

# Zusammenfassung TEIL 1

## Datenmanagement EDD

### Contents

1	Einführung.....	3
1.1	Mutationsanomalie .....	3
1.2	Flache Tabellen und Anomalien .....	3
1.2.1	Mutationsanomalie .....	3
1.2.2	Einfügeanomalie.....	3
1.2.3	Löschanomalie.....	3
1.3	Datenbankarchitektur .....	4
1.3.1	Externe Ebene .....	4
1.3.2	Konzeptuelle Ebene.....	4
1.3.3	Interne Ebene .....	5
1.3.4	Client und Server .....	5
1.4	Modellierung .....	6
1.4.1	Informationsmodel.....	7
2	Das Entitäten-Beziehungsmodel ERM.....	7
2.1	Entitäten und Entitätstypen .....	7
2.2	Assoziationen, Assoziationstypen, Beziehungen, Beziehungstypen .....	7
2.3	Multiplizität .....	8
2.4	Attribute .....	9
2.5	Modellierungsregeln .....	9
3	UML Klassendiagramm .....	10
3.1	Erweiterungen zum ERD.....	11
3.2	Aggregation / Komposition .....	12
4	Das Relationenmodell .....	13
4.1	Relationen .....	13
4.2	Primärschlüssel und Tupelintegrität.....	14
4.3	Beziehungstypen .....	14
4.4	Fremdschlüssel und referenzielle Integrität.....	16
5	Vom ERD zum relationalen ERD .....	16
5.1	Attribute und Domänen (Wertebereiche).....	16

5.2	Attribut oder Entitätstyp .....	17
5.3	Arten von Entitätstypen .....	17
5.3.1	Attributiver Entitätstyp (tkey) .....	17
5.3.2	Fundamentaler Entitätstyp (tbl) .....	17
5.3.3	Assoziativer Entitätstyp (tass) .....	18
5.4	Rezept zum relationalen Entwurf .....	18
5.4.1	Elemente .....	18
5.4.2	Behandlung von Beziehungstypen .....	18
6	Normalisierung (Skript Teil 1: S. 46 – 57) .....	19
6.1	Erste Normalform (1 NF) .....	19
6.2	Zweite Normalform (2 NF) .....	20
6.3	Dritte Normalform (3 NF) .....	21
6.4	Funktionale Abhängigkeit .....	21
7	Datenbank in ACCESS (Skript Teil 1: S. 58 – 96) .....	22
7.1	Ein Relationales Datenbanksystem (RDBMS) .....	22
7.2	Datenbasis .....	22
7.3	Systemkatalog .....	22
7.4	Eine Datenbank .....	22
7.5	Eine Tabelle .....	22
7.6	Ein Attribut .....	22
7.7	Als weitere Objekte werden wir uns befassen mit .....	22
7.8	Datentypen .....	23
7.8.1	Alphanumerische Typen: .....	23
7.8.2	Numerische Typen: .....	23
7.9	Indizes .....	23
7.9.1	Vorteile: .....	24
7.9.2	Achtung: .....	24
7.10	Beziehungen .....	24
7.11	Abfragen (Queries, nur einer Tabelle) .....	25
7.12	Joins (Abfrage mehrere Tabellen) .....	32
7.13	Abfragen aus mehreren Tabellen .....	34

# 1 Einführung

## 1.1 Mutationsanomalie

Wozu dienen Datenbanken:

- Um Redundanzen zu vermeiden → Mehrfachspeicherung von identischen Sachverhalten wird vermieden oder falls notwendig verwaltet
- Um Inkonsistenzen zu vermeiden → Synchronisation von Daten bei zeitgleicher Benutzung und Bearbeitung
- Um Datenschutz zu gewährleisten → Zugriffsrechte, Verschlüsselung
- Um Datensicherheit zu gewährleisten → Verlust von Daten verunmöglichen oder unwahrscheinlich machen
- Um Datenabhängigkeit zu gewährleisten → Daten sind nicht abhängig von anderen Anwendungen

## 1.2 Flache Tabellen und Anomalien

Eine Kundenliste ist ein geeignetes Beispiel für eine flache Tabelle. Eine sogenannte Tabellenkalkulation. Sie bergen grosse Gefahren für die Datenbestände.

### 1.2.1 Mutationsanomalie

Die Änderung für eine Entität erfordert mehrere Nachtragungen

### 1.2.2 Einfügeanomalie

Das Einfügen eines Datensatzes zwingt uns zu

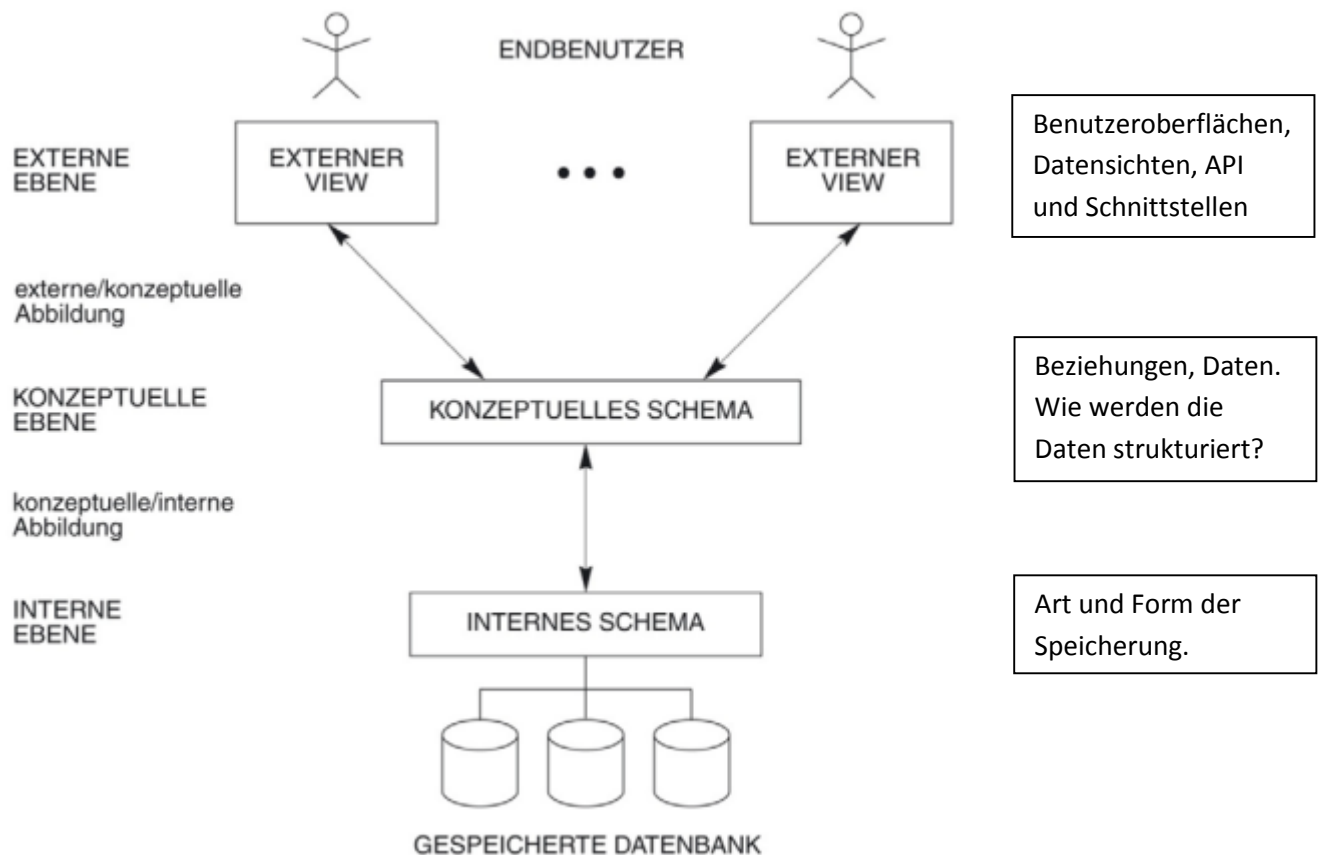
- Entweder viele NULL-Werte
- Viele redundante Werte mit zu erfassen

### 1.2.3 Löschanomalie

Die Löschung eines Datensatzes zwingt uns zu weiteren Löschungen oder es löscht unbeabsichtigt andere Informationen.

**Flache Tabellen dienen nicht für grosse Datenbestände.**

## 1.3 Datenbankarchitektur



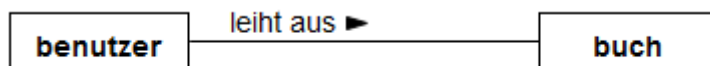
### 1.3.1 Externe Ebene

Die externe Ebene, die den Benutzern und Anwendungen individuelle Benutzersicht bereitstellt. Beispiele: Formulare, Masken-Layouts, Listen, Schnittstellen.

### 1.3.2 Konzeptuelle Ebene

Die konzeptionelle Ebene, in der beschrieben wird, welche Daten in der Datenbank gespeichert sind, sowie deren Beziehungen zueinander. Designziel ist hier eine vollständige und redundanzfreie Darstellung aller zu speichernden Informationen.

Als erstes entwerfen wir ein Konzept der Informationsstruktur aus sachlogischer Sicht.



Wenn die sachlogische Modellierung steht, wird dieses Modell in ein relationales Modell übersetzt.



Abbildung 4: Metadaten (Struktur der Daten)

pkBenutzer	name	vorname
1	Zwimpfer	Victor
2	Marfurt	Konrad
3	Fischer	Peter

fkBenutzer	fkBuch	rueckgabetermin
1	1	2013-12-01
1	2	2013-12-01
2	1	2013-10-01

pkBuch	titel	erscheinungsjahr	auflage
1	Datenbanksysteme	2007	1
2	SQL Server 2012	2013	1

### 1.3.3 Interne Ebene

Die interne Ebene (auch physische Ebene), ist die physische Sicht der Datenbank, die im Computer dargestellt wird. In ihr wird beschrieben, wie und wo die Daten in der Datenbank gespeichert werden. Designziel ist hier ein effizienter Zugriff auf die gespeicherten Informationen. Das wird meistens nur durch eine bewusst in Kauf genommene Redundanz erreicht (z.B. im Index werden die gleichen Daten gespeichert, die auch schon in der Tabelle gespeichert sind).

### 1.3.4 Client und Server

Client und Server sind zwei Prozesse. Es braucht beide, damit etwas Brauchbares zustande kommt. Client und Server können durchaus auf dem gleichen Computer laufen. In der Regel wird dies aber nicht gemacht.

