ZENTRUM FÜR
INFORMATIONSMODELLIERUNG
AUSTRIAN CENTRE FOR
DIGITAL HUMANITIES

KARL-FRANZENS-UNIVERSITÄT GRAZ
UNIVERSITY OF GRAZ

Information Modelling I

Introduction to Data Modelling

Sarah Lang

2022

Syllabus

Content

Theoretical and practical introduction to modelling data in tabular formats.

1. tabular data
2. relational model (Entity-Relationship-Model)
3. SQL databases

Preliminaries

How to get a positive grade on this class

Final Submission (60%)

- **Infomod:** Text, ER model, SQL database
- **DigEd:** Small digital edition or review of an existing edition.
- You can start in the last month of the semester and ask questions.
- You can collaborate but no plagiarism (Uni Graz zero tolerance policy).

Homework assignments (40%)

Communicated and to be completed within the week

Other aspects

1. attendance in class (you can miss max. 3, to be communicated beforehand).
2. Positive grade: at least 50% on all partial submissions.
3. "LVen mit immanentem Prüfungscharakter" → once you accept the first task you get a grade (i.e. first homework this week)
4. If you get a negative grade, the whole class needs to be retaken.

> Nichterbringung weiterer Teilleistungen ohne wichtigen Grund ist Prüfungsabbruch (Negativbeurteilung). Abmeldung nach bereits übernommener Teilleistung führt zu negativer Beurteilung.

see also: slides on grading & further info materials on the final submission

Deadlines

Hard deadlines

All deadlines are hard deadlines. You can get extensions for good reasons.

- Good reasons for example: care responsibility, being ill, etc.
- i.e. understandable reasons which are communicated asap

If you miss a deadline...

- If you didn't communicate: negative grade.
- Otherwise up for discussion according to the circumstances.

Learning Goals

1. get to know modelling theory („Allgemeine Modelltheorie“ nach Herbert Stachowiak)
   - try the Entity-Relationship-Diagramm (Peter Chen)
   - go from conceptual model to computer processable model
   - represent relational data structures
2. learn SQL (*Structured Query Language*)
3. get to know different data structures
4. basic practical modelling experience

Final project

Move from your own conceptual data model to a relational database in SQL and write about it (3–5 pages).

Literature on learning tech subjects

1. Carol Dweck, *Mindset: The New Psychology of Success* (New York: Random House 2006). (10min Video | blog post on the topic)
2. K. Anders Ericsson & Robert Pool, *Peak: Secrets from the New Science of Expertise*, Penguin: London 2016.
3. Barbara Oakley, *A Mind for Numbers*, NY 2014.

Working with computers as a humanities person

- Übungsblatt 0 is a prerequisite
- don’t panic

Present yourselves!

Name, pronouns, domain of origin, interests, etc.

References

1. **Stachowiak,** Herbert: *Allgemeine Modelltheorie*, Wien 1973.
2. **Chen,** Peter: *The Entity-Relationship Model – Toward a Unified View of Data*, 1976.
3. **Jannidis (Hg.):** *Digital Humanities. Eine Einführung*, Stuttgart 2017.
   - Online-Exemplar → gedrucktes Exemplar in der ZIM-Bibliothek.
   - Kap. 7 (*Datenmodellierung*) und 8 (*Datenbanken*).
4. **Flanders, Julia; Jannidis, Fotis:** *Knowledge Organization and Data Modeling in the Humanities*, 2015.
5. **Lothar Piepmeyer:** Grundkurs Datenbanksysteme. Von den Konzepten bis zur Anwendungsentwicklung, Hanser 2011.
6. **Ciula, Arianna & Eide, Øyvind & Marras, Cristina & Sahle, Patrick (Ed.):** *Models and Modelling between Digital and Humanities: A Multidisciplinary Perspective*, 2018.
7. **Julia Flanders, Fotis Jannidis:** *The Shape of Data in Digital Humanities Modeling Texts and Text-based Resources*, Routledge 2019.

Ressources

1. **OpenRefine:**
   - OpenRefine Download
   - OpenRefine site with tutorials
2. **SQL:**
   - SQLite Download
   - SQLite Browser
   - w3schools SQL Tutorial
3. **ER:**
   - ERDplus Tool for creating ER models
   - GeeksforGeeks-Tutorial/Intro to ER Models

Modelling theory: Models in research

Models in research

> In den Wissenschaften werden Modelle aus den unterschiedlichsten Gründen zur Originalrepräsentation herangezogen. (Stachowiak 1973, p. 138)

- demonstration models
- experimental models
- theoretical models
- operative models

For English summary of Stachowiak see: *https://modelpractice.wordpress.com/2012/07/04/model-stachowiak/*

All knowledge production happens in the context of models

> Der Gewinn dieser Vorgehensweise [Modellierung] liegt auf der Hand: Modellseitig gewonnene Einsichten und Fertigkeiten lassen sich – bei Erfülltsein gewisser Transferierungskriterien – auf das Original übertragen, der Modellbildner gewinnt neue Kenntnisse über das modellierte Original, er bekommt dieses besser als bisher in den Griff, kann es auf neue Weise zweckdienlich umgestalten oder als verbessertes Hilfsmittel für neue Aktionen verwenden. (Stachowiak 1973, p. 140)

Demonstration models

> Als Demonstrationsmodelle werden sie zur Veranschaulichung von (weniger anschaulichen oder unanschaulichen) Zusammenhängen benutzt, [...] . (Stachowiak 1973, p. 138)

> So können etwa Veränderungen sichtbar gemacht werden, die der Künstler während des Malens an einer Komposition vorgenommen hat. Auch Manipulationen, die von anderer Hand und zu einem späteren Zeitpunkt am Original vorgenommen wurden, können auf diese Weise „durchschaut“ werden.
>
> Link: Der Blick durch die Bilder: Alte Meister geröntgt, in: *arge-kunstgeschichte.de*, S. 2.

Experimental models i

> „ [...] als Experimentalmodelle dienen sie der Ermittlung oder Überprüfung von Hypothesen [...] “ (Stachowiak 1973, p. 139)

> Als Demonstrationsmodelle werden sie zur Veranschaulichung von (weniger anschaulichen oder unanschaulichen) Zusammenhängen benutzt, [...] . (Stachowiak 1973, p. 138)

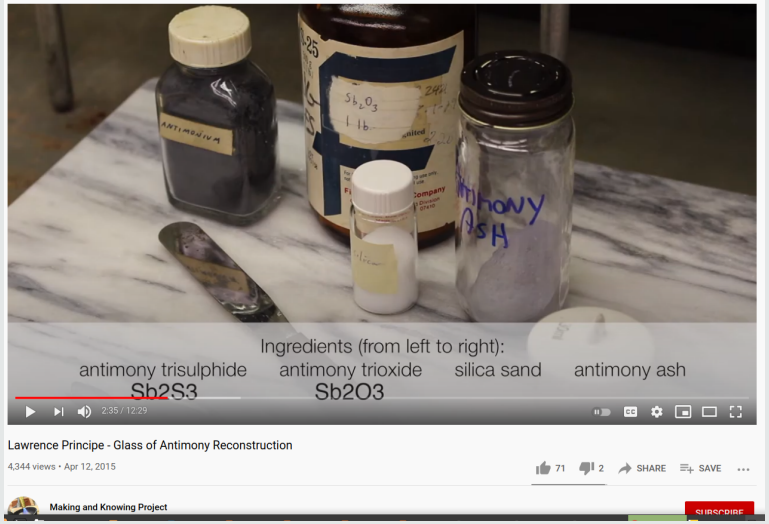
Example

1. **Hypothesis:** „The stronger the current, the lesser the pressure.“
2. **Testing the hypothesis** using experiments

Experimental models ii

Physics: Bernoulli effect

Recreating historical recipes



Theoretical models

Stachowiak

> „ [...] als theoretische Modelle vermitteln sie in logisch bündiger Form Erkenntnisse über Sachverhalte, [...] “ (Stachowiak 1973, p. 139).

1. **Rule:** All fish live in water.
2. **Example:** Gold fish Hermann is a fish.
3. **Result:** Hermann lives in water.

Logic

$p \vee \neg p$

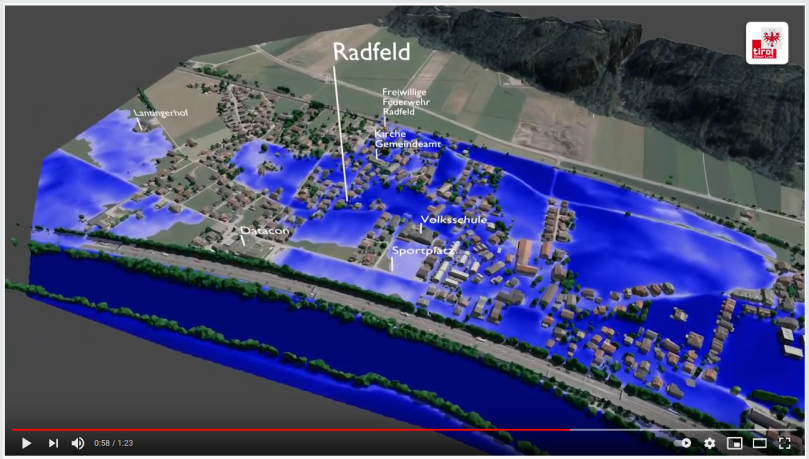
Operative models

> „Der wasserbauliche Modellversuch ist die beste Methode, um Vorhersagen zu treffen, wie es in der Natur aussieht, […] Die Ergebnisse der Untersuchungen sollen Aufschluss darüber geben, ob das geplante Einlaufbauwerk optimal dimensioniert ist und wann und wie die Verschlüsse zu öffnen sind, um die gewünschte Wirkung zu erzielen.
>
> *Hochwasserschutz bei Kramsach im Modellversuch*, in: ***meinbezirk.at***. Stand: 04.02.2017.

Stachowiak

> „ […] und als operative Modelle möglicher Zielaußenwelten stellen sie ihren Benutzern Entscheidungs- und Planungshilfen zur Verfügung.“ (Stachowiak 1973, p. 139)

Hochwasserschutz Radfeld



Homework: globe model

- the globe is a physical model of the world
- digital model = Google Maps
  - go on “Satellite“, zoom out.
  - Google Earth
- Which information or (abstract) knowledge is represented in those two models of the world?
- For which reasons or for what purpose can models (globe, Google Maps/Earth) be used to represent their original (the world)?

Homework

Take a globe or Google Maps/Earth, read chapters 2.1.3 of „Herbert Stachowiak: Allgemeine Modelltheorie“ (pdfs in the zip under ”LEKTUERE”, or in EN). If you can read it in German, please do so.

Try to integrate what you have learned from reading Stachowiak. Upload your solution (max. 1/2–1 page of text) to Moodle within the time period indicated. The text should be readable, not bullet points.

Reading for next week:

Stachowiak, all texts/PDFs (Nr. 02–04)

(ca. 10 pages)

General modelling theory II

Homework assignment solution: What knowledge is in a globe?

What information or (abstract) knowledge is embodied or encoded in a globe or Google Maps/Earth?

- form, rotation, tilt
- sizes, distances, topography: continents, bodies of water, mountains, vegetation
- Borders and names of: states, cities, streets, ...
- Places of Interest = POI (shops, traffic, tourism)
- Further information: population, traffic data, time zones

What can it be used for?

1. Navigation: plan your travel route
2. capture borders, distances and locations
3. explore the earth and its regions, places, etc.

How? (operating with the model)

1. directions / route planning form
2. turning
3. zooming in and out
4. checking or removing certain layers

3 proporties of a model following Stachowiak i

1) Mapping

> „Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können." „Der Abbildungsbegriff fällt mit dem Begriff der Zuordnung von Modell-Attributen zu Original-Attributen zusammen." (Stachowiak 1973, pp. 131–132)
> Models are always models of something, i.e. mappings from, representations of natural or artificial originals, that can be models themselves.

2) Reduction

> „Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant scheinen." (Stachowiak 1973, p. 132)
> Models in general capture not all attributes of the original represented by them, but rather only those seeming relevant to their model creators and/ or model users.

3 proporties of a model following Stachowiak ii

3) Pragmatism

> „Eine pragmatisch vollständige Bestimmung des Modellbegriffs hat nicht nur die Frage zu berücksichtigen, *wovon* etwas Modell ist [Abbildungsmerkmal], sondern auch, *für wen, wann* und *wozu* bezüglich seiner je spezifischen Funktionen es Modell ist.“ (Stachowiak 1973, p. 132)
>
> Models are not uniquely assigned to their originals per se. They fulfill their replacement function
>
> a) for particular – cognitive and/ or acting, model using subjects,
>
> b) within particular time intervals and
>
> c) restricted to particular mental or actual operations.

Mapping using attributes

Attributes versus predicates

> „ […] alle Attribute [sind] als primär perzeptiv-kogitative Gebilde wohlzuunterscheiden von ihren sprachlichen – gesprochenen oder geschriebenen oder sonstwie symbolisierten – *Artikulationen*.“ (Stachowiak 1973, p. 135)
>
> „Es sei angenommen, daß es zu jeder Attributklasse wenigstens *eine* sie elementweise repräsentierende Prädikatklasse gibt.“ (Stachowiak 1973, p. 137)
>
> → each attribute of the original has a predicate class in the model representing it.

Attributes = properties

> „Unter Attributen sind Merkmale und Eigenschaften von Individuen, Relationen zwischen Individuen, Eigenschaften von Eigenschaften, Eigenschaften von Relationen usw. zu verstehen.“ „Hiernach sind einem beliebigen Original stets wohlunterscheidbare, nicht weiter zu zerlegende Teilobjekte zuzusprechen, die potentielle oder tatsächliche Träger von Eigenschaften sind und denen gegebenenfalls eine Relationenstruktur aufgeprägt werden kann.“ (Stachowiak 1973, p. 134)
>
> → properties should ideally be atomic

Mapping = assigning attributes between modell and original

> „Der Abbildungsbegriff fällt mit dem Begriff der Zuordnung von Modell-Attributen zu

Mapping in the globe modell

Erde (Attribute des Originals)	repräsentiert/erfasst durch	Globus (Attribute des Modells)
Hat angenähert Form einer Kugel	▭	Hat die Form einer Kugel
Dreht sich um eine eigene Achse	▭	Dreht sich
Element des Sonnensystems	▭	Neigung der Rotationsachse
Wasser- und Landflächen	▭	Farben (Blau, Nicht-Blau)
Vegetation	▭	Farben (Weiß, Grün, Gelb)
Relief (Topographie)	▭	Schattierungen
Ländergrenzen	▭	Schwarze gestrichelte Linien
Länder- und Hauptstadtnamen	▭	Wörter
Koordinatensystem	▭	Längen- und Breitengrade

Discussion:

Which originals would you like to model?

Which attributes of the original are to be represented?

What reasons are there for that model?

Reading for next week:
McCarty, Modelling

Data modelling

Data Modelling: Stachowiak's *General Model Theory* i

His notion of a model „Alle Erkenntnis ist Erkenntnis in Modellen oder durch Modelle und jede menschliche Weltbegegnung überhaupt bedarf des Mediums Modell“ (Herbert Stachowiak 1973) → All knowledge-making is knowledge-making in or through models and all human perception of the world needs models as a medium.

Data modelling Model = snippet of the real world but it only covers the attributes I chose to be relevant for the task at hand. Thus, the model and the aspect of the real world it models (its subject) diverge.

Model = abstraction (=class) concrete applications of the model (i.e. one concrete cooking recipe, poem, etc.) are called *instances*.

Standardized models allow us to exchange and analyse data, search/query data. → only **formal models** can be processed digitally, i.e. every digital model is a formal model.

Data Modelling: Stachowiak's *General Model Theory* ii

1) Mapping Models are always models of something, i.e. mappings from, representations of natural or artificial originals, that can be models themselves.

2) Reduction Models in general capture not all attributes of the original represented by them, but rather only those seeming relevant to their model creators and/ or model users. For example, I can only photograph (or even see!) a statue from one perspective at a time. I never see 'the thing itself' but a partial aspect of all its defining properties.

3) Pragmatism Models are not uniquely assigned to their originals per se. They fulfill their replacement function

a) for particular – cognitive and/ or acting, model using subjects,
b) within particular time intervals and
c) restricted to particular mental or actual operations.

Willard McCarty, „Modeling: A Study in words and Meanings“ (2004)

1. A model is a representation of something, for example for research purposes.
2. Modelling = hermeneutic process by which models are created and used to solve problems.
3. *model of* (descriptive) vs. *model for* (i.e. architectural plan for building a house).
4. modelling as an experimental process with the goal of capturing aspects of reality

Modelling as an interative process

- include present knowledge
- create model
- use for model to research the subject
- learn from discrepancies between model and real world to refine your model.
- re-make the model to integrate new insights
- repeat

Models = simplified repres. of parts of the real world i

Model = mapping = representation $\neq$ original: just parts of the original are represented. Choosing which parts depends on the intended use by which you later measure how useful (i.e. good) the model is. Or is the best model the most accurate?

subjective & abstracted $\rightarrow$ the model has to be created with reference to a specific research question and will only be useful (or the most useful) with regard to this question. Perception and representation are always selective & subjective. Thus, so are the models. I can never capture all aspects at the same time. A photo of a vase only shows it from one side.

Metadata can be used to enrich data models (supplying additional information, such as administrative or technical)

Models = simplified repres. of parts of the real world ii

Disambiguation → often necessary to represent something in the digital sphere (practical aspect of modelling), for example by using normalization, geonames, GND etc. → conventions so that we can be sure we are referring to the same thing on the internet.

Where does the realistic ‘mapping’ of reality end? At which point are we left with only a subjective perspective of reality ?

Models = simplified repres. of parts of the real world iii

What do we even mean by original? The book whose materiality gets lost in the digital realm? The idea behind the book?

