Data modeling

Wiederholung: Software Engineering

Baupläne (Architektur) und Entwurfsregeln (Architekturkonzept)
Vorsicht: Statt *Architekturkonzept* sagt man oft mit *Architektur*!

Hausbau **Regeln für Baupläne**	**Produktentwicklung** **allgemein**	**Wirtschafts-Informatik** **Regeln für IS-Entwicklg.**
Wie **entwirft** man Pläne?	**Modellierungs-Methoden**	Phasenkonzepte (software process) und Entwurfsebenen
Welche dem Bauablauf entspr. **Verfeinerungen** braucht man?	**Modell-Ebenen**	
Welche **Ansichten** braucht man?	**Modell-Sichten** **Modell-Aspekte**	Modellaspekt-Matrix
Wie **zeichnet** man Pläne?	**Modell-Notationen**	

Wiederholung: Multiperspektivität

	Static models	**Dynamic models**
Data models	**Data (structure) models:** data structure diagrams; entity-relationship models (ERM); UML class diagrams	**Information flow models:** information / data flow charts / diagrams; Structured Analysis (SA); UML use case diagrams
Function models	**Function structure models:** compositional function trees; Jackson trees	**Behavior / process models:** algorithms (functions); Nassi-Shneiderman diagrams; (control) flow charts; business process models; UML activity diagrams; (UML sequence diagrams)

Wiederholung: Phasenkonzepte und Entwurfsebenen

Main phase	**Subphase, model level**	**Methods (ex.)**
Analytic phase: problem analysis	Elicitation of the current state of the org	Systems analysis, Reverse engineering
	Analysis of the current state of the organization	Requirements engineering, OOA
	Design of the planned state of the org. (sociotechnical IS)	Requirements engineering, BPM
	Design of the business concept of the IT system (technical IS)	Reference mod., test case description
Synthetic phase: IT system develop-ment	Design of the techn. concept of the IT system independ. of developm. tool	OO design, design patterns
	… depending on development tool	Unit tests
	Programming	Coding conv., agile programming
	Test	V model tests

Data modeling – Model levels

Auf den unmittelbar vorangehenden Folien geht es um eine Wiederholung aus Software Engineering, aus dem Sie die Termini "analytische Phase" und "synthetische Phase" brauchen.

Bemerkung: Im Folgenden werden wir von der analytischen und synthetischen Datenmodellierung sprechen. Die beiden Termini haben nichts mit der analytischen und synthetischen Phase im Software Engineering zu tun!

Für Ihr Verständnis von Datenmodellen ist sehr wichtig, dass es nicht einfach nur Datenmodelle gibt, sondern
Datenmodelle auf unterschiedlichen Modellebenen mit unterschiedlichen Modellzwecken.
Diese Modelle können sehr verschieden aussehen, da es abhängig vom Modellzweck unterschiedliche Modellierungsziele und Modellierungskriterien gibt.

Wenn Sie mit irgendwelchen Internetquellen oder Lehrbüchern arbeiten, dann wird der Unterschied zwischen verschiedenen Modellzwecken häufig nicht gemacht, und es entstehen furchtbare Modelle, die gleichzeitig verschiedene Modellzwecke erfüllen sollen, was natürlich nicht geht.

Im Folgenden unterscheiden wir konzeptuelle und logische Datenmodelle.
Wir fokussieren uns auf konzeptuelle Datenmodelle.

Data modeling – Model levels

Introductory example: customers → orders → order lines ← products

1 Conceptual model / Information model

(information-relevant level – **analytic phase**)
Goal: understandable for domain experts ("users"), easy to modify,
mathematically optimized, IT independent,
stable basis for a logical / implementation model

controlled redundancy (no modification anomalies):
– **basic attributes** only (no reconstruction possible if deleted)
– 3NF with foreign keys

primary data, domain data

This is the focus of our DB course.

2 Logical model / Implementation model

(implementation-relevant level – **synthetic phase**)
Goal: performance optimization for reports in an application area

uncontrolled redundancy (danger of inconsistency):
– **derived attributes** (in addition to, derived from basic attributes)
  data aggregation (Datenverdichtung)
– horizontal and vertical fragmentation
– denormalization (parts in 1NF or 2 NF, conditional relationships)

secondary data (in addition to primary data), administration data: **indexes**

The optimization methods often depend on the particularities of an individual DBMS which you best learn in a dedicated training.

This is not the focus of our DB course.

3 Internal model / Speichermodell

The form how DBMS store data (B trees etc.), not readable for humans

Data modeling – The two basic data modeling approaches

In data models, there are only two model layers: sets and attributes

This leads to two modeling approaches (not concurrent):
1 **synthetic method**: sets → attributes
First, you intuitively design tables and then, you look for suitable attributes.

2 **analytic method**: attributes → sets (normalization → 3NF)
You start with a list of attributes (UNF) and arrange them step by step in tables. This is called normalization.

The use of the two methods is not concurrent, but comparative with a mutual check of the results.

Bemerkung: Die Termini analytische und synthetische Datenmodellierung haben nichts mit der analytischen und synthetischen Phase im Software Engineering zu tun!

Data modeling – Atomization of attributes

In order to be as flexible as possible for reports, decompose the attributes into minimal meaningful units (kleinste bedeutungstragende Einheiten) from the very beginning – before starting normalization!
For this purpose, you have to use the knowledge about an application area.

This process is called **atomization** of attributes

(The word *atomicity* is also used in the context of transactions; next slide.)

Manche Internetquellen verwenden in völlig irreführender Weise das Wort Atomarität/Atomizität auch bei der 1NF. Dort sprechen wir aber eindeutig von Wiederholgruppen. Wiederholgruppen haben mit der Zerlegung von Attributen (bspw. Adresse) nicht das Geringste zu tun.

Example:
Decompose address
into country code, postal code, place, street, house number

Data modeling – Remark on transactions in relational databases

ACID principle

Atomicity: execution of transactions “either all or nothing”

Consistency: a transaction transforms a valid state into another valid state

Isolation: independency of transactions from one another
leads – in spite of parallel execution – to the same result
which would be obtained using serial execution

Durability: permanent storing of the results of transactions even in the case of power loss and crashes (non-volatile memory)

1 Mathematical modeling of sets (nodes)

Primary key

A primary key is an attribute (simple; einfacher Primärschlüssel) or
a group of attributes (compound; zusammengesetzter Primärschlüssel)
whose values uniquely identify the lines (entities) of a table (entity type).
This feature is called entity integrity.

The description of the primary key has to fit the description of the entity type.
(Die Namen der Primärschlüsselattribute müssen zum Namen der zugehörigen Tabelle passen.)

Bemerkung: Primärschlüssel sind heutzutage alphanumerisch.
Man verwendet leicht merkbare mnemotechnische Kürzel, z.B. für die Fakultäten der TH: IN, BW
Numerische Primärschlüssel (autoincrement integer) nur in folgenden Fällen:

- Lückenlos aufsteigender Nummernkreis (z.B. Rechnungs-, Buchungs-Nr.)
- Zur Bestimmung einer Prüfziffer (z.B. mod 11)
- Artifizieller Zähler (z.B. Statistik-Daten)

1 Mathematical modeling of sets (nodes)

Candidate key (Schlüsselkandidat)

A candidate key is an attribute (simple) or a group of attributes (compound) whose values uniquely identify the lines (entities) of a table (entity type). That is, candidate keys can work like primary keys.

For each table of a model, you have to choose one unique primary key from all of the possible candidate keys.

In everyday life, candidate keys don't play a big role.

Example: An invoice position table has three candidate keys:
PositionNr
InvoiceNr, ItemID
InvoiceNr, LineNr

1 Mathematical modeling of sets (nodes)

Relational databases are called relational as the tables are described in the form of *n*-dimensional mathematical **relations**,
and not – as you may read here and there – as they focus on relation(ship)s between tables.

The following concepts are only explained in class.

Mathematical concept of a relation

Subset of the cross product of 2 or more factor sets

2-dimensional in contrast to the concept of a function

n-dimensional

(EE_Slides_9_Meta, **Nodes**)

1 Mathematical modeling of sets (nodes)

There are four database terminologies depending on four different backgrounds.
In everyday life, you often switch between these four terminologies
so that you have to keep the entire terminological table in mind.

File management	Entities	Database tables	Algebra
File	Entity set	Table	Relation
File structure	Entity structure	Table structure	Relation structure
Data record	Entity	Row, line	Tuple
Data item	Attribute	Column	Factor (set)
Data item value	Attribute value	Elementary / cell value	Component value
Number of data items	Number of attributes	Number of columns	Degree (number of factors)
Number of data records	Number of entities	Number of rows	Cardinality (number of tuples)

2 Mathematical modeling of relationships (arcs) between sets

2.1 General relation of degree 2: many-to-many relationship

Example:
A Student can attend many courses.
A course can be attended by many students.

In the following, we will concentrate on two goals:

1 eliminate many-to-many relationships, focus on **functional relationships**
Eine n:m-Beziehung (relational) wird durch zwei funktionale Beziehungen und eine Zuordnungstabelle aufgelöst.

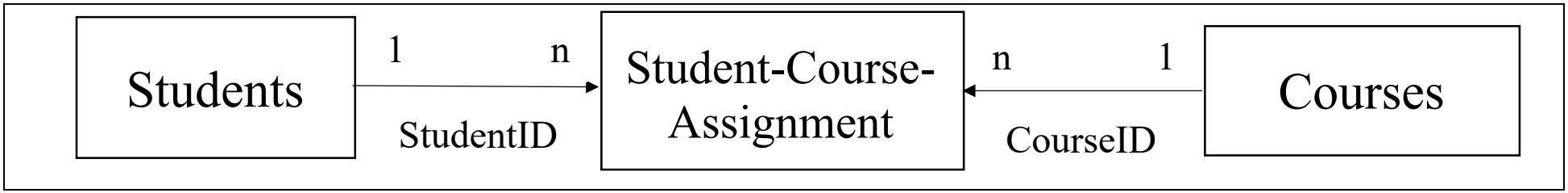
2 eliminate injective relationships, focus on **one-to-many relationships**
1:1-Beziehungen und 1:c-Beziehungen werden auf konzeptueller Ebene in jeweils einer Tabelle zusammengefasst.
Schauen Sie sich dazu auch meinen Foliensatz "Meta-models of information systems modeling approaches an", S. 12-14.

2.2 Transition to functional relationships: one-to-many relationship

A many-to-many relationship can always be replaced by two one-to-many relationships and an assignment table.

This should always be done in conceptual models.

Example (continued from 2.1):



PK: StudentID	PK: StudentID, CourseID FK: StudentID FK: CourseID	PK: CourseID

Note: An attribute can be part of a PK and a FK at the same time!

2.3 Keys

Primary key (entity integrity)
Foreign key / reference key (referential integrity)
A reference key is an attribute (simple) or a group of attributes (compound) which is a primary key in another table.

A foreign key always appears in the table on the ‘many’-side
of a one-to-many relationship.
The corresponding primary key is in the table on the ‘one’-side.

Note:
– an attribute can be part of a PK and a FK at the same time!
– there can be several FKs in one single table
– there can be compound FKs

2.4 Types of functional relationships

1. Let $y \in \mathbf{W}$.

Then there are three types of instance relationships:
(1) There is no $x \in \mathbf{D}$: $f(x) = y$ [customer without order]
(2) There is one $x \in \mathbf{D}$: $f(x) = y$ [customer with one order]
(3) There is more than one $x \in \mathbf{D}$: $f(x) = y$ [customer with several orders]

f: **D** (orders) → **W** (customers) (function)
orders ← customers (many-to-one relationship)

There are no further types!

2. Expansion to the entire domain

2.1 There is at least one instance relationship of type (3): non-injective

The following cases do not contain any inst. relationship of type (3).

2.2 There are only instance relationships of type (1): **D** = {} not interesting

2.3 There are only instance relationships of type (2): one-to-one relationsh.

2.4 There are only instance relationships of types (1) and (2): injective
Conditional relationship; 1:c with c ε {0; 1}
Ex.: customer (one invoice address) has either a different delivery address or not.

Functional relationships between two entity types can be classified as
1) one-to-many relationships (non injective)
2) one-to-one relationships (bijective)
3) conditional relationships (1:c with c ε {0; 1})

This classification is complete.

Relationships have to be verbalized in two directions.
Example: one-to-many relationship between customers and orders
1 in the 'many'-direction: One customer can have (0, 1 or) many orders
2 in the 'one'-direction: One order belongs to exactly one customer
(referential integrity)

Attention: one-to-many includes the case 0 on data record level,
e.g. a (new) customer without any order

2.5 Elimination of injective relationships

One-to-one and conditional relationships are injective (do not carry any additional information):
coincide in one database table.

Only one-to-many are not injective (carry additional information).

**In the field of conceptual data modeling,
only one-to-many relationships have to be considered.**

Modellierung der Referenzen bei 1:1- und 1:c-Beziehungen

Weil man diese beiden Beziehungstypen bei konzeptuellen Modellen praktisch ganz ausschließt, stellt sich dieses Problem nur bei logischen Modellen, die nicht Thema dieser Vorlesung sind.

Deshalb nur kurz:

1. Fall: Fragmentiert man eine DB-Tabelle vertikal, so haben die beiden neuen Tabellen, die ja zum gleichen konzeptuellen Entitätstyp gehören, das/ie gleiche/n Primärschlüssel-Attribut/e.
Dieser Fall ist trivial. Man braucht keine Fremdschlüssel.

Beispiel 1: Die Tabelle Kunden wird geteilt in Adressdaten und Finanzdaten, normalerweise 1:1.
(1:c kann vorkommen, wenn man von einem Kunden noch keine Finanzdaten kennt.)

2. Fall: Man will zwei verschiedene konzeptuelle Entitätstypen 1:1 oder 1:c miteinander verbinden.
Wie bei 1:n-Beziehungen (dort werden die zu einem PK-Wert gehörigen n-seitigen Entitäten über einen Index über den FK der n-seitigen Tabelle gefunden) genügt grundsätzlich ein einziger Fremdschlüssel in eine 1-Richtung.

Beispiel 2: Rechnungen und Mahnungen sollen 1:c verbunden werden. Dann ergänzt man in der c-seitigen Tabelle (Mahnungen) einen Fremdschlüssel, der auf die jeweilige Rechnung verweist.
Man kann die 1:c-Beziehung aber auch schon konzeptuell zu einer 1:n-Beziehung expandieren, wenn man zu einer Rechnung mehrere Mahnungen zulassen will.

Beispiel 3: Man will monogame, heterosexuelle menschliche Paare 1:1 modellieren, Entitätstypen Frauen und Männer.
Auch hier genügt (wie bei 1:n) ein einziger Fremdschlüssel in eine der beiden Richtungen.
Diese 1:1-Abhängigkeit könnte man übrigens auch zu n:m expandieren, wenn man menschliche Beziehungshistorien und polygame Beziehungen mit erfassen will.

Im Prinzip behandelt man im Datenmodell 1:1 und 1:c wie 1:n.
Das könnte aber bei Anwenderfehlern zu Problemen führen:
Sei A die Tabelle auf der „linken“ Seite der Beziehung. Dann könnte fälschlicherweise in Tabelle B bei verschiedenen B_IDs die gleiche A_ID stehen.
Das kann man durch wechselseitige Fremdschlüssel abfangen, die gleichzeitig die Funktion von alternativen Primärschlüsseln haben. Dadurch kann man Anwenderfehler, die das Verhältnis 1:1 oder 1:c zerstören würden, bereits bei der Eingabe verhindern.