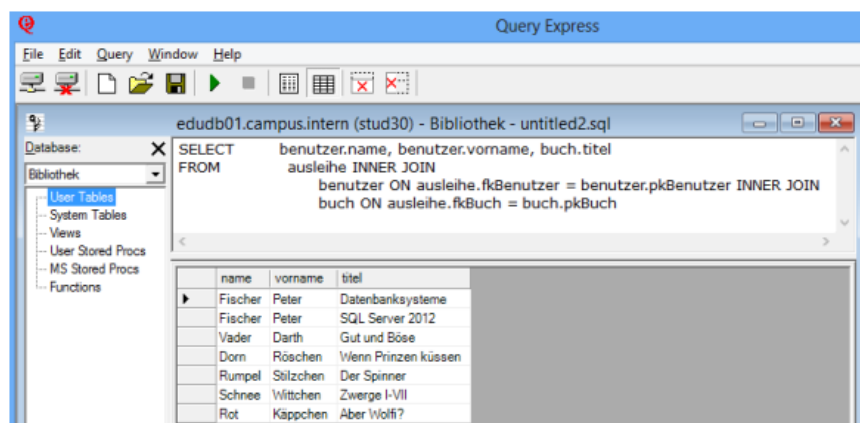
Client am Frontend: nimmt menschliche Befehle an, prüft sie, bereitet sie auf und sendet diese weiter. Wird bedient vom Menschen

Server am Backend: nimmt die befohlenen Datenmanipulationen und Berechnungen vor, sendet Quittungen und/oder Resultate zurück. Wird bedient vom Client unterstützt vom Betriebssystem.

zwei Beispiele:

#### Beispiel 1

- Server: MS SQL-Server 2008; Standort HSLU-T&A
- Client: Query Express (lokal installierte Applikation)
- Schnittstelle: SQL



## 1.4 Modellierung

Die Qualität von Modellen kann anhand drei semiotischen Aspekte beurteilt werden.

Syntax: Welche Konstruktionselemente dürfen gemäss welchen Verknüpfungsregeln miteinander in Beziehung gesetzt werden? (Qualität der Form)

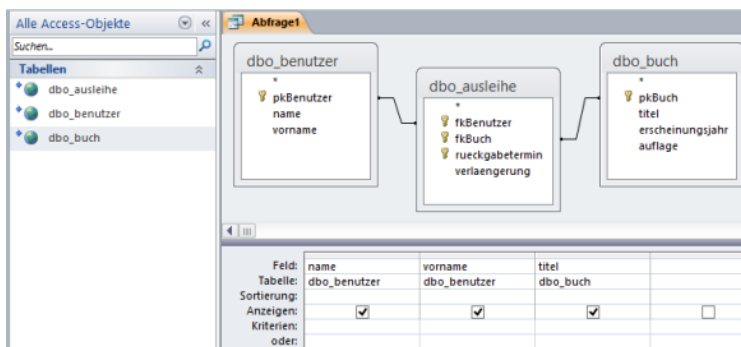
Semantik: Welche Bedeutung haben die einzelnen Konstruktionselemente? (Qualität der Bedeutung)

Pragmatik: Welche Wirkung wird beabsichtigt? (Qualität der Verwendung)

Bsp. Underground England

### Beispiel 2

- Server: MS SQL-Server 2012; eigener Laptop
- Client: MS Access als reines GUI mit Verknüpfungen zur Datenbasis auf dem Server
- Schnittstelle: Access-QBE (*Query By Example*)



name	vorname	titel
Fischer	Peter	Datenbanksysteme
Fischer	Peter	SQL Server 2012
Vader	Darth	Gut und Böse
Dorn	Röschen	Wenn Prinzen küssen
Rumpel	Stilzchen	Der Spinner
Schnee	Wittchen	Zwerge I-VII
Rot	Käppchen	Aber Wolfi?

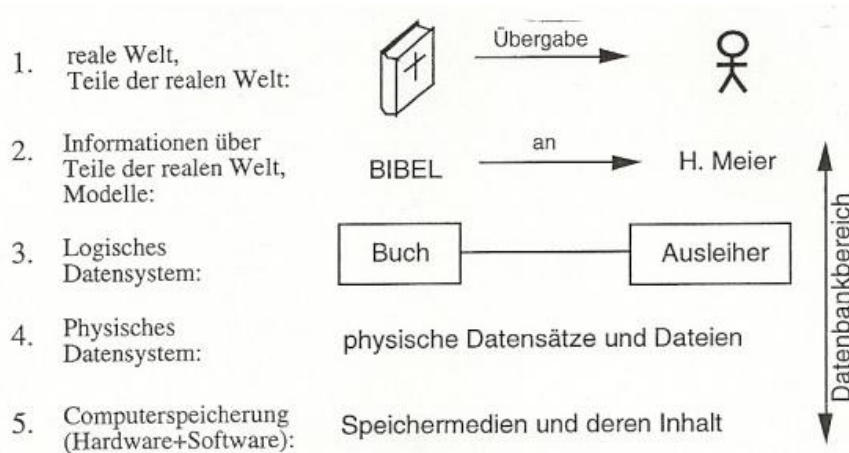


Syntax: O / ...

Semantik: Stationen, Linien & Zonen

Pragmatik: Wie komme ich von A nach B

### 1.4.1 Informationsmodell



## 2 Das Entitäten-Beziehungsmodell ERM

Um in eine Datenbank zu gelangen sind vier Schritte zu tun:

1. Analyse: Aufnahme der realen Gegebenheiten sowie der Anforderungen an die Datenbank.
2. Erstellung des konzeptionellen Schemas: Grafische Modellierung
3. Umsetzung in ein internes Schema: Die Datenbank im gewählten DBMS definieren
4. Abbildung der Geschäftslogik: Hier werden Programme geschrieben und ans DBMS angebunden, welche Daten anlegen, die gespeicherten Daten selektieren, manipulieren und löschen oder aufgrund der aktuellen Bestände Aktionen auslösen, also z.B. einen Lagerbestand überwachen, Bestellungen auslösen, Mahnungen versenden, für eine Patrouille genau zwei Mitarbeiter einteilen usw.

### 2.1 Entitäten und Entitätstypen

#### *Isolieren der Entitäten*

Wir befassen uns mit konkreten Objekten wie Personen, Bücher, Ausleihen, Buchhändler, Rechnungen etc. Nach diesen Entitäten (Objekte) wird sortiert. Der Schritt des Erkennens und Isolierens von Entitäten ist ein rein didaktischer. Es wird nichts modelliert.

#### *Festlegen der Entitätstypen*

Wir abstrahieren alle ermittelten Entitäten zu Entitätsklassen. Die Entitäten einer Klasse zeichnen sich alle dadurch aus, dass sie vergleichbare Eigenschaften mit individuellen Werten haben.

Beim Isolieren der Klasse haben wir Buch mit den Werten: „Datenbanken“, „Datenbankprogrammierung mit .NET 4.5“ usw.

Beim Festlegen der Entitätstypen heisst es einfach nur noch Buch.

### 2.2 Assoziationen, Assoziationstypen, Beziehungen, Beziehungstypen

#### *Isolieren der Assoziationen*

Zwischen den Entitäten unseres Systems laufen Informationsflüsse, es bestehen Informations-Beziehungen. Wir ermitteln und benennen Aktivitäten, die eine solche, konkrete Beziehung zweier Entitäten kennzeichnen.

- Autor schreibt Buch, Autorinnen und Autoren schreiben Bücher usw.
- Verlag veröffentlicht Bücher und Bücher werden von Autoren geschrieben...

Der Schritt des Erkennens und Isolierens von Assoziationen ist ein rein didaktischer, es wird noch nichts modelliert.

#### *Festlegen der Assoziationstypen und Beziehungstypen*

Die Assoziationen einer Klasse zeichnen sich alle dadurch aus, dass sie eine Geschäftsaktivität ausdrücken.

- Autor schreibt Buch
- Verlag veröffentlicht Buch

Die abstrakte Beziehung betrachtet den Informationsfluss jeweils nur in eine Richtung. **Diese werden als Assoziationstypen bezeichnet.** Assoziationstypen werden **nicht** modelliert sondern in auch in die andere Richtung ausgebaut.

Aus:

- Autor schreibt Buch

Wird:

- Autor schreibt Buch / Buch wird geschrieben von Autor

Aus:

- Verlag veröffentlicht Buch

Wird:

- Verlag veröffentlicht Buch / Buch wird veröffentlicht von Verlag

Man muss immer beide Entitätstypen betrachten.

Aus Entitätstyp A zu B → Jede Buchhandlung erhält eine Rechnung

Aus Entitätstyp B zu A → Jede Rechnung betrifft eine Buchhandlung

## 2.3 Multiplizität

Beziehungen werden durch Angaben der Multiplizität genauer spezifiziert. Mit der Multiplizität wird angegeben, **wie viele konkrete Beziehungen eine Entität mit einer anderen haben kann oder muss.** Sie muss das erlaubte Minimum (0 oder 1) sowie das erlaubte Maximum (1 oder m) ausdrücken.

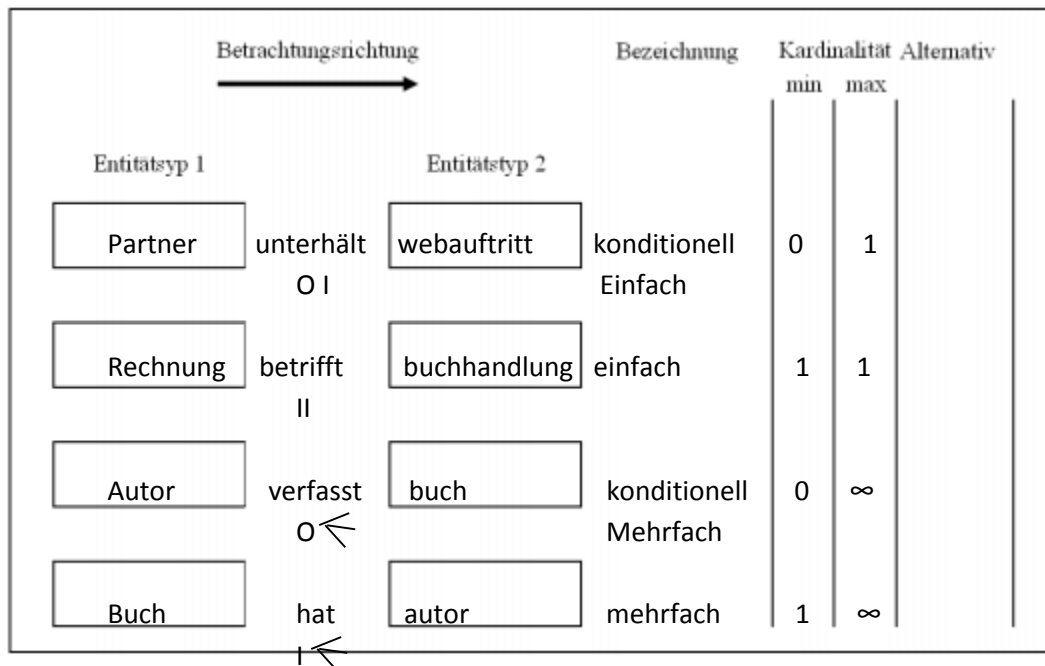
- Jeder Autor schreibt kein, ein oder mehrere Büche / Jedes Buch wird geschrieben von einem oder mehreren Autoren
- Jeder Verlag veröffentlicht null bis mehrere Bücher / Jedes Buch wird veröffentlicht von einem oder mehrere Verlagen

Die Zahleigenschaften von Beziehungen werden auch Kardinalität genannt.

