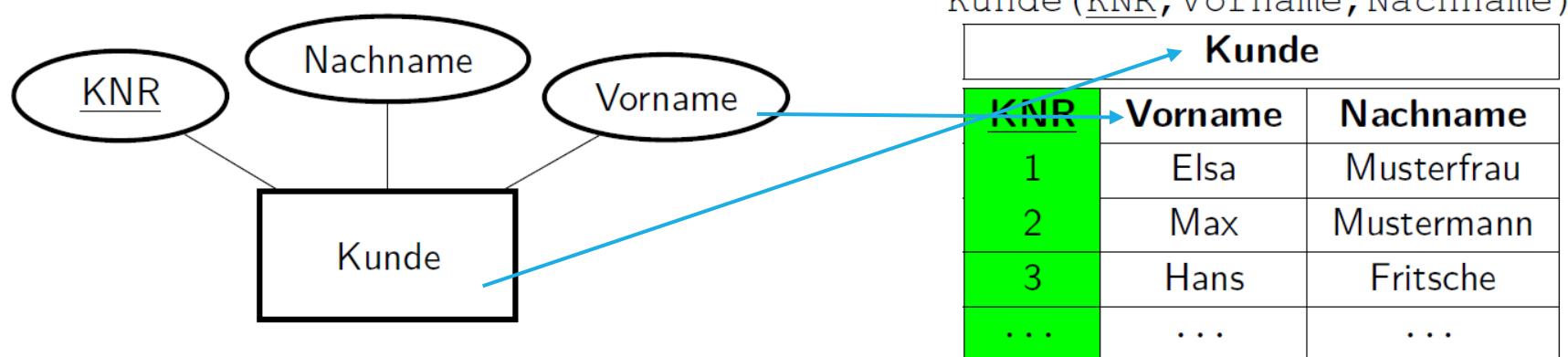
Entitätstypen entsprechen Relationenschemata

- Entitäten entsprechen einem Tupel einer Relation
- Beziehungen können durch Fremdschlüssel ausgedrückt werden

→ ER-Modelle können leicht in das relationale Datenmodell überführt werden. Es müssen jedoch einige Besonderheiten bei der Überführung des ER-Modells in ein relationales Datenmodell beachtet werden.

4.A ÜBERFÜHRUNG ER-MODELL ZU RELATIONEN

Entitäten und deren Attribute können direkt in ein Relationenschema überführt werden.



Der Name der Entität wird zum Namen des Relationenschemas, Attribute werden direkt übernommen, wobei auch die Schlüsseleigenschaften erhalten bleiben.

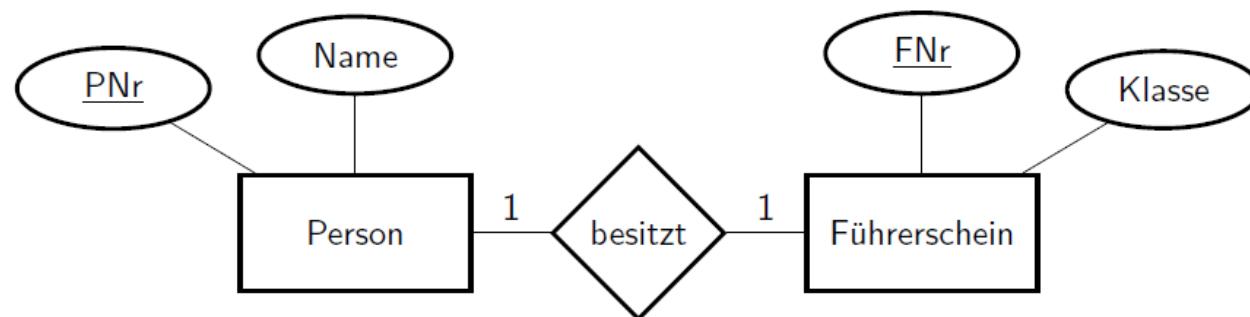
4.A ÜBERFÜHRUNG ER-MODELL ZU RELATIONEN

Für verschiedene Beziehungstypen und Besonderheiten gibt es unterschiedliche Handhabungen, um diese vom ER-Schema ins relationale Modell zu überführen:

- I. 1:1 Beziehungen
- II. 1:n Beziehungen
- III. Schwache Entitäten
- IV. N:m Beziehungen
- V. Beziehungen vom Grad >2
- VI. Mehrwertige und zusammengesetzte Attribute
- VII. Is-A Beziehungen

4.A.1 1:1 BEZIEHUNGEN

Beim Überführen von 1:1 - Beziehungen wird eine beteiligte Relation um das Fremdschlüsselattribut auf die jeweils andere beteiligte Relation erweitert.



Alternative 1:

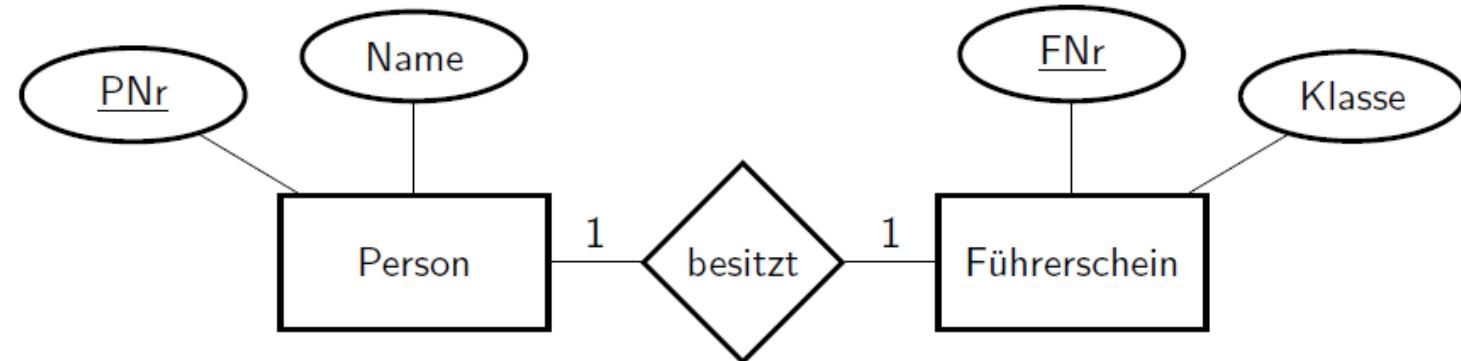
Person (PNr, Name, FNr)
Führerschein (FNr, Klasse)

Alternative 2:

Person (PNr, Name)
Führerschein (FNr, Klasse, PNr)

4.A.1 1:1 BEZIEHUNGEN

Alternative 1 als Relationen:

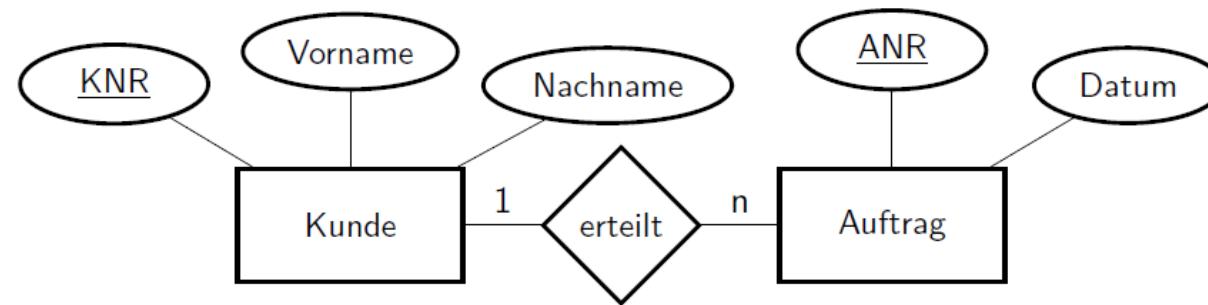


Person		
PNr	Name	FNr
1	Musterfrau	1002
2	Mustermann	1001
3	Fritsche	1003
...

Führerschein	
FNRL	Klasse
1001	1
1002	2
1003	3
...	...

4.A.II 1:N BEZIEHUNGEN

Beim Überführen von 1:n - Beziehungen wird die Relation bzw. [Entität auf der n-Seite](#) der Beziehung um einen Fremdschlüssel erweitert.



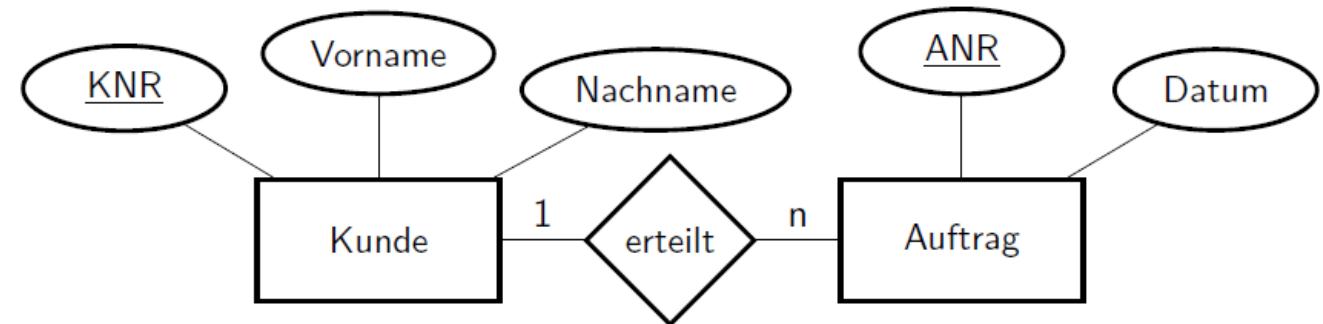
Relationenschema:

Kunde (KNR, Vorname, Nachname)

Auftrag (ANR, KNR, Datum)

4.A.II 1:N BEZIEHUNGEN

Als **Relationen**:

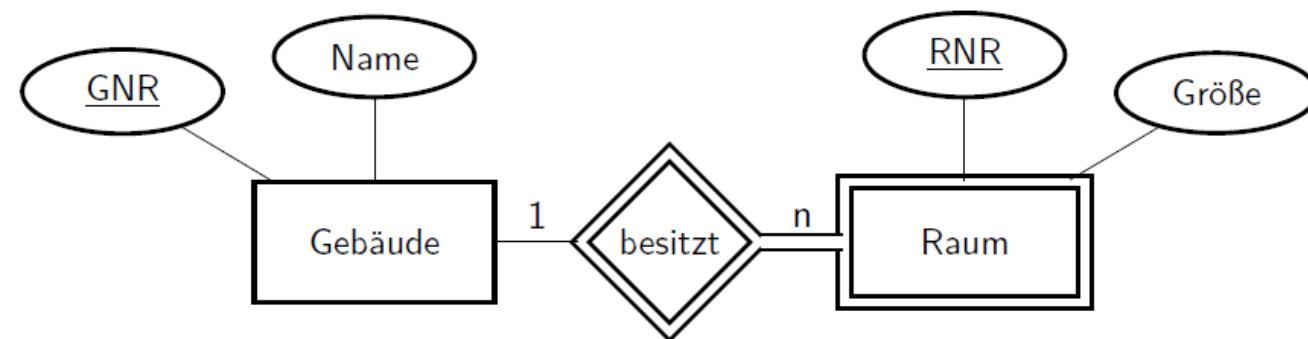


Kunde		
<u>KNR</u>	<u>Vorname</u>	<u>Nachname</u>
1	Elsa	Musterfrau
2	Max	Mustermann
3	Hans	Fritsche
...