Example Photogrammetry / Structure from Motion

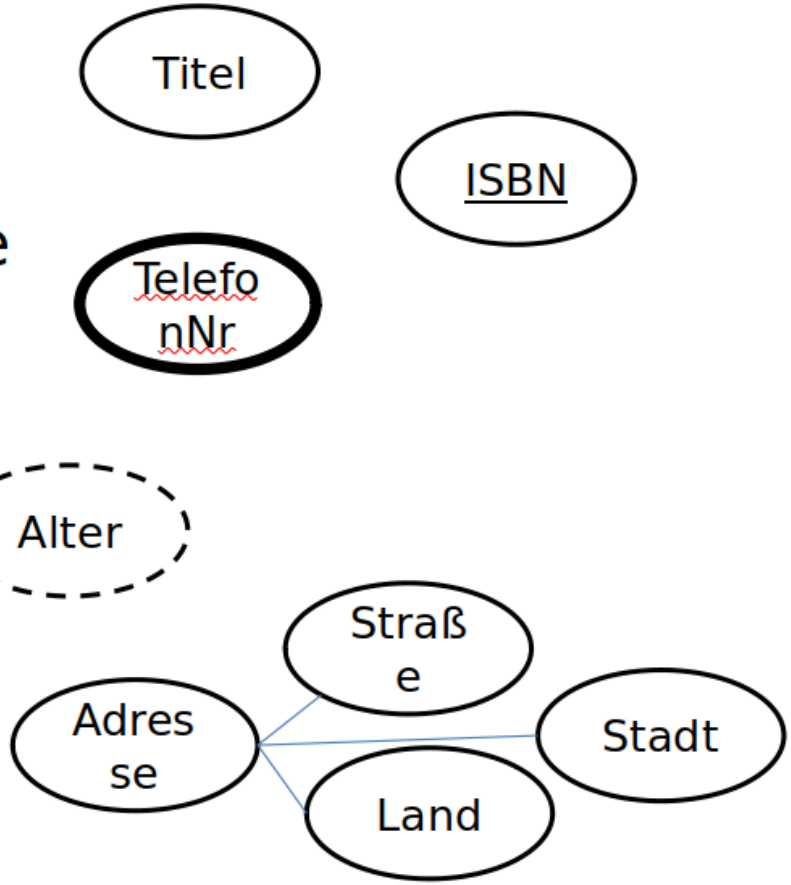
(= creating 3D models from overlapping photos):
reality → more info than on the images.
3D model: you need 8 Pixels (the same pixel on 8 different images) for one point in the model → less information than the sum of the (at least 50) images used to create the model from.

Readings – summary:
McCarty, Modelling

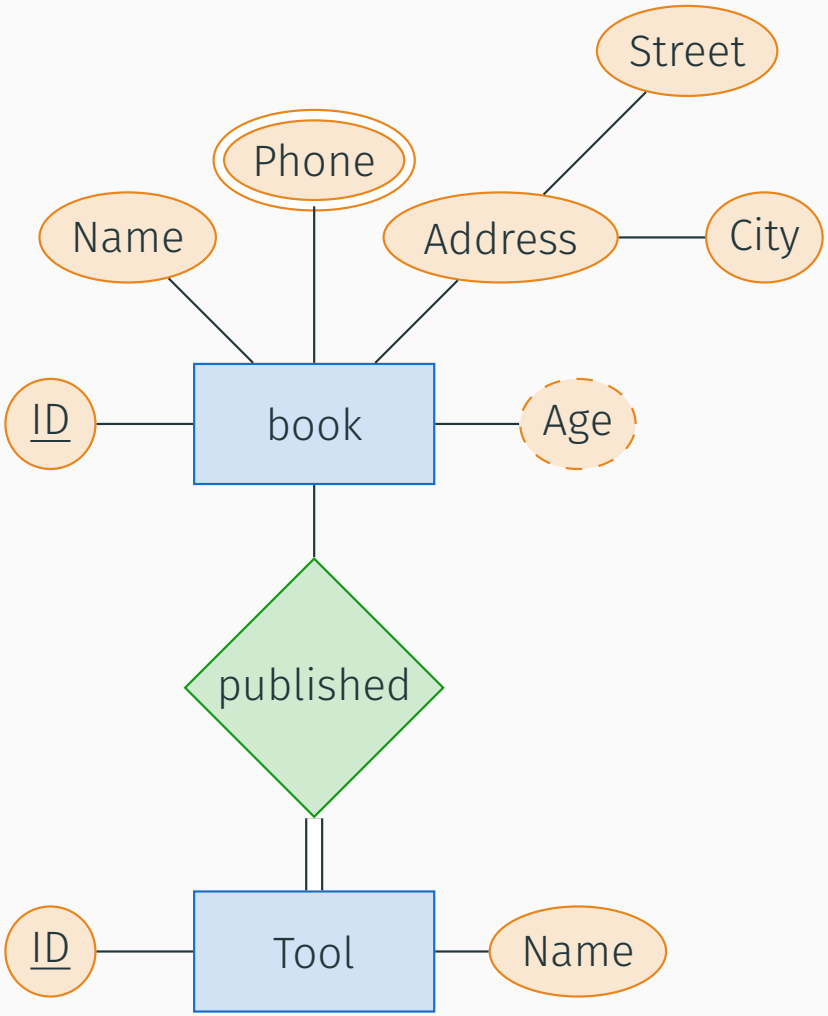
Please create a 1 slide summary for your part per group; designate 1 person to present to the group.

ER erweitert i

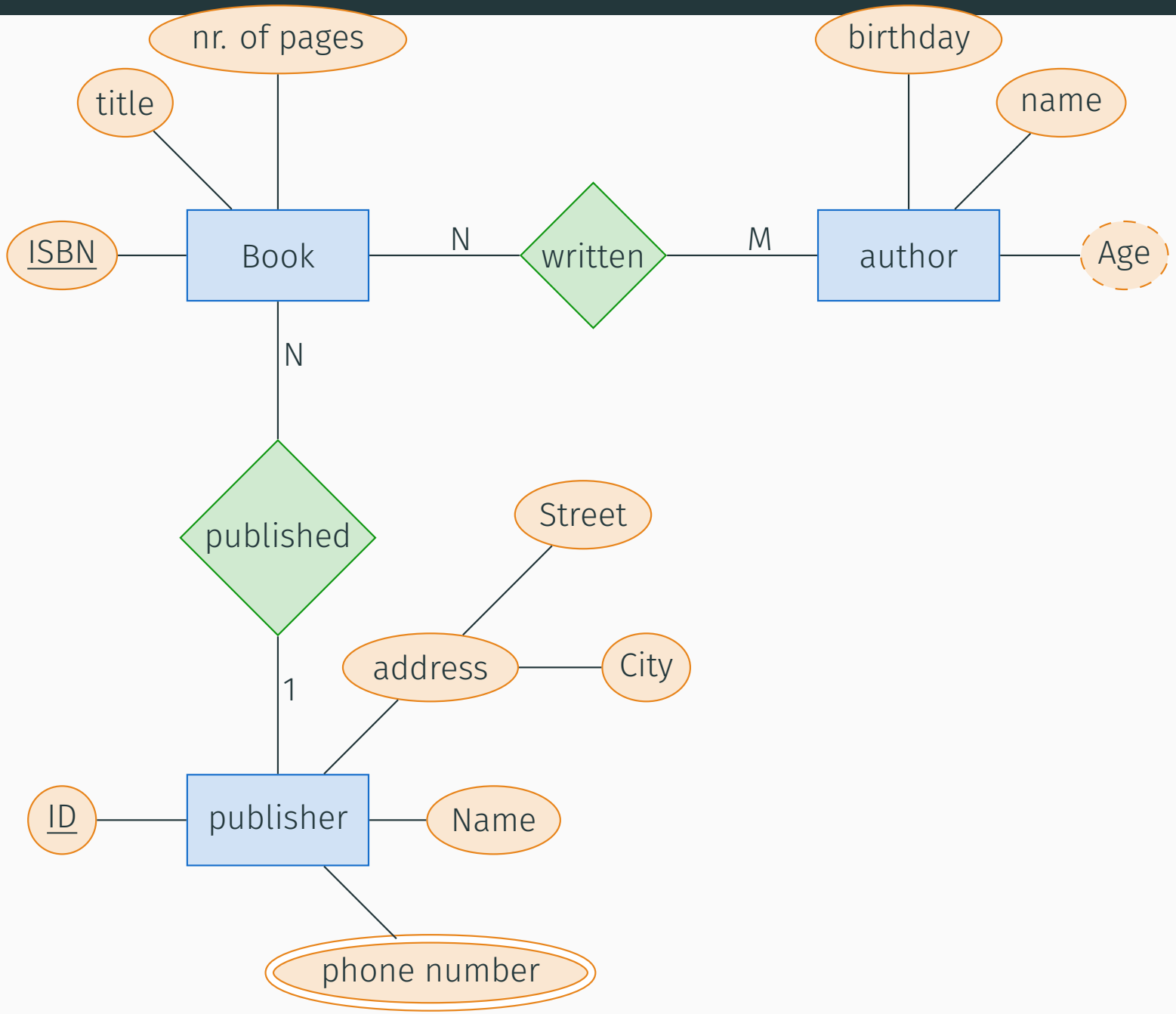
- Attribute
- Unique/Key Attribute
- Multi-valued Attribute
- Derived Attribute
- Composite Attribute



ER erweitert ii



ER erweitert iii



Entity-Relationship Model (ERM)

Das Entity-Relationship Model nach Peter Chen

Chen 1976, p. 9

> The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model.

Chen 1976, p. 10

> [Der Text] analyzes the network model, the relational model, and the entity set model, and describes how they may be derived from the entity relationship model.

Chen 1976, p. 19

> [Der Text stellt vor:] a diagrammatic technique for exhibiting entities and relationships: the entity-relationship diagram.

→ Konzeptuelles Modell

ER-Begriffe (Chen 1976, pp. 9–12)

Entität (Entity)

An entity is a „thing“ which can be distinctly identified. A specific person, company, or event is an example of en entity.

Beziehung (Relationship)

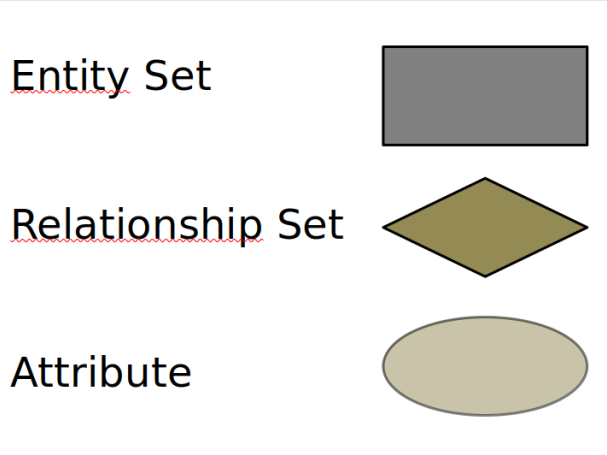
A relationship is an association among entites. For instance, „father-son“ is a relationship between two „person“ enitities.

Entitätsmenge (Entity Set)

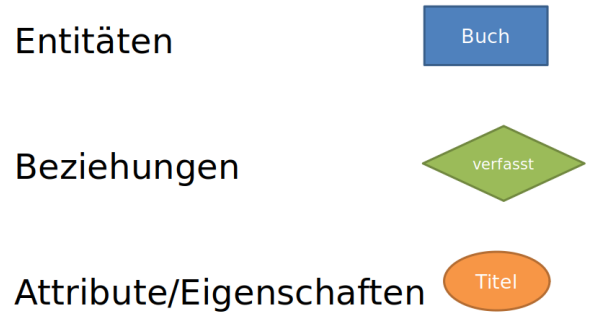
Alle Entitäten derselben Entitätsmenge haben dieselben Eigenschaften (properties).

„Entities are classified into different entity sets such as EMPLOYEE, PROJECT, and DEPARTMENT.“

ER-Modell-Bestandteile



ER-Modell-Bestandteile



ER-Begriffe (Chen 1976, pp. 9–12)

Menge von Beziehungen (Relationship Set)

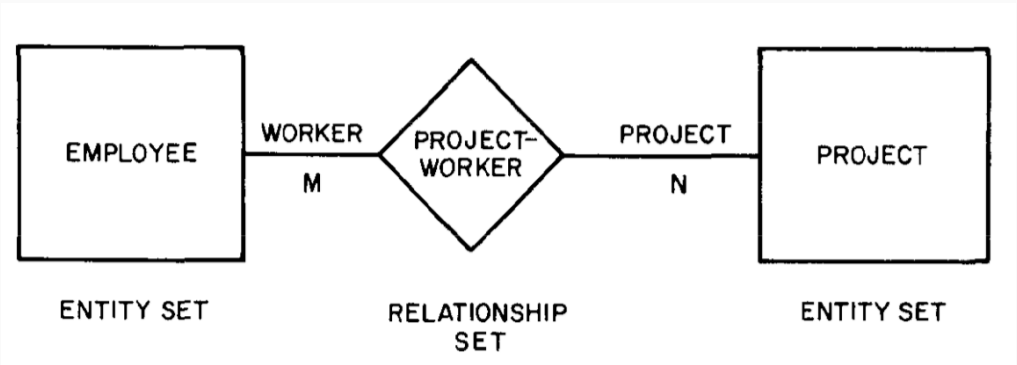
Gleiche Beziehungen zwischen Entitäten, die jeweils aus derselben Entitätsmenge stammen, lassen sich in einer Menge von Beziehungen zusammenfassen.

Rolle (Role)

The role of an entity in a relationship is the function that it performs in the relationship.

Eigenschaften (Properties)

Mithilfe von Eigenschaften lassen sich Entitäten (Entities) und Beziehungen (Relationships) beschreiben.



Attribut-Wert-Paare und Primary Keys

Eigenschaften (Properties) werden stets durch eine Kombination von Attribut und Wert ausgedrückt. Beispiel:

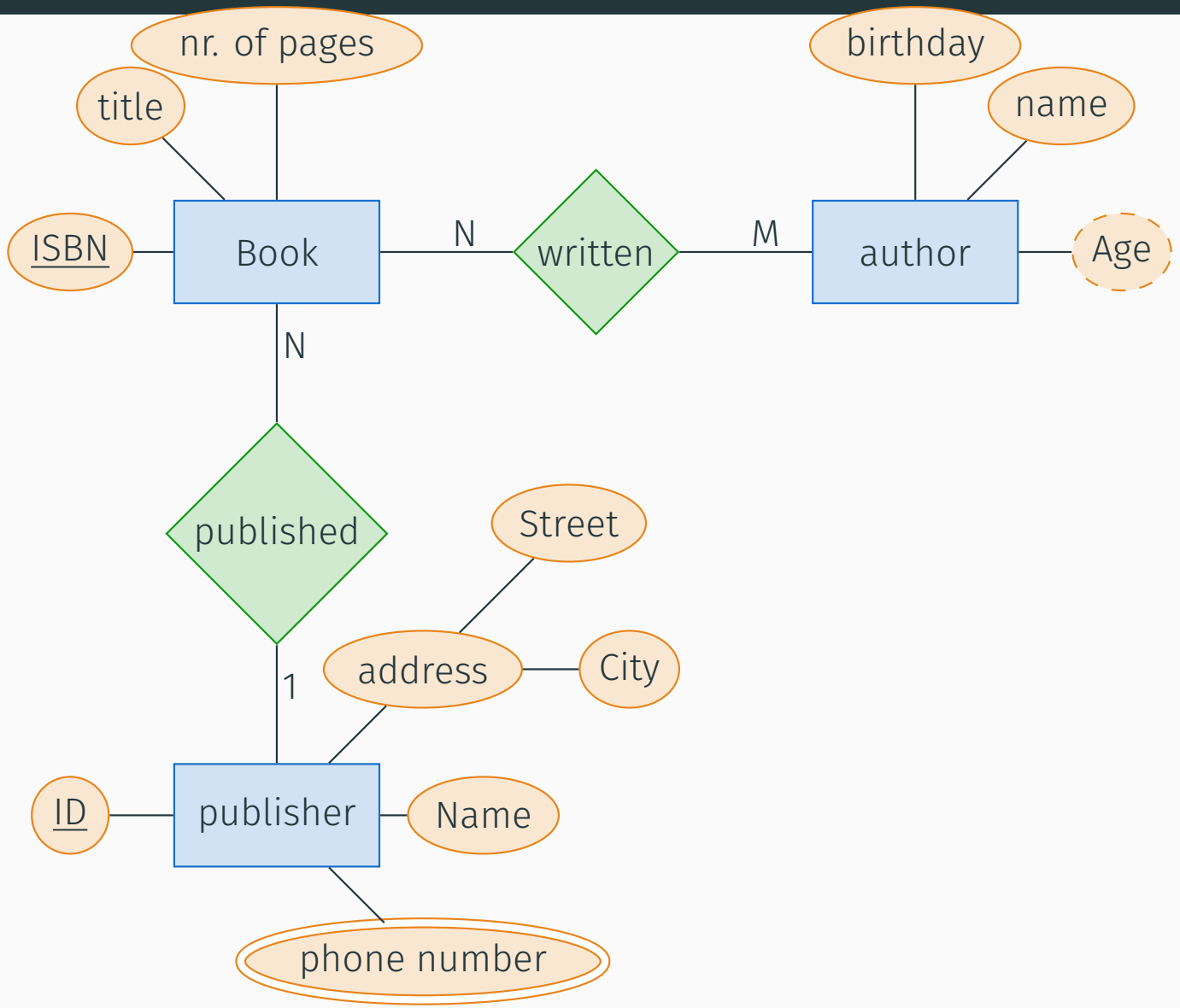
1. Der Stift ist blau = Die Entität ‘Stift’ hat das Attribut „Farbe“ mit dem Attributwert „blau“
2. Der Stift ist rot = Die Entität ‘Stift’ hat das Attribut „Farbe“ mit dem Attributwert „rot“

Primary Key: Identifizierende Eigenschaften

Entitäten und Beziehungen lassen sich über den Wert eines oder mehrerer Attribute identifizieren. Das oder die identifizierenden Attribute werden als Primary Key bezeichnet.

Beispiel ER-Modell für ein Buch i

Beispiel ER-Modell für ein Buch ii



Beispiele

Entitäten: Unikurs

- **Entitäten:** Kurs, Dozent:in, Studierende, Raum
- **Attribute:** Kurs (Nr, Titel), Dozent:in (Name), Studierende (Name, Matrikelnummer, Studium, Fach), Raum (Nr, Gebäude, Sitzplätze)
- **Relationen:**
  - Ein Kurs wird abgehalten von Dozent:in.
  - Er findet in einem Raum statt.
  - Die Studierenden nehmen am Kurs teil.