**Bei uns muss immer die minimale und die maximale Kardinalität vermerkt werden.**

Bsp aus Script





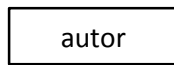
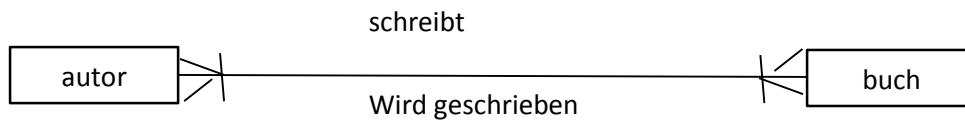
## 2.4 Attribute

Attribute zeigen die Eigenschaft einer Entität auf

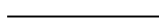
- Autor wird charakterisiert durch Familienname, Vorname, Geburtsdatum, Strasse, PLZ, Wohnort, Fon, Fax, Mobile, Fachgebiet u.v.a.m.
- Verlag wird charakterisiert durch Bezeichnung, Zusatzbezeichnung, Adresse (Strasse, PLZ, Ort), Kontaktperson u.v.a.m.

## 2.5 Modellierungsregeln

- Das Entitäten-Beziehungsmodell bildet den relevanten Realitätsausschnitt ab in ein Modellsystem mit *Entitätstypen*, *Beziehungstypen*, *Multiplizitäten* und *Attributen*.
- Entitätstypen erhalten *sprechende* Namen und stehen im Singular.
- Beschriften Sie mit Kleinbuchstaben, ohne Umlaute und in männlicher Form.
- Vermeiden Sie *Homonyme*, also die Namensgleichheit für mögliche, unterschiedliche Sachverhalte: *Lieferung* (An- oder Auslieferung?)
- Vermeiden Sie *Synonyme*, also die mögliche Bedeutungsähnlichkeit bei unterschiedlichen Namen: mit *Adressat* und *Empfänger* sind möglicherweise die gleichen Entitäten gemeint.
- Beziehungstypen erhalten in beide Richtungen je eine *sprechende* Bezeichnung. Diese drückt den sachlogischen Zusammenhang zwischen den Entitätstypen aus. Aus genanntem Grund sollte nie «hat» und «ist» verwendet werden.
- Die beiden Assoziationstypen geben mittels Angaben der Multiplizität Auskunft über Optionallität (minimal 0 oder 1) und mögliche Häufigkeit (maximal 1 oder mehrere) der konkreten Beziehungen an.
- Die Attributlisten der Entitätstypen werden in der sogenannten Strukturschreibweise «autor (name, vorname, ...)» festgehalten.



= Entitätstyp oder Entitätsmenge



= Beziehungstyp

$[0, 1] \text{ o } |$

= Minimal-Kardinalität

$[1, \infty] \text{ 1 } \leftarrow$

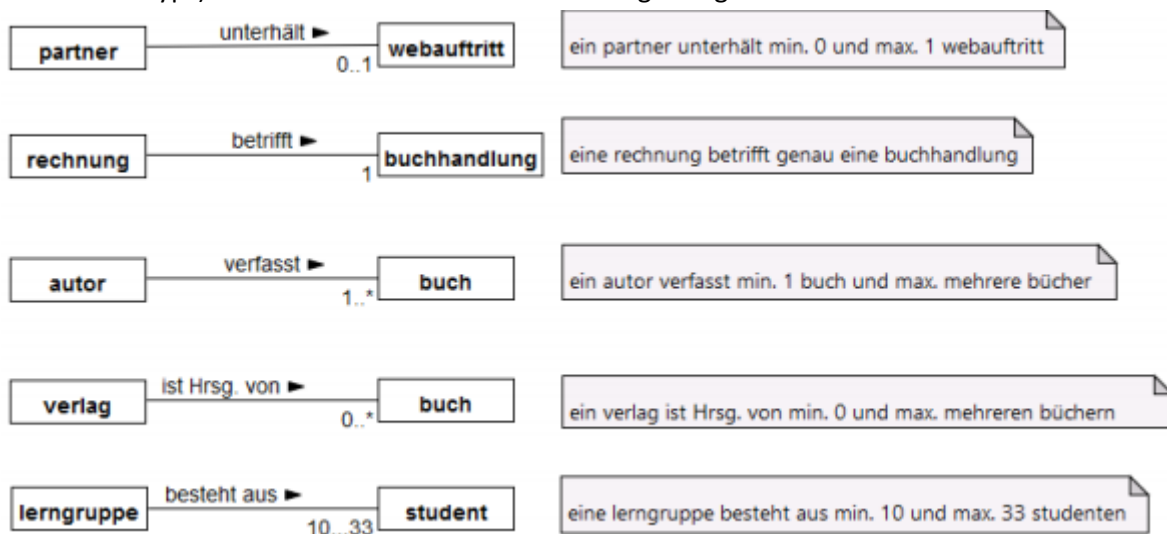
= Maximal-Kardinalität / Mehrfach Seite

Eigenschaften

= bsp. buch(titel, erscheinungsdatum...)  
Autor(name, vorname, gebJahr...)

### 3 UML Klassendiagramm

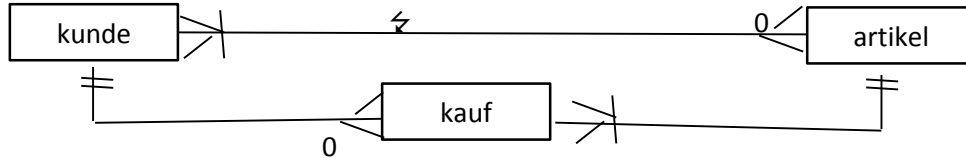
Im UML-Klassendiagramm wird der Entitätstyp Klasse und die Entität Objekt genannt. Im Gegensatz zum klassischen ERD kennt das Klassendiagramm neben den bereits bekannten Assoziationsstypen noch die Angabe von konkreten natürlichen Zahlen an der Unter-, bzw. Obergrenze (Präzisierung des Assoziationsstyps). Zudem ist es üblich die Leserichtung anzugeben.



Zusätzlich zur oben eingeführten IE-Notation für die grafische Darstellung von ERDs, besteht im UML-Klassendiagramm die Möglichkeit, Assoziationsklassen zu modellieren. Dabei wird die Assoziationsklasse mit einer gestrichelten Linie mit der Assoziation verbunden.

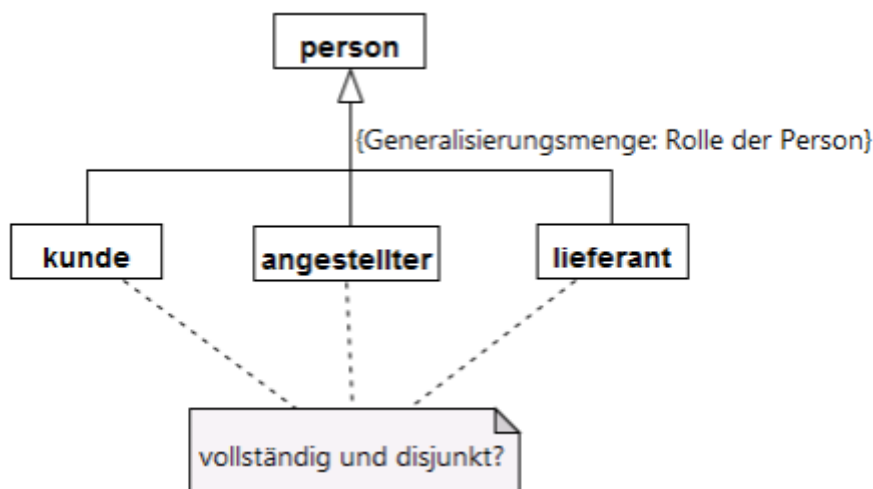


Vergleich zu ERD:



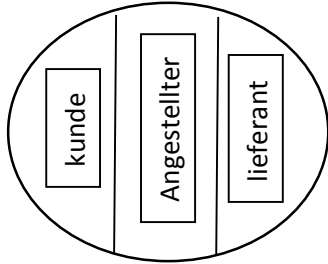
### 3.1 Erweiterungen zum ERD

Die Generalisierung / Spezialisierung ermöglicht die Darstellung einer hierarchischen Beziehung zwischen Entitätstypen, bzw. Klassen. Bei der Generalisierung – «vom Konkreten zum Allgemeinen» – werden gemeinsame Eigenschaften von Subtypen in einen Supertyp überführt. In der umgekehrten Modellierungsrichtung – der Spezialisierung – wird der Supertyp in Rollen oder Teilmengen aufgegliedert. Der Subtyp erbt alle Merkmale des Supertyps und kann um eigene Attribute ergänzt werden. Die Beziehung zwischen Subtyp und Supertyp wird durch eine «IS-A»-Beziehung charakterisiert: «kunde IS A person».

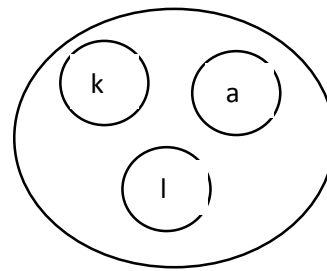


Ein System von Subtypen und Supertypen heisst disjunkt, wenn die einzelnen Subtypen keine gemeinsamen Elemente haben. Ein System von Subtypen und Supertypen nennt man vollständig, wenn der Supertyp keine eigenen Elemente enthält, also jedes Element in einem der Subtypen enthalten ist.

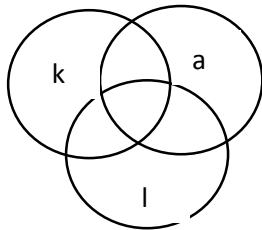
vollständig und disjunkt



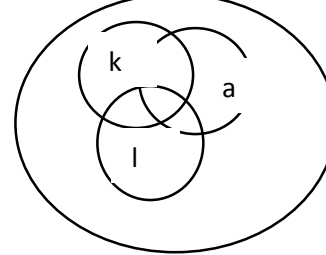
unvollständig und disjunkt



Vollständig überlappend

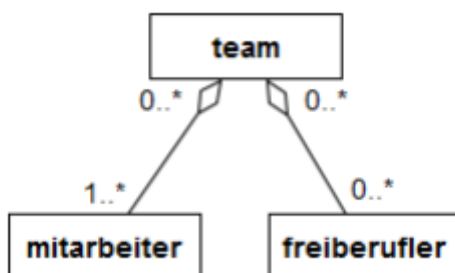


unvollständig überlappend



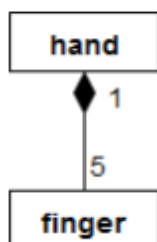
### 3.2 Aggregation / Komposition

Mit der Aggregation können Zusammenhänge als «Teile-Ganzes»-Beziehungen dargestellt werden. Eine Aggregation beschreibt also das strukturierte Zusammenfügen von Klassen zu einer komplexen Klasse. Wenn man die Beziehungen vom Ganzen aus betrachtet, handelt es sich um eine «HAS»-Beziehung, die zum Ausdruck bringt, dass das Ganze aus Teilen besteht: «team HAS mitarbeiter» etc. In umgekehrter Richtung gelesen sprechen wir von einer «IS PART OF»-Beziehung: «mitarbeiter IS PART OF team» etc.