Auftrag		
<u>ANR</u>	<u>KNR</u>	<u>Datum</u>
1001	1	02.01.2013
1002	2	04.05.2013
1003	2	03.01.2014
...

4.A.III SCHWACHE ENTITÄTEN

Für den schwachen Entitätstyp wird eine Relation erzeugt, die um einen Fremdschlüssel auf die identifizierende Relation **erweitert wird**. Der Schlüssel wird zusammengesetzt.



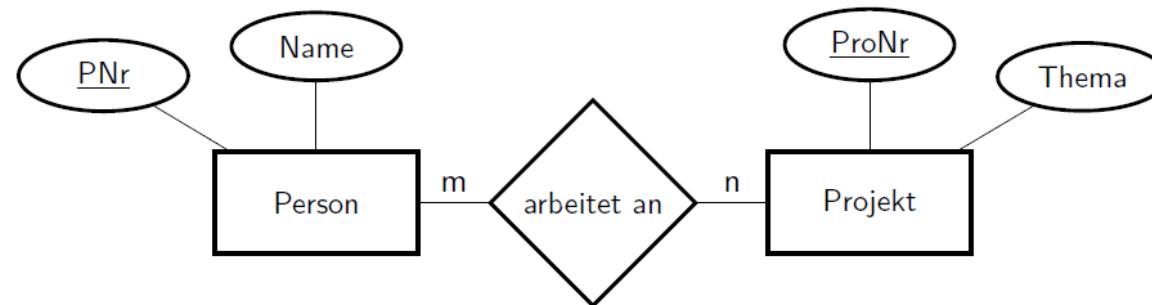
Relationenschema:

Gebäude (GNR, Ort)

Raum (GNR, RNR, Größe)

4.A.IV N:M BEZIEHUNGEN

Beim Überführen von m:n - Beziehungen muss eine **eigene Relation für die Beziehung** gebildet werden. Der Primärschlüssel der neuen Relation wird aus beiden Primärschlüsseln der Ursprungsrelationen zusammengesetzt.



Relationenschema:

Person (PNR, Name)

Projekt (ProNr, Thema)

Person_arbeitet_an_Projekt (PNR, ProNr)

4.A.IV N:M BEZIEHUNGEN

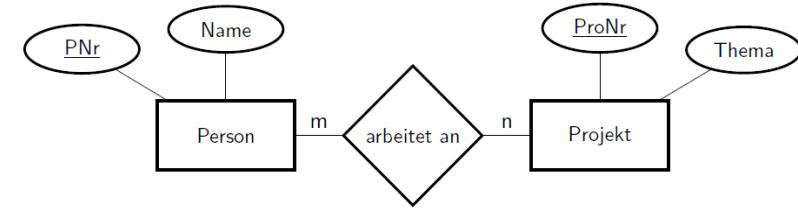


Abbildung als Relationen:

Person	
PNR	Name
1	Mustermann
2	Musterfrau
3	Fritsch
...	...

arbeitet_an	
PNR	ProNr
1	1001
1	1002
2	1002
2	1001

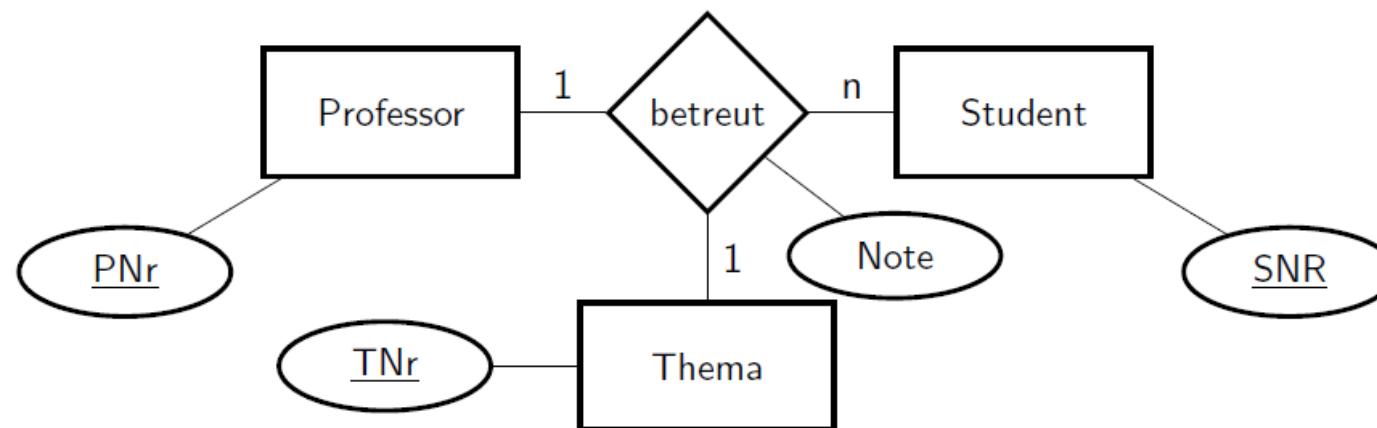
Projekt	
ProNr	Thema
1001	Entwicklung
1002	Marketing
1003	Innovation
...	...

Die Relation arbeitet an beinhaltet Fremdschlüssel auf die jeweiligen Relationen. Die Kombination der Fremdschlüssel ergibt den Primärschlüssel.

Ist im ER-Modell die Beziehung attributiert, so muss immer eine Relation für die Beziehung gebildet werden. In diese Relation werden dann die Attribute übernommen!

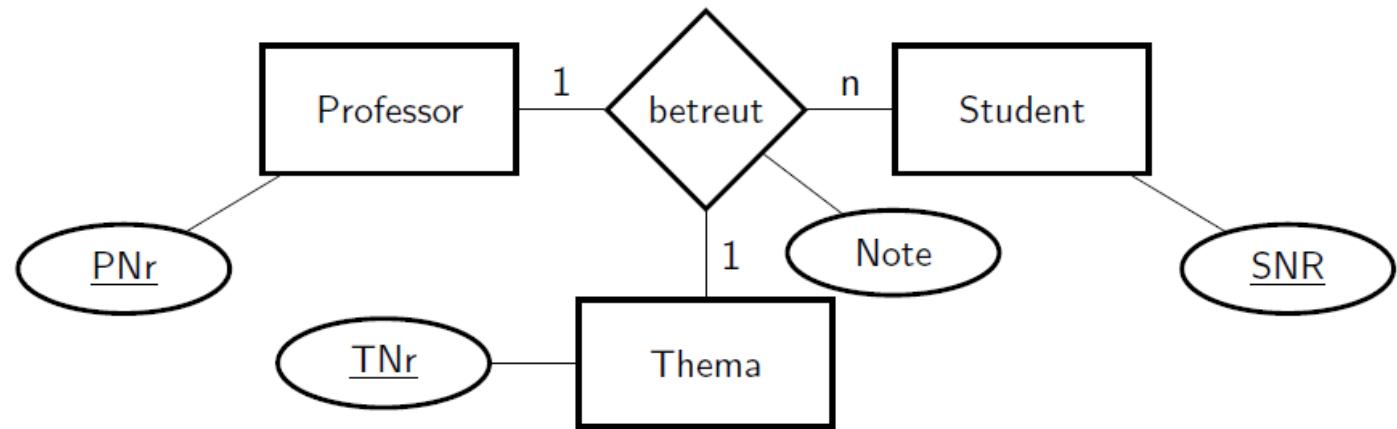
4.A.V BEZIEHUNGEN VOM GRAD >2

Für Beziehungen vom Grad >2 muss ebenfalls eine **Zwischenrelation** gebaut werden. Diese Relation beinhaltet die Primärschlüssel-Attribute **aller beteiligten Relationen**, als Fremdschlüssel, sowie alle ggf. beteiligten Beziehungstyp-Attribute. Der Primärschlüssel setzt sich i.d.R. aus den Fremdschlüsseln zusammen – auf die Semantik muss hier sehr geachtet werden!



4.A.V BEZIEHUNGEN VOM GRAD >2

Als **Relationen**:



Professor

<u>PNr</u>	Name
1	Meier
2	Schulz
...	...

Student

<u>SNr</u>	Name
1	Jörg
2	Lisa
...	...

Thema

<u>TNr</u>	Name
1	Abc
2	Xyz
...	...

betreut

<u>PNr</u>	<u>SNr</u>	<u>TNr</u>	Note
1	1	4	1,4
1	1	13	3,9
1	2	9	2,2
...

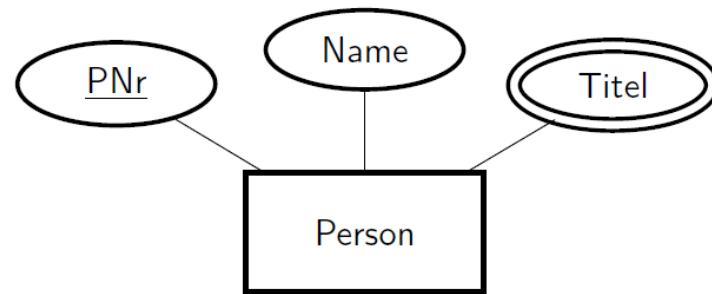
4.A.VI MEHRWERTIGE UND ZUSAMMENGESETZTE ATTRIBUTE

Hierfür gibt es im relationalen Datenmodell keine Entsprechung, diese Attribute müssen daher in atomare Attribute aufgelöst werden:

- Erzeugen von mehreren Attributen direkt in der Relation (bei zusammengesetzten Attributten)
- Erzeugen einer eigenen Entität für das mehrwertige Attribut

4.A.VI MEHRWERTIGE UND ZUSAMMENGESETZTE ATTRIBUTE

Beispiel für mehrwertige Attribute:



Für das mehrwertige Attribut wird eine eigene Relation erstellt. Der Primärschlüssel ist eine Kombination aus dem Attribut selbst und dem Primärschlüssel der Ursprungsrelation, der als Fremdschlüssel in die neue Relation aufgenommen wird.