2.6 Classification of types of relationships [meta-models]

1 numeric (cardinality)
2 syntactic (reference keys)
3 semantic (compositional, taxonomic)

2.7 Treatment of selected relational relationships

c:n → use a dummy on the ‘c’-side, then you get 1:n

c:c → different, depending on application; maybe expand to n:m

2.8 DBA-defined attribute descriptions

Example:
Goal: unique attribute descriptions in the entire data model
Use: table abbreviation plus semantic attribute description

Client_ID
In table Client: C_Client_ID
In table Orders: O_Client_ID

2.9 Notations for data models

Meine eigenen Notationen:
– PK durch Unterstreichen, FK durch ein Sternchen *
– 1:n-Beziehung durch Pfeil in n-Richtung mit Fremdschlüssel
– 1:n-Beziehung schließt auf Datensatzebene den Fall 0 mit ein

Wenn Sie die Notationsweise FK für Fremdschlüssel verwenden:
– unterscheiden Sie verschiedene FK in einer Tabelle durch FK1, FK2 etc.

The following notations are only explained in class.
Entity-Relationship Models
min max notation
Oracle diagrams
Krähenfuß-Notation
(EE_Slides_9_Meta, **Arcs**)

3 Normalization: Motivation

1 Bei fester UNF-Attributliste liefert die Normalisierung ein konzeptuelles Datenmodell (3NF) (vgl. oben Folien „Model levels“),
das mathematisch optimiert ist
das nur Basisattribute enthält
das von den Eigenheiten einzelner Modellentwickler unabhängig ist
das von der Wahl des UNF-Primärschlüssels unabhängig ist
(1/2NF können sich je nach UNF-Primärschlüssel unterscheiden)
das leicht änderbar ist
das eine stabile Basis für darauf aufbauende logische Datenmodelle bildet
das ein ähnliches Ergebnis wie intuitive synthetische Modellierung liefert

2 Analytische Datenmodellierung (Normalisierung) und synthetische Datenmodellierung ergänzen sich gegenseitig und werden in der Praxis kombiniert verwendet
(vgl. oben Folie „The two basic data modeling approaches“).

3 Normalization: Conditions starting from UNF

Die drei Normalisierungsbedingungen lauten:
- keine Wiederholgruppen (1NF)
- keine Schlüsselteil-Abhängigkeiten (2NF)
- keine transitiven Abhängigkeiten (3NF)

UNF-Attribute:

Die Liste der UNF-Attribute darf nur atomare Basisattribute enthalten (zerlegt in kleinste bedeutungstragende Einheiten).
Diese Liste darf während der Durchführung des Normalisierungskalküls keine Änderungen erfahren.
Es dürfen keine Attribute hinzugefügt (außer Fremdschlüssel bei neuen Tabellen), weggelassen oder zerlegt werden.

3 Normalization: Problems 1

1 Parallele Abhängigkeiten:
Bei jedem der drei Normalisierungsschritte werden Gruppen von Attributen, die die jeweils geprüfte Normalisierungsbedingung nicht erfüllen, gleichzeitig in neue Tabellen ausgeklammert.

2 Geschachtelte (nested) Abhängigkeiten:
Bei jedem der drei Normalisierungsschritte sind die neu entstandenen Tabellen nochmal auf die gerade geprüfte Normalisierungsbedingung zu prüfen und daraus ggf. weitere Tabellen auszuklammern.
Geschachtelte Abhängigkeiten werden in mehreren Schritten aufgelöst.
Für jede Schachtelungsebene entsteht ein neues Datenmodell.

3 Normalization: Problems 2

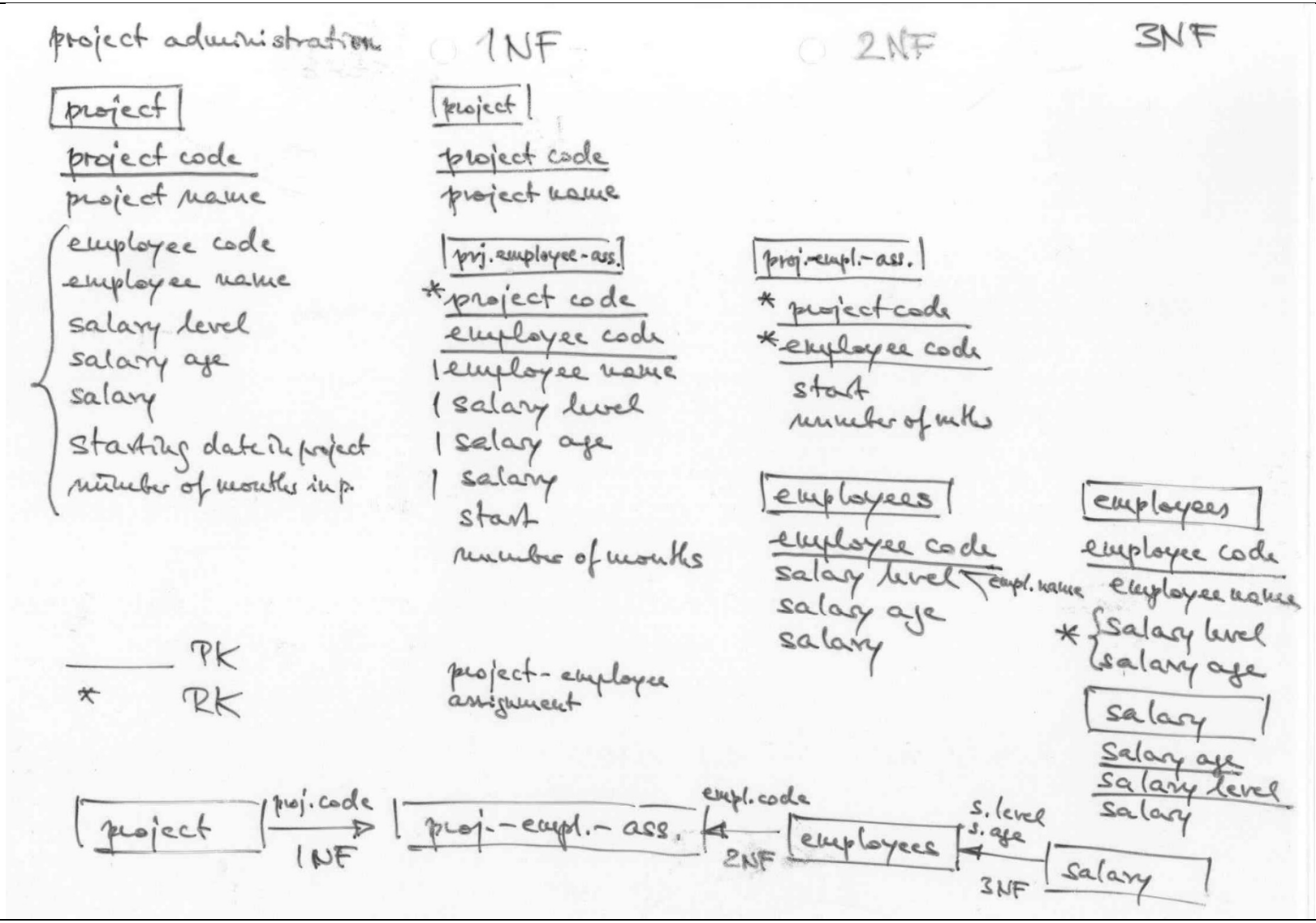
3 Hat man ausgehend von einer UNF die drei Normalisierungsschritte ausgeführt, so ist das Ergebnis nicht unbedingt schon ein 3NF-Modell. Ein Modell liegt erst dann in 3NF vor, wenn alle Tabellen in 3NF sind. Das erkennt man durch Überprüfung der drei Normalisierungsbedingungen für jede einzelne Tabelle.
Ist das entstandene Modell noch nicht in 3NF, beginnt man in einem zweiten Durchlauf von vorne mit dem ersten Normalisierungsschritt.

4 Das Ergebnis des Normalisierungskalküls ist immer ein wegeindeutiges Modell. Das entspricht häufig nicht der Wirklichkeit in Organisationen, so dass ggf. Beziehungen und Fremdschlüssel in einer manuellen Optimierung ergänzt werden.

5 Normalisierung ist nur für kleinere Modelle (10 Tabellen) gut geeignet. Größere Modelle zerlegt man in Teilmodelle, normalisiert diese und integriert sie zu einem Gesamtmodell.

3 Normalization: Example – Project administration

(only tables with changes are mentioned)



3 Normalization 0

UNF

List of attributes (atomized); need not be structured

Information-relevant (basic) attributes
Not: implementation-relevant (derived) attributes

Choose “suitable” one (or more) attribute(s) as UNF key
Note: different suitable UNF keys lead to the same 3NF.

Choose a name for the UNF table
Note: UNF key name has to fit UNF table name.

Result of mathematical normalization in 3NF:
unique-path graph

3 Normalization 1

1NF

Find repeating groups (Wiederholgruppen) of attributes.
  Choose a fix value of the UNF key and
  check whether the other attributes can have more than one value.
Remove them to subordinate tables.

Copy UNF key to subordinate tables as foreign key.
Determinate suitable primary keys of subordinate tables.

Note: new table in the direction of a one-to-many arrow

Es sind solche Attribute auszuklammern, die in Bezug auf einen festgehaltenen Wert (!!!)
des UNF-Schlüssels verschiedene Werte annehmen können.
Manche Internetquellen verwenden in völlig irreführender Weise das Wort Atomarität/Atomizität
(Zerlegung der Attribute in kleinste bedeutungstragende Einheiten) auch bei der 1NF.
Wiederholgruppen haben mit der Zerlegung von Attributen aber nicht das Geringste zu tun.

3 Normalization 1

Exceptions

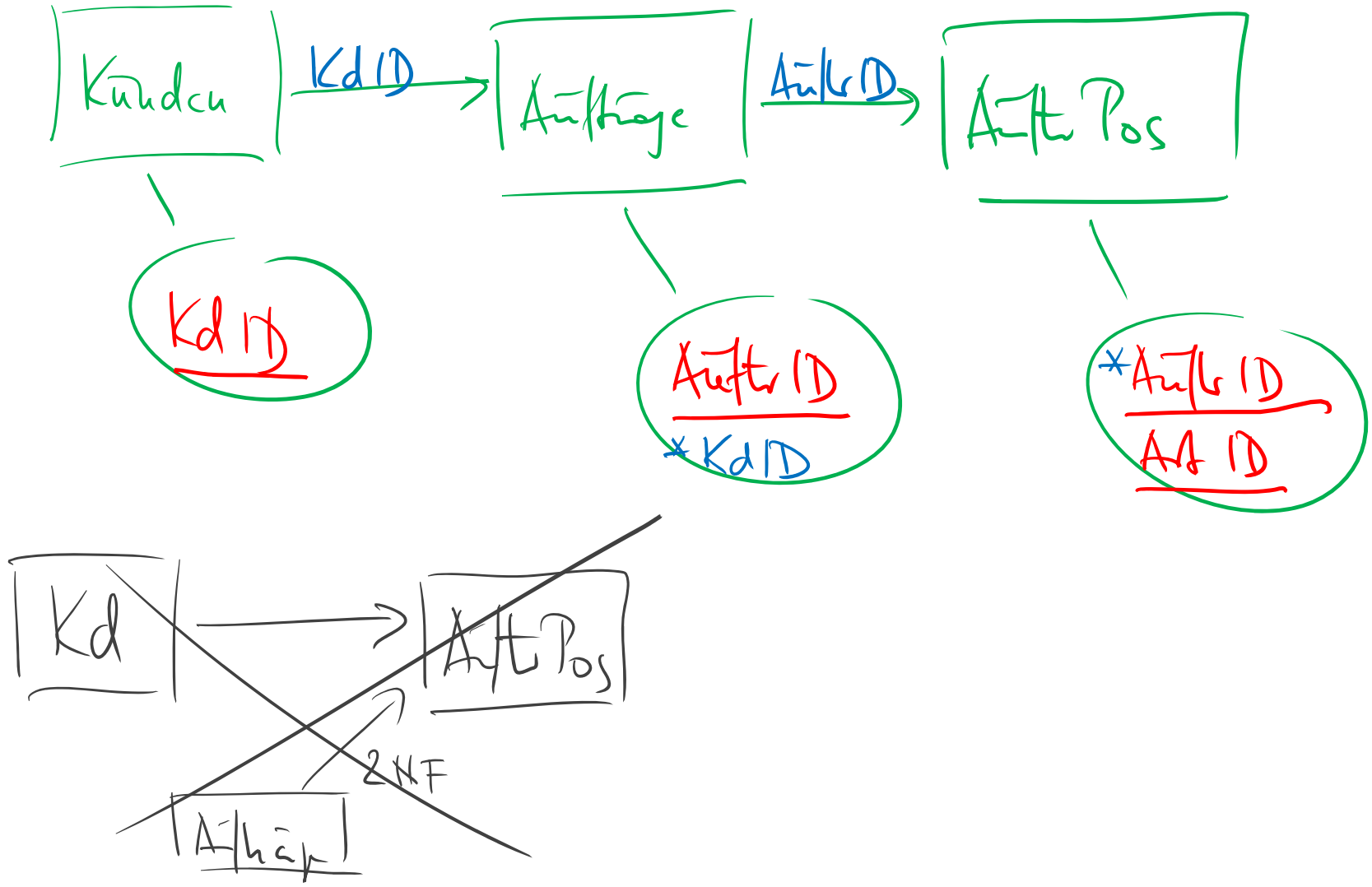
1 Nested (geschachtelt) repeating groups
isolation in at least two steps

2 Parallel repeating groups
simultaneous isolation

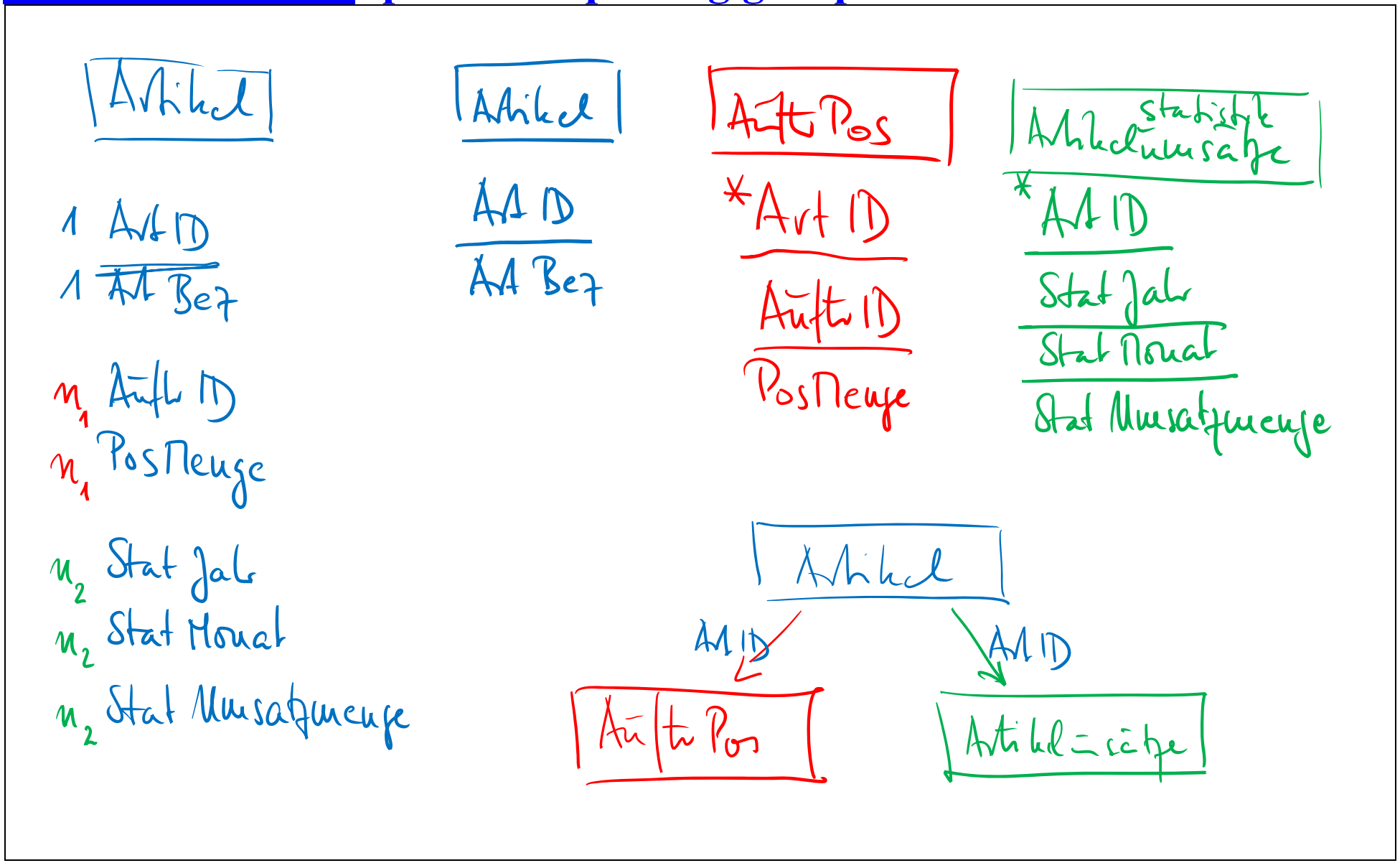
3 Dangerous thumb / heuristic rule:
Determination of primary key in 1NF:
Duplicated UNF key plus one or more attributes
Note: this rule does not apply in the case of functional (one-to-many) interdependencies between parts of the primary key.