Kann auch ohne

Handelt es sich bei der «Teile-Ganzes»-Beziehung um eine Existenzaussage, spricht man von einer Komposition oder starken Aggregation. In einem solchen Fall stellt man sich vor, dass die Teile ohne das Ganze nicht existieren können.



Kann nicht ohne  
(Existenz)

## 4 Das Relationenmodell

Grundlage:

Das Relationenmodell packt alle Entitätstypen und Beziehungstypen in flache Tabellen. Die Tabellen werden Relationen genannt. Doch diese Tabellen werden untereinander nach strengen Einschränkungen und Regeln verknüpft. Diese Regeln gewährleisten dann **Integrität und Redundanzfreiheit**.

### 4.1 Relationen

Jeder Entitätstyp wird zu einer Tabelle. Jeder Beziehungstyp wird (vorerst) zu einer Tabelle.

Die Theorie bezeichnet Tabellen als Relationen. Diese Bezeichnung ist glücklicherweise wenig gebräuchlich, da sie mit Relationship (Beziehung) verwechselt werden kann.

Für jede Tabelle gilt:

- eindeutiger Tabellenname
  - o wir übernehmen die Benennungen des ERDs; Beispiel: kunde
  - o bei der DB-Implementierung werden wir uns an ein Benennungskonzept halten und der Bezeichnung ein Präfix voranhängen; Beispiel: tbl\_kunde
- jede Entität - z.B. jeder Einzelkunde - ist darin eine Zeile
  - o die Zeilen heissen Tupel (alternativ: Datensätze, Records)
  - o die Reihenfolge der Tupel ist bedeutungslos, sie sind also z.B. nie sortiert
- jedes Attribut - z.B. Wohnort aller Kunden - ist darin eine Spalte
  - o ein Entitätstyp hat mindestens zwei Attribute, eine Tabelle also mindestens zwei Spalten
  - o Attribute erhalten eine in der Tabelle eindeutige Benennung
  - o sofern wir das ERD attribuiert haben, übernehmen wir die Benennungen des ERDs
  - o es gibt natürliche und künstliche Attribute: vorname ist natürlich, kunden\_nummer ist künstlich
- in jedem Koordinatenpunkt steht ein Feld
  - o Felder enthalten stets einen und nur einen Attributswert oder das Feld ist leer (genannt: NULL-Wert oder NULL-Marke)
  - o alle Werte, die ein Attribut annehmen kann, entstammen einem endlichem Wertebereich, einer Domäne; Beispiel: 0007 ist nicht in der Domäne Schweizerischer Postleitzahlen und deshalb kein gültiger Attributswert in einem Attribut plz
  - o die Reihenfolge der Attribute ist bedeutungslos; eine Anordnung mit von links nach rechts abnehmender Relevanz macht Sinn.

Die Relation ist eine Menge ungeordneter Tupel!

- Die waagrechte Ausdehnung, die Anzahl Attribute ist der Grad, also die Breite (Degree) einer Tabelle.
- Die senkrechte Ausdehnung, die Anzahl Tupel, ist die Kardinalität, also die Höhe (Card) einer Tabelle.

- Interessieren uns aus einer ganzen Tabelle nur gewisse Attribute und ihre Werte, dann sprechen wir von einer Projektion.
- Interessieren uns aus einer ganzen Tabelle nur gewisse Tupel und ihre Werte, dann sprechen wir von einer Selektion.

## 4.2 Primärschlüssel und Tupelintegrität

Jeder Datensatz (Tupel) darf nur einmal vorkommen. Zwei Tupel müssen sich also in mindestens einem Attributwert unterscheiden. Für die eindeutige Identifizierung (Tupelintegrität) sorgt der Primärschlüssel (PrimaryKey). **Tupelintegrität heisst deshalb auch Primärschlüsselintegrität.** Der PK ist immer ein Attribut oder eine geordnete Menge mehrere Attribute (zb aus mehreren FK).

Als PK eignen sich:

- jedes Tupel eindeutig und für immer identifizieren (deshalb auch: Identifikationsschlüssel) –
- keine Mehrfachnennungen tragen noch je tragen werden (SQL nennt sie: UNIQUE)
- keine Leereinträge tragen können noch je müssen (SQL: NOT NULL)

Bei PK gilt das Minimalitätsprinzip: möglichst wenige Attribute verwenden!

Die Begriffe im Überblick:

<i>ERM</i>	<i>Relationenmodell</i>	<i>alternativ</i>
Entität	<b>Tupel</b>	Zeile, Datensatz, Record
Entitätstyp	Relation	<b>Tabelle</b>
Beziehung	<b>Beziehung</b>	
Beziehungstyp	<b>Beziehungstyp</b>	
Kardinalität	<b>Kardinalität</b>	wenig sinnvoll: Anzahl
Attribut	<b>Attribut</b>	Spalte; wenig sinnvoll: Feld

The diagram shows a table named 'tbl\_buch' with the following columns: isbn, autor, titel, v\_name, v\_adr1, v\_adr2, ort, jahr. The 'isbn' column is marked as the primary key. Two rows are highlighted in orange, representing a selection. A bracket on the right side of these rows is labeled 'Selektion' and 'höhe'. A bracket at the bottom of the table is labeled 'Projektion' and 'abfrage auf Tabelle (Attribute beschränkt)'. Other handwritten notes include 'Primärschlüssel', 'Tabelle', 'Attribut', 'Attributswert', 'abfrage auf Tabelle', 'Tupel', 'Primäre (Verleger)', and 'Bezieh'.

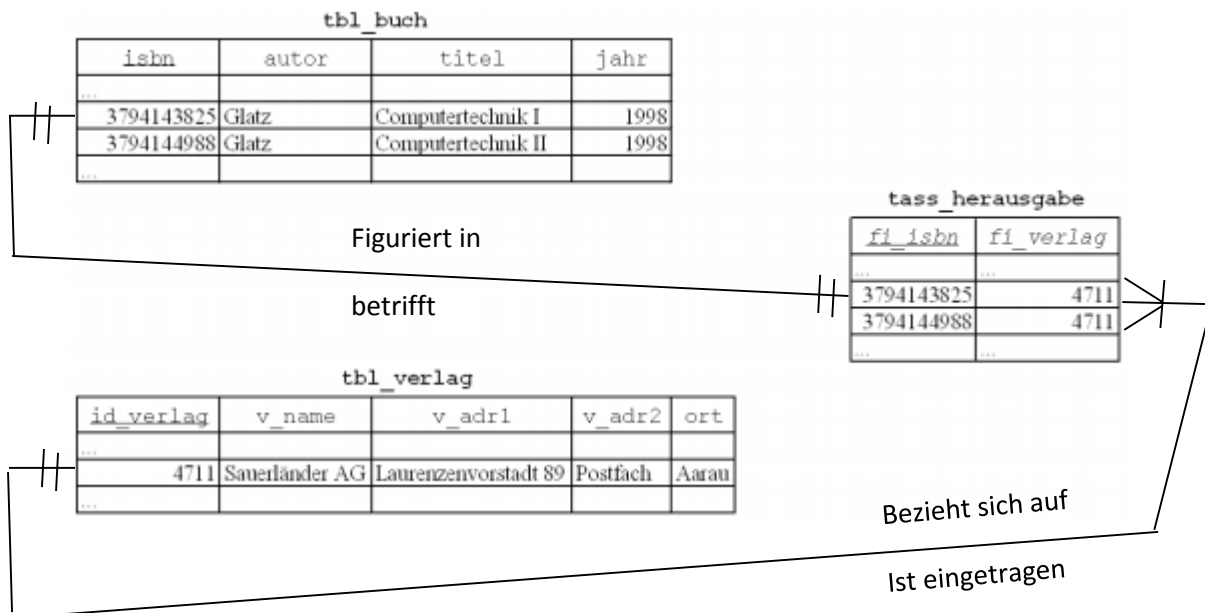
isbn	autor	titel	v_name	v_adr1	v_adr2	ort	jahr
3794143825	Glatz	Computertechnik I	Sauerländer AG	Laurenzenvorstadt 89	Postfach	Aarau	1998
3794144988	Glatz	Computertechnik II	Sauerländer AG	Laurenzenvorstadt 89	Postfach	Aarau	1998

## 4.3 Beziehungstypen

Wenn die Gewissheit oder nur schon die Möglichkeit besteht, dass in einer Tabelle Einträge redundant sind (z.B. mehrmalige Eintragung des gleichen Verlegers mit ganzer Adresse), dann werden die Tabellen aufgebrochen - dekomponiert - und in zwei Tabellen mit einer Beziehung (Relationship) zerlegt. Hilfreiche Regeln dazu lernen wir weiter unten kennen.

Obige Tabelle hat Redundanzen. Schuld daran ist immer die Tatsache, dass mehrere Attribute erfasst wurden, die eigentlich zu einer eigenen Entitätsmenge gehören:

1. Suche den Redundanz verursachenden Entitätstyp (hier: verlag).
2. Forme daraus einen eigenen Entitätstyp - natürlich mit einem Primärschlüssel.
3. Forme aus der Beziehung zwischen den beiden einen eigenen, dritten Entitätstyp. Diese (künstliche) Tabelle werden wir weiter unten eine assoziative (zuordnende) Tabelle nennen. Sie enthält mindestens zwei Attribute, nämlich die beiden zugeordneten Primärschlüssel.
4. Entwirf die Beziehungen als ERD.



In der Mitte ist ein assoziativer (auch: relationaler) Entitätstyp entstanden.

Das Gemachte ist relational vollständig korrekt und verletzt keinerlei Integritätsregeln! Aber die meisten so genannten "... zu-eins-Beziehungen" (und nur solche!) zweier Entitätstypen (Beispiel: buch und verlag) dürfen wir vereinfachen, indem wir die assoziative Entitätsmenge in die m-Seite "nehmen"; also:

5. Suche den Primärschlüssel der 1-seitigen Tabelle; z.B.: Jedes buch erscheint bei genau einem verlag; also ist verlag der Schlüssellieferant.
6. Richte in der m-seitigen Tabelle ein neues Attribut als so genannten Fremdschlüssel ein, benenne es entsprechend (gleiche Benennung wie Primärschlüssel oder sonst gut sprechend). Kennzeichne die Fremdschlüssel-Benennung durch Einkreisung (im Skript kursiv).
7. Fülle die korrespondierenden Primärschlüsselwerte der 1-seitigen Tabelle in die soeben angelegte Fremdschlüssel-Spalte der m-seitigen Tabelle ein.
8. Die neue Beziehung ersetzt die bisherige. Die assoziative Tabelle entfällt in diesem Fall. Entwirf die Beziehungen als ERD.