Person(PNr, Name)

PersTitel(PNr, Titel)

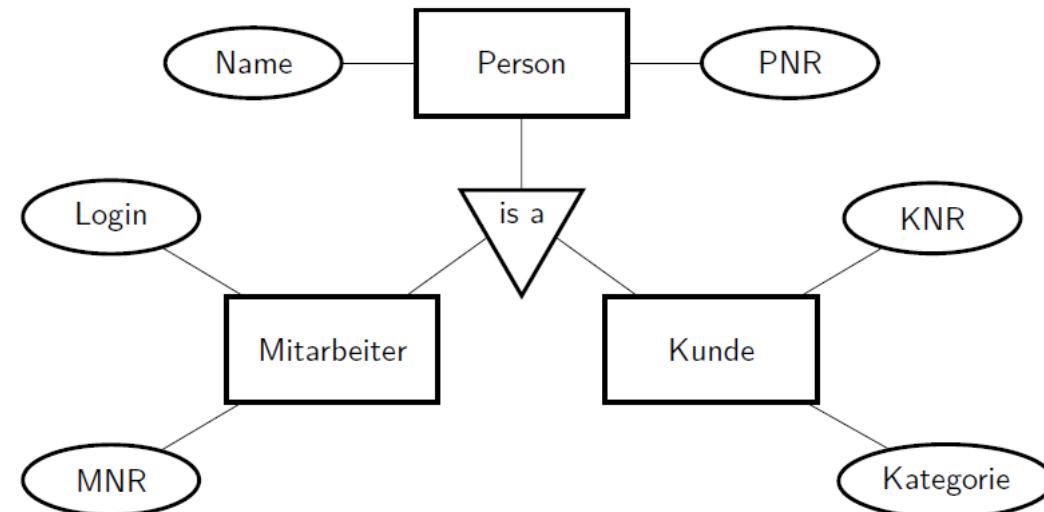
4.A.VII IS-A BEZIEHUNGEN

Is-A – Beziehungen müssen besonders beachtet werden:

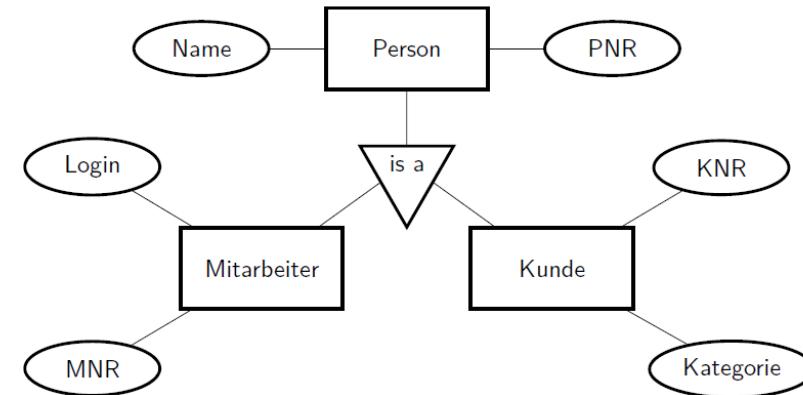
Die Entitäten können hier als Klassen im objektorientierten Modell aufgefasst werden.

Mögliche Ansätze:

- One Table per Class Hierarchy
- One Table per Concrete Class
- One Table per Class



4.A.VII IS-A BEZIEHUNGEN



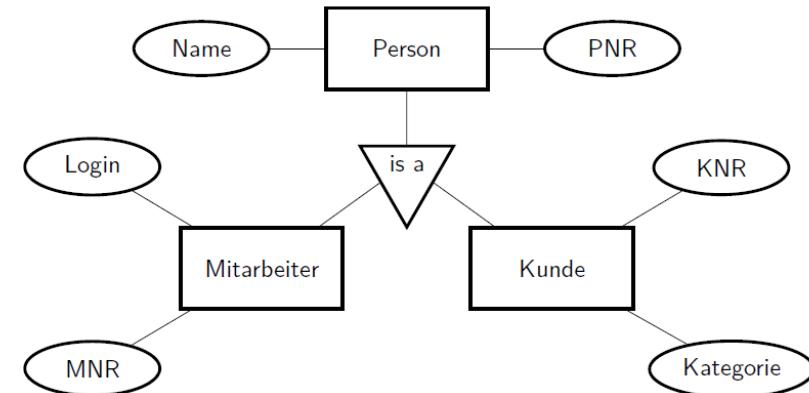
One Table per Class Hierarchy

Alle Attribute der Klassenhierarchie werden in eine gemeinsame Relation überführt und durch ein **Diskriminator-Attribut** unterschieden.

Personen						
<u>PNR</u>	Name	MNR	Login	KNR	Kategorie	Typ
1	Mustermann			1001	A-Kunde	K
2	Musterfrau	13	emfr			M
3	Schneider	14	schn	1002	B-Kunde	B

Das Attribut Typ dient der Unterscheidung zwischen Mitarbeiter (M), Kunde (K) oder beides(B). Der Ansatz ist einfach zu implementieren, provoziert jedoch null-Werte und wird bei vielen Kombinationen schwer wartbar.

4.A.VII IS-A BEZIEHUNGEN



One Table per concrete Class

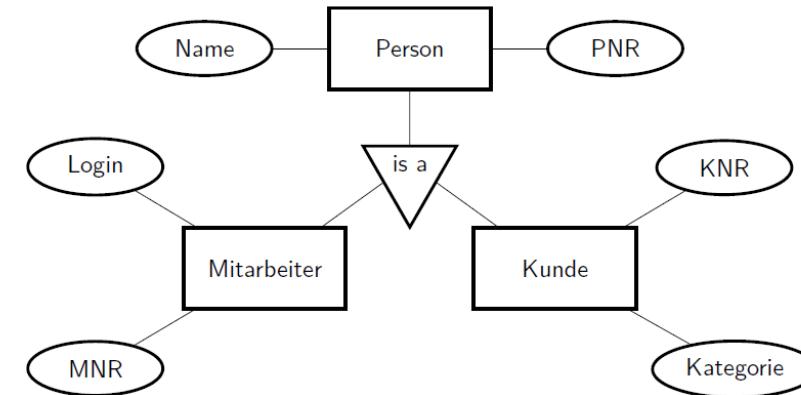
Alle konkreten Klassen werden in einer eigenen Relation abgebildet

Mitarbeiter			
<u>PNR</u>	Name	MNR	Login
2	Musterfrau	13	emfr
3	Schneider	14	schn

Kunde			
<u>PNR</u>	Name	KNR	Kategorie
1	Mustermann	1001	A-Kunde
3	Schneider	1002	B-Kunde

Da die PNR die Eindeutigkeit über alle Relationen hinweg garantiert, kann sie als Primärschlüssel herangezogen werden. Dieser Ansatz ist ebenfalls leicht zu implementieren, jedoch fehlen die Ableitungsinformationen: In diesem Fall existiert keine Person.

4.A.VII IS-A BEZIEHUNGEN



One Table per Class

Alle Klassen (auch die abstrakten) werden in einer eigenen Relation abgebildet

Person	
<u>PNR</u>	<u>Name</u>
1	Mustermann
2	Musterfrau
3	Schneider

Mitarbeiter		
<u>PNR</u>	<u>MNR</u>	<u>Login</u>
2	13	emfr
3	14	schn

Kunde		
<u>PNR</u>	<u>KNR</u>	<u>Kategorie</u>
1	1001	A-Kunde
3	1002	B-Kunde

Die PNR muss in allen Relationen vorhanden sein. Alle Ableitungsinformationen sind vorhanden, jedoch ist ein erhöhter Aufwand nötig, um die einzelnen Tupel mittels Verbundoperationen zusammen zu setzen.

ÜBERFÜHRUNG: FRAGEN

1. Welche Art von Beziehungen gibt es und wie werden diese in ein relationales Schema übertragen?
2. Wie werden Beziehungen vom Grad > 2 in ein relationales Schema übertragen?
3. Was ist das Problem an Is-A Beziehungen? Auf welche Arten ist dieses Problem zu lösen?
4. Wie werden mehrwertige Attribute übertragen? Wie geht das bei zusammengesetzten Attributen?



4.B PROBLEME & NORMALISIERUNG

- I. Anomalien
- II. Schmale vs. breite Relationen
- III. Basics zu Normalisierung
- IV. 1NF
- V. 2NF
- VI. 3NF
- VII. >3NF/BCNF

4.B.I ANOMALIEN

Mitarbeiter_Projekt

<u>MANr</u>	Vorname	Nachname	<u>PNr</u>	Projektname	Funktion
1	Max	Mustermann	4711	Entwicklung	Entwickler
1	Max	Mustermann	4712	Test	Unit-Tests
1	Max	Mustermann	4713	Vertrieb	Kampagnen
2	Erika	Musterfrau	4711	Entwicklung	SCRUM Master
2	Erika	Musterfrau	4712	Test	Integrations-Tests

4.B.I ANOMALIEN

Einfügeanomalie

Mitarbeiter_Projekt					
<u>MANr</u>	Vorname	Nachname	<u>PNr</u>	Projektname	Funktion
1	Max	Mustermann	4711	Entwicklung	Entwickler
1	Max	Mustrmann	4712	Test	Unit-Tests
1	Max	Mustermann			

Falsche Daten (z.B. durch Vertipper) führen zu inkonsistenten Datenbeständen

Entstehung von null-Werten, wenn ein Teiltupel auf einen Datensatz nicht anwendbar ist

4.B.I ANOMALIEN

Modifikationsanomalie

Mitarbeiter_Projekt					
MANr	Vorname	Nachname	PNr	Projektname	Funktion
1	Max	Mustermann	4711	Entwicklung	Entwickler
1	Max	Mustermann	4712	Test	Unit-Tests
2	Erika	Musterfrau	4711	Entwicklung	SCRUM Master
2	Erika	Mustermann	4712	Test	Integrations-Tests

Änderungen eines Attributwertes (z.B. der Nachname von Frau Musterfrau) müssen in allen Tupeln durchgeführt werden

Werden Tupel vergessen, so entstehen ebenfalls inkonsistente Datenbestände

4.B.I ANOMALIEN

Löschanomalie

Mitarbeiter_Projekt					
<u>MANr</u>	<u>Vorname</u>	<u>Nachname</u>	<u>PNr</u>	<u>Projektnname</u>	<u>Funktion</u>
1	Max	Mustermann	4711	Entwicklung	Entwickler
1	Max	Mustermann	4712	Test	Unit-Tests
2	Erika	Musterfrau	4711	Entwicklung	SCRUM Master
3	Frank	Meier	4713	R&D	Leitung

Das Löschen eines Datensatzes vernichtet Informationen über andere Sachverhalte / Entitäten.

4.B.II SCHMALE VS. BREITE RELATIONEN

Zu schmale Relationen:

- Zu viele kleine Abhängigkeiten
- Objekt-Semantik wird zerspalten
- Hoher Aufwand für eine joined Tabelle

MA_Vorname		MA_Str		MA_Nachname	
<u>MANr</u>	Vorname	<u>MANr</u>	Strasse	<u>MANr</u>	Nachname
1	Max	1	Gartenstrasse	1	Mustermann

MA_PLZ		MA_Ort	
<u>MANr</u>	PLZ	<u>MANr</u>	Ort
1	68165	1	Mannheim

4.B.II SCHMALE VS. BREITE RELATIONEN

Zu schmale Relationen:

- Zu viele kleine Abhängigkeiten
- Objekt-Semantik wird zerspalten
- Hoher Aufwand für eine joined Tabelle

MA_Vorname		MA_Str		MA_Nachname	
MANr	Vorname	MANr	Strasse	MANr	Nachname
1	Max	1	Gartenstrasse	1	Mustermann

MA_PLZ		MA_Ort	
MANr	PLZ	MANr	Ort
1	68165	1	Mannheim

Zu breite Relationen:

- Sind zwar abfragetechnisch einfach zu handhaben, leiden aber an Anomalien
- Vermischung unterschiedlicher Entitäten führt zu Unsauberkeiten

4.B.III BASICS ZU NORMALISIERUNG

Definition: Die Datennormalisierung ist der Zerlegungsprozess einer Relation R in mehrere Teilrelationen R_1, R_2, \dots, R_n anhand von Regeln, die die funktionalen Abhängigkeiten von R sowie die Schlüsseleigenschaften der Attribute aus R analysieren.