Nested and parallel dependencies can also occur in the derivation of 2/3NF.

3 Normalization 1: nested repeating groups (also example for exception 3)



3 Normalization 1: parallel repeating groups



3 Normalization 2

2NF

Isolate attributes which functionally depend on one part of the primary key only (Schlüsselteil-Abhängigkeiten).

New table with the corresponding part of the original primary key as new primary key

Note: consider tables with compound primary keys only.

Note: new table against the direction of a one-to-many arrow

3 Normalization 2: nested functional dependencies

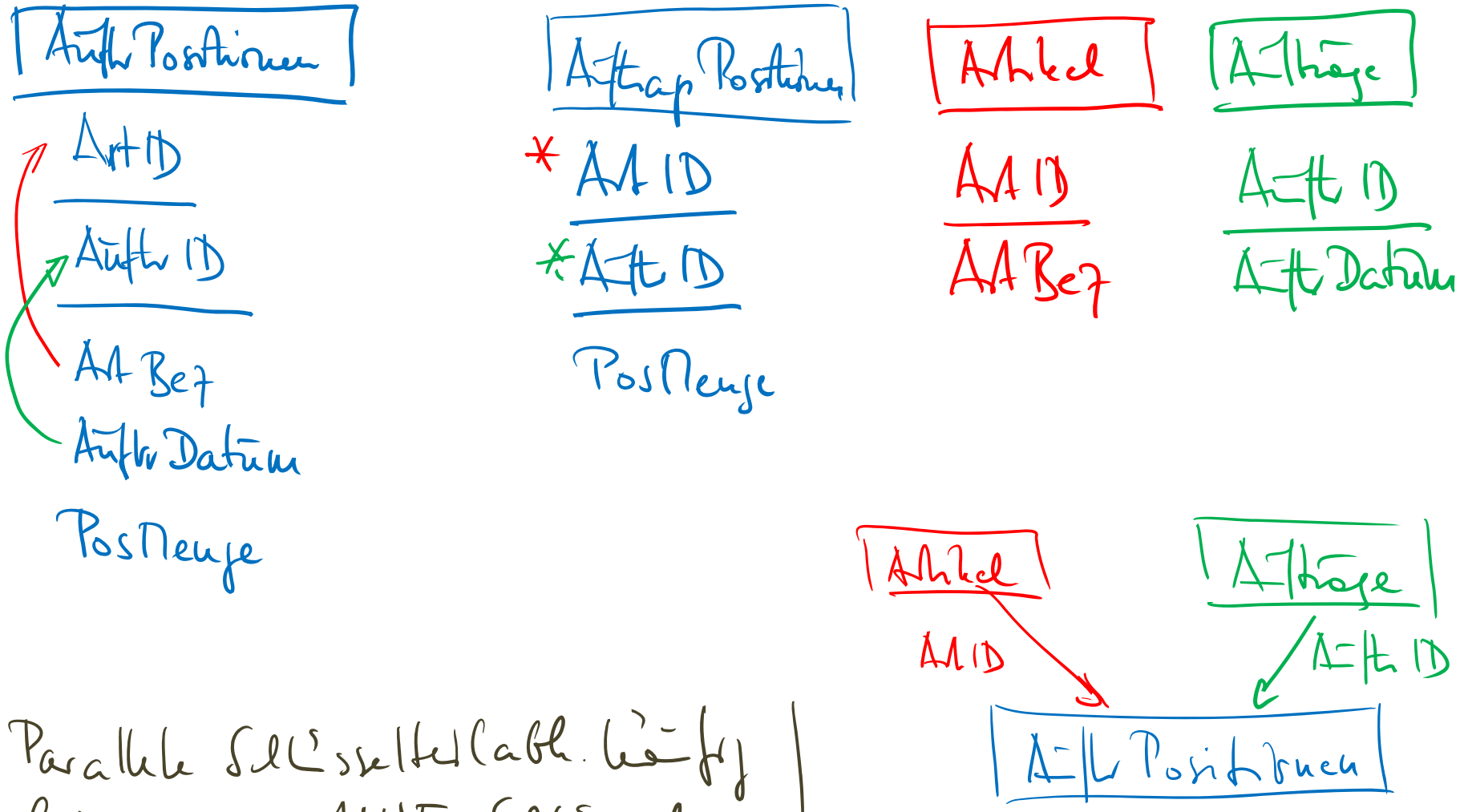
Nested dependencies only if the primary key consists of at least three parts

Auftragspositionen

KdID
KdName
AuftrLfdNr
AuftrDatum
ArtID
(ArtBez)
PosMenge

AuftrID = KdID + AuftrLfdNr

3 Normalization 2: parallel functional dependencies



3 Normalization 3

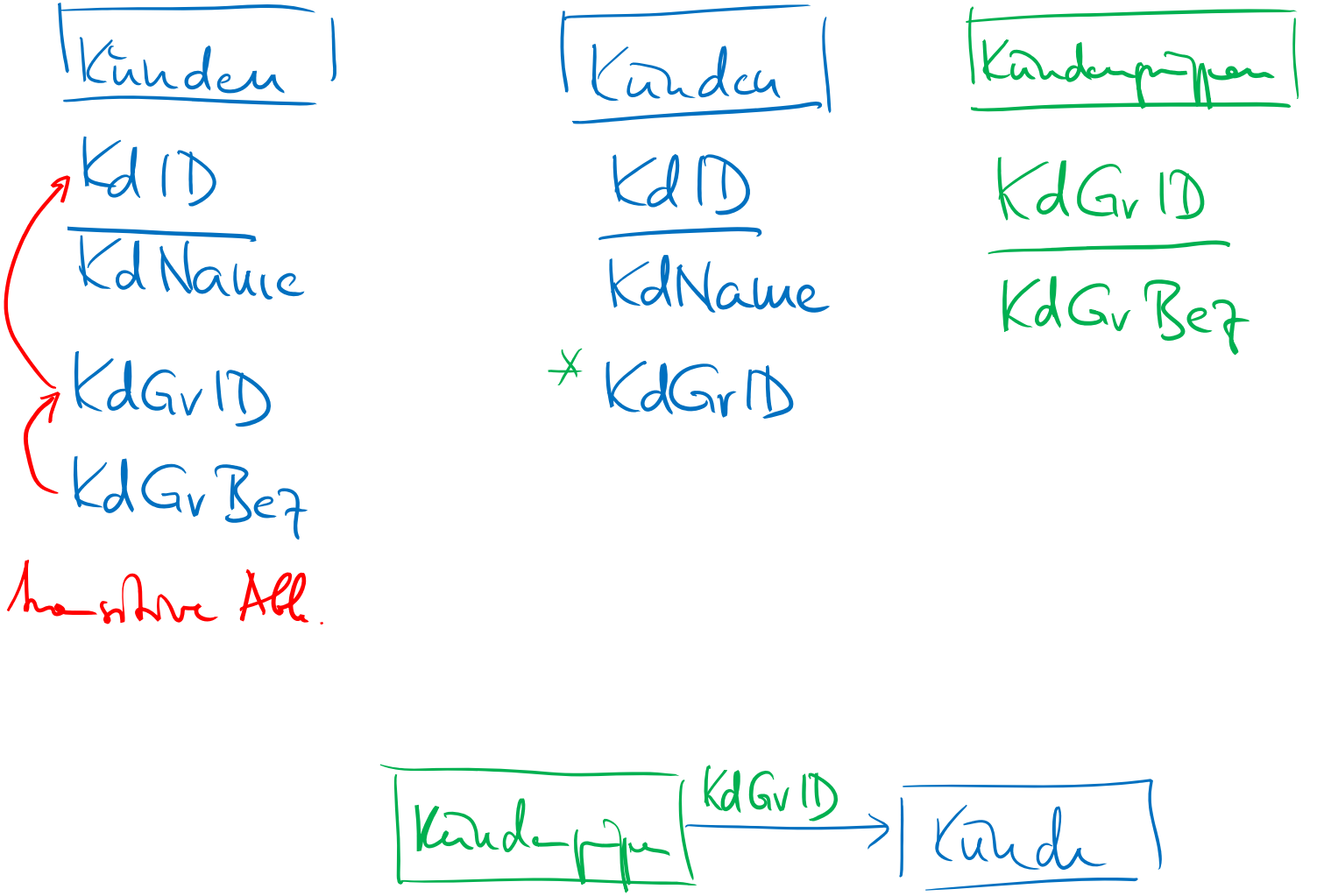
3NF

Isolate attribute(s) which do not directly depend on the primary key (or a part of it), but only via simple attributes.
Transitive dependency

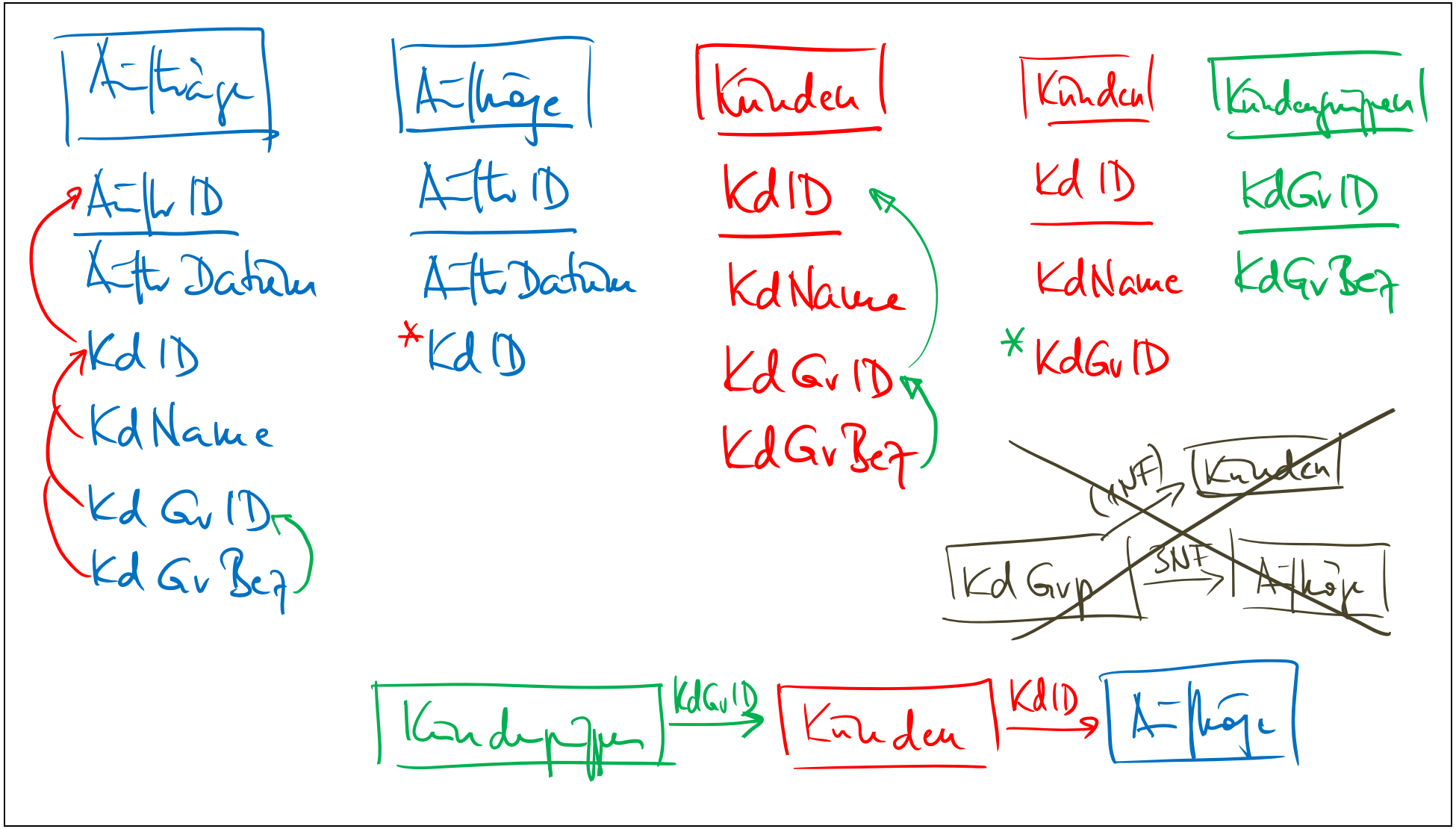
Note: new table against the direction of a one-to-many arrow

a ~ b and b ~ c => a ~ c

3 Normalization 3: transitive dependency



3 Normalization 3: nested transitive dependencies



3 Normalization 3: parallel transitive dependencies

Auftragspositionen

AuftrPosID
AuftrID
AuftrDatum
ArtID
ArtBez
PosMenge

AuftrDatum ist von AuftrPosID transitiv abhängig über AuftrID
ArtBez ist von AuftrPosID transitiv abhängig über ArtID

4 Normalization: particularities and first special cases

1 Stability of the UNF: no changes of attributes during normalization

2 Iteration of normalization

3 Result: unique-path graph

4 Ambiguity of decomposition:
most data models are not unique-path graphs (nicht wegeindeutig)
(order, delivery, order positions, delivery positions)
Then, several different ways of normalization are possible and correct.
Each of them leads to the same tables, but with different relationships.
– overlay different unique-path graphs (complicated)
– manually optimize one unique-path graph (quite simple)

4 First special cases

1 Transform a 2-dimensional table into a database table

Trivial: see students and courses in 2.2

2 Tree (bill of materials for one product)

Assume parent orientation towards product (root).
Each node without the root has got a parent node.

Data model:
Table ‘Nodes’ with a reflexive one-to-many relationship and
primary key: Node_ID
foreign key: Parent_node_ID

3 Network (bills of materials for several products)

Assume parent orientation towards the products
An arc is always defined by two nodes.

Tables ‘Nodes’ and ‘Arcs’ with two one-to-many relationships
and the foreign keys Parent_node_ID, Child_node_ID

Table ‘Nodes’
primary key: Node_ID
no foreign key

Table ‘Arcs’
primary key Parent_node_ID, Child_node_ID
foreign key 1: Parent_node_ID
foreign key 2: Child_node_ID

5 Reference models

Wenn man ein neues Modell für einen Anwendungsbereich entwickelt (egal ob Daten- oder Prozessmodell), gibt es dafür immer zwei Quellen:

– Beobachtungen und Interviews mit den Personen in diesem Anwendungsbereich

– Modelle, die man bereits kennt und die zu diesem Anwendungsbereich gewisse Ähnlichkeiten (Analogien) aufweisen.
Solche Modelle verwendet man als Referenzmodelle oder Vergleichsmodelle.
So kann ein Kunden-Auftragsverwaltungs-Modell als Referenzmodell für ein Lieferanten-Bestellverwaltungs-Modell dienen

Ein Modell ist also nicht schon von vornherein ein Referenzmodell, sondern wird erst durch seine Verwendung dazu.