CSV: Geburtsort

```
"Name","Address","Description","Longitude","Latitude"
"Vorname Nachname","Teststr. 1, PLZ","Geburtsort","7.018242","50.678667"
```

(Materialien für die) Hausübung i

Ressourcen

- Tool zum Entity-Relationship-Modelle erstellen
- GeeksforGeeks-Tutorial/Intro to ER Models
- DARIAH Geobrowser

Angabe HÜ ER-Model

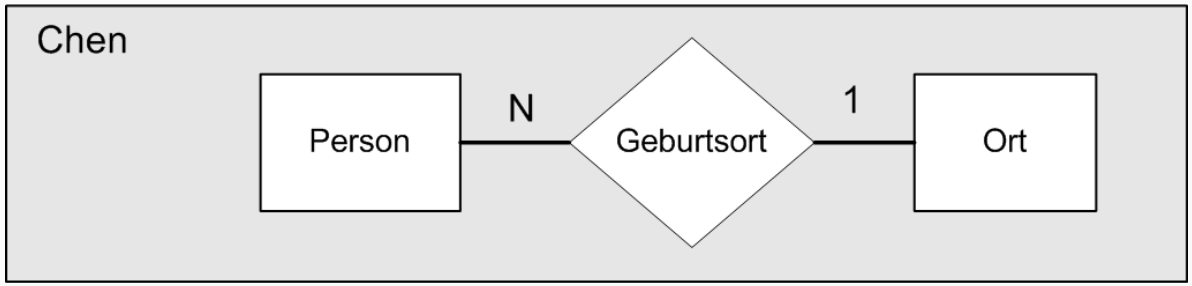
1. Konzipieren und visualisieren Sie das Model Ihres Originals (Attributklasse) mithilfe des Entity-Relationship-Diagramms. (z.B. ERDplus siehe oben)
2. Exportieren Sie ihr Diagramm als Bild und laden es auf Moodle hoch.
3. Bei Problemen können Sie sich auch gegenseitig im Hilfe-Forum unterstützen.

Kardinalitäten i

”Kardinalität” = Anzahl der an einer Beziehung beteiligten Entitäten
Dabei stehen *n* und *m* für eine beliebig hohe Anzahl.

Typen von Relationen

1:1 → Es gibt genau je eine Entität, die an der Beziehung beteiligt ist.
1:n → Für jede Entität auf der 1-Seite der Beziehung kann es beliebig viele Entitäten auf der n-Seite geben.
n:m → Für jede Entität auf der n-Seite der Beziehung kann es beliebig viel Entitäten auf der m-Seite geben und umgekehrt.

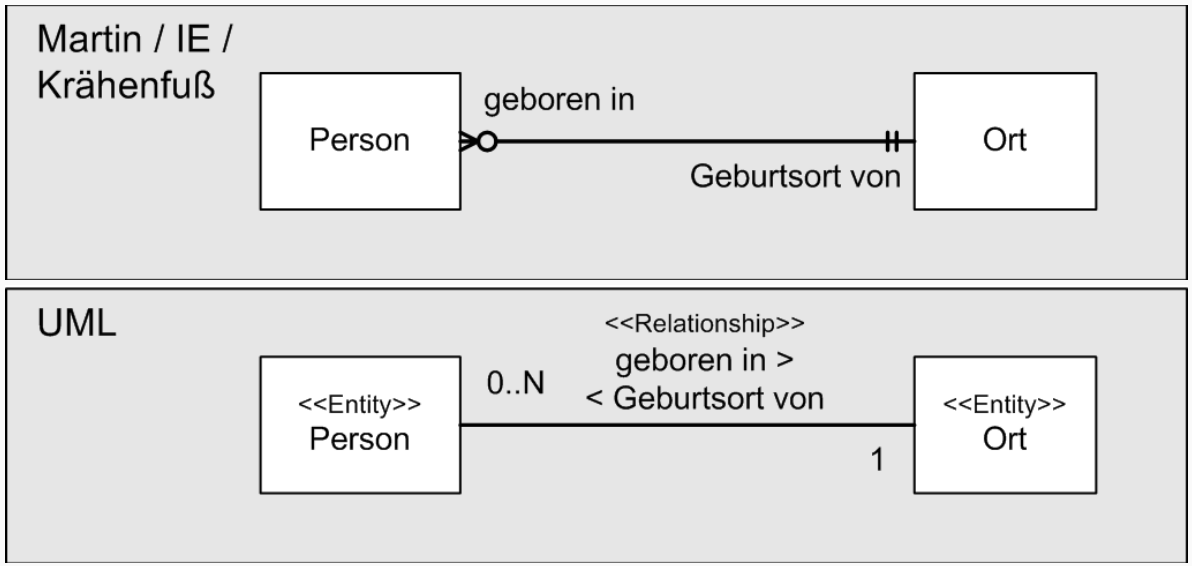


Kardinalitäten ii

Minimale Kardinalitäten

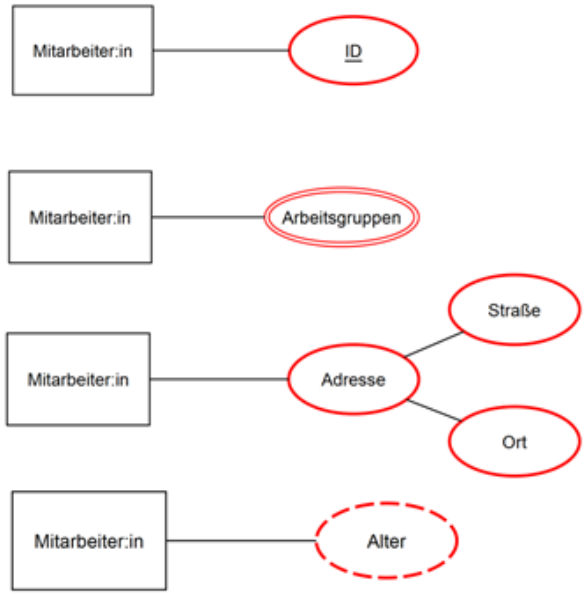
0 $\rightarrow$ Es muß keine Entität auf dieser Seite der Beziehung geben. (=*optional*)

1 $\rightarrow$ Es muß mindestens eine Entitäten auf dieser Seite der Beziehung geben. (=*verpflichtend*)



Attribute

- Schlüssel
- Mehrwertige Attribute
- Zusammengesetzte Attribute
- Abgeleitete / Berechnete Attribute



ER erweitert i

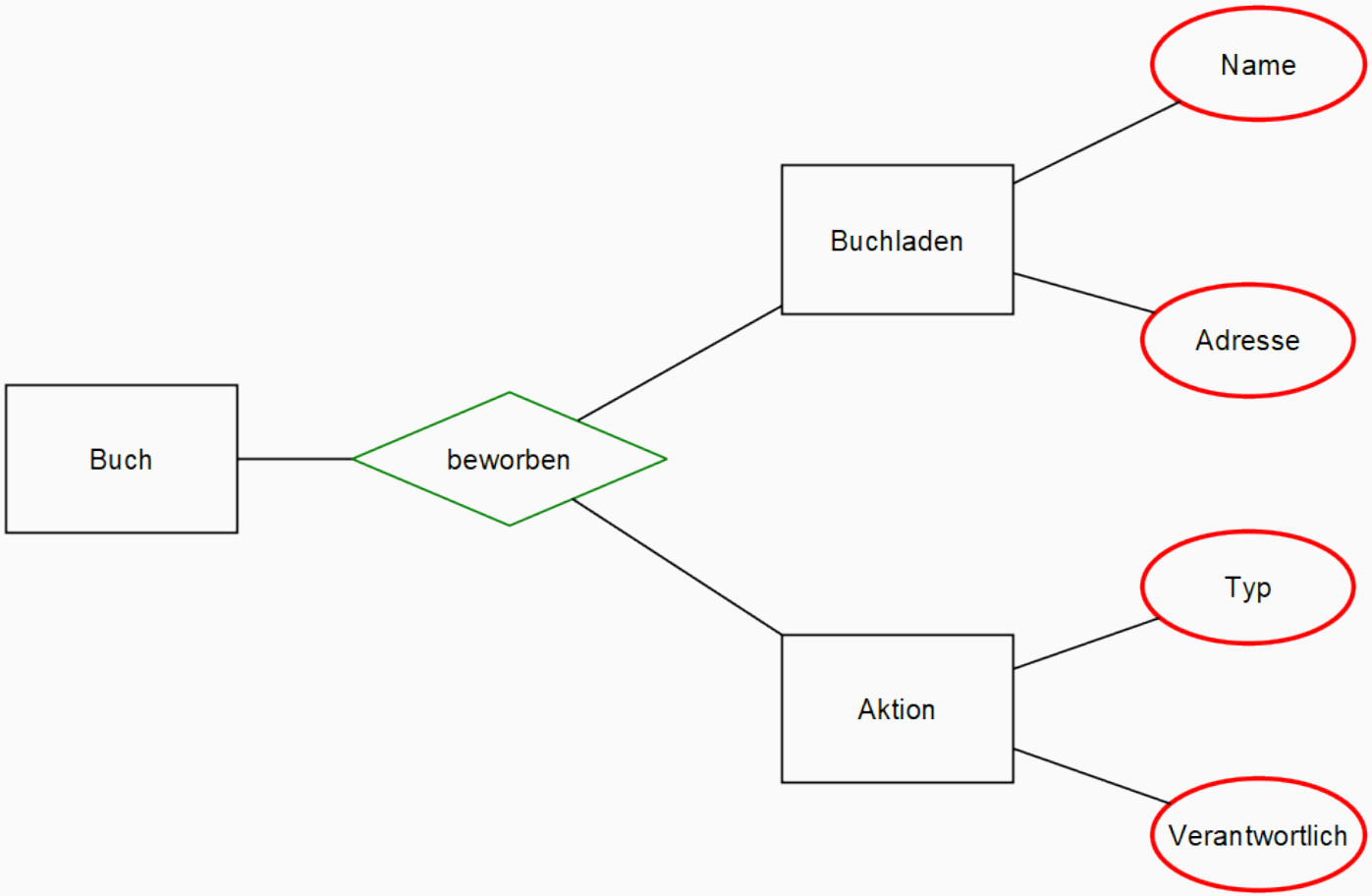
- **identifizierende Attribute (“Schlüssel“)**: Innerhalb der Entitätsklasse gibt es die Werte dieses Attributs nur genau einmal. Sie können damit die einzelne Entität identifizieren.
- **Mehrwertige Attribute**: Attribute, die mehr als einen Wert enthalten können.
- **abgeleitete Attribute**: Attribute, die sich aus einem anderen Attribut berechnen.
- **zusammengesetzte Attribute**: Attribute, die aus mehreren Teilen bestehen.

- **n-äre Beziehungen**: ein Relationship-Set kann prinzipiell mehrere Entity-Sets mit einander verbinden.

ER erweitert ii

- **Schwache Entität**: Kann nur existieren, wenn eine Relation zu einer anderen Entity existiert (Raum in Gebäude), Schlüssel ist abhängig vom Schlüssel einer Relationship (Chen-Notation: Doppelter Rand). Besitzen keine Eigenschaften, anhand derer man sie identifizieren könnte (besitzen keinen Primary Key). Sie werden nur über die Beziehung der schwachen Entität zu einer starken Entität identifiziert. Beispiel: der Angehörige eines Mitarbeiters eines Unternehmens

n-äre Beziehungen



Übung ER-Modell

Eine Kurseinheit einer Lehrveranstaltung

Eine Kurseinheit besteht aus folgenden Enitäten:

1. Studierende
2. Lehrende
3. Sitzplatz
4. Kursraum
5. Gebäude

Übung

1. Mit welchen Attributen lassen sich diese Entitäten beschreiben?
2. Anhand welcher Attribute lassen sich die Entitäten jeweils identifizieren?
3. Welche Beziehungen existieren zwischen den Entitäten?

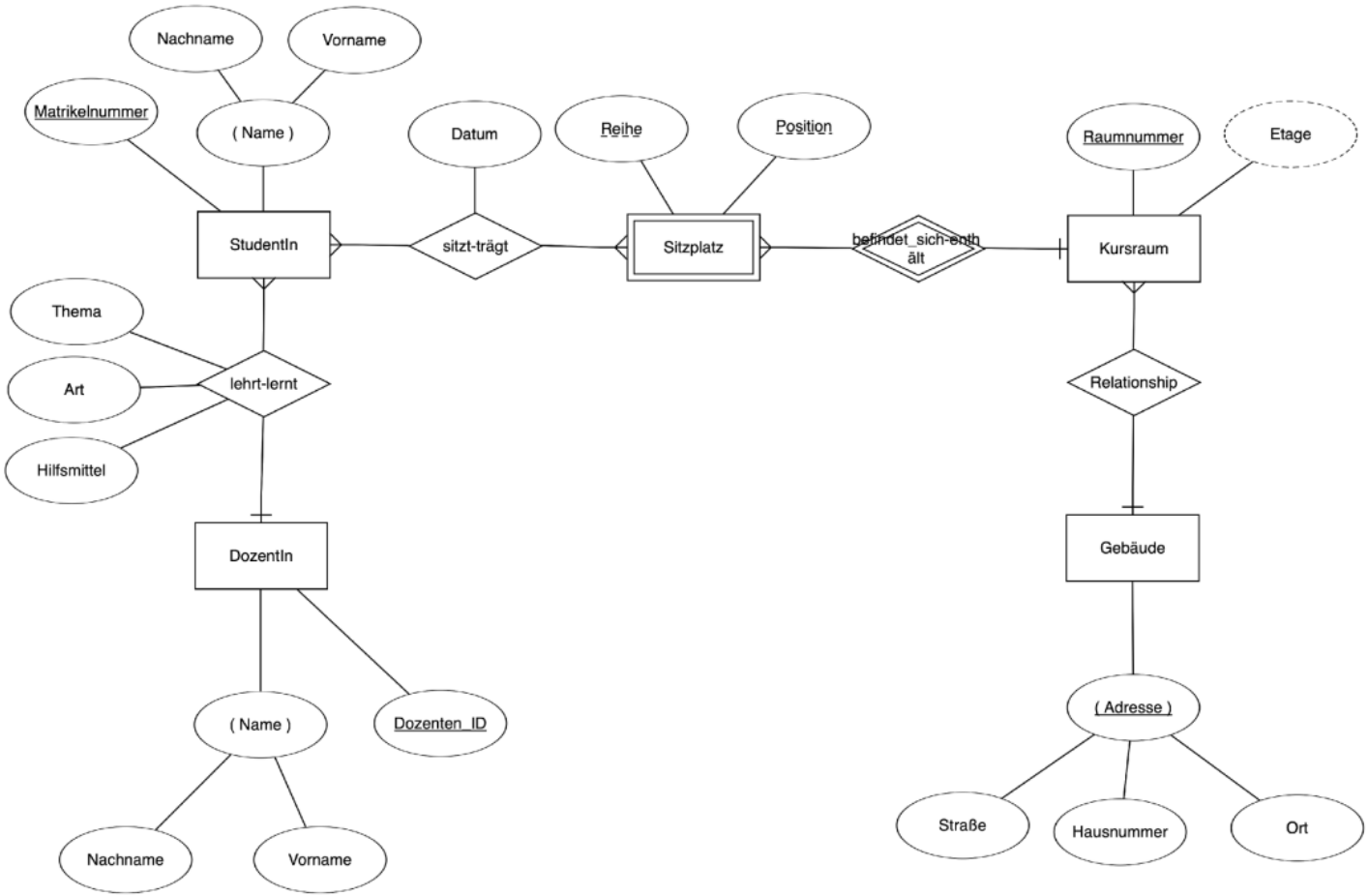
→ bezogen auf normalen Präsenzunterricht

Übung 2

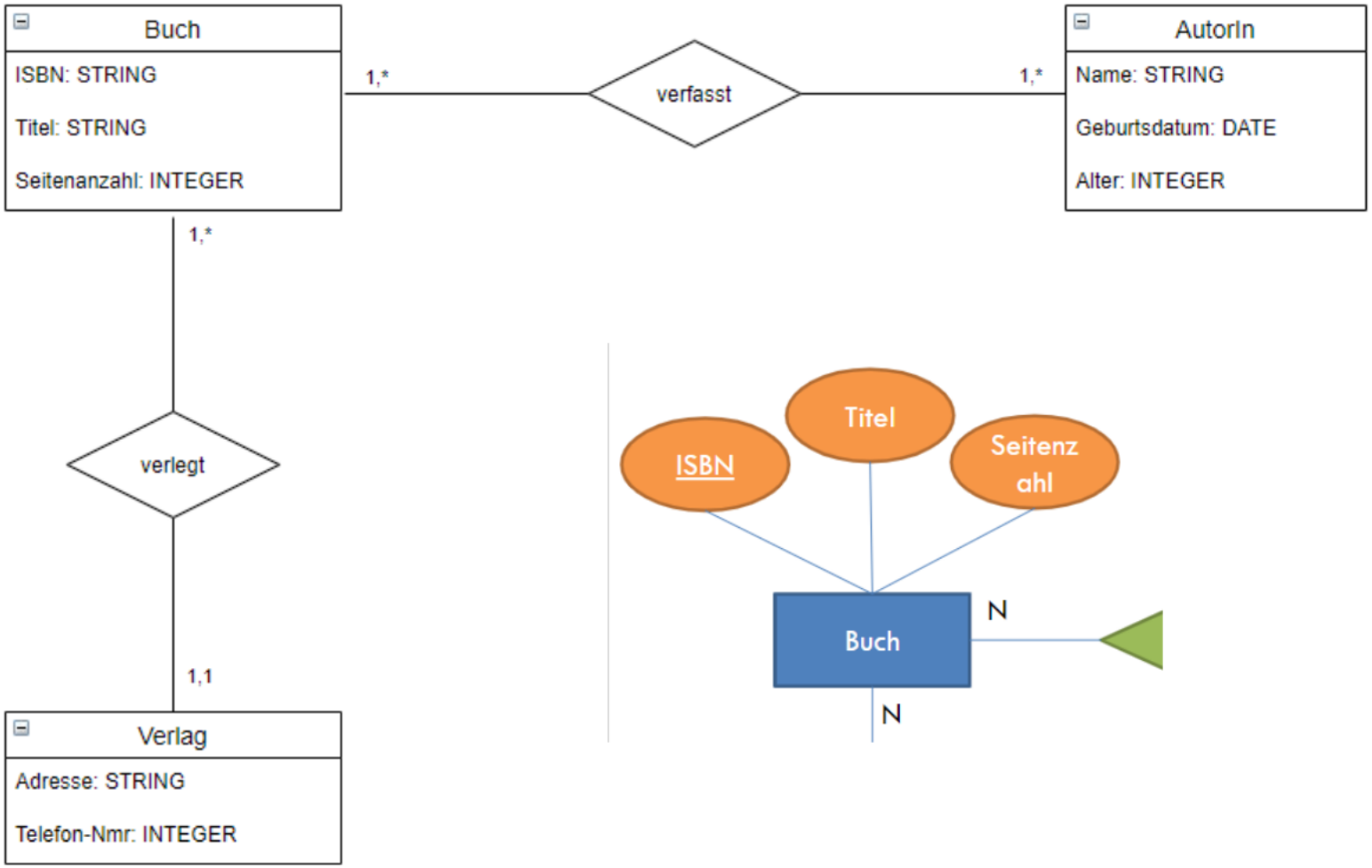
Wie müsste man das jetzt umändern, um auch Distance Learning einzukalkulieren?