- Die Zerlegung von R in die Teilrelationen R_1, R_2, \dots, R_n unterliegt der **Verbundtreue**, d.h. R muss aus den Teilrelationen mittels Joins rekonstruiert werden können.
- Die Zerlegung von R muss **abhängigkeitsstreu** sein, d.h. die funktionalen Abhängigkeiten von R müssen in einer der Teilrelationen enthalten sein.

Mit der Datennormalisierung werden systematisch die sogenannten **Normalformen** erstellt.

4.B.IV 1NF

Definition: Eine Relation R ist in 1NF genau dann, wenn alle Attribute von R ausschließlich **atomare** Werte annehmen können. Mehrwertige oder zusammengesetzte Attribute sind nicht gestattet.

Ob ein Attribut atomar ist oder nicht kann von der Verwendung des Attribus abhängig gemacht werden - es muss genau geprüft werden, ob die Zerlegung eines mehrwertigen Attributs **sinnvoll** ist oder ob es als atomares Attribut behandelt werden soll.

4.B.IV 1NF

Person			
<u>PNr</u>	<u>Titel</u>	<u>Vorname</u>	<u>Nachname</u>
1	B.Sc.	Hans	Schneider
2	Dipl.-Kfm.	Max	Mustermann
3	Dipl.-Inf. Dipl.-Kffr.	Erika	Musterfrau



Person		
<u>PNr</u>	<u>Vorname</u>	<u>Nachname</u>
1	Hans	Schneider
2	Max	Mustermann
3	Erika	Musterfrau

Titel	
<u>PNr</u>	<u>Titel</u>
1	B.Sc.
2	Dipl.-Kfm.
3	Dipl.-Inf.
3	Dipl.-Kffr.

Aus dem mehrwertigen Attribut wird eine **eigene Relation** erstellt und über den Primärschlüssel PNr mit der Ursprungsrelation verknüpft. Primärschlüssel der neuen Relation wird aus dem Fremdschlüsselattribut PNr und dem mehrwertigen Attribut zusammengesetzt. Es existiert eine 1:n - Beziehung zwischen den Relationen.

4.B.V 2NF

Definition: Eine Relation R ist in 2NF genau dann, wenn sie in 1NF ist und jedes Attribut von R , das nicht Teil der Menge der Kandidatenschlüssel ist, **voll funktional von den Kandidatenschlüsseln** abhängt und von diesem keine teilfunktionale Abhängigkeit zum Kandidatenschlüssel besteht.

Die Definition impliziert, dass jede Relation, die in 1NF ist und in der die Kandidatenschlüssel **lediglich ein Attribut** aufweisen, **automatisch in 2NF** ist. Der Test auf 2NF muss daher für diejenigen Relationen durchgeführt werden, in denen zusammengesetzte Kandidatenschlüssel enthalten sind.

4.B.V 2NF

Mitarbeiter_Projekt_Fkt				
<u>MANr</u>	Vorname	Nachname	<u>PNr</u>	Funktion
1	Max	Mustermann	4712	Unit-Tests
1	Max	Mustermann	4713	Kampagnen
2	Erika	Musterfrau	4712	Integrations-Tests



Person		
<u>MANr</u>	Vorname	Nachname
1	Max	Mustermann
2	Erika	Musterfrau

Mitarbeiter_Projekt_Fkt		
<u>MANr</u>	<u>PNr</u>	Funktion
1	4712	Unit-Tests
1	4713	Kampagnen
2	4712	Integrations-Tests

Funktionale Abhängigkeiten:

$$\{\text{MANr}, \text{PNr}\} \rightarrow \{\text{Vorname}, \text{Nachname}, \text{Funktion}\}$$

$$\{\text{MANr}\} \rightarrow \{\text{Vorname}, \text{Nachname}\}$$

Vorname, Nachname sind nicht voll funktional vom Schlüssel, sondern nur von einer Teilmenge des Schlüssels abhängig.

4.B.VI 3NF

Definition: Eine Relation R ist genau dann in 3NF, wenn sie in 2NF ist und kein Nicht-Schlüssel-Attribut transitiv von einem Schlüsselkandidaten abhängt.

Eine **transitive Abhängigkeit** liegt dann vor, wenn Y von X funktional abhängig und Z von Y , so ist Z von X funktional abhängig. Diese Abhängigkeit ist transitiv.

Für die 3NF darf es kein solches Y unter den Nicht-Schlüssel-Attributen geben.

4.B.VI 3NF

Mitarbeiter				
<u>MANr</u>	Vorname	Nachname	PLZ	Ort
1	Max	Mustermann	68165	Mannheim
2	Erika	Musterfrau	57072	Siegen
3	Hans	Schmidt	68165	Mannheim



Mitarbeiter				
<u>MANr</u>	Vorname	Nachname	PLZ	Ort
1	Max	Mustermann	68165	Mannheim
2	Erika	Musterfrau	57072	Siegen
3	Hans	Schmidt	68165	Mannheim

Ort	
<u>PLZ</u>	Ort
68165	Mannheim
57072	Siegen

Funktionale Abhängigkeiten:

$\{MANr\} \rightarrow \{Vorname, Nachname, PLZ, Ort\}$

$\{PLZ\} \rightarrow \{Ort\}$

Transitive Abhängigkeit: $\{MANr\} \rightarrow \{PLZ\} \rightarrow \{Ort\}$, Ort ist Nicht-Schlüssel-Attribut

4.B.VII > 3NF/BCNF

Definition: Eine Relation R ist genau dann in BCNF, wenn sie in 3NF ist und kein Attribut transitiv von einem Schlüsselkandidaten abhängt.

Verallgemeinerung der 3NF: die BCNF bezieht sich auf alle Attribute, nicht nur auf die Nicht-Schlüssel-Attribute. Sie ist daher eine Verschärfung der 3NF: Die BCNF verhindert, dass Teile von Schlüsselkandidaten funktionale Abhängigkeiten aufweisen.

4.B.VII > 3NF/BCNF

Definition BCNF (Boyce Codd Normalform): Eine Relation R ist genau dann in BCNF, wenn sie in 3NF ist und kein Attribut [(egal ob Schlüssel oder nicht)] transitiv von einem **Schlüsselkandidaten** abhängt.

Verallgemeinerung der 3NF: die BCNF bezieht sich auf alle Attribute, nicht nur auf die Nicht-Schlüssel-Attribute. Sie ist daher eine Verschärfung der 3NF: Die BCNF verhindert, dass Teile von Schlüsselkandidaten funktionale Abhängigkeiten aufweisen.

Einen Unterschied zwischen Dritter Normalform und Boyce-Codd Normalform gibt es nur, wenn es mehrere Schlüsselkandidaten mit überlappenden Attributen gibt.

4.B.VII > 3NF/BCNF

Wenn Schlüsselkandidaten voneinander funktional abhängig sind, ist eine Auslagerung **in einer weitere Tabelle vonnöten**. Gibt es keine voneinander abhängigen Schlüsselkandidaten, ist die BCNF erreicht.

<u>ReNr</u>	<u>ArtNr</u>	<u>LagerOrt</u>	Anzahl
100100	1010	22	1
100100	1020	15	2
100100	1030	9	5
100103	1040	13	10
100104	1040	13	6



<u>ReNr</u>	<u>ArtNr</u>	Anzahl
100100	1010	1
100100	1020	2
100100	1030	5
100103	1040	10
100104	1040	6

<u>ArtNr</u>	<u>LagerOrt</u>
1010	22
1020	15
1030	9
1040	13
1050	5

4.B.VII > 3NF/BCNF

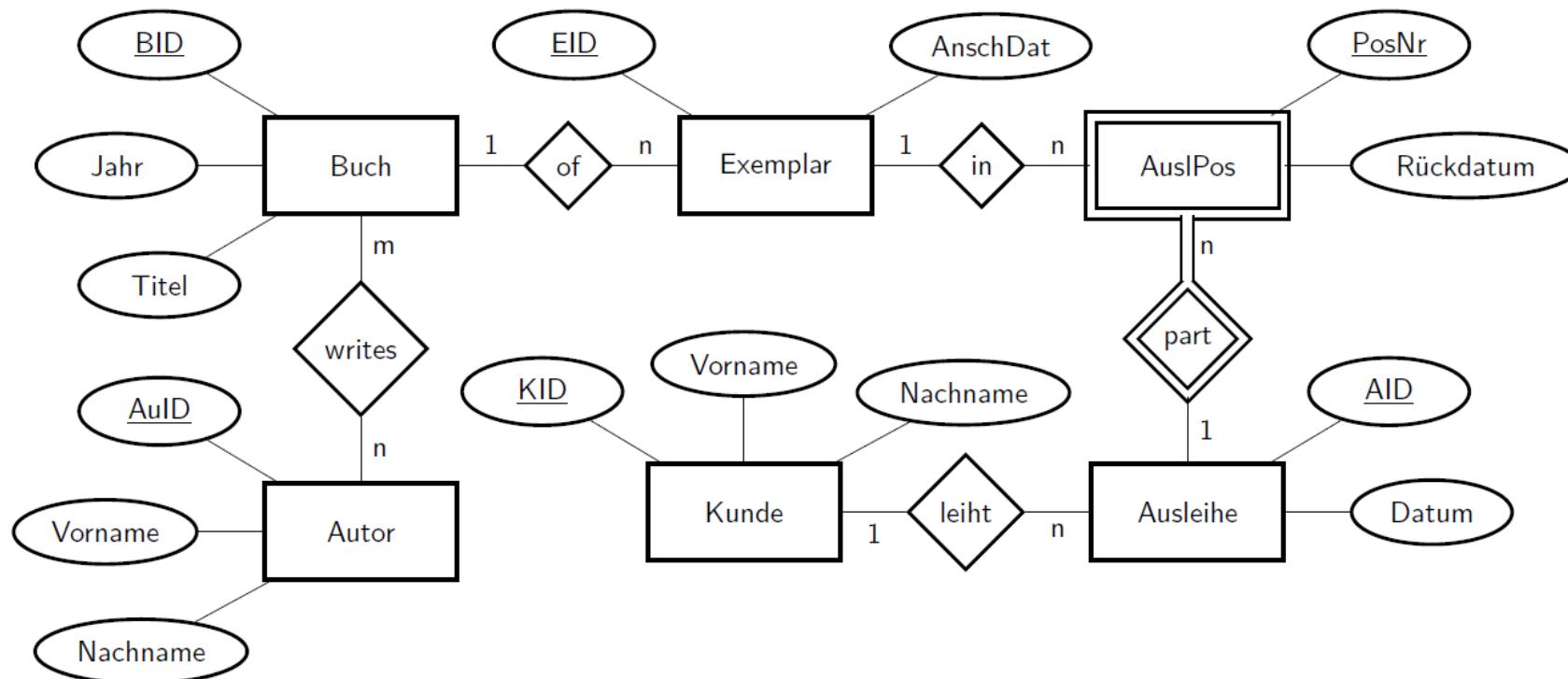
Definition 4NF: Ein Relationstyp befindet sich genau dann in der vierten Normalform (4NF), wenn er sich in der Boyce Codd Normalform (BCNF) befindet und für jede mehrwertige Abhängigkeit einer Attributmenge Y von einer Attributmenge X gilt:

- Die mehrwertige Abhängigkeit ist trivial ist oder
- X ist ein Schlüsselkandidat der Relation.