Sind zwei Modelle ähnlich, so gibt es paarweise für jeden ähnlichen Entitätstyp einen Oberbegriff.
Beispiel: Kunde und Lieferant haben den Oberbegriff Geschäftspartner, Auftrag und Bestellung den Oberbegriff Vertrag.

Aus derartigen Oberbegriffen kann man ein generisches Modell ableiten, das seinerseits wieder als Referenzmodell dienen kann.

WICHTIG (Teilanalogien): Die Ähnlichkeiten (Analogien) zwischen einem Referenzmodell und einem neu zu entwickelnden Modell sind häufig nur teilweise (es muss nicht zu jedem Entitätstyp des Referenzmodells im neuen Modell einen analogen Entitätstyp geben und umgekehrt!!!), manchmal aber auch vollständig.

Ein in der Wirtschaftsinformatik verbreitetes generisches Referenzmodell, mit dem man viele Anwendungsbereiche zumindest teilweise abdecken kann, ist das folgende – immer aus der Sicht einer Organisation gedacht:

Geschäftspartnergruppen → Geschäftspartner →
→ Verträge → Vertragspositionen ←
← Vertragsgegenstände (physisch) ← Vertragsgegenstände (logisch) ← Vertragsgegenstandsgruppen

Die Grundgedanken bei diesem Typ Referenzmodellierung lauten zusammengefasst:

1. Es werden Vertragspartner zu Vertragsgegenständen in Beziehung gesetzt.
   Das geschieht über Verträge entweder mit genau einer oder mit mehreren Vertragspositionen.

2. Vertragspartner und Vertragsgegenstände können zu Gruppen zusammengefasst werden.

3. Falls Vertragsgegenstände als Exemplare individuell eindeutig identifizierbar sind, braucht man den Entitätstyp „physische Vertragsgegenstände“ (z.B.). Alle gleichartigen phys. Vertragsgegenstände fasst man zu einem „logischen Vertragsgegenstand“ zusammen. Ansonsten arbeitet man nur mit logischen Vertragsgegenständen.

Mit physischen Vertragsgegenständen meint man individuell eindeutig identifizierbare Vertragsgegenstände, z. B. Ausleihgegenstände, Buchexemplare einer Bibliothek mit Inventarisierungsnummer, Leihgeräte eines Vereins, Mietwagen, Gebrauchtwagen mit Fahrgestellnummer, große technische Anlagen.

Beispiel Bibliothek, Carsharing, Autovermietung, Verein
Ein log. Vertragsgegenstand ist etwa ein Buchtitel, zu dem eine Bibliothek mehrere Exemplare hat.
Eine Reservierung bezieht sich immer auf einen log. Vertragsgegenstand, weil es egal ist, welches Exemplar (phys. Vertragsgegenstand) von lauter gleichen Exemplaren Sie bekommen.
Die Ausleihe selbst bezieht sich dann auf einen phys. Vertragsgegenstand.
Bei einer Bibliothek muss man beide Ebenen von Vertragsgegenständen modellieren.

Beispiel Supermarkt / Baumarkt:
Wenn Sie eine Schachtel Schrauben kaufen, handelt es sich um einen n i c h t individuell identifizierbaren Vertragsgegenstand. Diese Schachteln tragen keine individuellen Codes.
Wenn Sie zwei Brote des gleichen Typs kaufen, dann sind sie nicht durch irgendeine ID unterscheidbar. Deshalb werden bei einem Baumarkt oder Supermarkt nur logische Vertragsgegenstände modelliert und keine physischen.

Beispiel Neuwagenkauf:
Sie bestellen einen Wagen eines bestimmten speziell konfigurierten Typs. Das ist ein log. Vertragsgegenstand.
Es werden aber mehrere Exemplare dieses Typs hergestellt. Sie bekommen eines davon mit einer bestimmten Fahrgestellnummer. Das ist dann ein phys. Vertragsgegenstand.
Die Bestellung bezieht sich also auf einen log. Vertragsgegenstand, die Rechnung auf einen phys. Vertragsgegenstand.

Beispiel Reisebüro:
Es gibt die Unterscheidung zwischen phys. Vertragsgegenstand (Durchführung einer Reise mit best. Programm zu einem best. Zeitpunkt) und log. Vertragsgegenstand (Reise mit best. Programm).

Vermietung: Ein Mietvertrag bezieht sich auf eine Wohnung als phys. Vertragsgegenstand.

5 Reference models

1. The two sources of model construction
– empiric: observation of an organization, interviews of the employees
– rationalistic: analogical transfer of reference models

2. Debtor model, creditor model, contract model

3. Examples of reference model structures in different business branches

4. Generic reference models and analogy based upon umbrella terms

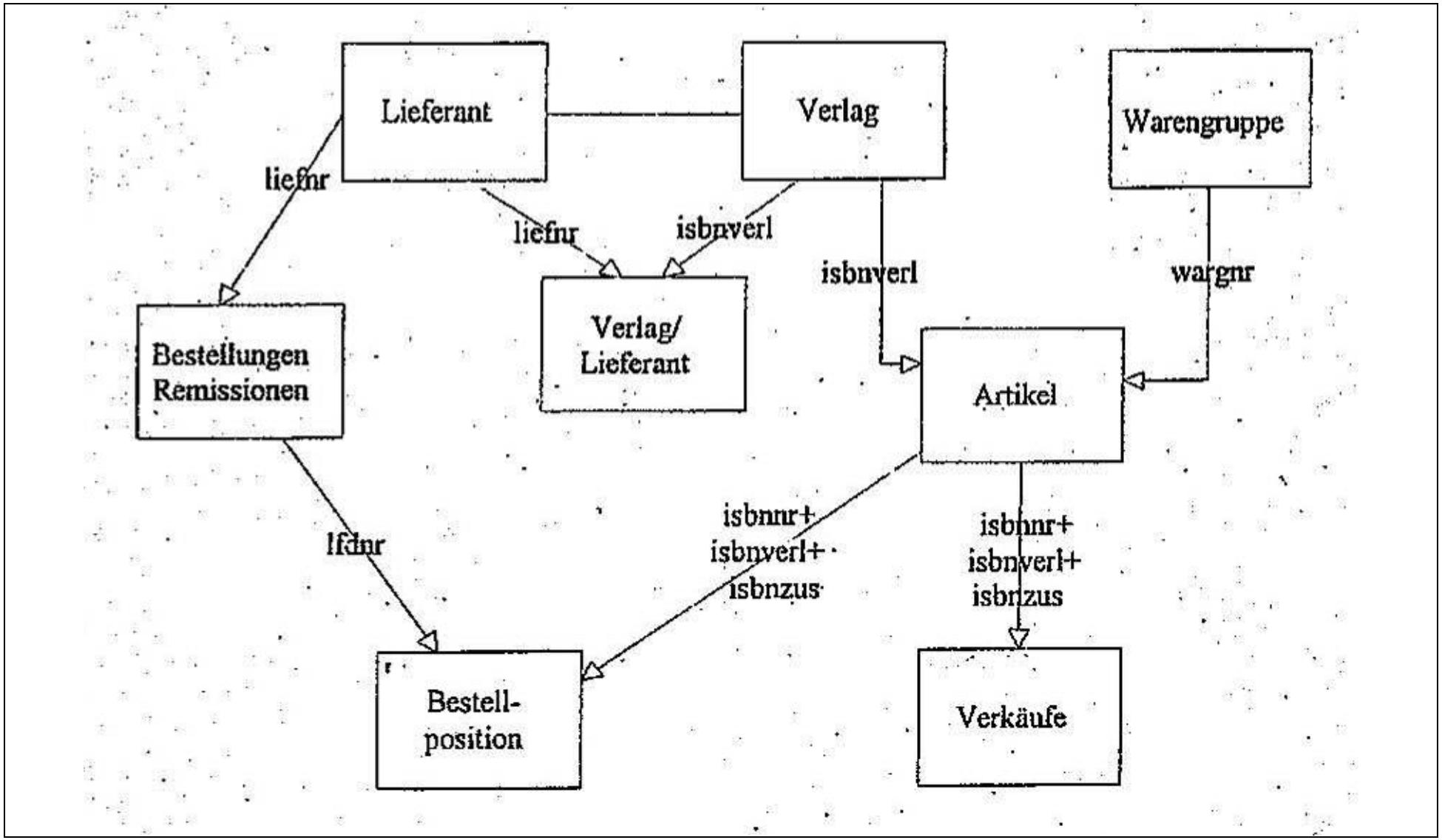
5. Partial analogies
– orders with one order position only
– individually identifiable items

(EE_Slides_7_Ratio)

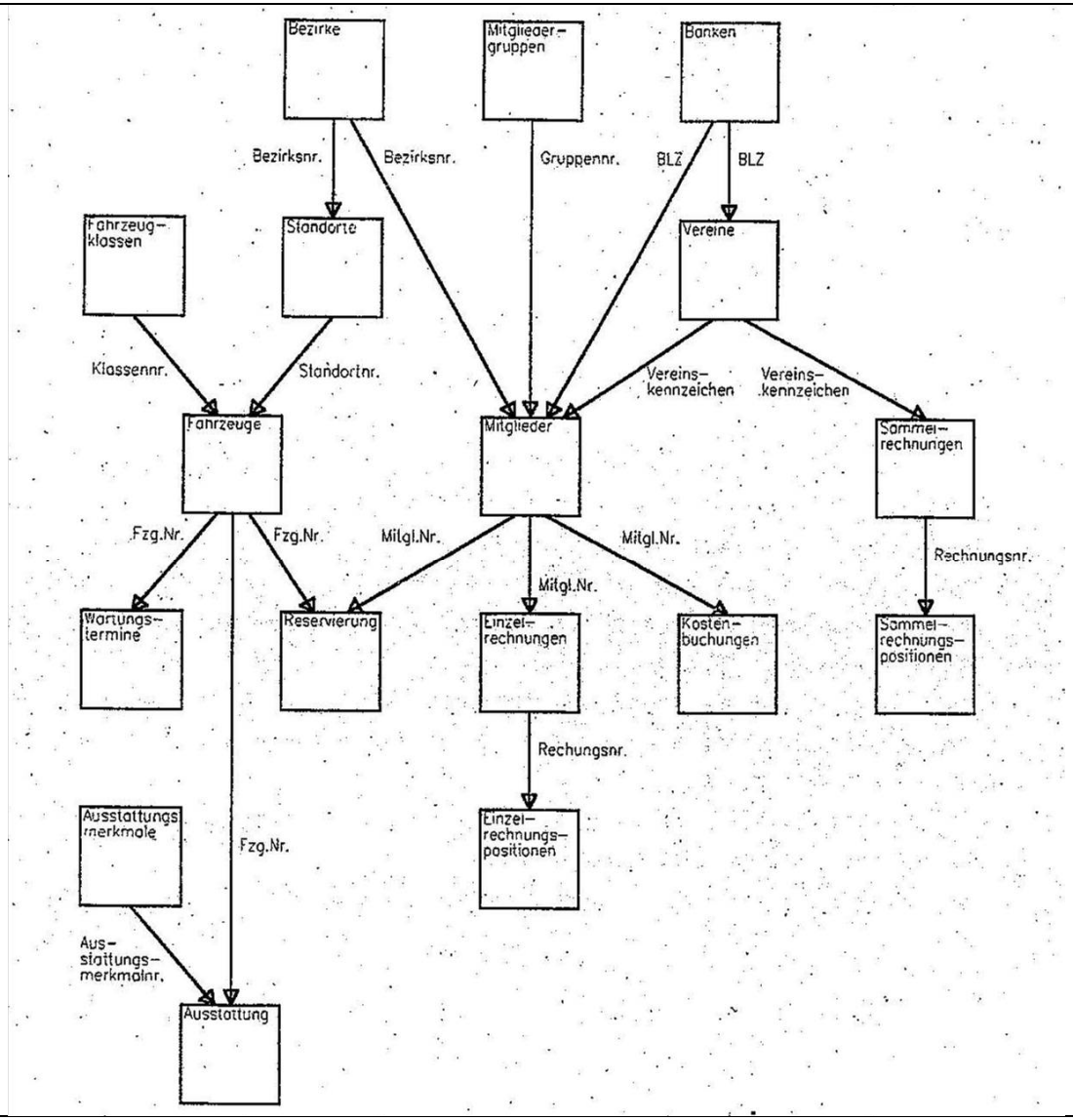
5 Reference models

creditor	debtor	umbrella terms generic model
supplier groups ↓ suppliers ↓ outgoing orders ↓ order lines ↑ raw materials ↑ material groups	customer groups ↓ customers ↓ incoming orders ↓ order lines ↑ products ↑ product groups	business partner groups ↓ business partners ↓ contracts ↓ contract positions ↑ contract items ↑ contract item groups

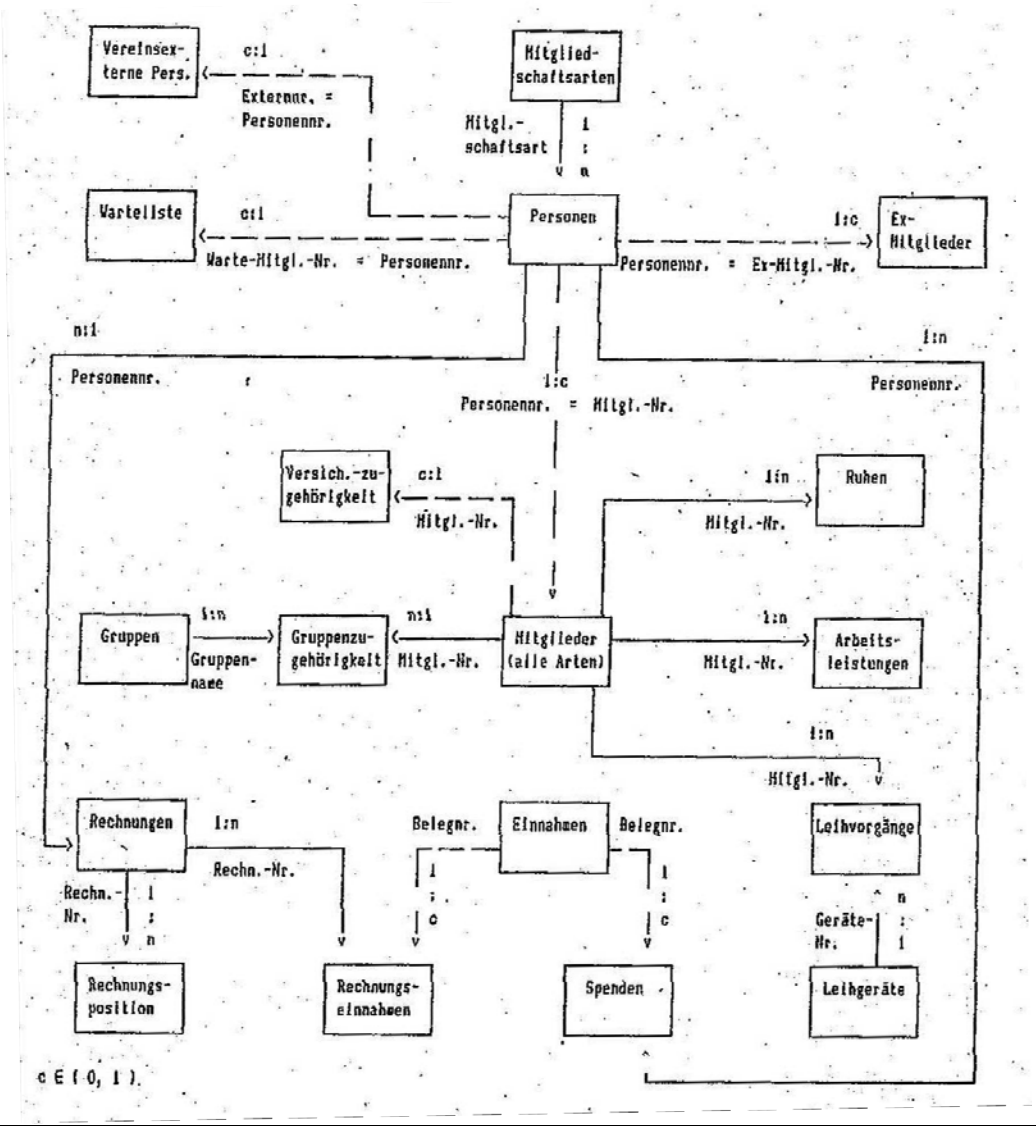
Comicladen


Lieferant
Verlag
Warengruppe
liefnr
liefnr
isbnverl
isbnverl
wargnr
Bestellungen
Remissionen
Verlag/
Lieferant
Artikel
lfdnr
isbnnr+
isbnverl+
isbnzus
isbnnr+
isbnverl+
isbnzus
Bestell-
position
Verkäufe