Für **Fremdschlüsselwerte** gilt folgendes:

- Sie müssen am Bezugsort vorhanden sein: **referenzielle Integrität**
- Sie haben die gleiche Domäne wie beim Bezugsort
- Sie dürfen leer, also NULL-Marken sein
- Sie dürfen mehrfach vorkommen

#### 4.4 Fremdschlüssel und referenzielle Integrität

Die referenzielle Integrität verlangt, dass ausnahmslos kein Fremdschlüssel-Wert vorkommen darf, der nicht auch als Primärschlüssel in der Herkunftstabelle vorhanden ist.

Bsp: Es darf keine Bestellung gegen mit einer Kundennummer, die nicht existiert. →

Fremdschlüsselintegrität.

- 1-seitig ein Primärschlüsselwert geändert wird. Falls m-seitig dieser Wert vorhanden ist, muss die Mutation
  - o weitergegeben (CASCADE) oder primärseitig
  - o unterbunden (RESTRICT) werden;
- 1-seitig ein Primärschlüsselwert gelöscht wird. Falls m-seitig dieser Wert vorhanden ist, sind dort
  - o Folgelösungen auszulösen (CASCADE) oder die primärseitige Löschung muss
  - o unterbunden werden (RESTRICT).
  - o Alternativ möglich ist, dass m-seitig NULL-Marken an die Stelle der Fremdschlüsselwerte gesetzt werden (SET NULL). Damit kann der Rest des Datensatzes bestehen bleiben.

(Die einseitige Tabelle heisst auch primär(schlüssel)seitige oder Elterntabelle)

(Die m-seitige Tabelle heisst auch fremdschlüsselseitige oder Kindtabelle)

## 5 Vom ERD zum relationalen ERD

### 5.1 Attribute und Domänen (Wertebereiche)

Mit der Identifikation von Entitäts- und den Beziehungstypen wurde eine wesentliche Vorarbeit zur Modellierung von Daten geleistet. Ist dieser Prozess erst einmal abgeschlossen, muss ein weiterer Entwurfsschritt folgen: die Attribuierung der verknüpften Entitätstypen.

Unter einer Domäne wird der Wertebereich eines Attributs verstanden. Dieser Wert kann logisch zwar unendlich sein, ist real indessen immer endlich.

Bsp:

- Zur Domäne von Familiennamen gehören alle denkbaren Familiennamen, nicht aber z.B. natürliche Zahlen.
- Zur Domäne von (alten) AHV-Nummern gehören Datenworte wie XXX.XX.XXX.XXX, (wobei X Ziffern sind), nicht aber Fließkommazahlen.
- Zur Domäne Schweizerischer Postleitzahlen gehören natürliche Zahlen zwischen 1000 und 9999, nicht aber Buchstaben und/oder Satzzeichen.
- Zur Domäne der Bindungsformen eines Buches gehören: geleimt, fadengeheftet, geklammert, nicht aber genagelt oder gelötet.



In einer Datenbank werden die Objekte, über die wir Daten sammeln, vollständig durch ihre Attributwerte beschrieben. Auch der Primärschlüssel und Fremdschlüssel sind Attribute.

- Primärschlüssel-Attribute und normale Attribute haben normalerweise einen statischen Wertebereich
- Fremdschlüssel-Attribute haben meist einen dynamischen Wertebereich, weil jeder neue Wert beim Primärschlüssel der entsprechenden Domäne um einen Wert vermehrt.

Für Attribute gilt:

- Pro Tabelle hat jedes Attribut eine eindeutige Benennung
- Attribute beziehen ihre Werte aus einer Domäne
- Die Attributwerte sind atomar, also ausnahmslos ein Wert pro Feld
- Attributwerte können NULL sein, wenn das die Definition zulässt

## 5.2 Attribut oder Entitätstyp

Werden zu einem Objekt der realen Welt mehrere Eigenschaften erfasst, wird das Objekt zum Entitätstyp

Interessiert ein Objekt nur eine Eigenschaft, dann bleibt es Attribut eines anderen Entitätstyps.

**Ein Entitätstyp hat Attribute, ein Attribut hat keine Unterattribute.**

Beispiel 1: Entitätstyp buch; der Autor interessiert in diesem Beispiel nur mit Name, Vorname (siehe Übung "Literaturliste eines Dozenten" weiter oben mit Lösungskommentaren): buch (isbn, autor, titel, jahr, fi\_verlag) verlag (id\_verlag, name, ... ) usw.

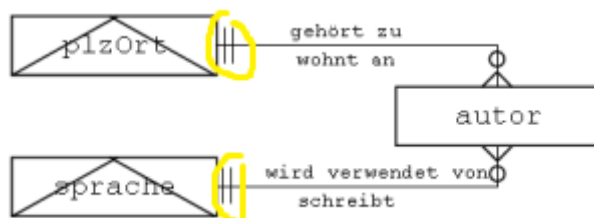
Beispiel 2: Entitätstyp buch, der Autor ist in diesem Beispiel ein eigener Entitätstyp: buch (isbn, fi\_aut, titel, jahr, fi\_verlag) autor (id\_aut, name, vorname, gebDat, tel, mobil, ... ) usw. verlag (id\_verlag, name, ... ) usw.

## 5.3 Arten von Entitätstypen

### 5.3.1 Attributiver Entitätstyp (tkey)

Attributive Entitätstypen sind reine „Attributlieferanten“. Sie helfen als „Spender“ von Fremdschlüssel. (tkey\_....) Sie haben selber aber **keine** Fremdschlüssel eingetragen.

Bsp: Tkey\_plzOrt



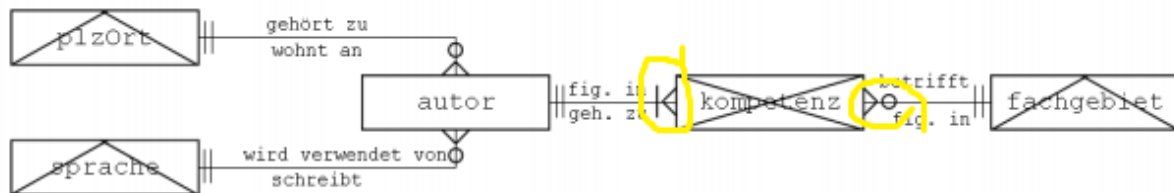
### 5.3.2 Fundamentaler Entitätstyp (tbl)

Fundamentale Entitätstypen modellieren reale Entitäten (Partner, Buch, Verlag, Autor...) und haben als solche mindestens auch mehrere Fremdschlüsselattribute von attributiven Entitätstypen.

### 5.3.3 Assoziativer Entitätstyp (tass)

m:m Beziehungen müssen zwingen aufgelöst werden. Bei dieser Dekomposition entsteht jeweils ein neuer, künstlicher Entitätstyp, der assoziative Entitätstypen verbindet.

Kompetenz eine tass Tabelle, da sie eine mehrfach zu mehrfach Beziehung zwischen autor und fachgebiet aufgelöst hat



## 5.4 Rezept zum relationalen Entwurf

### 5.4.1 Elemente

1. Entitätstypen: Analysiere das reale System (oder den beschriebenen Fall oder die vorliegende flache Tabelle usw.) und isoliere dessen Objekte als Entitäten. Fasse gedanklich gleichartige Entitäten zusammen und abstrahiere sie zu Entitätstypen.

**Entwurf jeden Entitätstyp je als ein Rechteck und benenne dieses.**

2. Beziehungstypen: Analysiere die Beziehungen zwischen Entitäten. Fasse gedanklich gleichartige Beziehungen zusammen und abstrahiere sie zu Beziehungstypen zwischen Entitätstypen.

**Entwurf jeden Beziehungstyp je als eine Linie. Benenne die beiden Assoziationstypen des Beziehungstyps in beide Leserichtungen korrekt.**

3. Multiplizitäten: Analysiere die Multiplizitäten an beiden «Enden» des Beziehungstyps.

**Entwurf beidseitig je die minimale und maximale Multiplizität in der jeweiligen Notation.**

4. Attribute: Ermittle die Eigenschaften der Entitäten. Fasse gedanklich gleichartige Eigenschaften zu Attributen zusammen.

**Entwurf die Attributliste aller Entitätstypen. Markiere für jeden Entitätstyp den Primärschlüssel durch Unterstreichung und die Fremdschlüssel durch Einkreisen.**

### 5.4.2 Behandlung von Beziehungstypen

In der relationalen Theorie sind alle Beziehungen ein eigener (assoziativer) Entitätstyp. Im relationalen Entwurf - also in der Praxis - gibt es zu dieser Regel Vereinfachungen: Es gilt hier, dass mindestens alle Beziehungen, die beidseitig ein m haben, aufzulösen sind und "in der Mitte" einen assoziativen Entitätstyp erhalten.

## 6 Normalisierung (Skript Teil 1: S. 46 – 57)

Warum Normalisierung?

Eine schlecht modellierte Datenbank zeigt zur Inkonsistenz/Redundanz<sup>1</sup> - zwar bei jeder Einfügung, Mutation oder Löschung. Das Relationenmodell und streng nach ihm modellierte Datenbanken **vermeiden solche Anomalien** langfristig und bewahren deshalb die Konsistenz.

Die Normalformen gewährleisten die langfristige Redundanzfreiheit von Datenbanken. Es gibt verschiedene ineinander geschachtelte Normalformen (Insgesamt 5 NFs, im Unterricht jedoch nur die ersten 3 NFs durchgenommen). Normalformen sind ein MUSS!

Die Punkte 1.1 – 1.3 dienen als Checkliste zur Normalisierung:

### 6.1 Erste Normalform (1 NF)

Ein Relationstyp (Tabelle) befindet sich in der **ersten Normalform (1NF)**, wenn die Wertebereiche der Attribute des Relationstypen atomar sind.

**Beispiel 1 NF:**

Vorher (Rot ist der so falsch):

ReNr.	Datum	Name	Straße	Ort	Artikel	Anzahl	Preis
187	01.01.2012	Max Mustermann	Musterstr. 1	12345 Musterort	Stift	2	1,00 €

Nachher (Grün stellt die Verbesserungen nach der Normalisierung dar):

ReNr	Datum	Name	Vorname	Straße	PLZ	Ort	Artikel	Anzahl	Preis
187	01.01.2012	Mustermann	Max	Musterstr. 1	12345	Musterort	Stift	2	1,00 €

**Erklärung:**

Der Name (vorher) ist in Name, Vorname (nachher) aufgeteilt worden. Ebenfalls wurde der Ort (vorher) in PLZ, Ort (nachher) aufgeteilt.

<sup>1</sup> Eine Nachricht ist dann redundant, wenn sie ohne Informationsverlust weggelassen werden kann.

## 6.2 Zweite Normalform (2 NF)

Ein Relationstyp (Tabelle) befindet sich genau dann in der zweiten Normalform (2NF), wenn er sich in der ersten Normalform (1NF) befindet und jedes Nichtschlüsselattribut von jedem Schlüsselkandidaten (auch Primärschlüssel genannt) voll funktional abhängig ist.