Definition 5NF: Ein Relationstyp befindet sich genau dann in der fünften Normalform (5NF), wenn er sich in der Vierte Normalform (4NF) befindet und für jede Abhängigkeit (R_1, R_2, \dots, R_n) gilt:
– Die Abhängigkeit ist trivial oder
– Jedes R_i aus (R_1, R_2, \dots, R_n) ist Schlüsselkandidat der Relation. Die Fünfte Normalform (5NF) wird unter anderem als Project Join Normalform (PJNF) bezeichnet.

4.C ÜBUNGEN

- Überführen Sie folgendes ER-Modell in ein logisches Datenmodell in mindestens dritter Normalform:





5. SQL-ABFRAGEN

KAPITEL-AGENDA:

5. SQL-ABFRAGEN

- a. Basics zu SQL
- b. Statements
- c. Erweiterte SQL Statements
- d. Views und Indices
- e. Übungen

5.A BASICS ZU SQL

Die Structured Query Language (SQL) wurde bereits in den 1970er Jahren von IBM entwickelt.

- Ursprünglicher Name: Sequel
- Entwicklung im Rahmen von IBM System/R
- Heute der Standard zur Arbeit mit RDBMS
- Theoretische Grundlage bildet die relationale Algebra
- Verschiedene Versionen, etabliert in den Normen ISO/IEC 9075 (Datenbanksprachen)
 - SQL89
 - SQL92
 - SQL99
- Viele proprietäre Erweiterungen und unterschiedliche Dialekte

5.A BASICS ZU SQL

SQL beinhaltet Befehle und Operationen aus den Bereichen der **Data Definition Language** (DDL) sowie der **Data Manipulation Language** (DML)

- Erzeugen von Schemata (DDL)
 - Definition von Relationen und Attributen mit ihren Wertebereichen
 - Beziehungen zwischen Relationen
 - Integritätsbedingungen
- Abfrage und Manipulation (DML)
 - Lesen (abfragen) von Datensätzen
 - Ändern von Attributwerten
 - Löschen von Datensätzen

Weiterhin kann mit SQL ein Zugriffs- und Berechtigungskonzept implementiert sowie physische Zugriffspfade (Indizes) definiert werden.

5.A BASICS ZU SQL

SQL ist eine **deskriptive** Sprache

```
1 SELECT * FROM Kunde  
2 WHERE PLZ = 68165;
```

```
1 ArrayList<Kunde> result = new ArrayList<>();  
2 for (Kunde tmpKunde : kundenListe) {  
3     if (tmpKunde.getPLZ() == 68165) {  
4         result.add(tmpKunde);  
5     }  
6 }
```

Während SQL (links) das gewünschte Ergebnis **beschreibt**, wird mittels Java (rechts) der Weg (der Algorithmus) angegeben, **wie** das Ergebnis herbeigeführt wird. SQL-Befehle können daher auf unterschiedliche Art implementiert werden (siehe Vorlesung *Datenbanktechnik*).

5.A BASICS ZU SQL

Nullwerte und dreiwertige Logik

- Nullwerte können in Tabellen vorkommen, die ggf. unterschiedliche Bedeutung haben:
 - Es existiert kein Wert
 - Es könnte ein Wert existieren, der aber unbekannt ist
 - Jeder Wert ist möglich
 - . . . (insgesamt 13 verschiedene Bedeutungen von NULL)

Eine einheitliche Semantik für NULL-Werte kann nicht existieren, daher muss SQL auf spezielle Weise mit NULL umgehen.

5.A BASICS ZU SQL

Welches Ergebnis liefern die untenstehenden Anfragen?

Kunden		
Vorname	Nachname	Ort
Max	Mustermann	Siegen
Erika	Musterfrau	Mannheim
Heiner	Ott	NULL

```
1 SELECT Vorname, Nachname  
2 FROM Kunden  
3 WHERE Ort = "Mannheim"
```

```
1 SELECT Vorname, Nachname  
2 FROM Kunden  
3 WHERE NOT (Ort = "Mannheim")
```

Die WHERE-Bedingungen können beim letzten Tupel **weder zu wahr noch zu falsch evaluiert werden!**

5.A BASICS ZU SQL

Dreiwertige Logik

- Es wird unterschieden zwischen wahr, falsch und unbekannt
- Tupel mit leeren Werten sollen herausgefiltert werden, damit das Anfrageergebnis nicht verzerrt wird
- In SQL ergibt ein Vergleich mit NULL immer den Wahrheitswert unbekannt
- Expliziter Test auf NULL in SQL: IS NULL (Nicht “= NULL”)

Richtig:

```
1 SELECT Ort FROM Kunden  
2 WHERE Ort IS NULL
```

Falsch:

```
1 SELECT Ort FROM Kunden  
2 WHERE Ort = NULL
```

5.A BASICS ZU SQL

Die [Wahrheitstafel](#) der zweiwertigen Logik:

X	Y	NOT X	X AND Y	X OR Y
f	f	t	f	f
f	t	t	f	t
t	f	f	f	t
t	t	f	t	t

5.A BASICS ZU SQL

Die [Wahrheitstafel](#) der dreiwertigen Logik in SQL:

X	Y	NOT X	X AND Y	X OR Y
f	f	t	f	f
f	u	t	f	u
f	t	t	f	t
u	f	u	f	u
u	u	u	u	u
u	t	u	u	t
t	f	f	f	t
t	u	f	u	t
t	t	f	t	t

5.A BASICS ZU SQL

Eine (kleine) Auswahl der Datentypen:

Bezeichnung	Beschreibung
bit, bool	Wahrheitswert, 0 oder 1, wahr oder falsch
char(n)	Character-Sequenz mit einer Fixlänge von n Zeichen
varchar(n)	Character-Sequenz mit einer Maximallänge von n Zeichen
nchar(n)	Siehe char(n), aber als Unicode-Character (auch nvarchar)
int	Ganzzahlen (4 Byte, von -2147483648 – 2147483647)
decimal(p,d)	Fixpoint-Zahl mit insgesamt p Stellen, davon d Nachkommastellen
real,float,double	Floatingpoint-Zahl mit einfacher (32 Bit) oder doppelter (64 Bit) Genauigkeit
date,time	Datums- und Zeitangaben
BLOB	Binary Large Object für Binärdaten

5.A BASICS ZU SQL

Besondere Datentypen (**propriätär!**)

- enum, um Aufzählungen abzubilden
- Geometrische Datentypen (z.B. point, circle oder polygon)
- Netzwerktypen (z.B. ipv4, cidr oder macaddr)
- Universally Unique Identifiers (z.B. uuid)
- Spezielle Datentypen für Volltextsuchen im Bereich des Natural Language Processing (z.B. tsvector)

Jedes RDBMS hat ein eigenes Set von proprietären Datentypen, lediglich die Basisdatentypen sind interoperabel!

SQL BASICS: FRAGEN

1. Warum ist NULL ein Problem? Wie geht SQL damit um?
2. Was für verschiedene Datentypen gibt es?
3. Was bedeutet proprietärer Datentyp?
4. Ist SQL für DDL oder DML anzuwenden?



Statement 1 Statement 2 Database Explorer 3

```
1SELECT reported.city,
2      reported.count_accidents::float /
3      (reported.count_accidents
4       + not_reported.count_accidents)::float AS ratio
5FROM
6  (SELECT city, COUNT(*) AS count_accidents
7   FROM accidents
8   WHERE reported = 'true'
9   GROUP BY city) as reported
10JOIN
11  (SELECT city, COUNT(*) AS count_accidents
12   FROM accidents
13   WHERE reported = 'false'
14   GROUP BY city) as not_reported
15ON reported.city = not_reported.city
16ORDER BY ratio DESC
17LIMIT 3;
```



Result 1 Messages

city	ratio
ROSTOCK	0.67
MAGDEBURG	0.67
BREMEN	0.55

5.B STATEMENTS

Arten von Statements:

- I. Umgang mit Strukturen
- II. Eingeben und Ändern von Daten
- III. Auslesen von Daten
- IV. Joins
- V. Nested Selects
- VI. Gruppierung und Aggregation

5.B.I UMGANG MIT STRUKTUREN

Erzeugen einer Datenbank innerhalb des Datenbankmanagementsystems.

Das Erzeugen einer leeren Datenbank wird durch den Befehl CREATE DATABASE durchgeführt.

CREATE DATABASE <schemaname>;

Dieser Befehl erzeugt eine leere Datenbank mit dem angegebenen Namen <schemaname>. In diesem Schema können nun Relationen usw. angelegt werden.

5.B.I UMGANG MIT STRUKTUREN

Eine Relation R mit den Attributen a_1, \dots, a_n und den dazugehörigen Datentypen d_1, \dots, d_n wird mit dem folgenden Befehl erzeugt:

```
CREATE TABLE R (
    a1 d1,
    a2 d2,
    ...
    an dn
);
```

Die Attributnamen a_1, \dots, a_n können (nahezu) beliebig sein (z.B. Vorname, Nachname, PLZ). Die Datentypen müssen aus der Menge der vom RDBMS unterstützten Datentypen stammen (z.B. int, nvarchar etc.).

5.B.1 UMGANG MIT STRUKTUREN

Beispiel:

```
CREATE TABLE Film (
    FilmNr int,
    Name nvarchar(50),
    ErschJahr int
);
```

In diesem Beispiel wird die Relation Film erzeugt. Die Attribute FilmNr und ErschJahr sind Ganzzahlen vom Typ int, der Name ist eine Unicode-Zeichenkette mit max. 50 Zeichen. In diesem Beispiel sind weder Primärschlüssel noch Nebenbedingungen (z.B. das der Filmname nicht leer sein darf) definiert worden! SQL bietet aber diese Möglichkeiten mittels Integritätsbedingungen.

5.B.I UMGANG MIT STRUKTUREN

Attribute können Integritätsbedingungen enthalten, die direkt hinter der Datentyp-Deklaration eingefügt werden können:

PRIMARY KEY: Definition eines Attributs als Primärschlüssel

REFERENCES: Setzt eine Fremdschlüssel-Beziehung auf eine andere Relation

UNIQUE: Attribut muss eindeutig sein, ist aber *nicht* der Primärschlüssel

NOT NULL: Attribut darf nicht null sein

DEFAULT: Es kann ein Default-Wert für ein Attribut festgelegt werden

CHECK: Es kann eine zusätzliche Prüfbedingung angegeben werden (z.B. keine negativen Altersangaben oder Jahreszahlen)

5.B.I UMGANG MIT STRUKTUREN

Parameter für Integritätsbedingungen mit **CHECK**

- Vergleichsoperatoren <,,=,,>
- [NOT] IN (<Werteliste>)
- [NOT] BETWEEN (<Werteliste>)
- [NOT] LIKE (<Textpattern>)

Beispiel:

CREATE TABLE Ort

...

PLZ int **CHECK(VALUE BETWEEN(0 AND 99999)),**

...