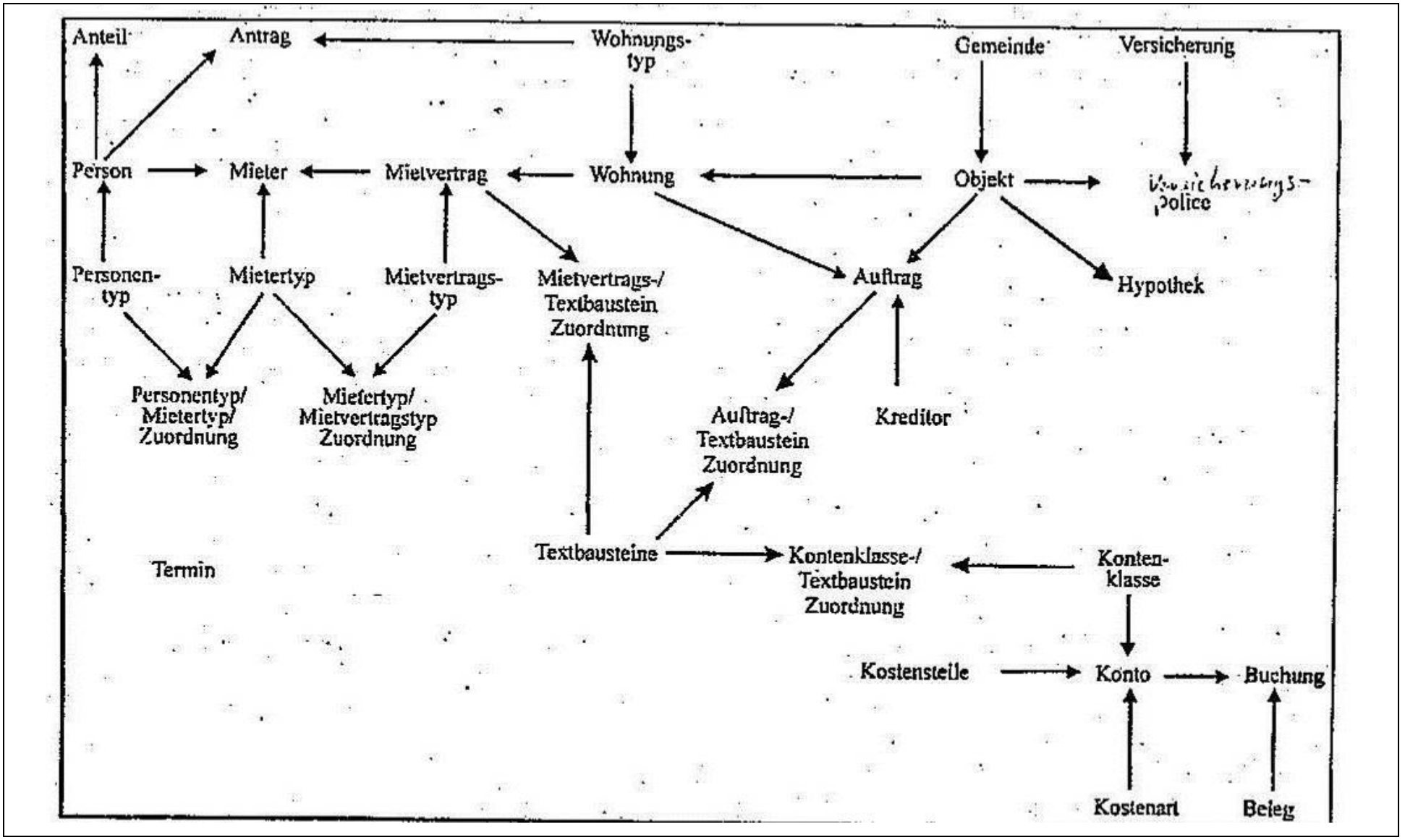
Car Sharing


Bezirke
Mitglieder-gruppen
Banken
Bezirksnr.
Bezirksnr.
Gruppennr.
BLZ
BLZ
Fahrzeug-klassen
Standorte
Vereine
Klassennr.
Standortnr.
Vereins-kennzeichen
Vereins-kennzeichen
Fahrzeuge
Mitglieder
Sammel-rechnungen
Fzg.Nr.
Fzg.Nr.
Mitgl.Nr.
Mitgl.Nr.
Rechnungsnr.
Mitgl.Nr.
Wartungs-termine
Reservierung
Einzel-rechnungen
Kosten-buchungen
Sammel-rechnungs-positionen
Rechungsnr.
Ausstattungs-merkmale
Fzg.Nr.
Einzel-rechnungs-positionen
Aus-stattungs-merkmalnr.
Ausstattung

Verein


Vereinsex-terne Pers.
c:1
Externnr. = Personennr.
Mitglied-schaftsarten
Mitgl.-schaftsart
1 : n
Warteliste
c:1
Warte-Mitgl.-Nr. = Personennr.
Personen
1:c
Personennr. = Ex-Mitgl.-Nr.
Ex-Mitglieder
n:1
Personennr.
1:c
Personennr. = Mitgl.-Nr.
1:n
Personennr.
Versich.-zu-gehörigkeit
c:1
Mitgl.-Nr.
1:n
Ruhen
Mitgl.-Nr.
Gruppen
1:n
Gruppen-name
Gruppenzu-gehörigkeit
n:1
Mitgl.-Nr.
Mitglieder (alle Arten)
1:n
Mitgl.-Nr.
Arbeits-leistungen
1:n
Mitgl.-Nr.
Rechnungen
1:n
Rechn.-Nr.
Belegnr.
Einnahmen
Belegnr.
Leihvorgänge
Rechn.-Nr.
1 : n
Geräte-Nr.
n : 1
Rechnungs-position
Rechnungs-einnahmen
Spenden
Leihgeräte
c ∈ { 0, 1 }.

Vermietergenossenschaft



6 Special cases

6.1 Transform a 2-dimensional table into a database table √

6.2 Tree (bill of materials for one product) √

6.3 Network (bills of materials for several products) √

6.4 Coupling of three or more tables n:m:p etc. (e.g. time-table)

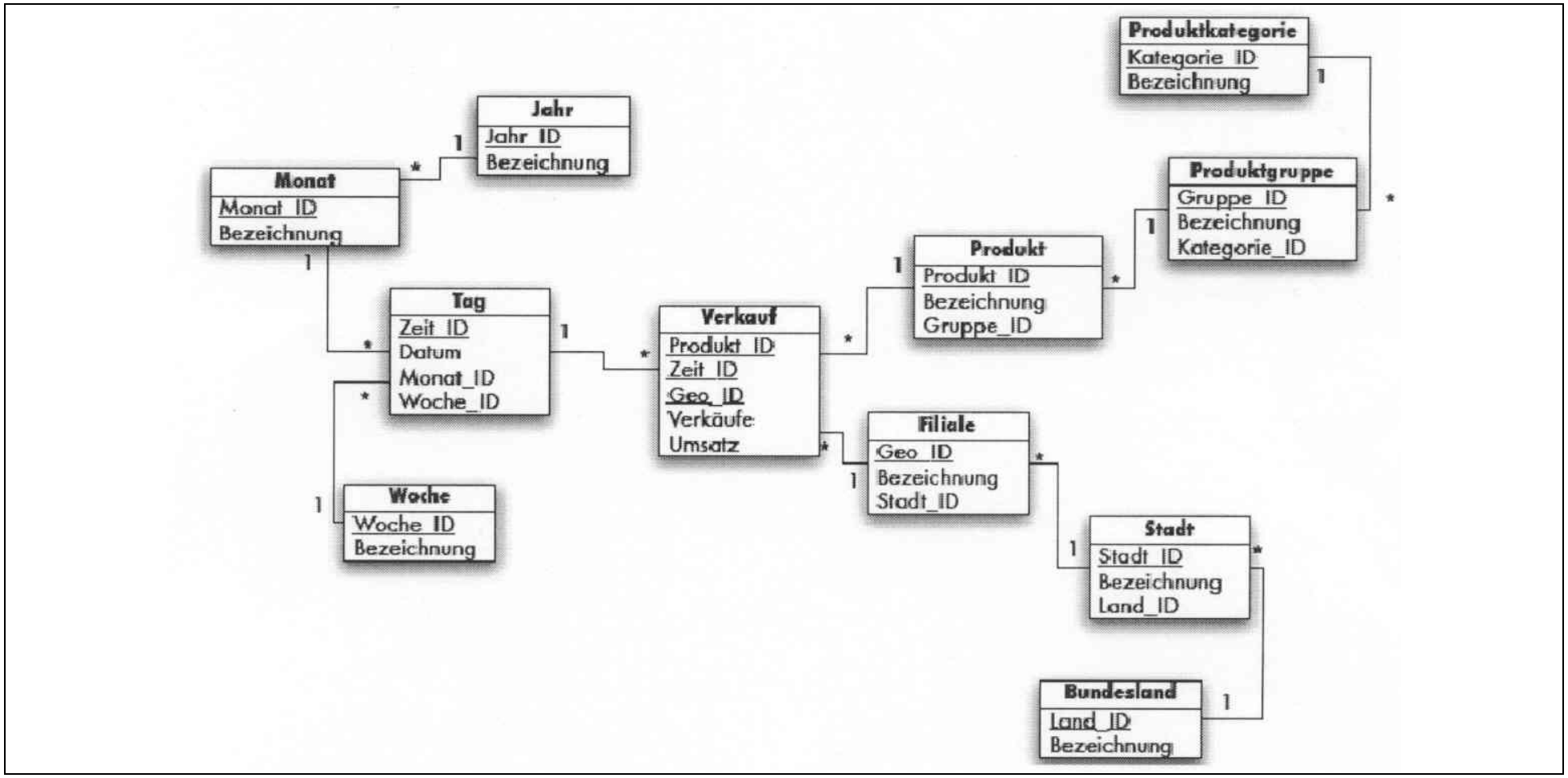
6.5 Customers, customer groups, customer categories

Only explained in class.