→ Übung in Gruppen. Erstellen Sie, wenn möglich, eine Zeichnung / einen Screenshot der Resultate, der dann vorgestellt wird.

Musterlösung Kurseinheit als ER



ER Notationsformen



Repräsentation von ER-Modellen als Tabellen

Repräsentation als Tabellen

Entitäten und deren Beziehungen können als Tabellen repräsentiert werden.
Die Attribute sind dann die Spalten

Umgang mit Kardinalitäten

1. Gleiches Buch aber mehrere Autoren? (Wieso nicht einfach als Attribut?)
2. Gleiches Buch mehrere Genre? (Entität oder Attribut)
3. Wie würde ich den Inhalt eines Buches modellieren?

Entitäten und Attribute in Tabelle(n) überführen: Bsp. Kursmodellierung

	A	B	C	D	E	F	G
1	Matrikelnummer	Vorname	Nachname	Sitzplatz	Kursraum	Kursnummer	Kurstitel
2	20001239			25	1.15	562.021	Grundlagen der Datenmodellierung
3	20081210			18	1.15	562.021	Grundlagen der Datenmodellierung
4	20092317			11	1.16	562.021	Grundlagen der Datenmodellierung
5	20101670			12	1.15	562.021	Grundlagen der Datenmodellierung
6	20001239			8	1.25	562.026	Einführung in die Webentwicklung
7	20092317			5	1.25	562.026	Einführung in die Webentwicklung
8							

```
2
3  Matrikelnummer;Vorname;Nachname;Sitzplatz;Kursraum;Kursnummer;Kurstitel
4  20001239;[illegible]25;1.15;562.021;Grundlagen der Datenmodellierung
5  20081210;[illegible];18;1.15;562.021;Grundlagen der Datenmodellierung
6  20092317;[illegible];11;1.16;562.021;Grundlagen der Datenmodellierung
7  20101670;[illegible]12;1.15;562.021;Grundlagen der Datenmodellierung
8  20001239;[illegible]8;1.25;562.026;Einführung in die Webentwicklung
9  20092317;[illegible];5;1.25;562.026;Einführung in die Webentwicklung
10
```

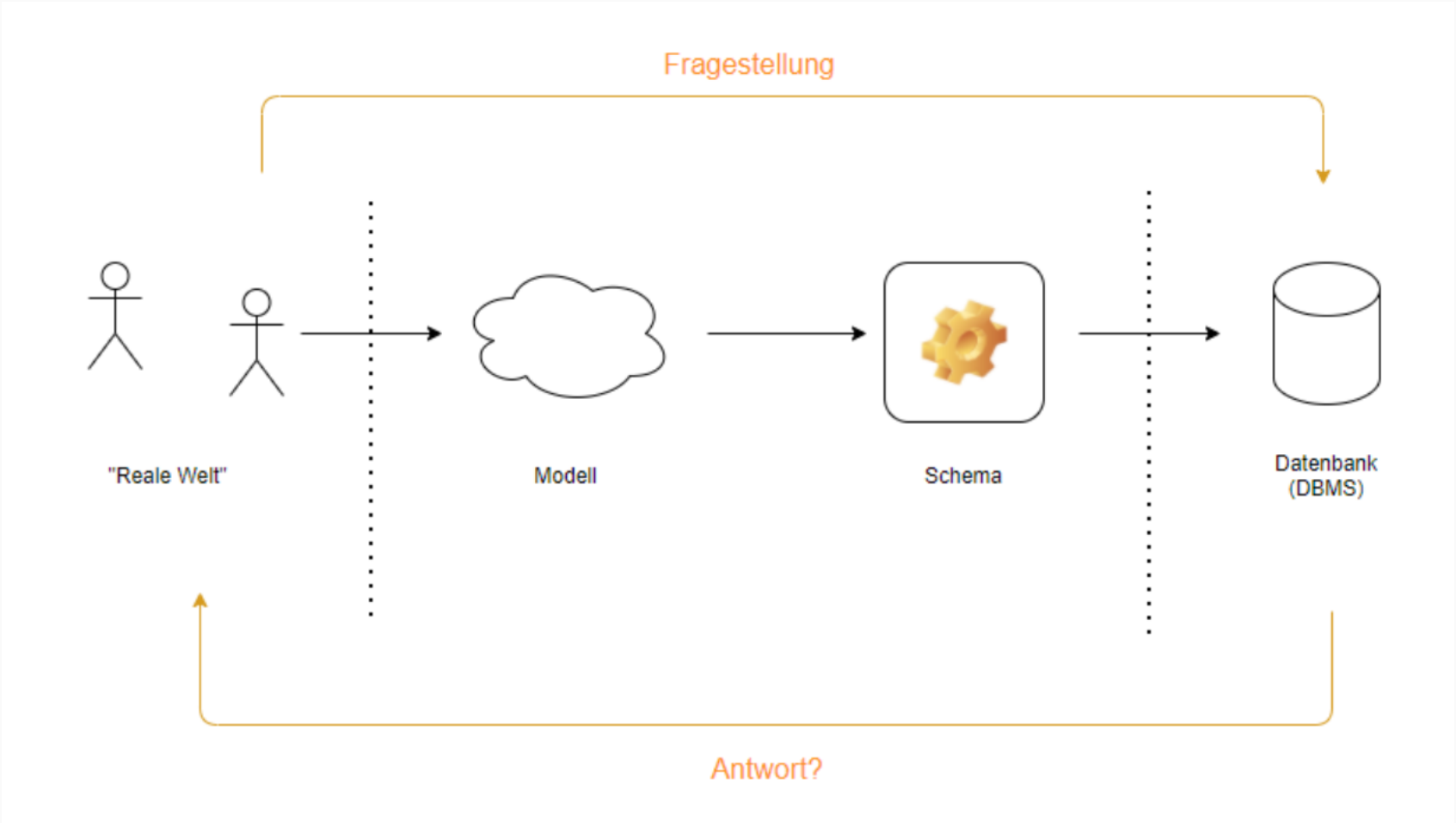
Repräsentation von ER-Modellen als Tabellen i

- Entitätsklasse → Tabelle
- Attributklasse → Spalte
  - mehrwertige Attribute → eigene Tabelle
- Relation
  - 1:1 → Schlüssel der einen Entität (Tabelle) wird als Referenz (”foreign key”/”Fremdschlüssel”) in der anderen Tabelle eingefügt (eigene Spalte).
  - 1:n → Schlüssel der 1-seitigen Entität (Tabelle) wird als Referenz (”foreign key”/”Fremdschlüssel”) in der n-seitigen Tabelle eingefügt (eigene Spalte)
  - n:m → eigene Tabelle, die die Schlüssel der beiden Entitäten als Spalten enthält
  - n-äre Relationen → eigene Tabelle, die die Schlüssel aller Entitäten als Spalten enthält

Repräsentation von ER-Modellen als Tabellen ii

- Attribute zu Relationen $\rightarrow$ eigene Tabelle, die die Schlüssel der beteiligten Entitäten als Spalten und Spalten für die Attribute enthält
- mehrwertige Attribute $\rightarrow$ eigene Tabelle, die den Schlüssel der Entität als Fremdschlüssel enthält (also wie eine 1:n-Relation modelliert wird) und eine Spalte für das Attribut enthält.
- zusammengesetzte Attribute $\rightarrow$ eigene Tabelle, die den Schlüssel der Entität als Fremdschlüssel enthält (also wie eine 1:n-Relation modelliert wird) und Spalten für die Teilattribute enthält.

Vom realweltichen Ding zur relationalen Datenbank?



Der Übergang zur Datenbank

Datenanalyse Bsp: Tabelle aus Orten

- Sind die Koordinatenpaare in der Tabelle einzigartig (oder gibt es Dopplungen)?
- Gruppiere und zähle die Orte nach Anzahl der Nachkommastellen des Längengrades etc.
- Was gibt es für Abfragen an Visual-isierungsmöglichkeiten?

Vorschau: Themenblock relationale Datenbanken

1. **Informations- und Datenmodellierung** = Der Weg in die Datenbank
2. **Datenanalyse** = Informationen aus der Datenbank wieder herausholen, um Fragestellungen zu beantworten / Strukturierte Information repräsentiert durch Daten abfragen

Hausübung: Installation von SQLite

ggf. in Gruppen zusammenfinden nach Betriebssystem (Linux, Windows, Mac)

wer es schon hat, kann gehen

von mir aus braucht man nicht den Browser, sondern nur SQLite

wer es noch nicht hat, jetzt geht und dann doch Probleme hat, muss es sich selbst organisieren

$\rightarrow$ Möglichkeit bei der Installation Hilfe zu bekommen, falls was nicht klappt

Tutorials: Tutorialspoint | SQLiteTutorial | SQLiteBrowser Github

Data modelling II

Re:

Data modelling

(but more technical now)

Data modelling

Representation of originals (attributes)

- image (photo, drawing, diagram etc.)
- text
- lists, tables
- audio
- objects (experimental setup, sculpture, 3D print, etc.)

What is data modelling?

- **data:** What does ‘data’ mean?
- **modelling:** What does ‘data modelling’ mean?

Data i

Definitions of ‘data’

1. Plural of Latin *datum* (’given’): data often isn’t exactly a given but a construct: data is created rather than given
2. [number]values, findings features, statements, details (collected/created through observation, measurement, statistical polls, phenomenotechnical devices)
3. (IT) digital numbers, information
4. (maths) values given to solve an equation

ISO definitions = standard

- ISO/IEC 2382:2015(en): Information technology – Vocabulary 2121272 data
- **Reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing**
- Note 1 to entry: Data can be processed by humans or by automatic means (source)

Data models

> Data models are models in the sense of Stachowiak's three main features of a mode but they have one more property: Models generally aren't necessarily computer-processable. To achieve this, they need to be given in an unambiguous, explicit way. Only then can they be machine-processed and only then are they formal models. (100 Jannidis 2017, translated from German)

Digital data representation i

→ i.e. computer processable

Digital representation

- images
  - raster graphics (*.png*, *.jpeg*)
  - vector graphics (*.svg*)
- Text
  - plain text (*.txt*)
  - formatted text (*.docx*)
- Lists, tables (*.csv*, *.xlsx*)
- Audio (*.wav*, *.midi*)
- **Objects:** (Simulation of) three dimensionality and physical attributes

Digital data representation ii

For advanced froms of digital representation → see other ZIM classes

- markup languages/OHCO (`.xml`, etc.)
- data objects (`.json`, etc.)
- graph databases (`.rdf`, etc.)
- relational databases → SQL (will follow soon...)

Fotis Jannidis' phases of data modelling

1) conceptual data model

1. identify relevant entities and their attributes and relations → relevant concerning their intended use
2. visualize entities, attributes and relations in a suitable format → lists, tables, diagrams (e.g. Entity Relationship Diagramm)

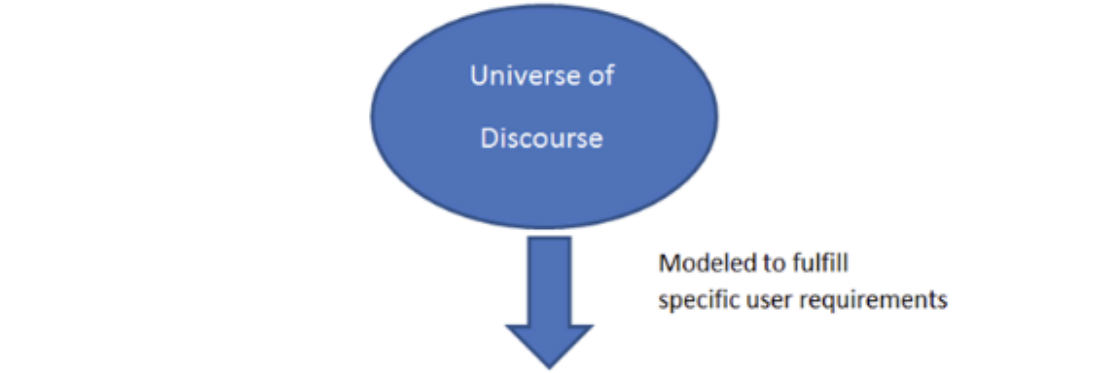
2) logical data model

Map the conceptual model onto the structure of a specific technology, e.g. database schema or XML schema

3) physical data model

Implementation and physical representation of the data model in the memory of a computer as a specific data structure

Jannidis, example diagram



Modeled Instance	Data Model	Meta Model	
	UML-diagram ER-diagram	UML Entity Relationship-Model	Conceptual Model
Table TEI encoded text file data set using an ontology	Structure of a table XML schema Ontology expressed in an RDF Schema	Relational Model XML-Schema-Language RDF-Schema-Language	Logical / mathematical Model
For example: data structure plus data in a database file	Physical Model / Data Structure	For example: An algorithm for database indexes	Physical Model

(Flanders / Jannidis 2012, S. 11)

Example: csv data format (=comma separated values)

1. simple exchange format for sheet calculation software
2. cells (attributes) are separated by a separator character (such as comma, semicolon, tab, etc.)
3. entries (datasets) separated by newline/line break
4. fields can be named using headers (optional)

Source: *ietf.org*, *https://tools.ietf.org/html/rfc4180*

csv example: birth place

```
"Name","Address","Description","Longitude","Latitude"
"forname surname","example street 1, postcode","birth place","7.018242","50.
```

Entity Relationship Model (ERM)

Entity Relationship Model (by Peter Chen)

Chen 1976, p. 9

> The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model.

Chen 1976, p. 10

> [His article] analyzes the network model, the relational model, and the entity set model, and describes how they may be derived from the entity relationship model.

Chen 1976, p. 19

> [The article presents:] a diagrammatic technique for exhibiting entities and relationships: the entity-relationship diagram.

→ Conceptual model

ER terminology (Chen 1976, pp. 9–12)

Entity (Entität)

An entity is a „thing“ which can be distinctly identified. A specific person, company, or event is an example of en entity.

Relationship (Beziehung)

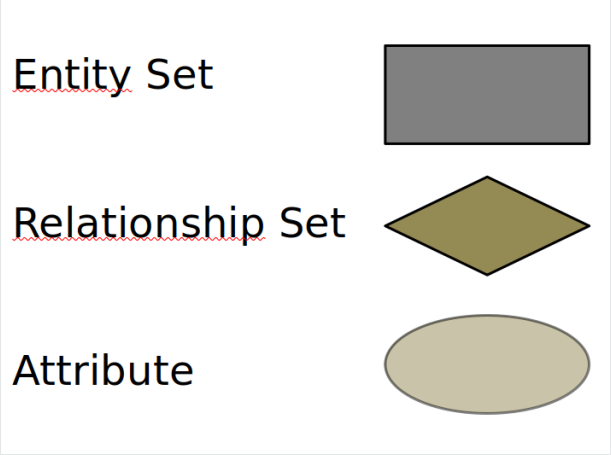
A relationship is an association among entites. For instance, „father-son“ is a relationship between two „person“ enitities.

Entity Set (Entitätsmenge)

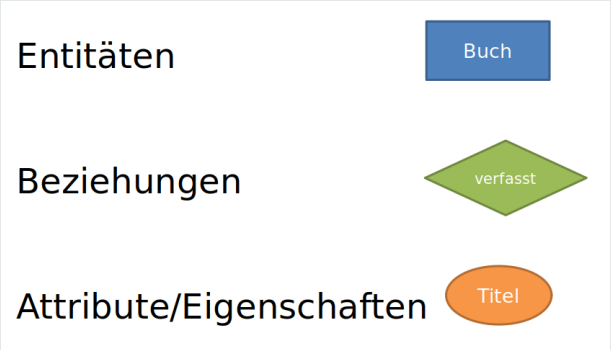
All entities of the same entity set have the same properties.

„Entities are classified into different entity sets such as EMPLOYEE, PROJECT, and DEPARTMENT.“

ER model elements



ER elements



ER terminology (Chen 1976, pp. 9–12)

set of relations (Relationship Set)

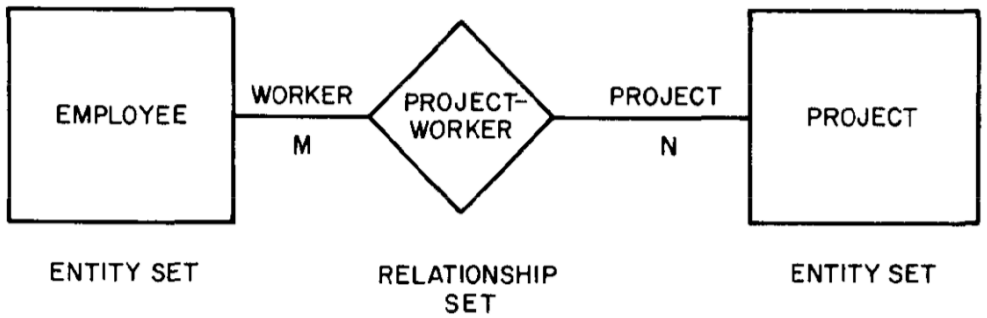
Same relationships between entities from the same entity sets: can be summarized in a set of relationships.