### Beispiel 2 NF:

Vorher:

CD_ID	Albumtitel	Interpret	Jahr der Gründung	Track	Titel
4711	I don't mind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Nachher:

CD				Lied		
CD_ID (PK)	Albumtitel	Interpret	Jahr der Gründung	CD_ID (FK)	Track	Titel
4711	Not That Kind	Anastacia	1999	4711	1	Not That Kind
4712	Wish You Were Here	Pink Floyd	1964	4711	2	I'm Outta Love
				4711	3	Cowboys & Kisses
4713	Freak of Nature	Anastacia	1999	4712	1	Shine On You Crazy Diamond
				4713	1	Paid my Dues

### Erklärung:

Man muss die gleichartigen Spalten sozusagen zu einer eigenständigen Tabelle zusammenfassen (CD, Lied). Die Daten in der Tabelle werden in zwei Tabellen aufgeteilt: CD und Lied. Die Tabelle CD enthält nur noch Felder, die voll funktional von CD\_ID abhängen, hat also CD\_ID als Primärschlüssel. Auch der Albumtitel allein sei eindeutig, also ein Schlüsselkandidat. Da keine weiteren (zusammengesetzten) Schlüsselkandidaten existieren, liegt die Tabelle damit automatisch in der 2. Normalform vor. Die Tabelle "Lied" enthält schließlich nur noch Felder, die voll funktional von CD\_ID und Track abhängen, liegt also auch in der 2. Normalform vor. Mit Hilfe dieser verlustfreien Zerlegung sind auch die genannten Redundanzen der Daten beseitigt.

### 6.3 Dritte Normalform (3 NF)

Ein Relationstyp befindet sich genau dann in der dritten Normalform (3NF), wenn er sich in der zweiten Normalform (2NF) befindet und kein Nichtschlüsselattribut transitiv von einem Kandidatenschlüssel abhängt.

#### Beispiel3 NF:

Vorher:

KundenNr	Name	Vorname	Straße	PLZ	Ort
007	Mustermann	Max	Musterstr. 1	12345	Musterort

Nachher:

Tabelle 1:

KundenNr	Name	Vorname	Straße	PLZ
007	Mustermann	Max	Musterstr. 1	12345

Tabelle 2:

PLZ	Ort
12345	Musterort

#### Erklärung:

In der Tabelle "Kunden" sind die Attribute "Vorname", "Straße" und "PLZ" abhängig vom Attribut "Name", nicht vom Primärschlüssel "KundenNr". Außerdem ist "Ort" abhängig von "PLZ".

Die transitiv abhängigen Spalten werden in eine weitere Untertabelle ausgelagert, da sie nicht direkt vom Schlüsselkandidaten abhängen, sondern nur indirekt.

### 6.4 Funktionale Abhängigkeit

Bestimmt jeder Wert aus einer Domäne A genau einen Wert aus einer Domäne B, dann ist B funktional abhängig von A.

$$B = f(A)$$

#### Beispiele

- A: Kalenderdatum;	B: Wochentage:	11.12.2006 → Montag
- A: ISBN;	B: Buchtitel:	9783642151255 → Fischer/Hofer: Lexikon der Informatik
- A: Internet-Domänen;	B: Domäneninhaber	nzz.ch → Neue Zürcher Zeitung, Zürich
- ...		

**Übungen dazu:** 25, 26, 27, 29, 30, 31 im Anhang

## 7 Datenbank in ACCESS (Skript Teil 1: S. 58 – 96)

Wir tun dies anhand eines - selbstverständlich - Relationalen DBMS, und zwar des Produkts Microsoft Access. Zu Beginn erklären wir noch einige Begriffe (Alle diese Begriffen werden wir mit Access benötigen).

### 7.1 Ein Relationales Datenbanksystem (RDBMS)

- packt alle Entitätstypen in Tabellen
- sichert die Tupelintegrität in jeder Tabelle durch die Definition eines Primärschlüssels ab
- drückt alle Beziehungen durch die Definition eines Primär-/Fremdschlüsselpaars aus
- sichert referenzielle Integrität durch die Definition der Löscho- bzw. Mutationsaktionen ab
- legt über alle Definitionen automatisch eine eigene relationale Datenbank ab

Die «grossen», kommerziellen RDBMS sind mehrbenutzerfähig, das heisst, sie

- regeln die Authentisierung und Autorisierung von Benutzenden
- sichern Anomalien durch gleichzeitige Zugriffe ab
- verhindern so Phantomeffekte.

### 7.2 Datenbasis

Die benutzerdefinierten Tabellen samt ihren Inhalten bilden die Datenbasis.

### 7.3 Systemkatalog

Enthält die Metadaten.

### 7.4 Eine Datenbank

- trägt einen systemweit eindeutigen Namen
- besteht aus zwei oder mehreren Tabellen und in der Regel weiteren Objekten
- besteht aus den (automatisch geführten) Metadaten.

### 7.5 Eine Tabelle

- trägt einen datenbankweit eindeutigen Namen
- hat zwei oder mehr Attribute (Spalten, Felder)
- hat einen Primärschlüssel aus einem oder mehreren Attributen
- hat eine (relativ statische) Breite, bestimmt durch die Anzahl Attribute
- hat eine (relativ dynamische) Höhe, bestimmt durch die Anzahl Tupel (Datensätze).

Tabellen (Tables) sind die wichtigsten Objekte einer Datenbank.

### 7.6 Ein Attribut

- trägt einen tabellenweit eindeutigen Namen
- entnimmt seine Werte einer bestimmten Domäne, die wir hier Datentyp nennen
- hat eine bestimmte Feldgrösse die u.U. durch den Datentyp bestimmt wird.

### 7.7 Als weitere Objekte werden wir uns befassen mit

- Indizes (Indexes)
- Abfragen (Queries) und Sichten (Views)
- Formularen (Forms)
- Berichten (Reports)

## 7.8 Datentypen

Wir haben gelernt, dass jeder Attributswert einer Domäne entnommen werden muss. In Relationalen Datenbanken heissen die Domänen Datentypen. Access hat viele Datentypen:

Text
Memo
Zahl
Datum/Uhrzeit
Währung
AutoWert
Ja/Nein
OLE-Objekt
Hyperlink
Nachschlage-Assis

### 7.8.1 Alphanumerische Typen:

Zeichentypen (Werte mit Buchstaben, Ziffern, Sonderzeichen) → Mit alphanumerischen Typen kann das System nicht rechnen. (Text, Memo)

### 7.8.2 Numerische Typen:

- Byte:  $0 \leq \text{Ganzzahl} \leq 255$   
(1 Byte pro Zahl)
- Integer:  $-32'768 \leq \text{Ganzzahl} \leq 32'767$   
(16 Bits = 2 Bytes pro Zahl)
- Long Integer:  $-2'147'483'648 \leq \text{Ganzzahl} \leq 2'147'483'647$   
(32 Bits = 4 Bytes pro Zahl)
- Single: pos./neg. Fließkommazahl mit einfacher Genauigkeit  
(4 Bytes pro Zahl)
- Double: pos. /neg. Fließkommazahl mit doppelter Genauigkeit  
(8 Bytes pro Zahl).

Autowerte sind Long Integers!

Ein Fremdschlüssel muss den gleichen Datentyp wie das zugehörige Primärschlüsselattribut haben

## 7.9 Indizes

Indizes sind eine Methode zur Beschleunigung des Suchens: Definieren wir einen Index auf ein Attribut, dann erstellt das DBMS von diesem Attribut eine Kopie, die aber bei jeder Erfassung, Mutation und Löschung sortiert wird.

Eine Suche erfolgt dann zweistufig:

1. Sequenzielles Suchen des ständig sortierten Suchschlüssels in der Indexdatei, dann Lesen der Tupelnummer
2. Sprung zum Tupel mit der zugehörigen Nummer (dies ist der Primärschlüssel oder eine maschinenintern vergebene Adresse bei grossen RDBMS) und Einlesen der Daten

Einen solchen Index erstellen Sie nicht direkt mit der Tabelle, sondern mit einer separaten Anweisung:

```
CREATE INDEX IndexAbteilung ON tblMitarbeiter(Vorname)
```

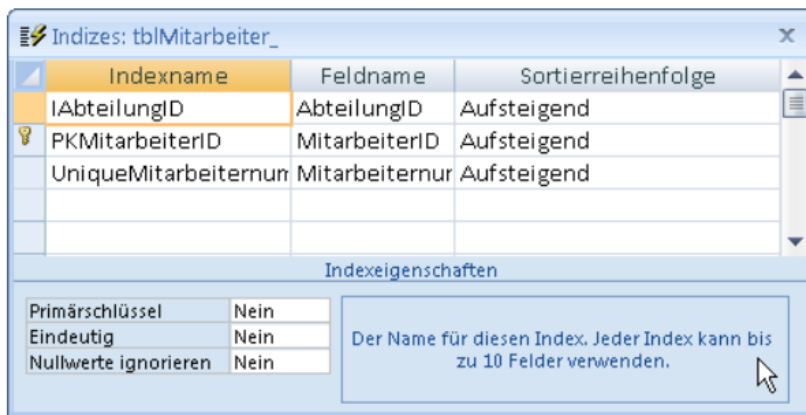


Abbildung 8.27: Auflistung der Indizes der Tabelle tblMitarbeiter

### 7.9.1 Vorteile:

Ein Index kann das Suchen und Sortieren dramatisch beschleunigen. Er belastet im Gegenzug das System leicht:

- Zusatzaufwand bei Tupelmutationen aller Art
- mehr Speicher

### 7.9.2 Achtung:

Das Vergeben eines Index' muss deshalb sehr sorgfältig und überlegt erfolgen. Vergeben Sie Indizes nur bei Attributen, die

- oft als Such- oder Sortierkriterien dienen
- eine hohe Selektivität aufweisen

## 7.10 Beziehungen

Access kennt nur Beziehungen vom Typ

- 1 : mc
- 1 : c

Ist das Fremdschlüsselattribut kein Primärschlüssel (Normalfall), "macht" Access eine **1 : mc**-Beziehung. Ist das Fremdschlüsselattribut auch alleiniger Primärschlüssel, "macht" Access daraus eine **1 : c**-Beziehung.