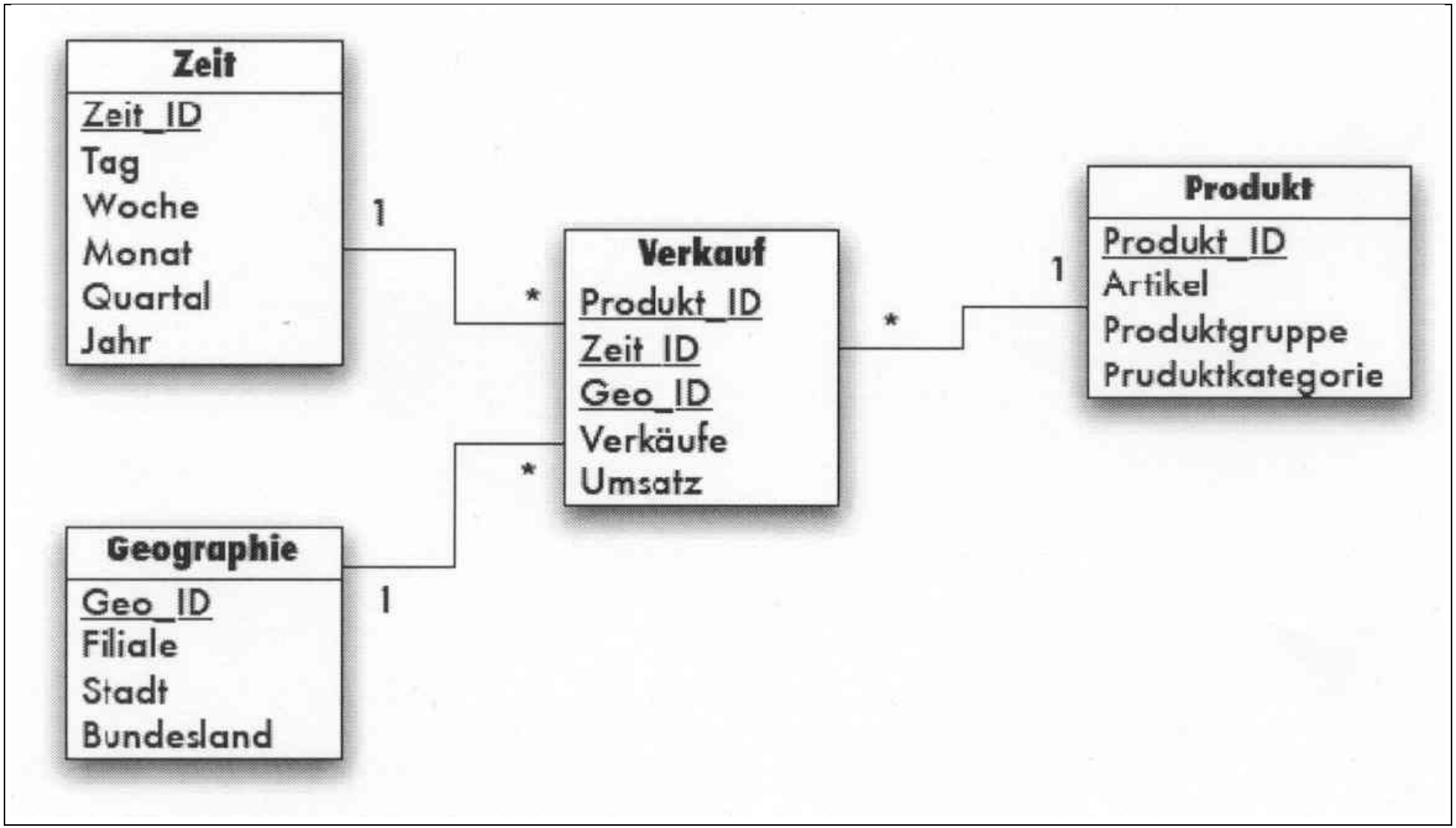
6 Special cases

6.6 Business Intelligence – Snowflake Schema (3NF, analytic phase)

Fact table(s) in the middle, dimension tables joined many-to-one



6.6 Business Intelligence – Star Schema (denormalized, synthetic phase)



6.6 Business Intelligence – Compacted snowflake schema

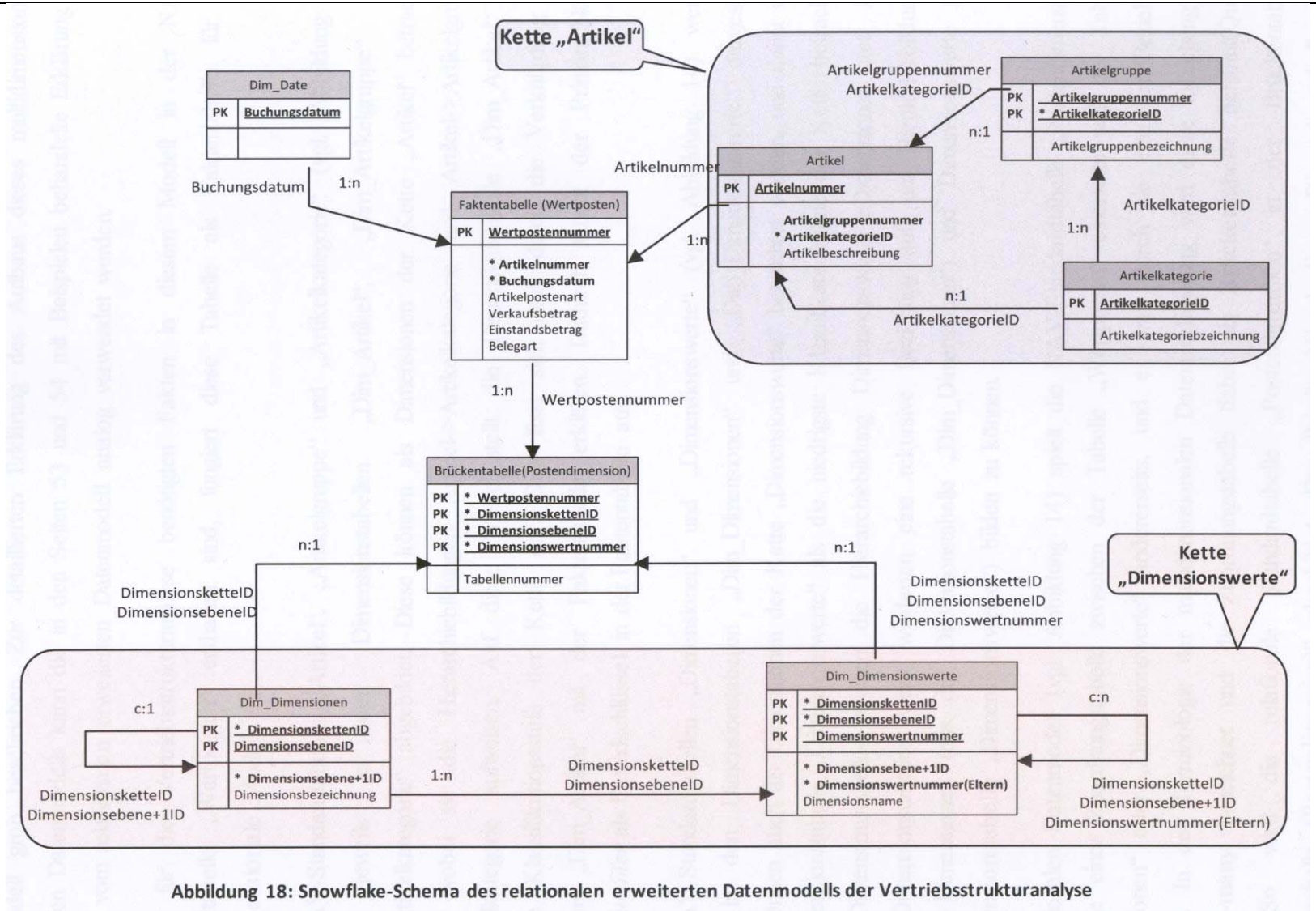
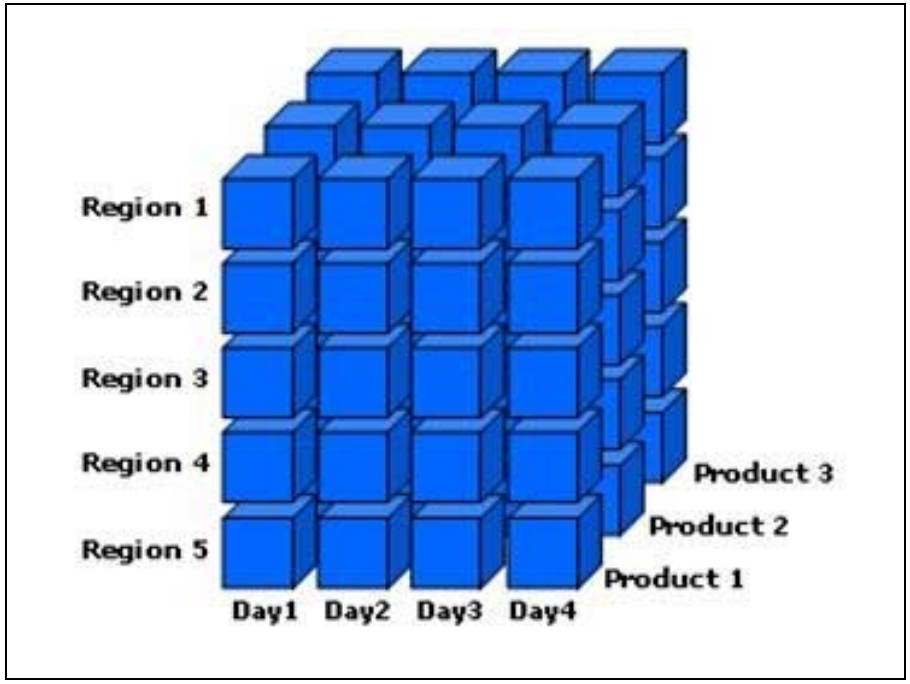


Abbildung 18: Snowflake-Schema des relationalen erweiterten Datenmodells der Vertriebsstrukturanalyse

6.6 Business Intelligence – Data Cube, Data Analysis

Online Analytical Processing OLAP



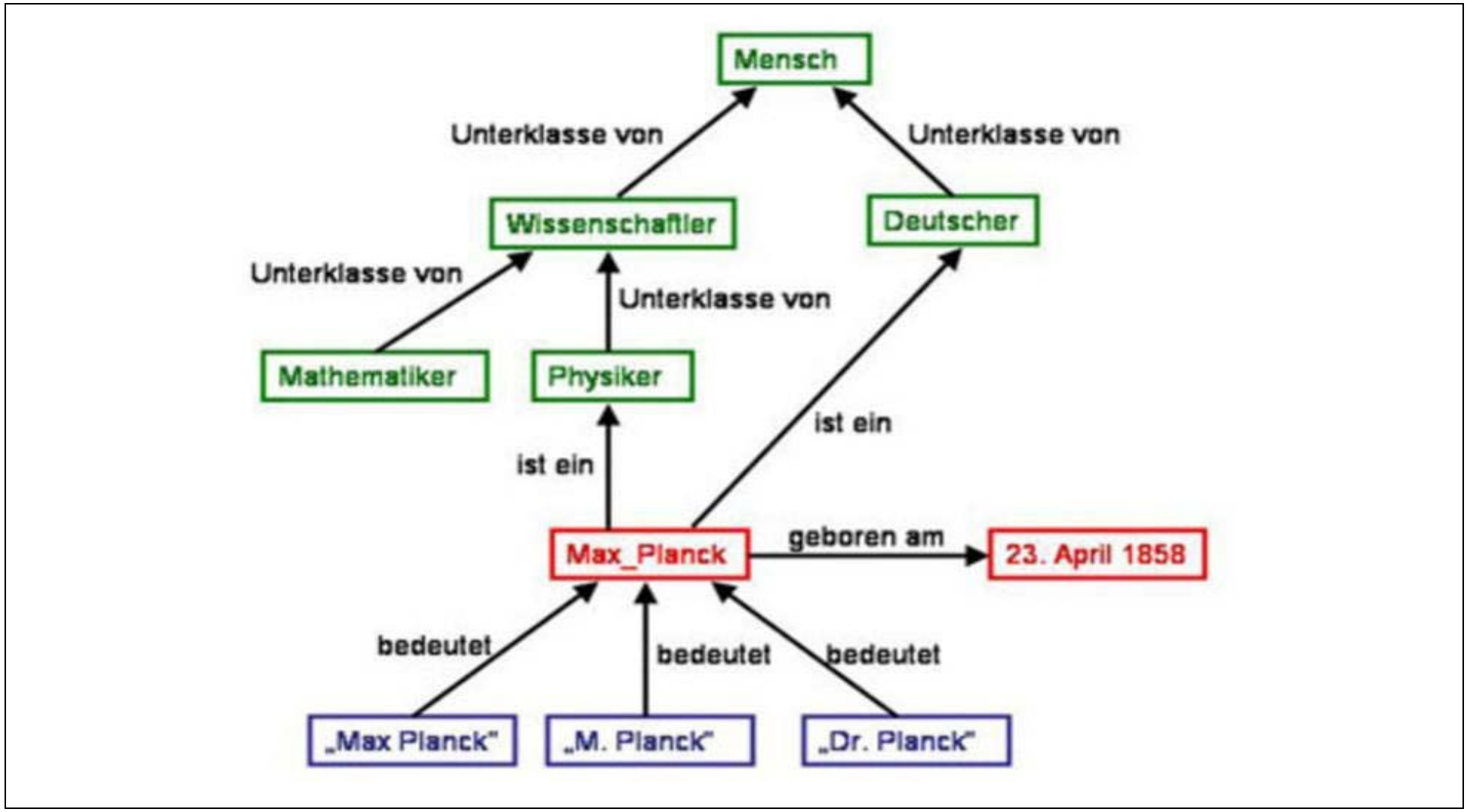
Slicing: Scheibe herausschneiden

Dicing: kleineren Würfel erzeugen

Roll-up: Herauszoomen

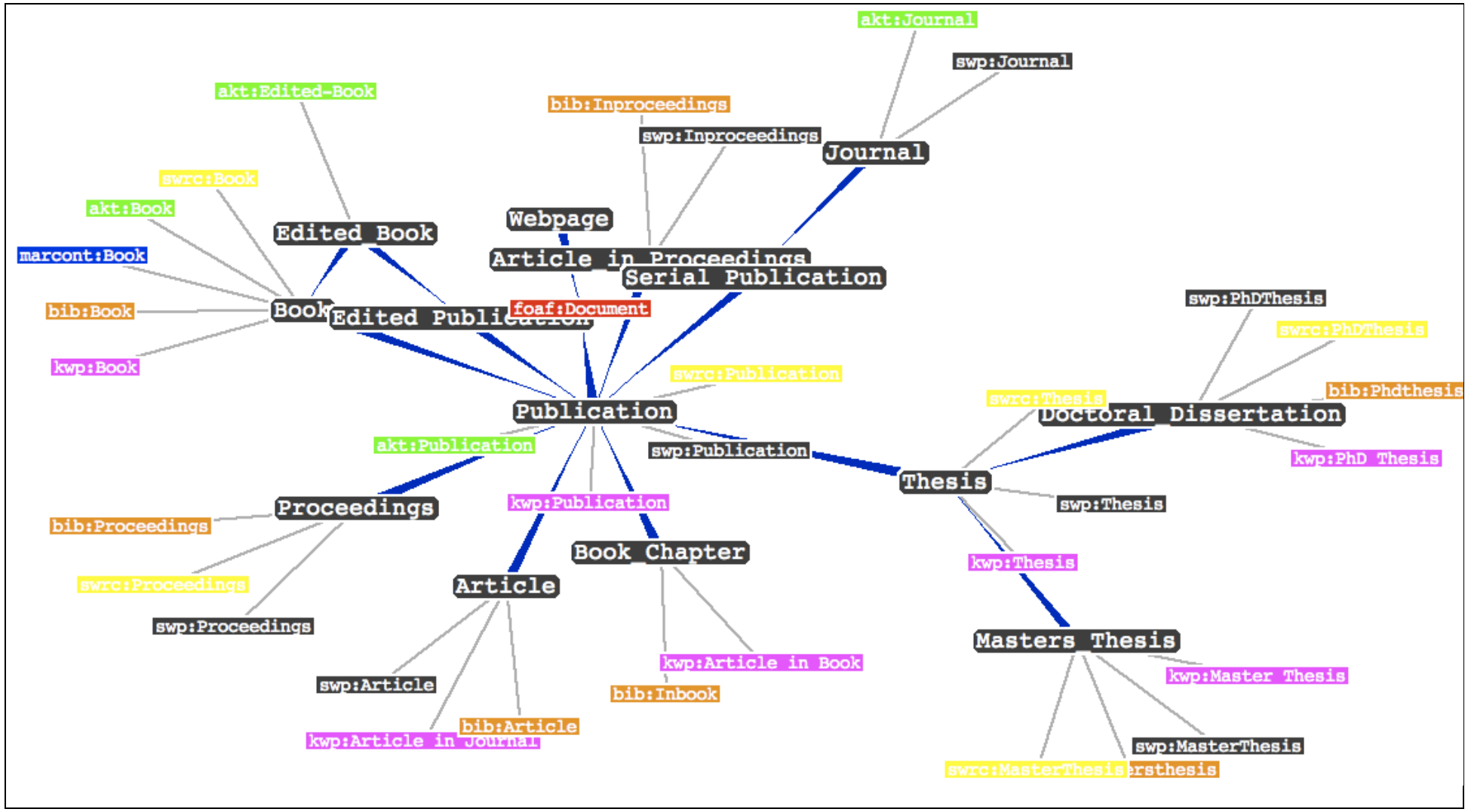
Drill-down: Hineinzoomen

6.7 Knowledge representation: ontology (semantic network)



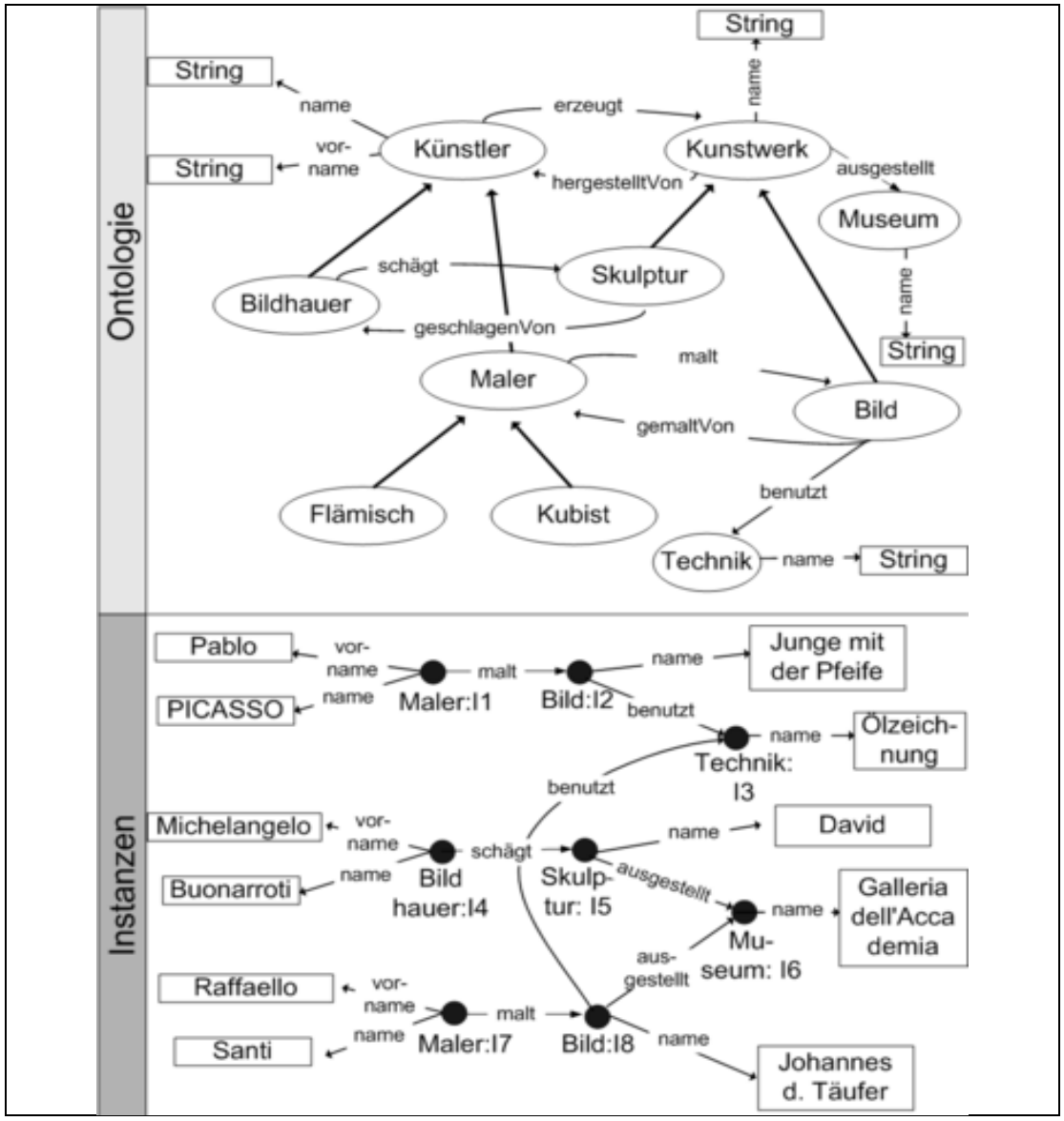
Ontologie über Personen

6.7 Knowledge representation: ontology (semantic network)



Ontologie über Veröffentlichungsarten

6.7 Knowledge representation: ontology (semantic network)



6.7 Knowledge representation: ontology representation languages

Resource Description Framework RDF:

XML-basiertes Rahmenwerk zur Beschreibung einzelner Ressourcen
Subjekt-Prädikat-Objekt-Tripel

RDF Schema RDFS

Web Ontology Language OWL: RDF-basiert

Web ontology:
OWL-Dokument, das eine Semantic-Web-Ontologie beschreibt

SPARQL Protocol And RDF Query Language SPARQL


SQL

SQL in der Lehrveranstaltung Datenbanken

Die Thematik SQL müssen Sie sich im Selbststudium erarbeiten.
Gegenstand dieser Lehrveranstaltung sind nicht bestimmte relationale DBMS (wie z. B. MS SQL Server, mySQL etc.) mit ihren Spezifika, sondern produktunabhängige Grundlagen!