Role (Rolle)

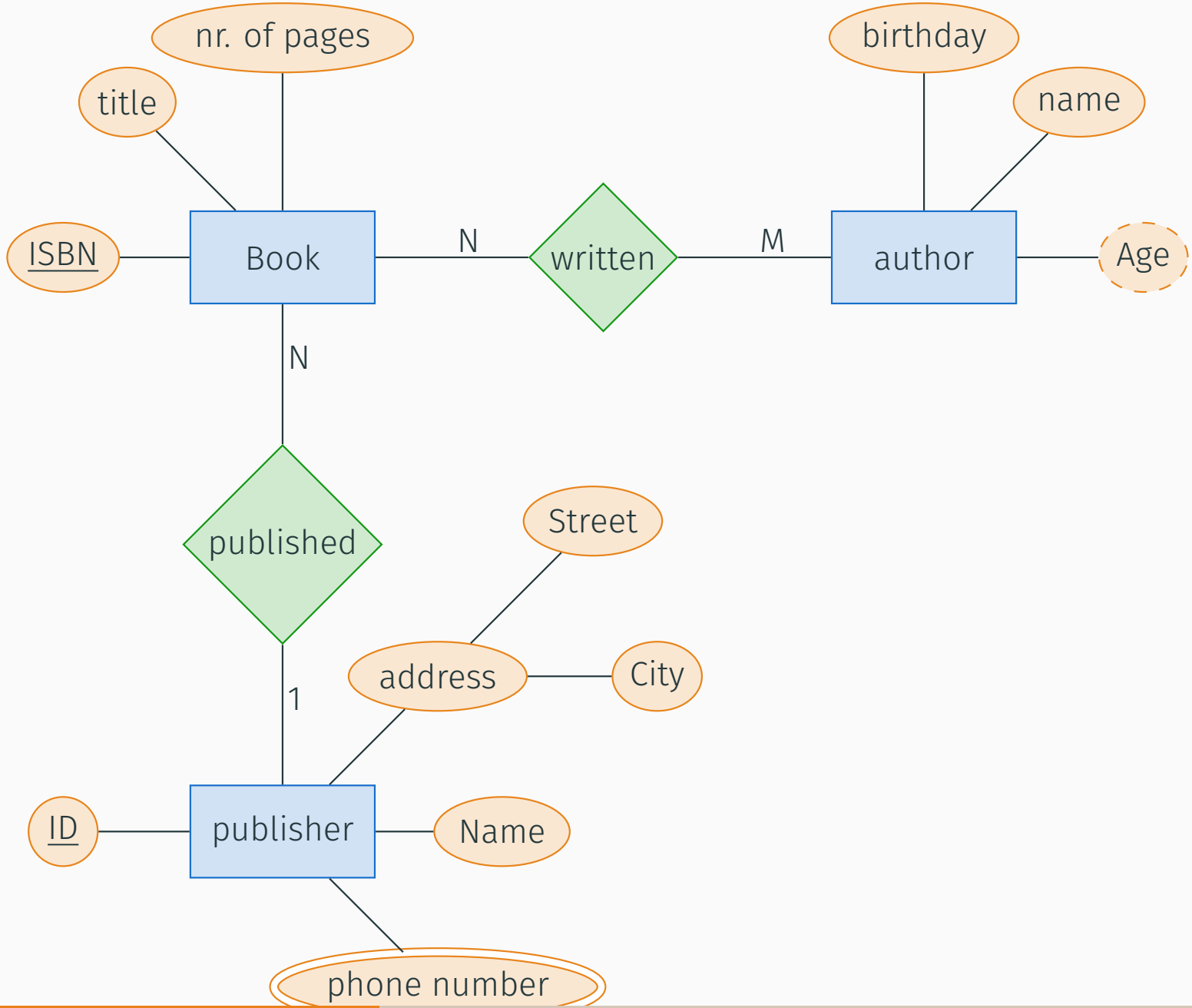
The role of an entity in a relationship is the function that it performs in the relationship.

Properties (Eigenschaften)

Using properties, entities and relationships can be described.

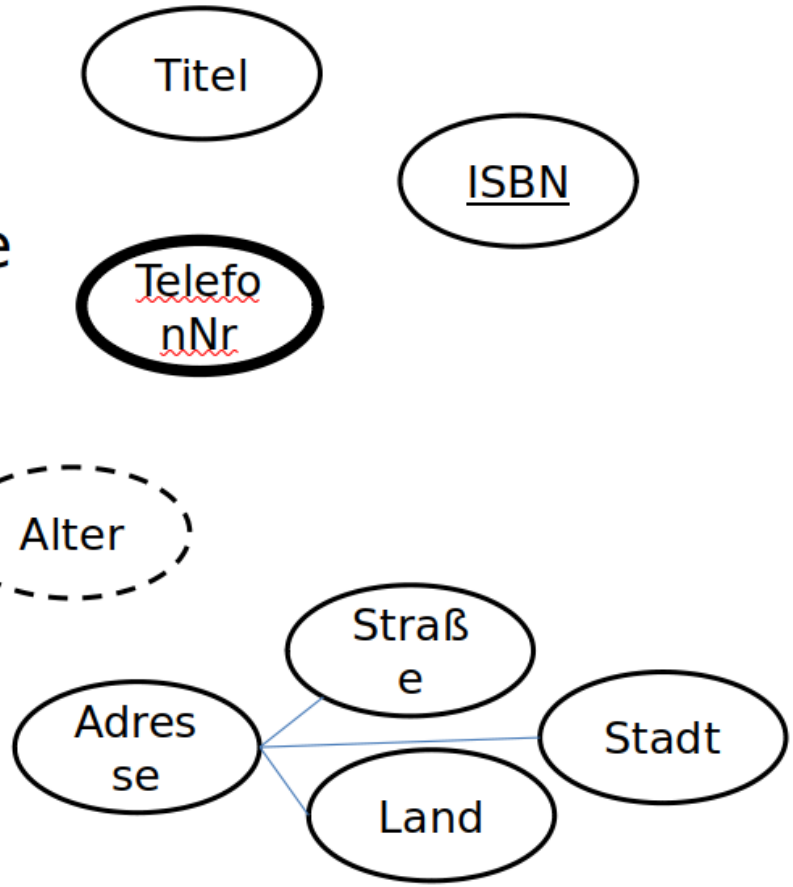


ER example for a book



ER example for a book

- Attribute
- Unique/Key Attribute
- Multi-valued Attribute
- Derived Attribute
- Composite Attribute



Examples

Entities: university class

- **entities:** class, teacher, students, room
- **attributes:** class (nr., title), teacher (name), students (name, student nr, degree programme, module), room (nr, building, seats)
- **relations:**
  - Teacher teaches class. Ein Kurs wird abgehalten von Dozent:in.
  - Class happens in room.
  - Students participate in class.

csv example: birth place

```
"Name","Address","Description","Longitude","Latitude"
"forname surname","example street 1, postcode","birth place","7.018242","50.
```

Homework: CSV

Assignment

1. Represent attributes (properties and relations) of your original in a computer processable table (`.csv` file).
2. To do that, put your attributes in a simple text-only file in a simple text editor (don't do this in MS Word or the like!), saving the file as `.txt` or `.csv` (UTF-8 encoding) or export from a table/sheet software like GoogleSheets, LIbre/OpenOffice Calc or Excel.

Please note:

Keep in mind that you can only represent attributes in a limited way using CSV:
You can representt the type of relatinship (e.g. 'made of', 'instance-class-relationship', 'bigger-than', etc.) using suitable labels in the CSV header and true/false answers as values for the cells – but you can't (at this point) say which relationships link entities (such as molecule – made of – H).
Relational databases allow for this more complete form of representation between entities using their relationships. For this homework, picking something that will easily fit in to this list / Excel sheet type format will suffice.

Action!

Let’s install SQLite3 and try out the terminal.

ER models (repetition & continuation)

Repetition: ER terminology (Chen 1976, pp. 9–12)

Entity (Entität)

An entity is a „thing“ which can be distinctly identified. A specific person, company, or event is an example of en entity.

Relationship (Beziehung)

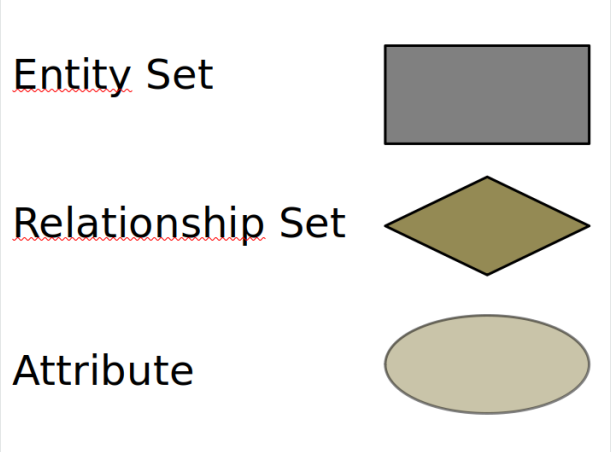
A relationship is an association among entites. For instance, „father-son“ is a relationship between two „person“ enitities.

Entity Set (Entitätsmenge)

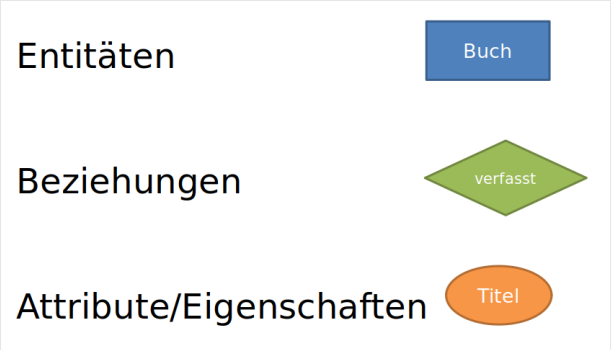
All entities of the same entity set have the same properties.

„Entities are classified into different entity sets such as EMPLOYEE, PROJECT, and DEPARTMENT.“

ER model elements

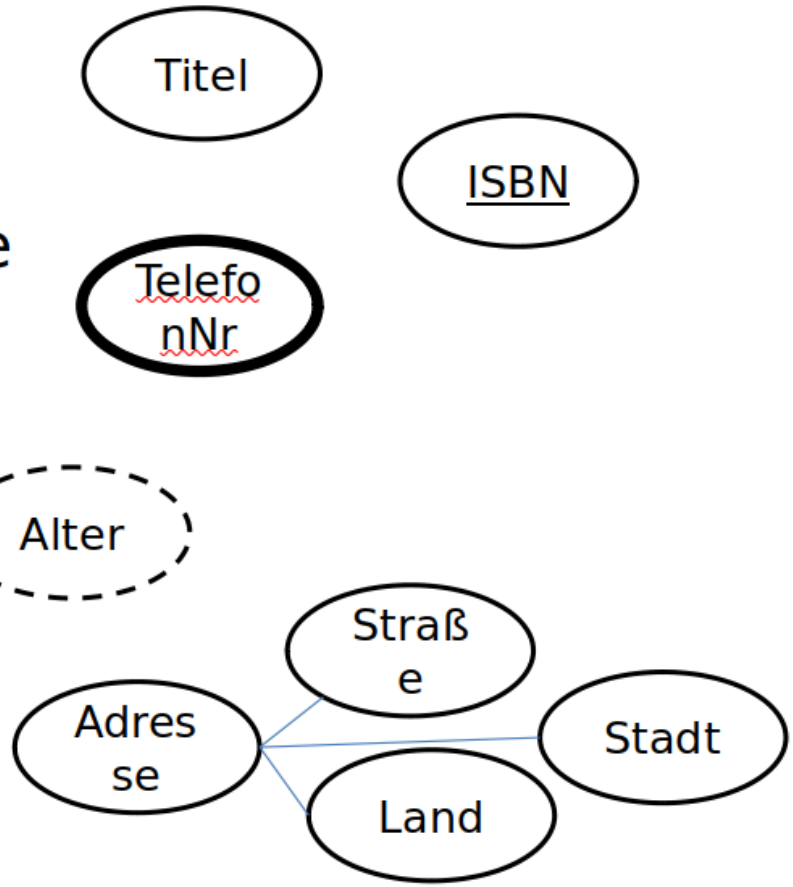


ER elements



Repetition ER

- Attribute
- Unique/Key Attribute
- Multi-valued Attribute
- Derived Attribute
- Composite Attribute



Attribute values pairs and Primary Keys

Properties (Eigenschaften) are expressed by a attribute-value combination like so:

1. The pen is blue = The ‘pen’ entity has the attribute ‘color’ with the value ‘blue’.
2. The pen is red = The ‘pen’ entity has the attribute ‘color’ with the value ‘red’.

Primary Key: Identifying properties

Entities and relationships can be uniquely identified via the value of one or multiple attributes combined. This attribute / the attribute combination is then called ‘primary key’.

Repetition exercise

A session of a university course

Such a session is defined by having:

1. students
2. teachers
3. seats
4. a room
5. in a building

Exercise 1

1. Which attribute can be used to describe these entities?
2. Using which attributes can the entities be uniquely identified?
3. Which relationships do they have amongst each other?

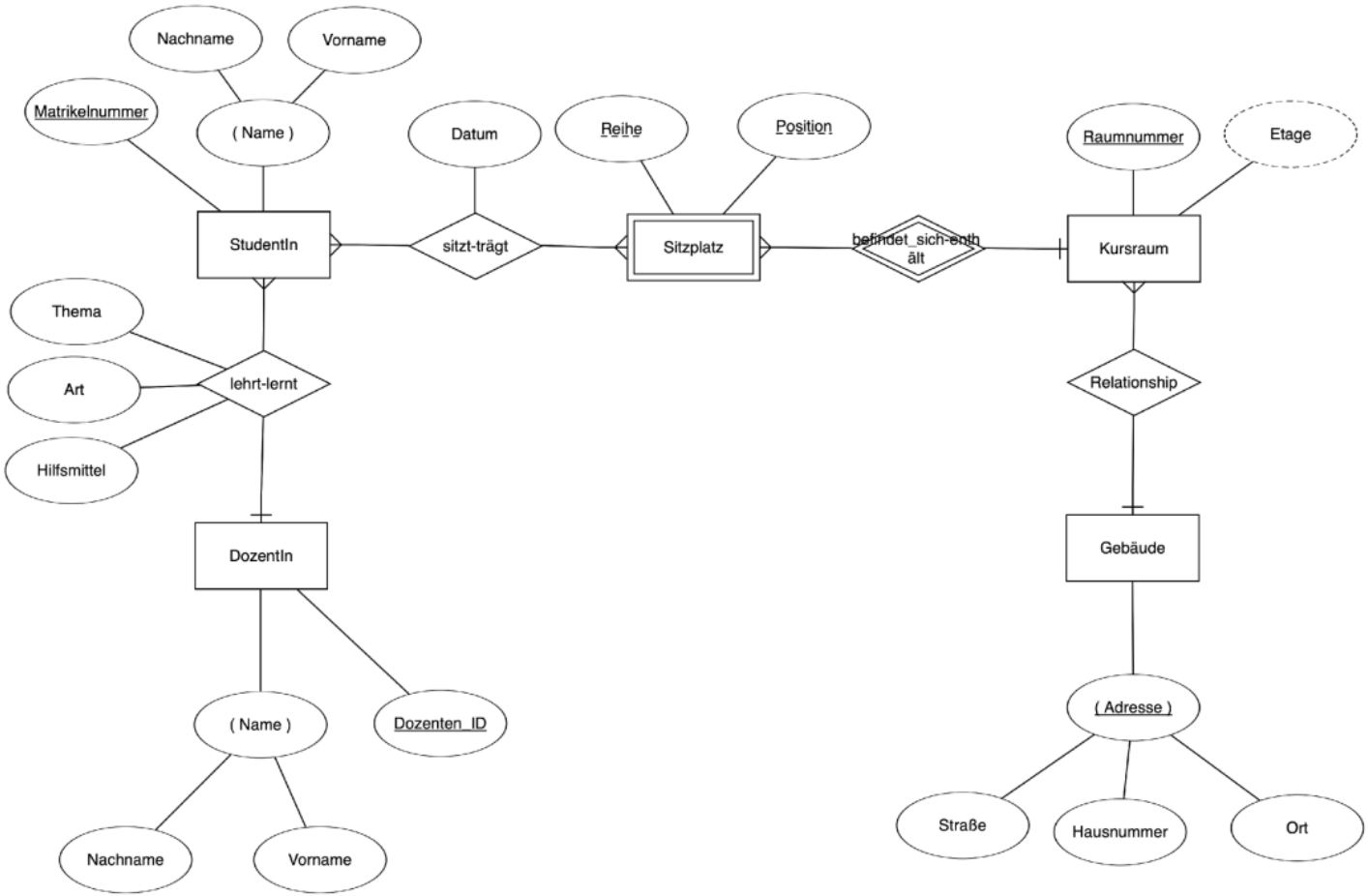
→ related to a normal in-person class

Exercise 2

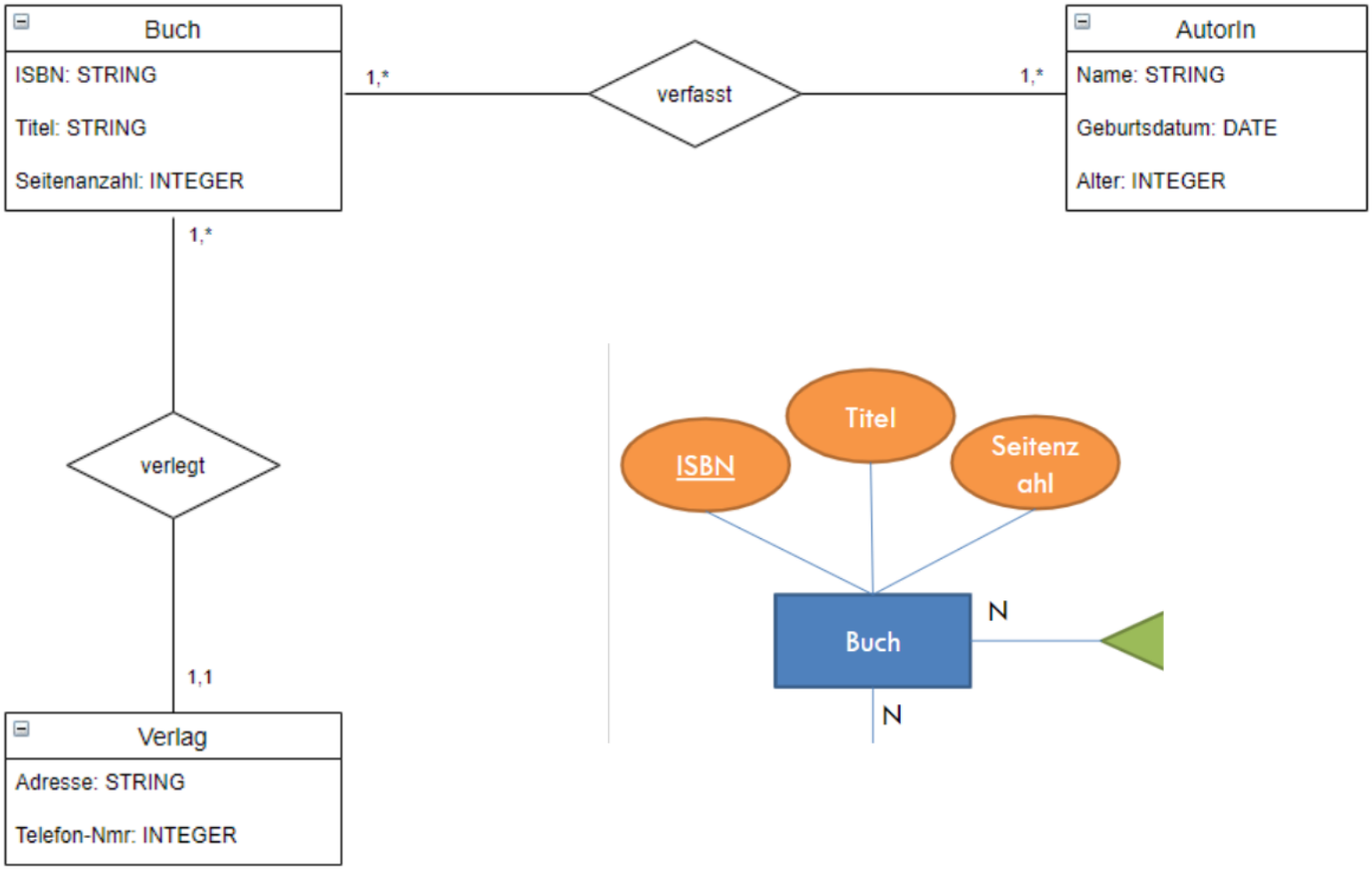
How do we have to change this to accomodate distance learning or even a hybrid setting?