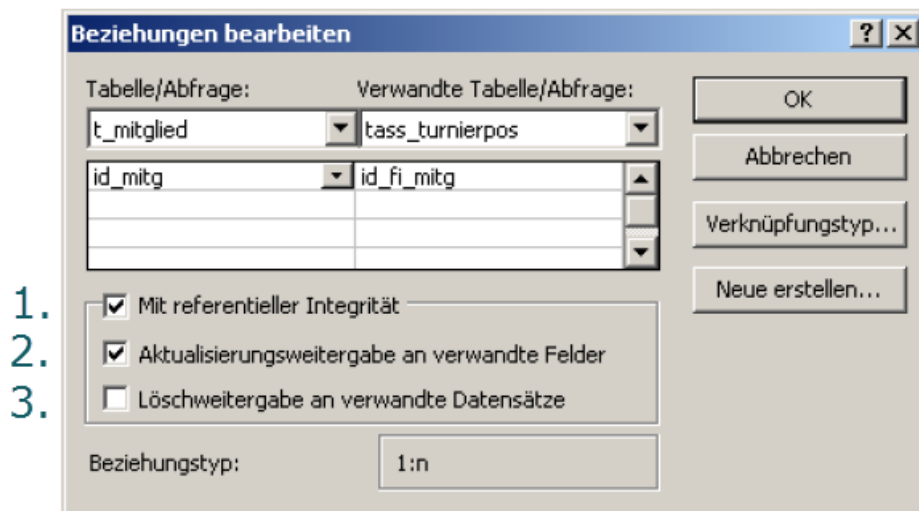


Abbildung 38: Definition einer Beziehung mit referenzieller Integrität

1. Referenzielle Integrität ist der Normalfall
2. **Mit aktivierter Aktualisierungsweitergabe** sichert das DBMS die referenzielle Integrität durch kaskadiertes Mutieren ab: Jede Änderung eines Primärschlüsselwerts in der Elterntabelle mutiert die zugehörigen Fremdschlüsselwerte in der/den Kindtabelle/n. Dies kann seinerseits wieder Kaskadierungen auslösen. Eine Mutation kann unumkehrbar eine Grosszahl weiterer Mutationen auslösen!

**Nicht aktivierte Aktualisierungsweitergabe** (auch genannt: bedingtes Mutieren) verhindert Mutationen an Primärschlüsselwerten in der Elterntabelle, solange entsprechende Fremdschlüsselwerte existieren.

3. **Mit aktivierter Löschweitergabe** sichert das DBMS die referenzielle Integrität durch kaskadiertes Löschen ab: Jede Löschung eines Primärschlüsselwerts in der Eltern-tabelle löscht die Tupel mit den zugehörigen Fremdschlüsselwerten in der/den Kind-tabelle/n. Dies kann seinerseits wieder Kaskadierungen auslösen. Eine Löschung kann also (unumkehrbar) eine Grosszahl weiterer Löschungen auslösen!

**Nicht aktivierte Löschweitergabe** (auch genannt: bedingtes Löschen) verhindert Löschungen von Primärschlüsselwerten in der Elterntabelle, solange entsprechende Fremdschlüsselwerte existieren.

## 7.11 Abfragen (Queries, nur einer Tabelle)

Abfragen (Queries)

- sind Datenbankobjekte, die wir definieren und somit anlegen müssen
- selektieren Daten aus einer oder mehreren Tabellen (genannt: Verbünde oder Joins)
- liefern als Resultat immer Tabellen
- werden ad hoc definiert oder als Definition gespeichert (dann eigentlich Sichten o-der Views genannt)
- können auf Tabellen und/oder ihrerseits auf Sichten basieren
- können wiederum Abfragen enthalten (genannt: Unterabfragen oder Subqueries)

## Die resultierende Tabelle einer Abfrage

- ist eine Menge (Set) mit 0, 1 oder n (Resultat-) Tupeln und den verlangten Attributen
- ist ohne unsere Veranlassung nie sortiert
- wird bei jedem Aktivieren der Abfrage neu berechnet
- ist also nicht gespeichert, wir sprechen deshalb von einer temporären oder virtuellen Tabelle
- kennt keine Primärschlüssel
- kann Redundanzen enthalten

Wir formulieren Abfragen in MS-Access vorwiegend mit der Maus. Das Vorgehen nennt sich etwas irreführend **Query By Example (QBE)**.










### Abfragen aus einer Tabelle:

Sehr gute Webseite: <http://www.access-tutorial.de/abfragen/abfrage.htm>

1.	Feld:	IngLand	IngLand
2.	Tabelle:	tblOrte	tblOrte
3.	Funktion:	Gruppierung	Anzahl
4.	Sortierung:		
5.	Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6.	Kriterien:		
7.	oder:		

1. **Feld:** Dort kommen die Daten von (z.B. txtOrt) = Spaltennamen. Mit Feldern können auch neue Berechnungen erstellt werden. SIEHE Punkt 6 unter Berechnungen.
2. **Tabelle:** Tabellennamen in der das Feld vorkommt (z. B. tblOrt)
3. **Funktion:** Abfragen können auch für verschiedene Berechnungen genutzt werden.

Verfügbare Funktionen sind:

-  Summe
-  Mittelwert
-  Min
-  Max
-  Anzahl
-  StAbw
-  Varianz
-  Erster Wert
-  Letzter Wert
-  Gruppierung

4. **Sortierung:** Es gibt nur Aufsteigend, Absteigend oder (nicht sortiert). Selbsterklärend
5. **Anzeigen:** Mit dem Häkchen bei "Anzeigen" definieren Sie, ob das Attribut anzuzeigen sei oder ob es lediglich der Formulierung von Kriterien dient, selber aber nicht angezeigt wird.  
Beispiel: alle Kunden im PLZ-Gebiet 8\* (8000, 8050, 8432) anzuzeigen sind, die PLZ selber aber nicht.

6. **Kriterien:** Eine der wichtigsten Fähigkeiten von Abfragen ist, Datensätze nach beliebigen Kriterien bzw. Bedingungen.

### Ausdrücke (Bsp)

Die Abfrage gibt jetzt nur noch Datensätze aus, bei denen im Feld txtOrt „München“ eingetragen ist

Feld:	tblOrte.*	txtOrt
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:		"München"
oder:		

Es ist auch möglich, mit **NICHT** eine Bedingung zu verneinen:

Feld:	tblOrte.*	txtOrt
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:		Nicht "B*"
oder:		

### Operatoren(Bsp)

So werden alle Datensätze ausgegeben, bei denen im Feld lngLand ein Wert größer als 2 steht

Feld:	tblOrte.*	lngLand
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:		>2
oder:		

Auflistung aller möglichen Operatoren im Bereich Kriterien:

Operatoren		
Operator	Beispiel	Bedeutung
=	=1	gleich
<>	<>1	ungleich
<	<1	kleiner als
>	>1	größer als
<=	<=1	kleiner oder gleich
>=	>=1	größer oder gleich
Zwischen ... und	Zwischen 1 und 3	Zwischen zwei Werten (jeweils einschließlich)
Wie	Wie "X*"	entspricht einem Textmuster
In	In ('Müller', 'Mayer', 'Schulze')	in einer Liste enthaltene Werte

Operatoren funktionieren auch für Textfelder:

Feld:	txtOrt	
Tabelle:	tblOrte	
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:	>"C"	
oder:		

Nun erhält man alle Ortsnamen, die alphabetisch nach „C“ kommen:

**Achtung: Leere Felder werden mit IST NULL gesucht. Leere Zeichenfolgen mit ""**

### Der WIE-Operator

Für Textfelder gibt es zusätzlich noch den Wie-Operator. Mit ihm können Platzhalterzeichen verwendet werden, um alle Elemente zu finden, die einem Muster entsprechen

Feld:	tblOrte.*	txtOrt
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:		Wie "B*"
oder:		

Die Abfrage gibt jetzt alle Datensätze aus, bei denen der Ortsname mit „B“ beginnt:

Abfrage1		
IDOrt	txtOrt	IngLand
2	Bamberg	2
5	Bayreuth	2
3	Berlin	3

Folgende **Platzhalterzeichen** sind erlaubt:

Platzhalterzeichen			
Symbol	Beispiel	Ergebnis	Verwendung
*	Wie "*er"	findet Maier, Müller, Junker	Eine beliebige Anzahl Zeichen
?	Wie "Ma?er"	findet Maier, Majer und Mayer	Ein beliebiges einzelnes Zeichen
#	Wie "1#3"	findet 103, 113, 123	Eine einzelne Ziffer
[ ]	Wie "Ma[iy]er"	findet Maier und Mayer, aber nicht Majer	Ein einzelnes Zeichen in den eckigen Klammern
!	Wie "Ma[!iy]er"	findet Majer, aber nicht Maier oder Mayer	Ein einzelnes, nicht aufgelistetes Zeichen
-	Wie "b[a-c]d"	findet bad, bbd und bcd	Ein einzelnes Zeichen im angegebenen Bereich

### Der IN-Operator

Der oben aufgeführte in-Operator ist eine Besonderheit: Man könnte mit In ('Müller', 'Mayer', 'Schulze') nach Datensätzen suchen, die in einer Liste enthalten sind. So etwas ist normalerweise mit „oder“ sinnvoller (siehe mehrere Kriterien). Spannend wird es erst im Zusammenhang mit Unterabfragen (Bsp)

Feld:	txtOrt	IngLand
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:		In (SELECT IngLand FROM tblOrte WHERE txtOrt = "Neustadt")
oder:		

### Mehrere Kriterien

Jetzt die Kriterien für alle Datensätze, die mit „B“ beginnen und in Bayern liegen:

Feld:	tblOrte.*	txtOrt	lngLand
Tabelle:	tblOrte	tblOrte	tblOrte
Sortierung:			
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Kriterien:		"B*"	2
oder:			

Nebeneinander liegende Kriterien sind automatisch mit UND verknüpft. Möchte man in einer Spalte ein UND dann kann einfach ein UND hinzugefügt werden.. Siehe unten.

### Weitere nützliche Beispiele zusammengefasst

### Unterabfragen

qry\_Bsp8\_Subquery : Auswahlabfrage

tass\_police

\*  
id\_fi\_kunde  
id\_fi\_vers\_art  
praem\_stufe  
ausbezahlt

Feld:	id_fi_kunde	id_fi_vers_art	ausbezahlt
Tabelle:	tass_police	tass_police	tass_police
Sortierung:			
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:			>(select avg(ausbezahlt) from tass_police)
oder:			

Feld:	txtOrt	IngLand
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:		In (SELECT IngLand FROM tblOrte WHERE txtOrt = "Neustadt")
oder:		

## Berechnungen

Beben Plus (+), Minus (-), Mal (\*), Geteilt (/) und Exponent (^) ist der Verkettungsoperator (&) wichtig, um Werte hintereinander zu setzen. Sind beide Operanden Texte, liefert Plus zwar in der Regel ein identisches Ergebnis. Wenn aber beide Operanden aus Zahlen bestehen, liefern [Feld1] + [Feld2] und [Feld1] & [Feld2] unterschiedliche Ergebnisse. Da man sich oft nicht darauf verlassen kann, was User in Textfeldern eingeben, ist bei Texten der Verkettungsoperator vorzuziehen.