);

5.B.I UMGANG MIT STRUKTUREN

Beispiel:

```
CREATE TABLE Film (
    FilmNr int PRIMARY KEY,
    Name nvarchar(50) NOT NULL,
    ErschJahr int CHECK (ErschJahr > 1900),
    GenreID REFERENCES Genre(GenreID)
);
```

In diesem Beispiel ist die FilmNr der Primärschlüssel der Relation, der Name darf nicht den Wert null annehmen und das ErschJahr muss mit Werten > 1900 belegt sein. Die neu hinzugekommene GenreID stellt einen Fremdschlüssel auf die Relation Genre und ihr Attribut GenreID dar.

5.B.I UMGANG MIT STRUKTUREN

Integritätsbedingungen können auch nachgelagert werden. So können z.B. Attributübergreifende Bedingungen formuliert werden:

```
CREATE TABLE Raum (
    GebaeudeNr int,
    RaumNr int,
    Groesse int,
    Beschreibung nvarchar(100),
    PRIMARY KEY(GebaeudeNr, RaumNR),
    FOREIGN KEY (GebaeudeNr) REFERENCES Gebaeude(GebaeudeNr)
);
```

Unten wird ein zusammengesetzter Primärschlüssel deklariert. Da auch mehrere Attribute zu einem Foreign Key zusammengesetzt werden können, wird die REFERENCES-Syntax zur FOREIGN KEY-Syntax erweitert.

5.B.I UMGANG MIT STRUKTUREN

Das DBMS verhindert das Löschen von Datensätzen, auf die durch eine Fremdschlüssel-Beziehung referenziert wird. Dieses Verhalten kann man in der FOREIGN KEY-Deklaration anpassen:

- **ON DELETE** legt fest, wie auf Löschoperationen reagiert wird
 - **CASCADE** löscht den Datensatz in der referenzierenden Tabelle
 - **SET NULL / SET DEFAULT** setzt das Attribut in der referenzierenden Tabelle null bzw. auf einen Default-Wert
 - **NO ACTION** erlaubt das Brechen der Fremdschlüsselbeziehung
- **ON UPDATE** legt fest, wie auf Updates reagiert wird.
 - **CASCADE** ändert den Wert des Attributs in der referenzierenden Tabelle
 - **SET NULL / SET DEFAULT** setzt das Attribut in der referenzierenden Tabelle null bzw. auf einen Default-Wert
 - **NO ACTION** erlaubt das Brechen der Fremdschlüsselbeziehung

5.B.I UMGANG MIT STRUKTUREN

Beispiel:

```
CREATE TABLE Anschrift (
...
  FOREIGN KEY (PLZ) REFERENCES Ort(PLZ)
  ON DELETE CASCADE
  ON UPDATE SET NULL
);
```

Achtung: Das Lösch- oder Update-Verhalten kann unabhängig voneinander gesetzt werden.

5.B.I UMGANG MIT STRUKTUREN

Relationen werden gelöscht mit dem DROP TABLE Befehl.

Beispiel:

DROP TABLE Raum;

Das Löschen der Tabelle beinhaltet gleichzeitig das Löschen aller Inhalte und kann nicht rückgängig gemacht werden. Wird die referentielle Integrität gefährdet, kann eine Tabelle nicht gelöscht werden.

Achtung: Die meisten RDBMS stellen keine Fragen vor der Befehlsausführung! DROP TABLE sollte daher mit Vorsicht angewendet werden!

5.B.I UMGANG MIT STRUKTUREN

Nachträgliche Änderungen an Tabellen können mit ALTER TABLE durchgeführt werden.

- Änderungen von Attributten
 - **ALTER TABLE Raum ADD COLUMN Verwendung nvarchar(20) NOT NULL**
 - **ALTER TABLE Raum DROP COLUMN Verwendung**
- Änderungen von Integritätsbedingungen
 - **ALTER TABLE Raum ADD PRIMARY KEY(RaumNr)**

Der Befehl ALTER TABLE bietet eine Vielzahl von Möglichkeiten, die – je nach verwendetem RDBMS – mitunter stark variieren.

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Datensätze werden in eine Tabelle mit dem Befehl **INSERT INTO** eingefügt.

Generelle Syntax

INSERT INTO <tabellenname> [(attributliste)] VALUES (werteliste)

- Die optionale Attributliste benennt jedes mit einem Wert aus der Werteliste zu belegenden Attribut.
- Reihenfolge von Attribut- und Werteliste müssen übereinstimmen
- Nicht angegebene Attribute werden mit null oder Default-Werten belegt
- Wird keine Attributliste angegeben, so ist das gesamte Tupel der Relation gemeint.

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Beispiel für eine Relation Person(ID,Vorname,Nachname)

```
INSERT INTO Person (ID, Nachname)  
VALUES  
(1,"Meier")
```

```
INSERT INTO Person  
VALUES  
(1,"Hans","Meier")
```

Durch die Angabe der Attribute ID, Nachname in der Attributliste wird der Vorname mit einem null-Wert belegt. Wird keine Attributliste angegeben, müssen alle Attribute der Relation (in der korrekten Reihenfolge) in der Werteliste vorkommen.

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Mit einem einzigen SQL-Befehl können auch mehrere Tupel in eine Tabelle geschrieben werden:

```
INSERT INTO Person  
VALUES  
(1,"Hans","Meier"),  
(2,"Freddy","Kronzucker"),  
(3,"Grete","Schneider")
```

Auf diese Weise können ggf. Performance-Vorteile bei Insert-Operationen genutzt werden. Achtung: Multiple Inserts werden nicht von allen RDBMS unterstützt.

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Datensätze können mittels des UPDATE-Befehls verändert werden.

Allgemeine Syntax

```
UPDATE <tabellenname>
SET <attribut> = <wert>
WHERE <selektionsbedingungen>;
```

- Ein Update wird immer nur auf genau einer Tabelle ausgeführt
- Dabei wird das benannte Attribut mit einem neuen Wert belegt
- Mittels einer WHERE-Bedingung kann die relevante Tupelmenge bestimmt werden. Die Selektionsbedingungen können logisch verknüpft werden (AND / OR)
- Wird keine WHERE-Bedingung angegeben, so bezieht sich das Update auf alle Tupel.

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Beispiel: Update-Operation mit einem betroffenen Tupel. Der Nachname eines bestimmten Mitarbeiters soll geändert werden.

```
UPDATE Person  
SET Name = "Mustermann,,  
WHERE ID = 3;
```

Beispiel: Update-Operation mit mehreren betroffenen Tupeln. Alle Mitarbeiter aus Mannheim mit einem Gehalt unter 20.000EUR sollen 10% mehr Gehalt bekommen

```
UPDATE Mitarbeiter  
SET Gehalt = Gehalt * 1.1  
WHERE Gehalt < 20000  
AND Ort = "Mannheim";
```

5.B.II EINGEBEN UND ÄNDERN VON DATEN

Datensätze können mittels des DELETE-Befehls gelöscht werden.

Allgemeine Syntax

```
DELETE FROM <tabellenname>  
WHERE <selektionsbedingungen>;
```

- Eine Delete-Operation wird immer nur auf genau einer Relation ausgeführt
- Alle Tupel, die der Selektionsbedingung entsprechen, werden gelöscht
- Wird keine WHERE-Bedingung angegeben, so bezieht sich die Operation auf *alle* Tupel

Erinnerung: Die meisten RDBMS fragen nicht nach! Eine Delete-Operation wird sofort ausgeführt.

Beispiel: Alle Kunden aus Mannheim sollen gelöscht werden:

```
DELETE FROM Kunden  
WHERE Ort = "Mannheim";
```

5.B.III AUSLESEN VON DATEN

Datensätze können mit dem SELECT-Statement gelesen werden

Allgemeine Syntax

```
SELECT [DISTINCT] <Attribute>  
FROM <Tabellen>  
WHERE <Bedingungen>;
```

- Das SELECT bezeichnet die die *Projektion* der relationalen Algebra - *nicht* die Selektion!
- Es werden diejenigen Attribute angegeben, die die Projektion enthalten soll (* erlaubt).
- Die Attribute können aus einer oder mehreren Tabellen stammen – siehe [5.B.IV JOINS](#)
- Durch WHERE können *Selektionsbedingungen* i.S.d. relationalen Algebra gesetzt werden.
- Duplikate werden durch optionales DISTINCT eliminiert.

5.B.III AUSLESEN VON DATEN

Beispiele

1) **SELECT * FROM Person;**

2) **SELECT Vorname FROM Person WHERE Name = "Meier";**

3) **SELECT DISTINCT Vorname FROM PERSON WHERE Name = "Meier";**

- Bsp. 1 gibt jedes Attribut aller Tupel der Relation Person aus
- Bsp. 2 gibt nur den Vornamen derjenigen Tupel aus, in denen das Attribut Name den Wert Meier besitzt. Sollten mehrere identische Vornamen zum Namen Meier existieren, so wird der Vorname ggf. mehrfach ausgegeben
- Bsp. 3 eliminiert doppelte Vornamen durch DISTINCT

5.B.III AUSLESEN VON DATEN

Relationen sind Mengen von Tupeln, d.h. sie besitzen keine innere Ordnungsstruktur. Im Ergebnis kann eine bestimmte Ordnung durch ORDER BY hergestellt werden

```
SELECT *  
FROM Person  
WHERE PLZ = 68165  
ORDER BY Vorname ;
```

Es können auch mehrere Attribute im ORDER BY vorkommen. So kann eine Ordnungshierarchie hergestellt werden (z.B. ORDER BY Nachname, Vorname). Das Herstellen einer Ordnungsstruktur ist nicht umsonst, der Aufwand bei großen Ergebnismengen sollte nicht unterschätzt werden!

5.B.III AUSLESEN VON DATEN

Umbenennung von Attributnamen mit AS

Gegeben ist die Relation Produkt(ProdId, Name, Nettopreis)

SELECT

ProdId,

Name **AS** ProduktName

FROM Produkt;

Ergebnis:

Produkt	
<u>ProdId</u>	ProduktName
1	Zahnpasta
2	Kartoffeln

5.B.III AUSLESEN VON DATEN

Berechnungen im SELECT-Statement sind ebenfalls erlaubt.

Gegeben ist die Relation Produkt(ProdId, Name, Nettopreis)

SELECT

```
ProdId, Name, Nettopreis,  
Nettopreis*1.19 AS Bruttopreis  
FROM Produkt;
```

Ergebnis:

Produkt			
<u>ProdId</u>	Name	Nettopreis	Bruttopreis
1	Zahnpasta	1.00	1.19
2	Kartoffeln	1.50	1.785

5.B.IV JOINS

Abfragen auf mehreren Relationen können durch Verbundoperationen abgebildet werden

- Wiederholung: Ein Verbund entspricht einer Selektion aus dem kartesischen Produkt zweier Relationen.
- Dies wird bereits in der Grundstruktur eines SQL-Statements wiedergespiegelt.

Durch eine Verbundoperation können Attribute aus unterschiedlichen Relationen abgefragt werden, wodurch Fremdschlüsselbeziehungen abfragetechnisch berücksichtigt werden.