→ do this exercise in groups/pairs and create a visualisation of the results, for example using the erdplus.com tool (be ready to present this briefly)

Solution: class session in ER (in-person)



ER Notationsformen



Representation of ER models as tables

Representation as tables

Entities and their relationships can be represented as tables.
But where to put the attributes?

Handeling cardinalities

1. Same book, multiple authors? (Why not just as an attribute?)
2. Same book, multiple genres? (entity or attribute?)
3. How to model the content of the book? (if at all?)

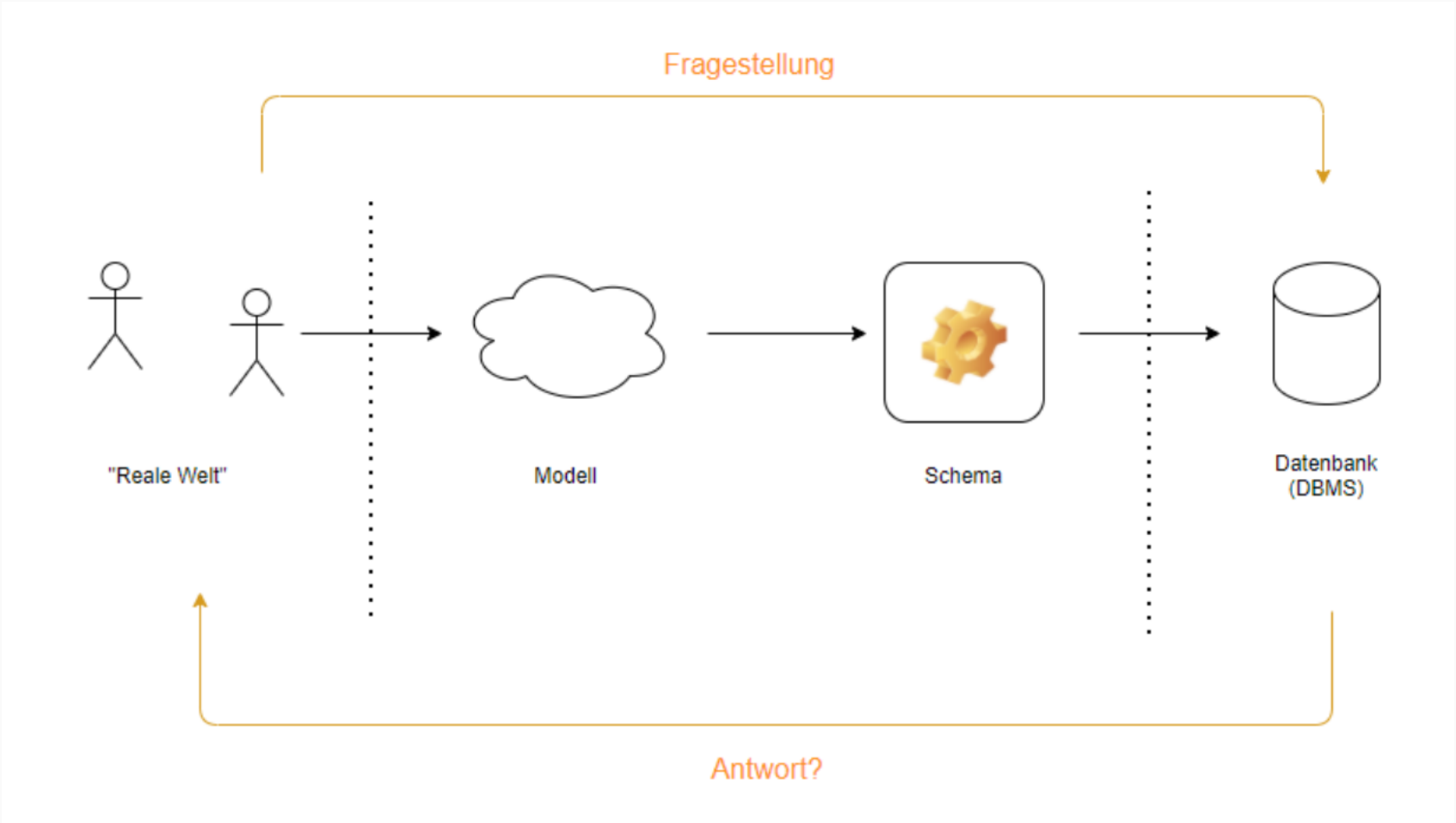
$\rightarrow$ keep these questions in mind for a discussion later (on cardinality)

Transform entities & attributes to tables: e.g. class

	A	B	C	D	E	F	G
1	Matrikelnummer	Vorname	Nachname	Sitzplatz	Kursraum	Kursnummer	Kurstitel
2	20001239			25	1.15	562.021	Grundlagen der Datenmodellierung
3	20081210			18	1.15	562.021	Grundlagen der Datenmodellierung
4	20092317			11	1.16	562.021	Grundlagen der Datenmodellierung
5	20101670			12	1.15	562.021	Grundlagen der Datenmodellierung
6	20001239			8	1.25	562.026	Einführung in die Webentwicklung
7	20092317			5	1.25	562.026	Einführung in die Webentwicklung
8							

```
2
3  Matrikelnummer;Vorname;Nachname;Sitzplatz;Kursraum;Kursnummer;Kurstitel
4  20001239;[illegible]25;1.15;562.021;Grundlagen der Datenmodellierung
5  20081210;[illegible];18;1.15;562.021;Grundlagen der Datenmodellierung
6  20092317;[illegible];11;1.16;562.021;Grundlagen der Datenmodellierung
7  20101670;[illegible]12;1.15;562.021;Grundlagen der Datenmodellierung
8  20001239;[illegible]8;1.25;562.026;Einführung in die Webentwicklung
9  20092317;[illegible];5;1.25;562.026;Einführung in die Webentwicklung
10
```

From real-world object to relational database?



Moving towards databases...

Data analysis example: table of places

- Are coordinates unique in the table or do some appear more than once?
- Group and count places using the last digits of the latitude values, etc.
- What visualizations would work for this data?

Preview: Relational Databases

1. **information and data modelling** = the way into the database
2. **data analysis and retrieval** = query/retrieve information from the DB to answer questions: query structured data

Homework: Install SQLite

maybe work in groups with those sharing your operating system (Linux, Windows, Mac)

if you already have it, you can leave early – but then figure it out yourself

we don't necessarily need SQLiteBrowser (sometimes causes trouble)

$\rightarrow$ there are tutorials; you can ask me or the others

Tutorials: Tutorialspoint | SQLiteTutorial | SQLiteBrowser Github

(materials for the) homework assignment

Resources

- Tool for creating ER models
- GeeksforGeeks tutorial/Intro to ER Models
- DARIAH Geobrowser
- Everything you didn't want to know about CSV

ER model homework assignment

1. Create and visualize the model for your original (attribute class) using an ER diagram (like ERDplus)
2. Export it as an image and upload it on Moodle
3. In case of problems, help each other via Slack or in-person.
4. **Bonus:** Look into the subject of cardinality and work it into your model.

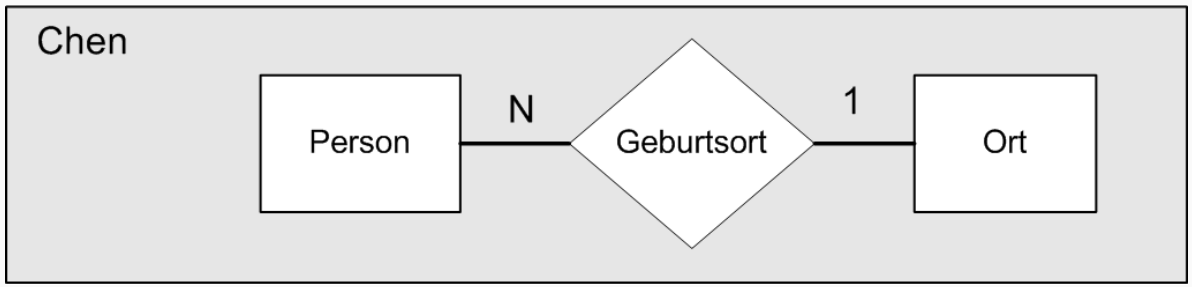
Cardinality

Cardinalities i

”cardinality” = the number of entities participating in a relationship where *n* & *m* stand for an arbitrary number.

Types of relationships

1:1 → exactly one entity participates in the relationship.
1:n → There can be an arbitrary number of entities (on the n-side) for each single entity on the 1-side.
n:m → There can be an arbitrary number for each entity on the n-side on the m-side and vice versa.

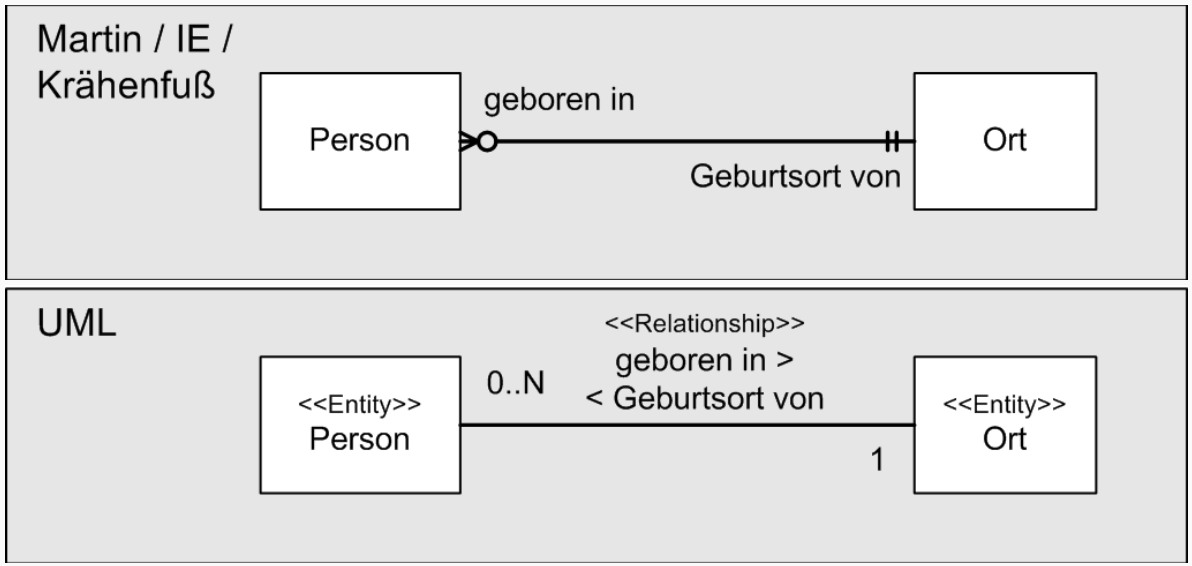


Cardinalities ii

Minimal cardinalities

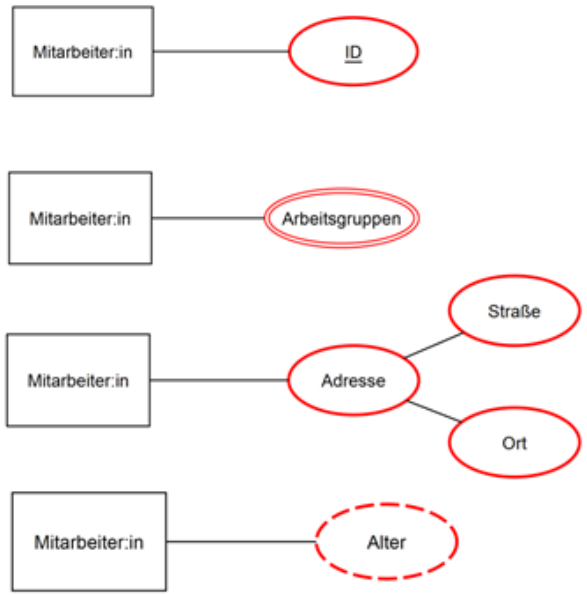
0 $\rightarrow$ there doesn't have to be an entity on this side of the relationship. (=*optional*)

1 $\rightarrow$ There has to be at least one entity on this side of the relationship. (=*obligatory*)



Attribute

- Schlüssel
- Mehrwertige Attribute
- Zusammengesetzte Attribute
- Abgeleitete / Berechnete Attribute



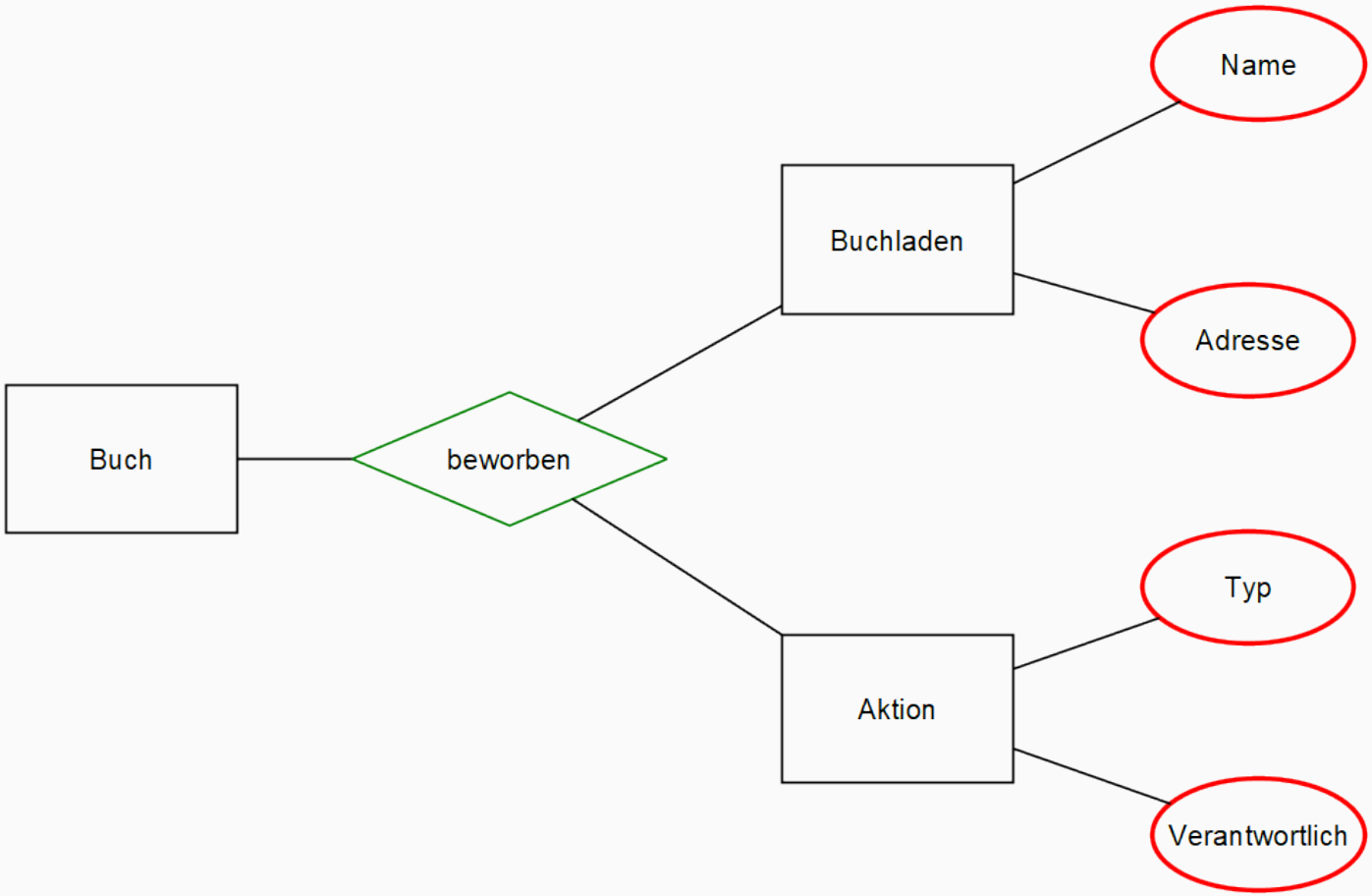
ER advanced i

- **identifying attributes (“keys“)**: these values are unique within the entity class and can thus be used to identify single entities.
- **multivalued attributes**: attributes containing more than one atomic value.
- **derived attributes**: attributes which can be calculated from other attributes.
- **composite attributes**: attributes consisting of multiple parts.