Welche Teile von SQL müssen Sie lernen?
Es geht nicht um den vollen Umfang von SQL, sondern ausschließlich um den SELECT-Befehl!

Welche Bedeutung hat der SELECT-Befehl für die Klausur?
Aufgaben zum SELECT-Befehl bilden die Hälfte der Klausur!

Mit welchen Quellen können Sie sich einarbeiten?
– auf meiner Homepage: dieser Foliensatz ("Ergänzende Folien zu Datenbanken")
– auf meiner Homepage: "Einführung und Aufgaben zu SQL". Das ist eine Bachelorarbeit, die vor Jahren speziell zur Unterstützung des Selbststudiums von SQL geschrieben wurde
– im Prinzip jedes vernünftige Buch über SQL

SQL in der Lehrveranstaltung Datenbanken

Es gibt vier Dinge beim SELECT-Befehl, die Sie von Anfang an verstehen müssen:

1. Wenn man eine Aufgabenstellung (meistens sind das irgendwelche Auswertungen) in SQL übersetzen will, tut man das in zwei Schritten:
– zunächst vergegenwärtigt man sich die Aufgabenstellung in der Sprache der Mengenlehre (die Konzepte Vereinigung, Durchschnitt, Differenz von Mengen müssen Sie beherrschen!)
– erst dann formuliert man einen SELECT-Befehl

2. SQL ist eine Viertgenerationssprache (4GL, deklarative Sprache).
Jeder SELECT-Befehl wird aber von einem Interpreter prozedural (wie bei einer 3GL) abgearbeitet, d. h., Sie müssen sich über die Abarbeitungsreihenfolge der einzelnen Klauseln genau im Klaren sein (Folie „1.6 Abarbeitungsreihenfolge …“)

3. SQL ist keine orthogonale Sprache, d. h., Sie können eine Aufgabe syntaktisch häufig auf verschiedene Arten mit SELECT lösen.
Eine Klausuraufgabe kann von Ihnen also verlangen, eine Problemstellung mit einem bestimmten SQL-Syntax-Konzept zu lösen!

4. Abfragefilter: WHERE nur bei echten Satzfiltern, ON nur bei Join-Filtern.

1 SQL-Grundlagen

1.1 Relationale Grundoperationen

1. Selektion (Zeilenauswahl, Satzauswahl)
Beispiel: Alle Kunden in Nürnberg
WHERE-Bedingung: Satzauswahl, Satzfilter

2. Projektion (Spaltenauswahl)
Beispiel: Adressliste aller Kunden

3. Join (Verknüpfung, Verbund)
Theta-Verbund oder condition join
ON-Bedingung: Joinauswahl, Joinfilter
Beispiel: Auftragsdaten mit zugehörigen Kundendaten

WHERE- und ON-Bedingung sind nach ANSI-Standard zu unterscheiden.

1.1 Relationale Grundoperationen

4. Selektion und Projektion
Beispiel: Adressliste der Nürnberger Kunden

5. Join und Selektion
Beispiel: Alle Kunden- und Auftragsdaten der Nürnberger Kunden

6. Join und Projektion
Beispiel: Alle Auftragsdaten mit Kundennamen

7. Join und Selektion und Projektion
Beispiel: Auftragsdaten der Nürnberger Kunden mit Kundennamen

1.1 Relationale Grundoperationen: Satzfilter vs. Joinfilter

Ein Joinfilter (JOIN … ON) verbindet z w e i Tabellen
(meist über die Primärschlüssel-Fremdschlüssel-Beziehung)
Ein Satzfilter (WHERE) wählt Zeilen aus e i n e r Tabelle aus
(diese Tabelle kann selbst durch einen Join entstanden sein)

Beispiel: Liste der offenen Aufträge der Nürnberger Kunden
(AuftragsNr, AuftragsDatum, KdID, KdName)

```
SELECT AuftragsNr, AuftragsDatum, KdID, KdName
FROM Aufträge JOIN Kunden ON Auf_KdID = KdID
WHERE AuftragsStatus = ‘offen‘
AND KundenOrt = ‘Nürnberg‘
```

Man könnte Joinfilter auch mit WHERE formulieren, aber das wäre intransparent und entspräche nicht dem ANSI-Standard. Also verboten!

1.2 Sonderfälle des Join

1. [INNER] JOIN (kommutativ): Verknüpfung wo möglich, Default

2. LEFT / RIGHT [OUTER] JOIN (nicht kommutativ):
Inner Join plus nicht verknüpfbare Zeilen der Driving Table
Driving Table (unabhängig): alle Zeilen werden ausgegeben
Driven Table (abhängig): Zeilen nur ausgegeben, sofern Join möglich

Beispiel: Alle Kunden mit Aufträgen plus alle Kunden ohne Aufträge

Vorsicht: Anders als beim Inner Join, können WHERE (Satzfilter) und ON (Joinfilter) beim Outer Join unterschiedliche Wirkungen haben!
Eine WHERE-Bedingung in einem Outer Join
wird immer erst nach Erstellung des Outer Join ausgewertet und
kann so den Outer Join zerstören.
Besonders gefährlich: WHERE-Bedingung auf Attribute der Driven Table.

1.2 Sonderfälle des Join

3. Anti-Join oder negativer Join:
   nur die nicht verknüpfbaren Zeilen der Driving Table

Beispiel: Alle Kunden ohne Aufträge

```
SELECT KdID, KdName
FROM Kunden LEFT [OUTER] JOIN Aufträge
ON KdID = Auf_KdID
WHERE Auf_KdID IS NULL
```

Das geht auch ohne Join mit WHERE KdID NOT IN Subselect …
oder mit einer Mengendifferenz

Es gilt: Outer Join = Inner Join (mit ON) disjunkt vereinigt mit Anti-Join
→ dazu Mengendiagramm!

1.2 Sonderfälle des Join **(nach GROUP BY besprechen!)**

Problem-Beispiel für ON und WHERE beim OUTER JOIN

Alle Artikel, die in einem Statistikzeitraum geliefert wurden, und
alle Artikel, die in diesem Zeitraum nicht geliefert wurden, mit Mengen.

```
SELECT A_ArtID, A_ArtBez, SUM(L-Liefermenge)
FROM Artikel LEFT JOIN Lieferpositionen ON A_ArtID = L_ArtID
WHERE L_Datum BETWEEN StatAnf AND StatEnd
      OR L_Datum IS NULL
GROUP BY A_ArtID, A_ArtBez
```

Falsch, denn es gibt 3 Fälle:
1. u.a. im Statistikzeitraum gelieferte Artikel (L_Datum BETW … AND …)
2. gar nicht gelieferte Artikel (L_Datum IS NULL)
3. nur außerhalb des Statistikzeitraums gelieferte Artikel

→ bei diesen ist die WHERE-Bedingung nicht erfüllt

1.2 Sonderfälle des Join

Richtig:

```
SELECT A_ArtID, A_ArtBez, SUM(L_Liefermenge)
FROM Artikel LEFT JOIN Lieferpositionen ON A_ArtID = L_ArtID
AND L_Datum BETWEEN StatAnf AND StatEnd
GROUP BY A_ArtID, A_ArtBez
```

OUTER JOIN erzeugt Inner Join (mit ON-Bedingung) und
erweitert diesen um Anti-Join auf Outer Join.

1.2 Sonderfälle des Join

Oder klassisch:

```
SELECT A_ArtID, A_ArtBez, SUM(L_Liefermenge)
FROM Artikel JOIN Lieferpositionen ON A_ArtID = L_ArtID
WHERE L_Datum BETWEEN StatAnf AND StatEnd
GROUP BY A_ArtID, A_ArtBez
UNION
SELECT A_ArtID, A_ArtBez, 0 FROM Artikel
WHERE A_ArtID NOT IN
(SELECT DISTINCT L_ArtID FROM Lieferpositionen
WHERE L_Datum BETWEEN StatAnf AND StatEnd)
```

Inner Join disjunkt vereinigt mit Anti-Join

Mengenstruktur des Select-Ergebnisses für UNION angeglichen

1.2 Sonderfälle des Join

Ein äquivalentes Beispiel:

Alle Kunden, die (u. a.) in I/2019 Aufträge erteilt haben,
mit ihren Aufträgen und
alle Kunden, die in I/2019 keine Aufträge erteilt haben,
d. h. entweder noch gar keine Aufträge erteilt haben oder
nur Aufträge außerhalb des Statistikzeitraums I/2019 erteilt haben.

1.3 Gruppenbildung, Aggregatfunktionen, Gruppenauswahl

Gruppenbildung
ist eine Form der Datenverdichtung (engl. aggregation)
“Töpfe”, deren Etiketten bestimmte Wert-Tupel der GROUP-Attribute sind
z.B. GROUP BY KdID: jeder Wert der KdID (Kunde) bildet einen “Topf”

GROUP
Beispiel: Alle Kunden, die 2012 Aufträge erteilt haben (aus Auftragstab.)

Die Einschränkung auf 2012
funktioniert mit WHERE … oder mit HAVING YEAR(AuftragsDatum).
HAVING auf Gruppenattribute ist grundsätzlich erlaubt.
Aber es werden überflüssige Gruppen erzeugt (Performance!),
deren Irrelevanz ohne Aggregatfunktion
bereits an jedem einzelnen zugehörigen Datensatz erkennbar ist.
Deshalb besser mit WHERE filtern.

1.3 Gruppenbildung, Aggregatfunktionen, Gruppenauswahl

GROUP und Aggregatfunktion SUM
Beispiel: Alle Kunden mit jeweiligem Gesamtauftragswert in 2012

GROUP, SUM und HAVING
Beispiele: 1. Alle Kunden mit Gesamtauftragswert > 5000 € in 2012
2. Alle Nürnberger Kunden mit Gesamtauftragswert > 5000 € in 2012

Aggregatfunktionen ohne GROUP: Die gesamte Tabelle bildet eine Gruppe
Beispiel: Gesamtauftragswert über alle Kunden in 2012

HAVING kann mit den gleichen syntaktischen Bedingungsformen verbunden werden wie WHERE:
<, >, =, LIKE, (NOT) IN (Subselect), Tupelabfrage etc.
Aggregatfunktionen sind mit HAVING, nicht mit WHERE kombinierbar.

1.3 Gruppenbildung, Aggregatfunktionen, Gruppenauswahl

HAVING: Gruppenauswahl, -filter, Teilmengenauswahl
WHERE: Satzauswahl, -filter, Elementauswahl
ON: Joinauswahl, -filter

Aggregatfunktions-Attribute:
HAVING-Aggregatfunktionen brauchen nicht im SELECT zu erscheinen.
Aggregatfunktionen dürfen nur in SELECT und HAVING vorkommen!
Aggregatfunktionen dürfen in GROUP und WHERE nicht vorkommen!

Normale Attribute:
Jedes HAVING-Attribut muss auch ein GROUP-Attribut sein.
→ gilt nicht für Argumente von Aggregatfunktionen,
z.B. GROUP BY KdID HAVING SUM(Auftragswert) > 500 und 2.1 Nr. 2
Jedes SELECT-Attribut muss auch ein GROUP-Attribut sein.

1.4 Sonderthemen

Aliasing von Tabellen- und Attributnamen

DISTINCT
teilweise gleichwertig mit GROUP ohne SELECT-Aggregatfunktionen

Subselect
teilweise gleichwertig mit Join:
WHERE … (NOT) IN (Subselect)
HAVING … (NOT) IN (Subselect)
SELECT … FROM (Subselect)

1.5 Strukturierte Kommentierung

Benötigte Attribute (Ausgabeattribute, Attribute in Aggregatfunktionen, Gruppenattribute, Satzfilter-, Joinfilter-, Gruppenfilterattribute)
Benötigte Tabellen
Joinfilter (meist so formuliert: „Join der Tabellen A und B über ihre Primärschlüssel-Fremdschlüssel-Beziehung“ – es gibt auch andere!)
Satzfilter
Gruppierung
Gruppenfilter

Zusammengesetzte SELECT-Befehle können geschachtelte und/oder parallele Teilselects enthalten, die folgendermaßen zu kommentieren sind:
Obige Angaben für jeden Teil-SELECT,
dazu die Semantik jedes Teil-SELECTs (beschreibt dessen Ergebnis),
nicht aber für den äußersten SELECT eines Schachtel-SELECTs
(dessen Semantik ist gleich der Aufgabenstellung)

1.6 Abarbeitungsreihenfolge der Klauseln des SELECT-Befehls

vgl. Software Engineering: Gruppenverarbeitung (breakpoint analysis)

Input	
Verknüpfung mit Joinfilter	[INNER] JOIN mit ON
Ergänzung um Anti-Join	LEFT / RIGHT [OUTER]
(Hinzunahme der nicht verknüpfbaren Sätze der Driving Table)	
Satzfilter (Selektion)	WHERE
Gruppenbildung und –auswertung	GROUP
Gruppenfilter	HAVING
Spaltenauswahl (Projektion)	SELECT
Output	

2 SQL-Aufgaben zu Logik und Mengenlehre

Mengenoperationen (Differenz, Vereinigung, Durchschnitt) verlangen, dass die beteiligten Mengen die gleiche Struktur haben.

Ausgangspunkt: Lieferpositionstabelle mit FS KdID und ArtID
Artikelauswahl für einen bestimmten statistischen Berichtszeitraum

Struktur der Lösungen:
SELECT L_ArtID FROM Lieferpositionen
WHERE …
GROUP BY L_ArtID
HAVING …
WHERE … (NOT) IN (Subselect) / MINUS / UNION / INTERSECT
SELECT …
Die folgenden Lösungsangaben sind in verkürzter Form dargestellt!