And, Or, Not, Imp, Xor, Eqv zurechtkommen), und zahlreiche Funktionen. Z.B. kann man mit Right([Name], 1) das letzte Zeichen eines Textes ermitteln.

tass\_police

\*  
id\_fi\_kunde  
id\_fi\_vers\_art  
praem\_stufe  
ausbezahlt

Feld:	ausbezahlt	Täglich: [ausbezahlt]/360
Tabelle:	tass_police	
Funktion:	Summe	Summe
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:		
oder:		

	Summe von ausbezahlt	Täglich
►	CHF 18922.00	52.5611111111

Feld:	tblVerkäufe.*	Bruttopreis: [curNettopreis]*1,19
Tabelle:	tblVerkäufe	
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:		
oder:		

### Eigene Felder erstellen (Abgesehen von Berechnungen)

, Kriterien: (empty)." data-bbox="174 257 489 472"/>

Feld:	Unsere höchste Leistung: ausbezahlt
Tabelle:	tass_police
Funktion:	Max
Sortierung:	
Anzeigen:	<input checked="" type="checkbox"/>
Kriterien:	

#### 7. Oder: Selbsterklärend

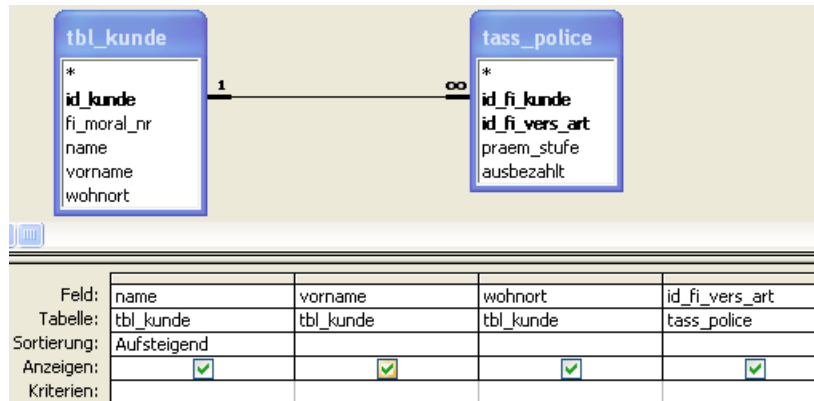
Feld:	tblOrte.*	txtOrt
Tabelle:	tblOrte	tblOrte
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kriterien:		"S*"
oder:		"M*"

**Skript-Übungen zu Abfragen einer Tabelle: 32 – 39**

## 7.12 Joins (Abfrage mehrere Tabellen)

### INNER JOIN

Gehen wir einleitend von einer ganz normalen Fragestellung aus: Welche Kunden haben welche Versicherungsart?



Das Ergebnis ist der der gewollte «Normalfall»: ein Verbund (Join) von zwei oder mehr Tabellen. Zwecks Abgrenzung zu den weniger normalen Fällen nennen ihn künftig Inner Join:

	Kundenname	Kundenvorname	Wohnort	Versicherungscode
►	Hauser	Martin	Kriens	1700
	Hauser	Martin	Kriens	1800
	Hauser	Martin	Kriens	1500
	Hauser	Martin	Kriens	1300
	Hauser	Martin	Kriens	1200
	Klausen	Heidi	Zürich	1500
	Klausen	Heidi	Zürich	1600
	Klausen	Heidi	Zürich	1200
	Meier	Max	Luzern	1000
	Meier	Max	Luzern	1300
	Menzer	Claudia	Horw	1300
	Menzer	Claudia	Horw	1500
	Menzer	Claudia	Horw	1700
	Menzer	Claudia	Horw	1900
	Menzer	Claudia	Horw	1800
	Müller	Eva	Bern	1000
	Müller	Eva	Bern	1400
	Müller	Eva	Bern	1300
	Müller	Eva	Bern	1200
*				

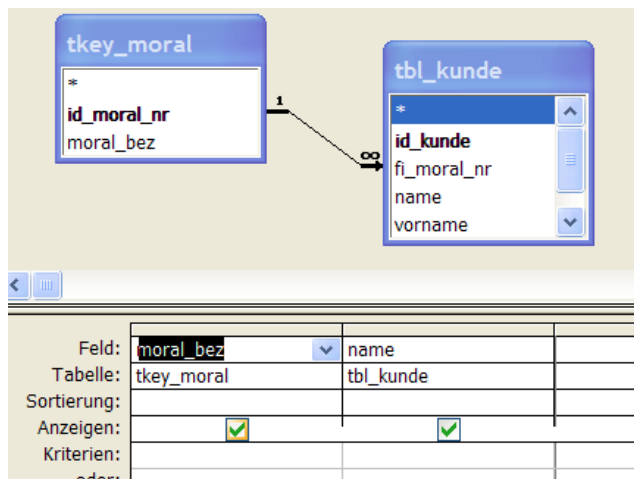
### CROSS JOIN

Beim Cross Join werden einfach alle Zahlungsmorale für alle Kunden angezeigt. Es werden alle Daten ausgegeben.

	Kundenname	Kundenvorname	Wohnort	Zahlungsmoral
►	Düsentrieb	Daniel	Entenhausen	gut
	Düsentrieb	Daniel	Entenhausen	schlecht
	Düsentrieb	Daniel	Entenhausen	sehr gut
	Hauser	Martin	Kriens	sehr gut
	Hauser	Martin	Kriens	gut
	Hauser	Martin	Kriens	schlecht
	Hilton		Malibu	gut
	Hilton		Malibu	sehr gut
	Hilton		Malibu	schlecht
	Klausen	Heidi	Zürich	sehr gut
	Klausen	Heidi	Zürich	gut



## LEFT JOIN

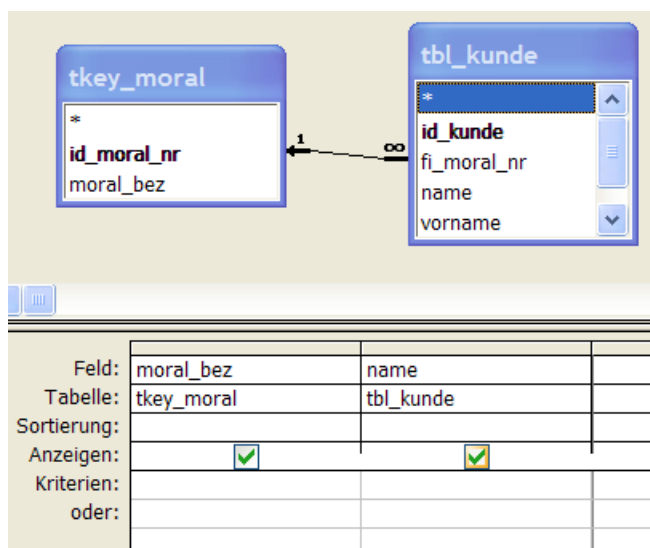


Wir haben einen Left Outer Join produziert: Zeige mir alle Tupel aus der links genannten Tabelle, auch wenn in der rechts genannten Tabelle keine Entsprechung vorhanden ist.

Resultat:

Zahlungsmoral	Kundenname
schlecht	Hilton
schlecht	Hauser
schlecht	Düsentrieb
gut	Meier
gut	Klausen
sehr gut	Müller
sehr gut	Menzer
unbekannt	

## RIGHT JOIN

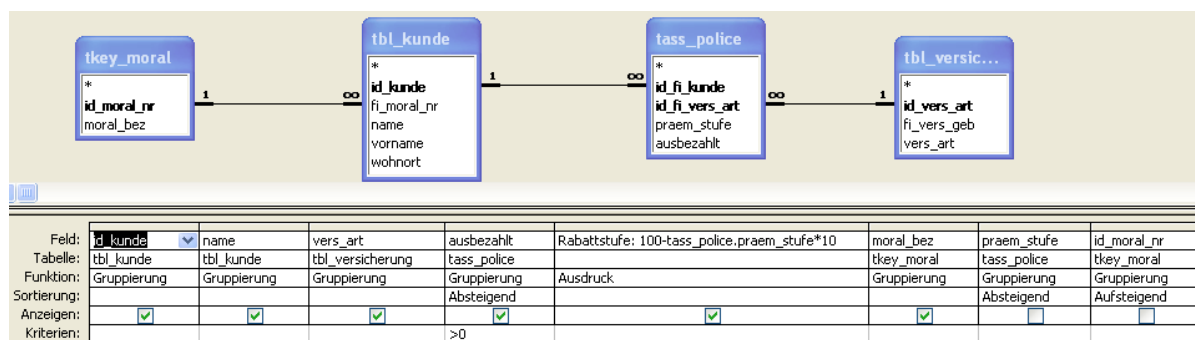


Wir haben einen Right Outer Join produziert: Zeige mir alle Tupel aus der rechts genannten Tabelle, auch wenn in der links genannten Tabelle keine Entsprechung vorhanden ist.

Resultat:

Zahlungsmoral	Kundenname
	Pellegrini
schlecht	Hilton
schlecht	Hauser
schlecht	Düsentrieb
gut	Meier
gut	Klausen
sehr gut	Müller
sehr gut	Menzer

## 7.13 Abfragen aus mehreren Tabellen



**Skript-Übungen im Anhang: 40 – 55**

<b>A</b>		Funktion Access	26
		Funktionale Abhängigkeit	21
Abfragen Access	25	<b>G</b>	
Abfragen aus mehreren Tabellen	34	Generalisierung	11
ACCESS	22	<b>I</b>	
Aggregation	12	Indizes	23
Aktualisierungsweitergabe Access	25	Informationsmodel	7
Alphanumerische Typen	23	INNER JOIN Access	32
Anzeigen Access	26	IN-Operator Access	28
Assoziationen	7	Interne Ebene	5
Attribute	9, 16	<b>K</b>	
Ausdrücke (Bsp) Access	27	Kardinalität	8
<b>B</b>		Klassen	11
Berechnungen Access	30	Komposition	12
Beziehungen Access	24	Konzeptuelle Ebene	4
Beziehungstypen	14	Kriterien Access	27
<b>C</b>		<b>L</b>	
Client und Server	5	LEFT JOIN Access	33
CROSS JOIN Access	32	Löschanomalie	3
<b>D</b>		Löschweitergabe Access	25
Datenbankarchitektur	4	<b>M</b>	
Datenbasis	22	Mehrere Kriterien Access	29
Datentypen	23	Modellierungsregeln	9
Disjunkt	11	Multiplizität	8
Domänen	16	Mutationsanomalie	3
Dritte Normalform	21	<b>N</b>	
<b>E</b>		Numerische Typen	23
Ein Attribut	22	<b>O</b>	
Eine Datenbank	22	Operatoren(Bsp) Access	27
Eine Tabelle	22	<b>P</b>	
Einfügeanomalie	3	Pragmatik	6
Entitäten und Entitätstypen	7	Primary Key	14
ERD	11		
ERM	7		
Erste Normalform	19		
Externe Ebene	4		
<b>F</b>			
Feld Access	26		
Felder erstellen Access	31		
Fremdschlüssel	16		

---

## ***Q***

Query By Example 26

---

## ***R***

RDBMS 22  
Referenzielle Integrität 16  
Referenzielle Integrität Access 25  
RIGHT JOIN Access 33

---

## ***S***

Semantik 6  
Sortierung Access 26  
Subtyp 11  
Supertyp 11  
Syntax 6  
Systemkatalog 22

---

## ***T***

Tabelle Access 26  
Tass 18  
Tbl 17  
Tkey 17  
Tupel 14  
Tupelintegrität 14

---

## ***U***

UML-Klassendiagramm 10  
Unterabfragen Access 30

---

## ***W***

WIE-Operator Access 28

---

## ***Z***

Zweite Normalform 20