ER advanced ii

- **n-ary relationships**: a relationship set can connect multiple entity sets (in principle).
- **weak entity**: can only exist if there is a relation to another entity (room in building) but the key depends on the key of another relationship (Chen notation: double border). They have no properties of their own by which they can be identified (have no primary key). They are defined only by their relationship to a strong entity. Example: relatives of an employee in a company.

n-ary relationships



How to represent entities going from ER to table?

- **entity** → table
- **attribute** → column
  - multi-valued attributed → helper table
- **relationship**
  - **1:1** → Key of one entity (table) is stored as reference ('foreign key'/'Fremdschlüssel') in the other table as a column/attribute of its own
  - **1:n** → key of the 1-ary entity/table stored as reference in the n-ary table/entity
  - **n:m** → table of its own containing the keys of both related entities
  - **n-ary relations** → helper table containing the keys of all entities as columns
  - **relationship attributes** → helper table with the keys of participating entities and columns for the attributes
  - **multi-valued attributes** → helper table with entity key as foreign key (like a 1:n relation) and a column for the attribute
  - **composite attributes** → helper table with entity key as foreign key (like a 1:n relation) and columns for the partial attributes

Using the commandline / terminal

Terminal/Commandline basics i

Navigating in the terminal

1. search for “terminal”/“cmd”. The prompt will tell you your current location in the file system, e.g. `c:` in Windows
2. To show contents of current directory:
   - `dir` (Windows)
   - `ls` (Linux/Unix)
3. change directory = `cd directoryname` or use absolute path:
   - `cd my/path` (Windows)
   - `cd my\path` (Unix)
4. back one directory: `cd ..`
5. open Sqlite3 (you have to be in the folder where `sqlite3.exe` lies or is accessible from):

   ```
   sqlite3
   ```

Most important commands for navigating the filesystem in the terminal

dir/ls show directory contents:

- `dir` (Windows)
- `ls` (Linux/Unix)

cd path change directory (`cd directoryname` or absolute path):

- `cd my/path` (Windows)
- `cd my\path` (Unix)

cd .. back one directory

Terminal/Commandline basics ii

E.g.. Linux / Unix / ~Mac

```
sary@laptop:~$ cd Desktop
sary@laptop:~/Desktop$ ls
 studis.db
sary@laptop:~/Desktop$ sqlite3 studis.db
SQLite version 3.32.3 2020-06-18 14:00:33
sqlite> .quit
sary@laptop:~/Desktop$ sqlite3
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a
persistent database.
sqlite> .open studis.db
sqlite> .tables
fachgebiet
```

If no success: install first or...

```
sary@laptop:~/Desktop$ which sqlite3
/home/oem/anaconda3/bin/sqlite3
sary@laptop:~/Desktop$ sqlite3 -version
3.32.3 2020-06-18 14:00:33 ...
```

E.g. Windows

Search: 'Terminal/cmd'

Show contents of currenct directory

```
C:\> dir
 Datenträger in Laufwerk C:
 ist System ....
 Verzeichnis von C:\Users
 18.01.2022 14:22 <DIR> sarah
 22.03.2022 10:07 <DIR> test
```

Change directory

```
C:\> cd Users\sarah
```

Change harddrive

```
C:\Users\sarah> Z:
```

Open SQLite3 in the terminal

Sqlite3 Setup i

How to get (to) Sqlite3?

- Installation: *https://www.tutorialspoint.com/sqlite/sqlite_installation.htm*
- If Windows: *https://www.sqlite.org/download.html* > Precompiled Binaries for Windows > *sqlite-tools-win32-x86-3360000.zip*
- Unzip file, in case of Windows: put *sqlite3.exe* in the same directory as *studis.db* & *urlaub.db*
  - best not a complicated long file path
  - best no spaces/special characters in the filenames or paths
- *Alternatively:* Just use the *.zip* from Moodle, unzip (!), put all data in one same directory (*sqlite.exe* & *.db* files have to be in the same directory!).

Sqlite3 Setup ii

Example ouput from a fromer student under Windows:

```
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Theresa>cd C:\Users\Theresa\Documents\Uni\Datenmodellierung\sqlite

C:\Users\Theresa\Documents\Uni\Datenmodellierung\sqlite>sqlite3
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open studis.db
sqlite> .tables
studis  titel
sqlite>
```

Sqlite3 Setup iii

In Sqlite3 (you can tell you're in because it says *sqlite3>* in front):

```
.open studis.db
.tables
SELECT * FROM studis;
```

Don't forget the semicolon at the end!

Maybe before that run:

```
.mode columns
.headers on
```

Sqlite3 Setup iv

Troubleshooting

- After `.open studis.db` ideally there is no output → check using `.tables` if there's any database content yet.
- If there's nothing, maybe try `which sqlite3` in Mac (might not open the file you wanted but another path → navigate to correct location).
- Upon clicking `sqlite3.exe` this should work under Windows but your database `studis.db` file really has to be in the same directory!
- Does `studis.db` seem empty? Did you open it with DB-Browser previously? This can sometimes delete the contents → re-download from Moodle and open directly via the terminal.

Potential problems

1. There is no data where you currently are, meaning that SQLite3 will not find anything.
2. You are where the data are but from there, SQLite3 cannot be accessed (maybe set environment variable or put `sqlite3.exe` into the folder).

First SQL exercise i

Try these commands for the first exercise:

1. First open a *.db* file as indicated above:

```
.tables -- 2 should appear
.headers on -- maybe improve the visuals
.mode column
```

2. We create a table called *disciplines/fachgebiete*:

```
CREATE TABLE fachgebiete (id INT, name TEXT);
.tables
```

3. → a new table should appear but it's still empty when you try the following command:

```
SELECT * FROM fachgebiete;
```

First SQL exercise ii

4. Add a new column *fachgebiet_id* to *studis*:

```
ALTER TABLE studis
ADD COLUMN fachgebiet_id INT;
```

5. Then add both values:

```
INSERT INTO studis (name, fachgebiet_id)
VALUES ("Max Mara", 1);
```

or update if this row already exists (all get overwritten with value 1):

```
UPDATE studis SET fachgebiet_id=1;
```

First SQL exercise iii

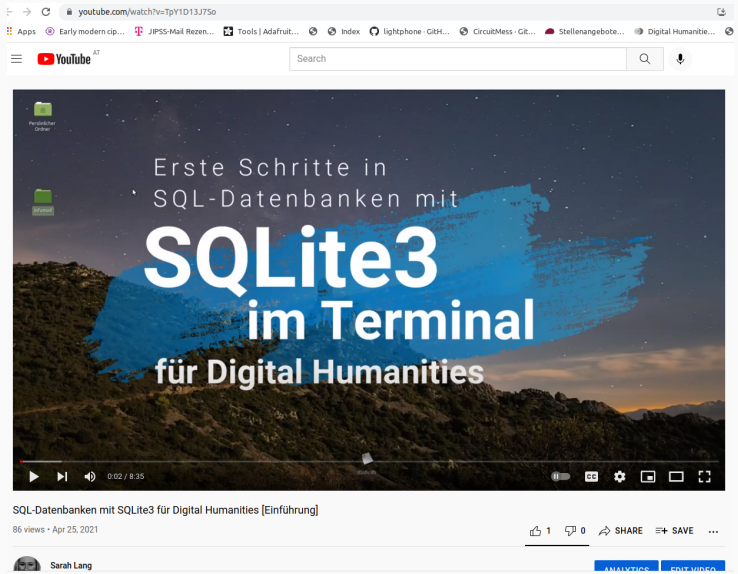
6. You can put conditions → e.g.. update *studis* with *id=1* & *id=4*, then set their discipline id to *1*:

```
1 UPDATE studis SET fachgebiet_id=1
2 WHERE (id = 1) AND (id = 4);
```

7. Using *.schema* I can check the setup of a table again. And save like so:

```
1 .save studis_neu.db
```

→ Pressing the arrow-up key gives you your last commands so you don't need to retype them each time.
→ Youtube-Video to go along with the SQL basics exercise: *https://www.youtube.com/watch?v=TpY1D13J7So*



CSV Import

Usually you can download/export spreadsheets as *.csv*. You should be able to import this in SQLite3 as a table using the following command (example *city.csv* imported as a newly created table called *cities*).

```
sqlite> .import c:/sqlite/city.csv cities
```

Create table

```
CREATE TABLE cities (
  name TEXT NOT NULL,
  population INTEGER NOT NULL
);
```

CSV Import

```
.mode csv
.import c:/sqlite/city.csv cities
```

Tutorial: *https://www.sqlitetutorial.net/sqlite-import-csv/*

Saving your database

```
oem@sary-ThinkPad-X270:~$ sqlite3
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Save results

How to use *.dump* / *.save* and *.open* again

- *https://www.sqlitetutorial.net/sqlite-dump/*
- *https://sqlite.org/cli.html*

```
1 .save example.db
```

https://sqlite.org/cli.html → under *.save*

SQL

Structured Query Language (SQL) i

Database functions

- **SQL:** *standardized* query language for relational databases
- **Data Definition Language (DDL):** create database
- **Data Manipulation Language (DML):** query and modify data
- **Data Storage Description Language (DSDL):** write data on physical storage structure

Ressources/Tutorials

- SQL: *https://www.w3schools.com/sql/*
- SQLite: *https://www.sqlitetutorial.net*
- *https://sqlite.org/cli.html*

Structured Query Language (SQL) ii

Good to know

`*` ..... sog. *Wildcard*, matches/means: everything

`;` expected as an end marker for SQL statements (not just ENTER)

`.dot` commands versus `;`

Using `.` tells the software to use an internal command from a fixed set of commands and not wait for `;` → only works for this predetermined set of commands!

Open SQLite-shell (SQL commands)

`.shell cd directory` ..... change directory

`.save dbname` ..... save database

`.open dbname` ..... open previously saved db

Structured Query Language (SQL) iii

Using SQL

1. via the Terminal (=commandline); in Windows *cmd* (search for it)
   → click *sqlite3.exe* or type *sqlite3*:
   - if there is no error, it's working and waiting for prompts!
   - more info on SQLite CLI: *https://sqlite.org/cli.html*
2. via SQLiteBrowser (not recommended)
3. possible file endings to import: *.csv*, *.db*, *.sql*

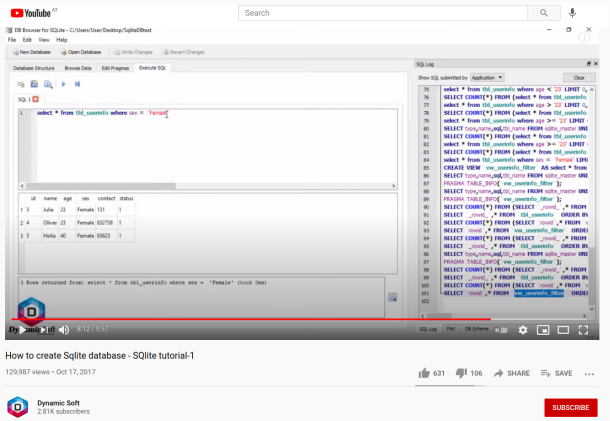
Structured Query Language (SQL) iv

Special commands to sqlite3 (dot commands)

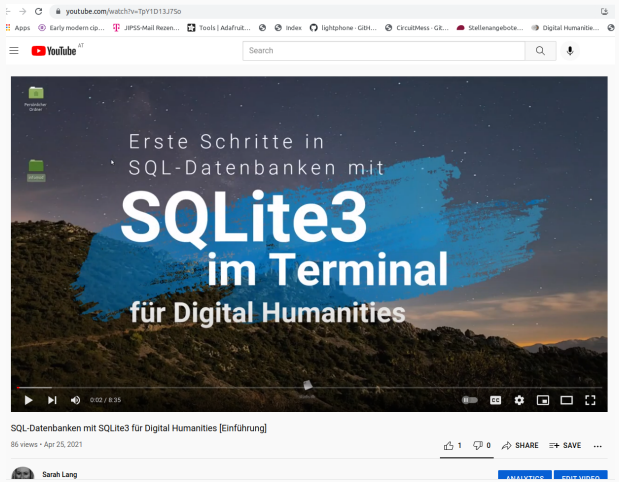
.help ..... overview over dot commands
.open ?OPTIONS? ?FILE? ..... open or create db
.databases ..... show existing dbs
.tables ?TABLE? ..... show existing tables
.import FILE TABLE ..... import data to table
.mode MODE ?TABLE? ..... how to display output
.mode column ..... show *SELECT* results as tabular format
.headers on ..... show *SELECT* results with headers
.schema ?TABLE? ..... displays *CREATE* command used
.save ?FILE? ..... save the db
.dump ?TABLE? .... show database contents as SQL commands

Tutorial: Beginner SQL using SQLiteBrowser or Terminal

```
> sqlite3 studis.db
```



ca. 10min Video
https://www.youtube.com/watch?v=Pni6WxHFTUg



ca. 8min
https://www.youtube.com/watch?v=TpY1D13J7So&t=413s

Some more SQL

selection criteria

- Use operators like *=, !=, >, <, LIKE, REGEX, MATCH, ISNULL, NOTNULL*
- Combine using *AND* or *OR*

Print results

```
1 SELECT content
2 FROM Tabelle
3 WHERE selectionCriteriaMatch
```

Reduce set of results

```
1 SELECT content
2 FROM Tabelle
3 WHERE selectionCriteriaMatch
4 LIMIT max number of results
5 OFFSET results starting from...
```

Delete or modify tables (DDL)

Delete table

```sql
DROP TABLE tableName;
```

Rename table

```sql
ALTER TABLE tableName RENAME TO tableName;
```

Add column

```sql
ALTER TABLE tableName
ADD COLUMN columnname1 datatype options;
```

Insert data = data representation (DML)

Add new row

```
INSERT INTO tableName
       (columnname1, columnname2, columnname3 [etc.])
       VALUES (value1, value2, value3 [etc.]);
```

add multiple rows at a time

```
VALUES (value1, value2, value3 [etc.]),
       (value1, value2, value3 [etc.]),
       (value1, value2, value3 [etc.]) [etc.];
```

Example: delete or change entries (DML) i

Change entries

```
UPDATE tableName
SET columnname_to_modify = new_value
WHERE columnname = referenceValue;
```

Delete entries

```
DELETE FROM tableName
WHERE columnname = value;
```

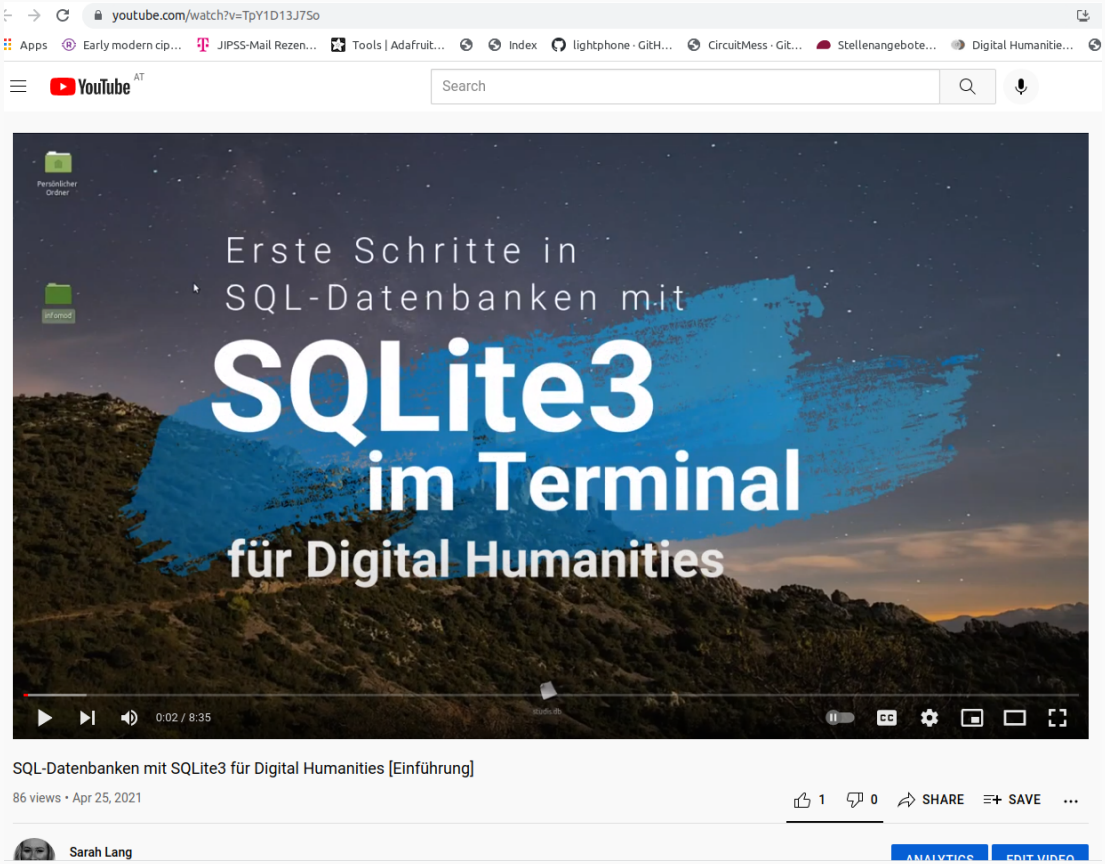
Example: delete or change entries (DML) ii

Query the db (DML)

```
SELECT * FROM tableName;
SELECT table1, table3 FROM tableName;
```

Example: delete or change entries (DML) iii

German video slowly showing all steps from the first basic example:



Example: delete or change entries (DML) iv

Our example from the intro.