5.B.IV JOINS

Das kartesische Produkt ist ein Verbund ohne Verbundbedingung

SELECT * FROM Kunde, Auftrag

Kunde		
<u>KNR</u>	Vorname	Nachname
1	Elsa	Musterfrau
2	Max	Mustermann

Auftrag		
<u>ANR</u>	KNR	Datum
1001	1	02.01.2013
1002	2	04.05.2013

R					
KNR	Vorname	Nachname	ANR	KNR	Datum
1	Elsa	Musterfrau	1001	1	02.01.2013
1	Elsa	Musterfrau	1002	2	04.05.2013
2

5.B.IV JOINS

Die Vergleichsbedingung für einen Verbund kann wie folgt angegeben werden:

SELECT * FROM Kunde, Auftrag WHERE Kunde.KNR = Auftrag.KNR

Kunde		
<u>KNR</u>	Vorname	Nachname
1	Elsa	Musterfrau
2	Max	Mustermann

Auftrag		
<u>ANR</u>	KNR	Datum
1001	1	02.01.2013
1002	2	04.05.2013

R					
KNR	Vorname	Nachname	ANR	KNR	Datum
1	Elsa	Musterfrau	1001	1	02.01.2013
2	Max	Mustermann	1002	2	04.05.2013
...

5.B.IV JOINS

Die Verbundoperation kann auf unterschiedliche Arten ausgedrückt werden:

```
SELECT *  
FROM Kunde, Auftrag  
WHERE Kunde.KNR = Auftrag.KNR
```

```
SELECT *  
FROM KUNDE INNER JOIN Auftrag  
ON Kunde.KNR = Auftrag.KNR
```

Beide Statements liefern das gleiche Resultat. Die zweite Variante erzwingt eine genauere (und z.T. auch bedarfsgerechtere) Formulierung.

5.B.IV JOINS

Verschiedene Typen des Verbundes

- Einen Verbund mit einer Gleichheitsbedingung (z.B. Auftrag.KNR = Kunde.KNR), der auf jeder Seite des Operators einen Verbundpartner erwartet, nennt man Inner Join.
- Es sind auch Ungleichheits-Operatoren zulässig ($<$, \neq , $>$) – dies kommt jedoch eher selten vor
- Werden mehr als zwei Relationen miteinander verbunden, so spricht man auch von einem Multijoin (bzw. einem Multiple Join)
- Ein Verbund mit sich selbst nennt man Self Join.
 - Problematisch: Wie werden die einzelnen Attribute voneinander unterschieden?
 - Problem wird über sog. *Tupelvariablen* gelöst.

5.B.IV JOINS

Gegeben ist die folgende Relation:

MitarbVorges		
<u>MNR</u>	Name	Vorgesetzter
1	Meier	
2	Müller	1
3	Schulze	1

Jedem Mitarbeiter wird – falls vorhanden – ein Vorgesetzter zugeordnet. Aufgabe:
Liste aller Mitarbeiter mit den entsprechenden Vorgesetzten

5.B.IV JOINS

MitarbVorges		
MNR	Name	Vorgesetzter
1	Meier	
2	Müller	1
3	Schulze	1

So nicht:

SELECT

MitarbVorges.MNR, MitarbVorges.Name, MitarbVorges.Vorgesetzter

FROM

MitarbVorges

WHERE

MitarbVorges.Vorgesetzter = MitarbVorges.MNR;

In diesem Fall ist das Ergebnis durch die WHERE-Bedingung in der WHERE-Klausel eine leere Menge von Tupeln, da ein Mitarbeiter nicht sein eigener Vorgesetzter sein kann. *Tupelvariablen* lösen das Problem, in dem verschiedene Instanzen der Relation unterschieden werden.

5.B.IV JOINS

MitarbVorges		
MNR	Name	Vorgesetzter
1	Meier	
2	Müller	1
3	Schulze	1

Korrekter Self Join:

```
SELECT  
M1.MNR, M1.Name, M1.Vorgesetzter,  
M2.MNR, M2.Name, M2.Vorgesetzter,  
FROM  
MitarbVorges M1,  
MitarbVorges M2  
WHERE  
M1.Vorgesetzter = M2.MNR;
```

Zur besseren Unterscheidung sollten in diesem Beispiel noch die Attribute durch den Operator AS in aussagekräftige Attributnamen umgewandelt werden.

5.B.IV JOINS

Für die aus der relationalen Algebra bekannten Mengenoperationen *Schnittmenge*, *Vereinigungsmenge* und *Differenzmenge* existieren ebenfalls Operatoren:

UNION: Bilden der Vereinigungsmenge

EXCEPT: Bilden der Differenzmenge

INTERSECT: Bilden der Schnittmenge

Die Operatoren werden zwischen zwei SELECT-Statements eingebettet. Alle Mengenoperationen setzen die Schemagleichheit der beteiligten Relationen voraus, d.h. die Anzahl der Attribute sowie deren Wertebereiche müssen übereinstimmen.

Achtung: Die Mengenoperatoren sind in RDBMS nicht einheitlich implementiert.

Lediglich der Operator UNION ist übergreifend vorhanden.

5.B.IV JOINS

Beispiel für die Vereinigung:

```
SELECT * FROM S  
UNION  
SELECT * FROM T ;
```

In diesem Beispiel wird die Vereinigungsmenge der beiden Relationen *S* und *T* gebildet. Beide Relationen müssen dasselbe Schema besitzen.

An Stelle von UNION können auch INTERSECT oder EXCEPT treten.

5.B.V NESTED SELECTS

SQL-Anfragen können geschachtelt werden, um komplexe Fragestellungen abzubilden.

Gegeben sind die folgenden Relationen:

Lieferung		
LNR	KNR	Betrag
65	1001	2500
66	1002	900
67	1003	1100

Kunden		
KNR	Vorname	Nachname
1001	Max	Mustermann
1002	Erika	Musterfrau
1003	Hans	Schulze

Aufgabe: Generierung einer Liste mit allen Kunden, die schon einmal eine Lieferung mit einem Einzelwert von mehr als 1000 Euro erhalten haben.

5.B.V NESTED SELECTS

Lösung:

```
SELECT * FROM Kunde  
WHERE  
KNR IN (SELECT DISTINCT KNR FROM Lieferung WHERE Betrag > 1000);
```

Ergebnis:

Ergebnis		
<u>KNR</u>	Vorname	Nachname
1001	Max	Mustermann
1003	Hans	Schulze

Das Inner Select erzeugt die Liste mit Kundennummern, die für das äußere Select von Relevanz sind.

5.B.VI GRUPPIERUNG UND AGGREGATION

Häufig sollen bestimmte Attributwerte einer Relation anhand einer Aggregatfunktion zusammengefasst werden.

- **GROUP BY** ermöglicht das Zusammenfassen gleicher Attributwerte zu Gruppen anhand eines Gruppierungsattributs
 - Für jeden unterschiedlichen Wert x des Gruppierungsattributs wird eine Gruppe gebildet.
 - Ausabeattribute müssen entweder Gruppierungsattribute oder Aggregate sein
- Eine Aggregatfunktion berechnet einen Wert anhand einer Menge von Eingangswerten
 - **SUM()** Berechnet eine Summe einer Menge von Attributwerten
 - **AVG()** Berechnet den Durchschnitt einer Menge von Attributwerten
 - **MIN()** Berechnet das Minimum einer Menge von Attributwerten
 - **MAX()** Berechnet das Maximum einer Menge von Attributwerten
 - **COUNT()** Berechnet die Anzahl der Elemente in einer Menge von Attributwerten

5.B.VI GRUPPIERUNG UND AGGREGATION

Beispiel: Wie viele Lieferungen hat ein bestimmter Kunde bekommen und welchen Wert haben diese Lieferungen?

```
SELECT  
KNR,  
count(*) AS AnzahlLief,  
sum(Betrag) AS Gesamtwert  
FROM  
Lieferung  
GROUP BY KNR
```

Ergebnis		
KNR	AnzahlLief	Gesamtwert
1001	12	13245
1002	7	10245
1003	1	98

5.B.VI GRUPPIERUNG UND AGGREGATION

Bedingungen auf Aggregaten können nicht in eine WHERE-Klausel aufgenommen werden, da diese vor der Aggregatbildung ausgewertet wird.

Falsch:

```
SELECT  
KNR, count(*) AS AnzahlLief,  
sum(Betrag) AS Gesamtwert  
FROM  
Lieferung  
WHERE Gesamtwert > 10000  
GROUP BY KNR ;
```

Richtig:

```
SELECT  
KNR, count(*) AS AnzahlLief,  
sum(Betrag) AS Gesamtwert  
FROM  
Lieferung  
GROUP BY KNR  
HAVING Gesamtwert > 10000 ;
```

Der HAVING-Operator agiert wie eine WHERE-Bedingung, er wird jedoch erst nach vollendeter Aggregation ausgewertet.

STATEMENTS: FRAGEN

1. Welche Arten von Statement gibt es?
2. Mit welchem Statement kann eine Tabelle erschaffen werden? Mit welchem Statement kann sie gelöscht werden?
3. Wie kann eine bestehende Datenstruktur angepasst werden? Wie können bestehende Datensätze angepasst werden?
4. Wie kann das SELECT-Statement erweitert werden?
5. Welche Möglichkeiten gibt es, über mehrere Tabellen hinweg Datensätze zu kombinieren?



5.C ERWEITERTE SQL STATEMENTS

Abfrage von Primary Keys über alle Tabellen hinweg.

```
SELECT * FROM Sys.Objects WHERE Type='PK'
```

5.C ERWEITERTE SQL STATEMENTS

Abfrage von Foreign Keys über alle Tabellen hinweg.

```
SELECT * FROM Sys.Objects WHERE Type='f'
```

5.C ERWEITERTE SQL STATEMENTS

Vertauschen von Werten in Attributen

UPDATE Customers SET Zip=Phone, Phone=Zip

5.C ERWEITERTE SQL STATEMENTS

Kopieren von Werten zwischen Tabellen

INSERT INTO Yearly_Orders

SELECT * FROM Orders

WHERE Date<=1/1/2018

5.C ERWEITERTE SQL STATEMENTS

TOP-Werte in SELECT

```
SELECT TOP 25 FROM Customers WHERE Customer_ID>3;
```

5.C ERWEITERTE SQL STATEMENTS

Wildcards in SELECT

Entspricht einer unscharfen Textsuche mit dem allgemeinen Ersatzkürzel "%"

SELECT * From Customers WHERE Name LIKE 'Herb%'

5.C ERWEITERTE SQL STATEMENTS

Verschachtelte SELECTS mit ALL, ANY, EXISTS, IN

SELECT Item FROM Orders

WHERE id = ALL

(SELECT ID FROM Orders

WHERE quantity > 50)

SELECT Name FROM Customers WHERE EXISTS

(SELECT Item FROM Orders

WHERE Customers.ID = Orders.ID AND Price < 50)

5.D VIEWS UND INDICES

Ein View ist eine *virtuelle Tabelle* einer Datenbank, mit der einem Benutzer eine Sicht auf die Daten ermöglicht wird (siehe auch logische Datenunabhängigkeit).