2 SQL-Aufgaben zu Logik und Mengenlehre

Beispiel für eine Lieferpositionstabelle

KdID	ArtID	Liefermenge	Lieferdatum
X	4711	…	…
Y	4712		
X	4715		
Y	4715		
Z	4713		
X	4717		
Z	4717		
Y	4718		
Z	4718		
X	4719		
Y	4719		
Z	4719		

2 SQL-Aufgaben zu Logik und Mengenlehre

			von genau 1 Kd gekaufte Artikel	von genau 2 Kd gekaufte Artikel	von > 2 Kd gekaufte Artikel
u.a. von Kd X gekaufte Artikel		nur von Kd X gekaufte Artikel			
	u.a. von mind. 1 anderen Kd als X gekaufte Artikel	von Kd X und mind. 1 anderen Kd gekaufte Artikel		von Kd X und genau 1 anderen Kd gekaufte Artikel	von Kd X und mind. 2 anderen Kd gekaufte Artikel
		nur von mind. 1 anderen Kd als X gekaufte Artikel	nur von genau 1 anderen Kd als X gekaufte Artikel	nur von genau 2 anderen Kd als X gekaufte Artikel	nur von mind. 3 anderen Kd als X gekaufte Artikel

2.1 Auswahlbedingungen für e i n e n Kunden

1. Artikel, die u.a. von Kunde X gekauft wurden
(es kann auch andere Kunden geben, die diese Artikel kauften)
= Artikel, zu denen es mindestens eine Lieferposition gibt,
die sich auf den Kunden X bezieht / die den Kunden X betrifft

SELECT DISTINCT L_ArtID … WHERE L_KdID = 'X'

2. Artikel, die jeweils von genau einem beliebigen Kunden gekauft wurden
(es ist egal, welcher Kunde es ist, es darf aber nur einer sein)
= Artikel, deren Lieferpositionen sich auf genau einen Kunden beziehen

```
SELECT L_ArtID …
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1
```

2.2 Negation, Komplement

3. Artikel, die u.a. von mind. einem anderen Kunden als X gekauft wurden (können auch von X gekauft worden sein)

```
SELECT DISTINCT L_ArtID … WHERE L_KdID <> 'X'
```

4. Artikel, die u.a. von genau einem anderen Kunden als X gekauft wurden (können auch von X gekauft worden sein)

```
SELECT L_ArtID …
WHERE L_KdID <> 'X'
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1
```

2.2 Negation, Komplement

Übung:
Artikel, die vom Kd X und genau einem anderen Kunden gekauft wurden

Lösung:
Artikel, die u.a. von X gekauft wurden (1.)
geschnitten mit
Artikel, die u.a. von genau einem anderen Kunden als X gekauft wurden (4.)
(oder: Artikel, die von genau zwei Kunden gekauft wurden)

falsch:
SELECT L_ArtID …
WHERE L_KdID = 'X'
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 2
Satzfilterung vor Gruppierung, also liefert SELECT die leere Menge

2.3 Mengendifferenz

5.1 Artikel, die ausschließlich von Kunde X gekauft wurden
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)

```
SELECT DISTINCT L_ArtID …
WHERE [L_KdID = 'X'
AND] L_ArtID NOT IN / MINUS
(SELECT L_ArtID …
WHERE L_KdID <> 'X')
```

Artikel, die [u.a. von Kunde X] gekauft wurden
ohne / minus
Artikel, die u.a. von mind. einem anderen Kunden als X gekauft wurden

falsch: WHERE L_KdID = 'X' AND NOT (L_KdID <> 'X')
logisch äquivalent zu L_KdID = 'X'

2.3 Mengendifferenz

5.2 Artikel, die ausschließlich von Kunde X gekauft wurden
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)

```
SELECT L_ArtID …
GROUP BY L_ArtID
HAVING COUNT(DISTINCT L_KdID) = 1
MINUS
SELECT DISTINCT L_ArtID …
WHERE L_KdID <> 'X'
```

Artikel, die von genau einem Kunden gekauft wurden
ohne / minus
Artikel, die u.a. von mind. einem anderen Kunden als X gekauft wurden

2.3 Mengendifferenz

5.3 Artikel, die ausschließlich von Kunde X gekauft wurden
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)

```
SELECT DISTINCT L_ArtID …
WHERE L_KdID = 'X'
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID …
GROUP BY L_ArtID
HAVING COUNT(DISTINCT L_KdID) > 1)
```

Artikel, die u.a. von Kunde X gekauft wurden
ohne / minus
Artikel, die von mehr als einem Kunden gekauft wurden

2.3 Mengendifferenz

5.4 Artikel, die ausschließlich von Kunde X gekauft wurden
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)
als Mengendurchschnitt (kommutativ)

```
SELECT L_ArtID …
WHERE L_ArtID IN
   (SELECT L_ArtID …
    WHERE L_KdID = 'X')
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1
```

Innerer Select: Artikel, die u.a. von Kunde X gekauft wurden
Äußerer Select: Artikel, die von genau einem Kunden gekauft wurden

WHERE L_KdID = 'X'ohne inneren Select liefert die u.a. von X gekauften

2.3 Mengendifferenz

5.5 Artikel, die ausschließlich von Kunde X gekauft wurden
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)
als Mengendurchschnitt (kommutativ)

```
SELECT DISTINCT L_ArtID …
WHERE L_KdID = ‘X’
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID FROM L
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1)
```

Artikel, die u.a. von Kunde X gekauft wurden
geschnitten mit
Artikel, die von genau einem Kunden gekauft wurden

2.3 Mengendifferenz

6.1 Artikel, die nur von mind. einem anderen Kunden,
aber nicht von Kunde X gekauft wurden

```
SELECT DISTINCT L_ArtID …
WHERE L_ArtID NOT IN / MINUS
(SELECT L_ArtID FROM L
WHERE L_KdID = 'X')
```

Alle verkauften Artikel
ohne / minus
Artikel, die u.a. von Kunde X gekauft wurden

falsch:
WHERE L_KdID <> 'X'
findet Artikel, die u.a. von mind. 1 anderen Kunden als X gekauft wurden

2.3 Mengendifferenz

6.2 Artikel, die nur von mind. einem anderen Kunden,
aber nicht von Kunde X gekauft wurden

```
SELECT L_ArtID FROM Lieferpositionen
MINUS
SELECT L_ArtID FROM Lieferpositionen
GROUP BY L_KdID, L_ArtID
HAVING COUNT(*) > 0 AND L_KdID = 'X'
```

Alle verkauften Artikel
ohne / minus
Artikel, die u.a. von Kunde X gekauft wurden

2.3 Mengendifferenz

7.1 Artikel, die von genau einem Kunden, aber nicht von X gekauft wurden
= Artikel, die ausschl. von genau einem anderen Kd. als X gekauft wurden

```
SELECT L_ArtID …
WHERE L_ArtID NOT IN
(SELECT DISTINCT L_ArtID …
WHERE L_KdID = 'X')
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1
```

Innerer Select: Artikel, die u.a. von Kunde X gekauft wurden
Äußerer Select: Artikel, die von genau einem Kunden gekauft wurden

2.3 Mengendifferenz

7.2 Artikel, die von genau einem Kunden, aber nicht von X gekauft wurden
= Artikel, die ausschl. von genau einem anderen Kd. als X gekauft wurden

```
SELECT L_ArtID …
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1
MINUS
SELECT DISTINCT L_ArtID …
WHERE L_KdID = 'X'
```

Artikel, die von genau einem Kunden gekauft wurden
ohne / minus
Artikel, die u.a. von Kunde X gekauft wurden

2.3 Mengendifferenz

7.3 Artikel, die von genau einem Kunden, aber nicht von X gekauft wurden
= Artikel, die ausschl. von genau einem anderen Kd. als X gekauft wurden
als Mengendurchschnitt (kommutativ)

```
SELECT L_ArtID …
WHERE L_ArtID IN
(SELECT DISTINCT L_ArtID …
WHERE L_KdID <> 'X')
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1)
```

Innerer Select: Artikel, die
u.a. von mind. einem anderen Kunden als X gekauft wurden
Äußerer Select: Artikel, die von genau einem Kunden gekauft wurden

2.3 Mengendifferenz

7.4 Artikel, die von genau einem Kunden, aber nicht von X gekauft wurden
= Artikel, die ausschl. von genau einem anderen Kd. als X gekauft wurden
als Mengendurchschnitt (kommutativ)

```
SELECT DISTINCT L_ArtID …
WHERE L_KdID <> 'X'
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID …
GROUP BY L_ArtID
HAVING COUNT (DISTINCT L_KdID) = 1)
```

Artikel, die u.a. von mind. einem anderen Kunden als X gekauft wurden
geschnitten mit
Artikel, die von genau einem Kunden gekauft wurden

2.4 Durchschnitt / Und (kommutativ)

8. Artikel, von denen jeder einzelne u.a. sowohl von Kunde X als auch von Kunde Y gekauft wurde
(es kann auch andere Kunden geben, die diese Artikel kauften)

```
SELECT DISTINCT L_ArtID …
WHERE L_KdID = ‘X’
AND L_Art ID IN / INTERSECT
(SELECT L_ArtID …
WHERE L_KdID = ‘Y’)
```

Artikel, die u.a. von Kunde X gekauft wurden
geschnitten mit
Artikel, die u.a. vom Kunden Y gekauft wurden

falsch: WHERE L_KdID = ‘X’ AND L_KdID = ‘Y’

2.4 Durchschnitt / Und (kommutativ)

9. Artikel, die sowohl von Kunde X als auch von mindestens einem anderen Kunden gekauft wurden

```
SELECT DISTINCT L_ArtID …
WHERE L_KdID = 'X'
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID …
WHERE L_KdID <> 'X')
```

Artikel, die u.a. von Kunde X gekauft wurden
geschnitten mit
Artikel, die u.a. von mind. einem anderen Kunden als X gekauft wurden
(oder: Artikel, die von mindestens zwei Kunden gekauft wurden)

2.4 Durchschnitt / Und (kommutativ)

10. Artikel, von denen jeder einzelne ausschließlich sowohl von Kunde X als auch von Kunde Y gekauft wurde
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)

```
(SELECT L_ArtID … WHERE L_KdID = 'X'
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID … WHERE L_KdID = 'Y'))
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID … WHERE L_KdID <> 'X' AND L_KdID <> 'Y')
```

Artikel, von denen jeder einzelne u.a. sowohl von Kunde X als auch von Kunde Y gekauft wurde
ohne / minus
Artikel, die u.a. von mind. 1 anderen Kunden als X und Y gekauft wurden

2.5 Vereinigung / Inclusives Oder (kommutativ)

11. Artikel, von denen jeder einzelne u.a. von Kunde X oder (incl.) von Kunde Y gekauft wurde
(es kann auch andere Kunden geben, die diese Artikel kauften)
WHERE L_KdID = 'X' OR L_KdID = 'Y'

12. Artikel, von denen jeder einzelne ausschließlich von Kunde X oder (incl.) von Kunde Y gekauft wurde
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)
SELECT L_ArtID … WHERE L_KdID = 'X' OR L_KdID = 'Y'
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID … WHERE L_KdID <> 'X' AND L_KdID <> 'Y')

Artikel, die u.a. von Kunde X oder (incl.) von Kunde Y gekauft wurden ohne / minus
Artikel, die u.a. von mind. 1 anderen Kunden als X und Y gekauft wurden

2.6 Mengenkomplement / symmetrische Differenz / Exclusives Oder (nur der Vollständigkeit wegen)

13. Artikel, von denen jeder einzelne u.a. entweder von Kunde X oder (excl.) Kunde Y gekauft wurde
(es kann auch andere Kunden geben, die diese Artikel kauften)

Vereinigung \ Durchschnitt

SELECT L_ArtID … WHERE L_KdID = 'X' OR L_KdID = 'Y'
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID … WHERE L_KdID = 'X'
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID … WHERE L_KdID = 'Y'))

2.6 Mengenkomplement / symmetrische Differenz / Exclusives Oder (nur der Vollständigkeit wegen)

14. Artikel, von denen jeder einzelne ausschließlich entweder von Kunde X oder (excl.) Kunde Y gekauft wurde
(es darf keinen anderen Kunden geben, der diese Artikel kaufte)

Vereinigung \ Durchschnitt \ von mind. 1 anderen Kunden gekaufte Artikel

```
SELECT L_ArtID … WHERE L_KdID = 'X' OR L_KdID = 'Y'
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID … WHERE L_KdID = 'X'
AND L_ArtID IN / INTERSECT
(SELECT L_ArtID … WHERE L_KdID = 'Y'))
AND L_ArtID NOT IN / MINUS
(SELECT L_ArtID … WHERE L_KdID <> 'X' AND L_KdID <> 'Y')
```

3. Spannendere SQL-Aufgaben

1. Auftragsverwaltung

Anzahl / Liste der Artikel (ID, Bezeichnung), die in einem Statistikzeitraum die geringste/größte in diesem Zeitraum vorkommende Auftragsmenge hatten

Anzahl / Liste der Kunden (ID, Name), die in einem Statistikzeitraum den größten in diesem Zeitraum vorkommenden Auftragswert erbrachten

Die Orte (PLZ, Ortsname), an denen die meisten Kunden wohnen

Anzahl / Liste der Kunden (ID, Name), die erstmalig 2019 etwas bei unserem Unternehmen bestellt haben

Für jeden Kunden (ID, Name) der/die teuerste(n) Artikel (ID, Bezeichnung), den/die er in einem Statistikzeitraum gekauft hat (Tupel-Satzfilter!)

3. Spannendere SQL-Aufgaben

Zu jedem Kunden (ID, Name) die an dessen jüngstem Auftragsdatum bestellten Artikel (ID, Bezeichnung), sofern noch nicht anonymisiert

1. Zu jedem Kunden das jüngste Auftragsdatum bestimmen
2. KdID und jüngstes Auftragsdatum mit Aliasnamen versehen
3. Ergebnis aus 2 als virtuelle Tabelle im SELECT-Befehl
   in Join-Kette (Joinfelder: u.a. KdID und jüngstes Auftragsdatum)
   mit Auftragspositionen verbinden

Die Orte, an denen die meisten Kunden wohnen

1. Zu jedem Ort die Anzahl der Kunden bestimmen (COUNT)
2. Anzahl mit Aliasname „Kundenanzahl“ versehen
3. Maximum der Kundenanzahlen bestimmen
   (Ergebnis aus 2 als virtuelle Tabelle im SELECT-Befehl)
4. Die gesuchten Orte bestimmen (Kundenanzahl = Maximum)
   (Die Kundenanzahl muss in diesem Befehl 2mal bestimmt werden!)

3. Spannendere SQL-Aufgaben

2. Projektverwaltung

Anzahl / Liste der Projekte (ID, Bezeichnung) mit der geringsten Anzahl von Mitarbeitern je Projekt in einem Statistikzeitraum

Anzahl / Liste der Mitarbeiter (ID, Name), die in einem Statistikzeitraum an der größten Anzahl von Projekten mitarbeiten

Anzahl / Liste der Projekte (ID, Bezeichnung) mit der höchsten, in einem Statistikzeitraum für ein Projekt geleisteten Arbeitszeit

3. Spannendere SQL-Aufgaben

3. Bibliothek

(Anzahl der) Benutzer (ID, Name), die heute die höchste heute vorkommende Anzahl von offenen Ausleihvorgängen je Benutzer erreicht haben, d. h. die heute die höchste heute vorkommende Anzahl von Büchern der Bibliothek in Besitz haben

(Anzahl der) Benutzer (ID, Name), die heute die höchste heute vorkommende Anzahl von neuen Ausleihvorgängen je Benutzer erreicht haben

Zu jedem Benutzer die an dessen jüngstem Ausleihdatum ausgeliehenen Medien (Verfasser, Titel), sofern noch nicht anonymisiert

3. Spannendere SQL-Aufgaben

4. Lieferpositionen (Kunden → Lieferpositionen ← Artikel)

Liste derjenigen Artikel, die in einem Berichtszeitraum von allen / keinem Kunden gekauft wurden (Hinweis: über Anzahl der Kunden)

Alle Artikel und alle in einem Statistikzeitraum verkauften Artikel mit den jeweiligen Liefermengensummen
Vorsicht: Lieferpositionstabelle kann auch Lieferungen aus anderen Jahren enthalten, dann wird OUTER JOIN falsch.

Anzahl aller Artikel, von denen in einem Statistikzeitraum insgesamt mehr als 10 Stück verkauft wurden

3. Spannendere SQL-Aufgaben

5. Lieferantenverwaltung

Zu jedem Lieferanten den/die angebotenen Artikel mit dem höchsten vorkommenden Einzelpreis (Tupel-Satzfilter!)

4. Weitere SQL-Konzepte

Zählen mit ROW-NUMBER OVER PARTITION

TOP, LIMIT