Beispiel:

```
CREATE VIEW AnzahlGesamtwert AS  
SELECT  
    KNR, count(*) AS AnzahlLief,  
    sum(Betrag) AS Gesamtwert  
FROM  
    Lieferung  
GROUP BY KNR  
HAVING Gesamtwert > 10000 ;
```

Der View kann nun wie eine Relation (z.B. `SELECT * from AnzahlGesamtwert`) abgefragt werden. Die Werte werden jedes mal neu berechnet.

5.D VIEWS UND INDICES

Ein Index ist ein Verzeichnis, welches zu einem bestimmten Attributwert den genauen Speicherort angibt.

Indizes werden im Rahmen der Vorlesung *Datenbanktechnik* noch ausführlich behandelt. An dieser Stelle ist die Erstellung eines Index für uns ein Weg der Anfragebeschleunigung und Teil der physischen Konzeption.

Beispiel:

CREATE INDEX idxPLZ ON Anschriften(PLZ)

Mit diesem Befehl wird ein Index (Werteverzeichnis) auf dem Attribut PLZ der Relation Anschriften erzeugt, um entsprechende Datensätze schneller auffinden zu können. **Mehr zu Index-Strukturen in der Vorlesung *Datenbanktechnik*!**

5.E ÜBUNGEN

Erstellen Sie eine Datenbank für das Streaming-Portal „Nerdflix“. In Nerdflix sollen Vorlesungen von Dozenten verschiedenster Fachrichtungen gespeichert werden, die ein Thema, eine Zusammenfassung und Stichworte haben. Ein Dozent gibt natürlich verschiedene Vorlesungen von ggf. verschiedenen Fachbereichen. Ein Dozent kann zu einer Vorlesung weiterführende Literatur von Autoren empfehlen. Ein Dozent kann natürlich auch selbst Autor eines Buches sein. Bücher haben einen Titel, ein Erscheinungsdatum und Autoren. Ein User muss nicht an einer Universität eingeschrieben sein, wenn er es aber ist, senkt sich der Preis je nach Status der Universität. Ein User kann an so vielen Kursen verschiedener Fachrichtungen teilnehmen, wie er möchte. Zu einem User werden Namen, Universität und Emailadresse hinterlegt.



6. EXKURSIONEN

6.A SAP/ABAP

Ablauf

- I. Basics SAP
- II. Live-Demo SAP
- III. Unterschied Sap GUI/NWBC
- IV. ABAP

6.A.I BASICS

SAP steht für Systeme, Anwendungen, Produkte in der Datenverarbeitung. SAP wurde 1972 in Walldorf gegründet und verfügt heute über Niederlassungen auf der ganzen Welt.

Ursprünglich bekannt als führendes Unternehmen für Enterprise Resource Planning (ERP)-Software, hat sich SAP zu einem Marktführer für End-to-End-Unternehmenssoftware, Datenbanken, Analysen, intelligente Technologien und Experience Management entwickelt. Als Top-Cloud-Company mit 200 Millionen Anwendern weltweit unterstützt SAP Unternehmen jeder Größe und Branche dabei, profitabel zu arbeiten, sich kontinuierlich anzupassen und ihre Ziele zu erreichen.

6.A.II LIVE-DEMO SAP

6.A.III UNTERSCHIED SAP GUI/NWBC

Der SAP NetWeaver Business Client (NWBC, Business Client) ist ein Rich Client, welcher eine optimale durchgehende Performance, eine vertraute Desktop-Integration und eine deutlich verbesserte Benutzerfreundlichkeit bietet. Er erlaubt den Gebrauch von Portal Services, Applikations-Content und Aufgaben direkt aus dem Backend. Der NWBC überbrückt die Kluft zwischen den heutigen Thick Clients (wie z.B. SAP GUI) und dem Browser-Zugriff, der verwendet wird, um auf das SAP NetWeaver Portal zuzugreifen.

6.A.IV ABAP

ABAP ist eine proprietäre Programmiersprache der Softwarefirma SAP, die für die Programmierung kommerzieller Anwendungen im SAP-Umfeld entwickelt wurde und in ihrer Grundstruktur der Programmiersprache COBOL entfernt ähnelt.

Ursprünglich stand die Abkürzung für „**Allgemeiner Berichtsaufbereitungsprozessor**“, da mit dieser Sprache nur Auswertungen (Reports) programmiert wurden, aber keine Datenbankveränderungen vorgenommen werden konnten. Im Zuge der Weiterentwicklungen der Sprache steht die Abkürzung nun für „**Advanced Business Application Programming**“. Der Sprachumfang ist nicht fest definiert und wurde in der Vergangenheit immer wieder erweitert, z. B. um die objektorientierten Sprachbefehle von ABAP Objects.

6.A.IV ABAP

- a. Um ein „Programm“ zu schreiben, wird der Befehl **REPORT** genutzt
- b. ABAP-Reports werden mit der Transaktion **SE38** geschrieben
- c. Jede Zeile in ABAP wird mit einem Punkt beendet
- d. Für Variablen wird der Befehl **DATA** genutzt
 - a. Die Domäne der Variable wird mit dem Befehl **TYPE** eingegeben
 - b. Typen sind z.B. **i** für Ganzzahlen, **c** für Texte, **d** für Datum
 - c. Lokale Variablen sollten mit dem Präfix „**I_**“ begonnen werden
 - d. Standardwerte werden mit dem Zusatz **DEFAULT** angegeben
 - e. Beispiel: **DATA: I_result TYPE i DEFAULT 5.**
- e. Eine Usereingabe kann mit dem Befehl **PARAMETERS** realisiert werden

6.A.IV ABAP

- a. Eine Variable kann mit dem Befehl **WRITE** ausgegeben werden
 - a. **WRITE** '[STRING]': für eine Zeile
 - b. **WRITE** '[STRING]'.
WRITE '[STRING]'.
WRITE '[STRING]': für mehrere Zeilen
 - c. **WRITE:** '[STRING]',
 '[STRING]',
 '[STRING]': für mehrere Zeilen
 - d. **WRITE:** '[STRING]', '[STRING]', '[STRING]': für mehrere Zeilen
- b. Für eine Leerzeile kann der Befehl **SKIP** genutzt werden
- c. Für eine Linie kann der Befehl **ULINE** genutzt werden

6.A.IV ABAP

- a. Bedingungen können über den Befehl **IF** abgebildet werden
 - a. Es gibt die Verzweigungsoperationen **ELSE** und **ELSEIF**
 - b. **IF** muss mit **ENDIF** beendet werden
- b. Größere Verzweigungen können mit dem Befehl **CASE** abgebildet werden
 - a. Bei **CASE** muss eine Variable angegeben werden, deren Wert die Verzweigung definiert
 - b. Jede Verzweigung wird mit dem Stichwort **WHEN** und der Ausprägung angegeben
 - c. Nicht angegebene Werte können unter **WHEN OTHERS** abgefangen werden
 - d. **CASE** muss mit **ENDCASE** beendet werden

6.A.IV ABAP

Operatoren:

a. Arithmetische Operatoren:

a. +, -, *, /, MOD

b. Vergleichsoperatoren:

a. =, <>, <, >, <=, >=, BETWEEN, IS INITIAL, IS NOT INITIAL

c. Stringoperatoren:

- a. CO (Contains only)
- b. CN (Contains not only)
- c. CA (Contains any)
- d. NA (Not contains any)
- e. CS (Contains a string)
- f. NS (Not contains a string)
- g. CP (Contains a pattern)
- h. NP (Not contains a pattern)

6.A.IV ABAP

- a. Schleifen können über den Befehl `WHILE` abgebildet werden
 - a. `WHILE` benötigt einen boolschen Ausdruck
 - b. `WHILE` muss mit `ENDWHILE` abgeschlossen werden
 - c. Der Code zwischen `WHILE` und `ENDWHILE` wird so lange ausgeführt, bis der boolsche Ausdruck `FALSE` ergibt
- b. Alternativ kann der Befehl `DO` genutzt werden
 - a. `DO` benötigt eine Anzahl, wie oft die Schleife durchlaufen werden soll
 - b. `DO` muss mit `ENDDO` abgeschlossen werden
 - c. Der Code zwischen `DO` und `ENDDO` wird so lange ausgeführt wie nach `DO` angegeben
- c. In einer Schleife können folgende Kontrollbefehle genutzt werden:
 - a. `CONTINUE` überspringt den Rest des Codes und springt wieder an den Anfang der Schleife
 - b. `CHECK` prüft auf einen boolschen Ausdruck. Ist die Prüfung `FALSE`, funktioniert es wie `CONTINUE`, bei `TRUE` läuft der Code weiter
 - c. `EXIT` bricht den Loop komplett ab

6.A.IV ABAP

Für fixierte Textausgaben kann der in SAP integrierte **Textpool** genutzt werden. In diesen fest formulierten Texten kann auch eine Übersetzung für verschiedene Sprachen angegeben werden, die userspezifisch gewählt werden kann.

Die Texte können mit dem Präfix „text-“ und der ID des Textinhalts aufgerufen werden:

WRITE text-006. würde hier „Defined Differently in report“ ergeben.

ABAP Text Elements: Change Text Symbols Language English			
Program		Z_TEXTSYMBOL_TEST active / revised	
Text symbols		Selection texts	List Headings
	S... Text	006	dLen mLen
		Defined Differently in report	5 50
	011	Unused Text	11 11
			0
			0
			0
			n

6.A.IV ABAP: BEISPIEL

```
REPORT z_calculator.  
PARAMETERS: pa_first TYPE i, pa_operator(1), pa_second TYPE i.  
DATA: l_result TYPE i.  
  
CASE pa_operator.  
  WHEN '+'.  
    l_result = pa_first + pa_second.  
  WHEN '-'.  
    l_result = pa_first - pa_second.  
  WHEN '*'.  
    l_result = pa_first * pa_second.  
  WHEN '/'.  
    IF pa_second = 0.  
      WRITE text-001.  
      EXIT.  
    ELSE  
      l_result = pa_first / pa_second.  
    ENDIF.  
  WHEN OTHERS.  
    WRITE text-002.  
    EXIT.  
ENDCASE  
WRITE: l_result
```

6.A.IV ABAP

- a. Es können interne Tabellen erstellt werden, um mit großen Datenmengen effizienter arbeiten zu können
- b. Hierfür wird der Befehl TYPES: genutzt
 - a. TYPES wird mit BEGIN OF [Tabellenname] begonnen und mit END OF [Tabellenname] beendet
 - b. Zwischen BEGIN und END können jetzt Variablen definiert werden, die als Attribute der internen Tabelle genutzt werden
 - c. Variablen haben Domänen
- c. Beispiel:
TYPES: BEGIN OF Kunden,
KndnID TYPE I,
KndnName (20) TYPE C,
END OF Kunden.

6.A.IV ABAP

- a. Alle Objekte (also Variablen etc.) gehören zu einer PACKAGE
- b. Alle lokalen Variablen gehören zu der PACKAGE \$TMP
- c. PACKAGES und REPORTS können zwischen Systemlinien transportiert werden, dafür müssen sie aktiviert werden
- d. So können Entwicklungen im Testsystem explizit gesteuert werden und unterschiedliche Stände gehalten werden.

6.B FRONT-END PROGRAMMIERUNG

- a. UI-Konzeption
 - a. Userstories
 - b. Clickdummies
- b. Segregation of duties
- c. Forced Data Integrity
- d. Usability