```
.tables

ALTER TABLE studis
ADD COLUMN fachgebiet_id INT;

CREATE TABLE fachgebiet (
name TEXT,
id INT PRIMARY KEY
);

.tables
```

```
INSERT INTO fachgebiet (name, id)
VALUES ('dh', 1), ('geschichte', 2);

UPDATE studis
SET fachgebiet = 2
WHERE id = 1;

SELECT * FROM studis;

SELECT name, fachgebiet
FROM studis;
```

What can you do with `urlaub.db`?

Homework 4: Get started creating your database!

Create a project database for your final project in SQL. Write 1/2-1 page of text (submitted as PDF) explaining the following questions:

- Which original are you modelling?
- What is the purpose of the database? Which queries do you want to answer with it?
- Which entities of the original are represented in your database?
- Which attributes (incl. Primary Keys) do the classes have?

Also submit a dump of your database please.

SQL basics

1. Our terminal expects that all statements are terminated by `;` – this allows you to use line breaks to structure your query for better overview. If you forget the semicolon the terminal will keep waiting for input. It might look like it's stuck: Just type semicolon plus enter to fix it.
2. Asterisk (`*`) is a so-called *wildcard* and means 'everything'.
3. Our simple first statement thus means: 'Select *everything* from the table named *table*.'
4. You can open databases (`.db` or `.sql`) using `sqlite3 databasename.db` from the terminal.

My first SQL statement

Before running your first command, you might need to use the `.tables` command in `sqlite3` to learn which tables even exist in your database and/or how they are named (dot-prefixed commands exceptionally need no semicolon to end them, just enter).

```
1 SELECT * FROM table;
```

SQL basics 2

1. If there was a mistake in your command, you get an error message. If there is none we assume all is good and whatever you prompted the program to do actually happened. There might just not be a success message. But what you prompted the program to do might not have been exactly what you had *intended* it to do, so inspect the results by displaying your database anew!
2. After you change a database, check its contents using `SELECT * FROM table;` → make sure the change effected was the one intended.
3. By convention, SQL keywords are capitalized in AllCaps style to make them easily identifiable visually. This makes your code more readable but the program would also work with lowercase commands.

More complex example:

```
1 SELECT DISTINCT column_list
2 FROM table_list
3 JOIN table ON join_condition
4 WHERE row_filter
5 ORDER BY column
6 LIMIT count OFFSET offset
7 GROUP BY column
8 HAVING group_filter;
```

SQL SELECT i

```sql
SELECT list_of_columns FROM list_of_tables;
```

What the options mean...

list_of_columns	you can select multiple columns separated by comma or use asterisk (*) to select all
list_of_tables	defines which tables to select from.
AS (alias)	allows you to name the column header for the output table
DISTINCT	only prints each unique value once; usage: *SELECT DISTINCT*

SQL SELECT ii

To select specific columns from multiple tables which share common column names, specify as follows:

```
SELECT hotels.name, places.name FROM hotels, places;
```

For more clarity, give them more meaningful headers:

```
SELECT hotels.name AS hotel_name,
       places.name AS place_name
FROM hotels, places;
```

Not specifying from which table will either print a cartesian product or produce an error like so:

```
SELECT name FROM hotel, place;
-- Error: ambiguous column name: name
```

SQL WHERE

Entries are selected depending on the values of their attributes which are communicated using the following operators:

Comparison operators

`=`	equals
`>`	greater than
`<`	less than
`>=`	greater or equal
`<=`	less or equal
`!=` or `<>`	not equal

Logical operators

`AND`	both are true
`OR`	at least one part is true
`BETWEEN`	value is within a defined range
`IN`	value in included in defined set
`LIKE`	value corresponds to a pattern
`%`	any sequence of characters (none or multiple)
`_`	any one character (exactly one)
`NOT`	results in the opposite of the truth value of the expression

```
1 SELECT name FROM hotel WHERE price < 150;
```

SQL CREATE VIEW i

If you need to rearrange the order of your table after it is already defined or just getting an output that you often need is very complicated and annoying, you might want to create a *VIEW* so that you can access the result of a complex query over and over again effortlessly:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

It will always show up-to-date data as it's just a shorthand for the actual complex query.

SQL CREATE VIEW ii

W3C Tutorial 'CREATE VIEW' → **examples:**

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';

-- display as:
SELECT * FROM [Brazil Customers];
```

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
-- call as:
SELECT * FROM [Products Above Average Price];
```

Creating tables (DDL)

```
CREATE TABLE tableName (
  columnName1 datatype option,
  columnName2 datatype option,
  [...]
  );
```

What the options mean...

tableName represents the entity set, written in lowercase.

columnName represents an attribute or relationship of the entity, in lowercase.

datatype sets the data type from a list of possibilities such as *TEXT* or *INTEGER*, etc.

options

- Primary Key: *PRIMARY KEY* → sets value as primary key.
- Foreign Key: *FOREIGN KEY columnName REFERENCES tableName_pk(attribute_pk)* → sets value as foreign key.
- No empty values allowed: *NOT NULL*.
- Values have to be unique: *UNIQUE*.

Database organization, Primary and Foreign Keys

Data types

Data types

- **NULL** nothing
- **INTEGER** positive integer values (Ganzzahlen)
- **REAL** 8-byte IEEE floating point values
- **TEXT** text encoded according to DB standard (UTF-8, UTF-16BE or UTF-16LE).
- **BLOB** The value is a blob of data, stored exactly as it was input.

Other database management systems (DBMS) have a slightly different choice of datatypes due to different implementations. **Remember:** SQL is a standard which is implemented slightly differently by different software solutions such as *SQLite3* or *MySQL* (check out this list).

In MySQL, instead of *TEXT* you would use this:

```
1 LastName varchar(255),
```

This tells the computer exactly for how many characters it needs to reserve space. This used to be more important especially when computer memory was more rare a commodity than it is today.

Primary and Foreign Keys i

Id	Titel	Autorid	Id	Autor
1	Die Stunde der Stümper	1	1	Andrew Keen
2	The Big Switch	2	2	Nicholas Carr
3	Wer bin ich wenn ich online bin	2	2	Nicholas Carr
4	Web 2.0	3	3	Tom Alby

Id	Titel	Autorid
1	Die Stunde der Stümper	1
2	The Big Switch	2
3	Wer bin ich wenn ich online bin	2
4	Web 2.0	4

Id	Autor
1	Andrew Keen
2	Nicholas Carr
4	Tom Alby

- Because relations are sets, each tuple needs to be different from all others. To address one specific tuple, we need a key or an id. This key needs to be unique in the whole relationship set.
- A key (*primary key*, **Primärschlüssel**) can consist of one or multiple attributes (as few as possible, it has to be irreducible or minimal). In practice, we often use artificial keys (like a running id number).
- **Foreign keys** (*Fremdschlüssel*) reference the primary key of a tuple in a different relation. Because foreign keys don't need to be unique, they aren't technically keys but are still called that way. Comparing foreign and primary keys, we can create new relations ad hoc.

Primary and Foreign Keys ii

Id	Titel	Autorid
1	Die Stunde der Stümper	1
2	The Big Switch	2
3	Wer bin ich wenn ich online bin	2
4	Web 2.0	4

Id	Autor
1	Andrew Keen
2	Nicholas Carr
4	Tom Alby

Referential integrity

Referential integrity is a constraint which ensures that a dataset can only be manipulated if it doesn't render other datasets inconsistent.

Example: Deleting dataset with `id=4` in the table `author` would destroy referential integrity because at least one foreign key in the table *book* references this entry. For the machine to automatically know that, we need to declare these keys and their relationships when setting up the database (more on that later).

Unique Identifiers: Primary and Foreign Keys i

Identifier

unique identifier / key: refers to exactly one element. Examples:

1. ISBN no.
2. matriculum nr.
3. ORCID-ID
4. DOI
5. car plate
6. norm data (GND): list of standardized terminoloy, gives a definite name form and an ID to refer to

Unique Identifiers: Primary and Foreign Keys ii

Indirection

An ID doesn't directly point to a value but rather to a number which, in turn, points to the value.

Separating contents into multiple tables minimizes redundancy!

Data can be reunited (joined) using *`VIEW`*s. Redundancy is a source of errors.

Unique Identifiers: Primary and Foreign Keys iii

Each table needs to have its primary key. You can define it in the two following ways:

1. as an added option when defining the column:

```
id INT PRIMARY KEY
```

2. as an extra option after all column definitions:

```
PRIMARY KEY(id)
PRIMARY KEY(forname, lastname, birthday)
```

Resources

1. W3Schools Primary Key: unique identifier for each table entry
2. W3Schools UNIQUE
3. W3Schools NOT NULL

Linking tables

Example: Linking tables using keys

1. **Class:** ClassID, LecturerID, subject, RoomID, hours
2. **Room:** RoomID, building, capacity
3. **Lecturer:** LecturerID, name, area of expertise
4. **Student:** StudentID, name, area of study

Generic example:

```
CREATE TABLE person (
    id      INTEGER PRIMARY KEY AUTOINCREMENT,
    name    TEXT    NOT NULL,
    age     INT     NOT NULL,
    address TEXT
);
```

Class example:

```
CREATE TABLE student (
    student_id      INTEGER PRIMARY KEY,
    name            TEXT    NOT NULL,
    degreeProgramme TEXT    NOT NULL
);
```

Using Foreign Keys join tables i

Each table only contains one type of entity (e.g. student's names and ID numbers). But these might have relationships to other tables, for example, for each student we could list course numbers. Using this course number we get a link to a course table where we write down the name of the class.

Using Foreign Keys join tables ii

Foreign Keys (*Fremdschlüssel*)

It is not mandatory to specify Foreign Keys but they allow the database to run internal optimization in the background. They are essential for maintaining referential integrity.

You can define them as follows:

1. as part of the respective column definition:

```
REFERENCES tableNname(fieldName)
```

2. after the column definitions:

```
FOREIGN KEY(columnName) REFERENCES tablenName(fieldName)
```

Using Foreign Keys join tables iii

Assuming there were another table called *places* with the primary key *id*

```
CREATE TABLE persons (
  place_id INT UNSIGNED REFERENCES places(id),
  ...
);

CREATE TABLE persons (
  place_id INT UNSIGNED ,
  ...
  FOREIGN KEY (place_id) REFERENCES places(id)
);
```

Using Foreign Keys join tables iv

This *Foreign Key* refers to another table's *Primary Key*. Assuming there is a *disciplineID* in the *disciplines* table where it is the*Primary Key*. In the *Students* table would be a column called *disciplineID* which is a foreign key *Foreign Key there*.

Foreign Key constraints in CREATE TABLE

```
1 FOREIGN KEY (columnName)
2 REFERENCES tableName_pk (attribute_pk)
3 PRAGMA foreign_keys = ON;
```

Using *FOREIGN KEY* constraints we could define what happens if, for example, one of the related tables gets updated or something is deleted.

Constraints i

W3Schools SQL Constraints

NOT NULL	makes sure a field can never be *`NULL`* (important for IDs).
UNIQUE	makes sure all values of a column are distinct (also important so that IDs remain unique).
PRIMARY KEY	automatically creates a combination of *`NOT NULL`* and *`UNIQUE`*, making the primary key an ideal means of referencing a table entry.
FOREIGN KEY	prevents relationships between tables from accidentally getting destroyed (by the deletion of a cell, for example, to which the cells of another table still refer).
CHECK	makes sure the values of a column correspond to certain constraints (such as age is in between 0-99 or similar). This prevents you from accidentally inserting nonsensical values into the table which could later corrupt any analyses you might want to use the database for.

Constraints ii

Value constraints

Many DBMSs allow you to define any allowed range for value for a given column.
Syntax:

```
CHECK ([ fieldname ] [ condition ])
```

Example

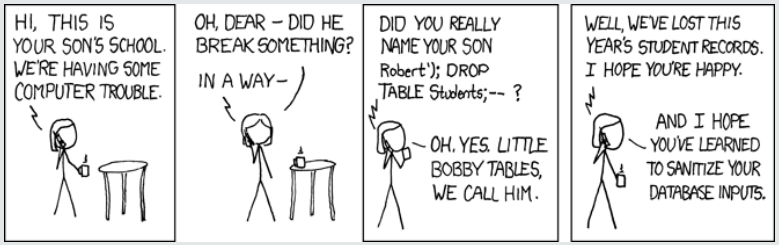
```
CREATE TABLE teachers (
salary DECIMAL (4 ,2) ,
CHECK ( salary > 0)
)
...
CHECK (grade >0 AND grade <=5)
```

Excursus: SQL Injection (definition)

Wikipedia

SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).

related xkcd comic



source

W3Schools

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

Excursus: SQL Injection (example from W3Schools)

Starting point

A website gets data using a form to later run a database query whose results are to be displayed to the user. Example in javascript:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId ="
         + txtUserId;
```

Always True Inject

In case there are no security checks when parsing the submitted query, users can submit SQL code instead of just a normal value. *OR 1=1* is always true, thus all users will be displayed. Resulting SQL query (obviously it was not intended by the database owners to use the form like this):

```
SELECT * FROM Users
WHERE UserId = 105 OR 1=1;
```

Batched Statement Inject

Instead of just running one statement, one could also add *;* and send a second query after it. For example by typing into the form: *105; DROP TABLE Users*. Resulting SQL code:

```
SELECT * FROM Users
WHERE UserId = 105;
DROP TABLE Users;
```

Cardinality i

Cardinality

Cardinality describes how many entities participate in a relationship. The following cardinalities are possible:

1:1 exactly one-to-one relationship, quite rare in practice.
Example: *Student can only have one passport photo.*

1:n (directionality can also be n:1 !) Exactly one entity is linked to one or multiple others.
Example: *Student can have multiple adresses.*

n:m Both parts of the relationship can participate in multiple relationships.
Example: *Students can go to multiple classes; most classes should have more than one student.*

Remember: How to represent entities in a database

- **entity** → table
- **attribute** → column
  - multi-valued attributed → helper table
- **relationship**
  - **1:1** → Key of one entity (table) is stored as reference (‘foreign key’/‘Fremdschlüssel’) in the other table as a column/attribute of its own
  - **1:n** → key of the 1-ary entity/table stored as reference in the n-ary table/entity
  - **n:m** → table of its own containing the keys of both related entities
  - **n-ary relations** → helper table containing the keys of all entities as columns
  - **relationship attributes** → helper table with the keys of participating entities and columns for the attributes
  - **multi-valued attributes** → helper table with entity key as foreign key (like a 1:n relation) and a column for the attribute
  - **composite attributes** → helper table with entity key as foreign key (like a 1:n relation) and columns for the partial attributes

More advanced queries

Functions, sorting, counting, etc. i

COUNT

```
SELECT COUNT(columnname)
FROM tablenname
WHERE bedingung ;
```

AS

Using *AS* you can rename columns in the output. *PricePerPage* would then be a derived attribute.

```
SELECT title AS Titel ,
price AS Price ,
price / pages AS PricePerPage
FROM books ;
```

Functions, sorting, counting, etc. ii

BETWEEN

Only prints entries upon the condition that they are between two values:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

DISTINCT

Only prints distinct values (*unique values*):

```
SELECT DISTINCT columnname
FROM table;

SELECT COUNT(DISTINCT columnname)
FROM table;
```

Functions, sorting, counting, etc. iii

ORDER BY

Sort the results of a query by one or multiple columns (separated by commata `,`)

ASC	ascending order
DESC	descending order

You can also sort by letters (e.g. alphabetically).

```
1 ORDER BY fieldnames
2 ORDER BY fieldname DESC
3 SELECT name FROM hotel
4 ORDER BY price ASC;
```

GROUP BY

Lists each value as a row of its own:

```
1 SELECT AVG(columnnameX)
2 FROM tablenname
3 GROUP BY columnnameY;
4
5 SELECT rating, AVG(price)
6 FROM hotel
7 GROUP BY rating;
```

Functions, sorting, counting, etc. iv

Aggregate functions

Functions offered by the database management system which aggregate data: They don't output single database entries but the result of a function the system calculated, e.g. *COUNT()*, SUM(), *MIN()*, *MAX()* and *AVG()*.

→ data fields with number values can be aggregated using:

AVG() ..... average
SUM() ..... sum
MIN() ..... minimum value
MAX() ..... maximum value

```
1 SELECT MIN(column_name)
2 FROM table_name
3 WHERE condition;
```

There are many more functions, operators, etc. than can be presented in the limited time here.

Instead of = you can use the *LIKE* operator which allows for the use of wildcards (placeholders).

Wildcard % and LIKE

Very powerful, in SQL use *%* which stands of any characters (0-n).

```
1 SELECT title
2 FROM books
3 WHERE title LIKE 'Datenbank %';
4
5 SELECT title
6 FROM books
7 WHERE title LIKE '% Datenbank';
```

Wildcard _ and LIKE

_ = exactly one character.

```
1 SELECT firstname, lastname
2 FROM authors
3 WHERE firstname LIKE 'g_nter';
```

Select all three-letter fornames:

```
1 SELECT firstname
2 FROM authors
3 WHERE firstname LIKE '___';
```

Special comparison operators: IN and NOT IN

IN compares against a list of values:

```
SELECT title, year
FROM books
WHERE year IN (2000, 2004)
ORDER BY year;
```

NOT IN gives you the opposite of the set for which the condition is true:

```
SELECT title, year
FROM books
WHERE year NOT IN (2000, 2008)
ORDER BY year;
```

Combining conditions with AND

```
SELECT title, pages, price
FROM books
WHERE pages > 500 AND price < 10;
```

gives you all books which have more than 500 pages and whose price is still below 10€.
Combine as needed with other operators such as *OR* and *AND NOT*. If queries get very complicated, make sure to introduce clarity by the use of parentheses.
Here are books appeared after 2012 which are more than 500 pages long *or* cost more than 30€:

```
SELECT title , year , pages , price
FROM books
WHERE year > 2012
AND (pages > 500 OR price > 30) ;
```

Functions, sorting, counting, etc. vii

LIMIT

Limits results in result table:

```
1 SELECT name FROM hotel
2 LIMIT 2;
```

OFFSET

Defines an offset for a starting position to print results:

```
1 SELECT name FROM hotel
2 LIMIT 2 OFFSET 1;
```

UNION

Unites the result of multiple db queries:

```
1 SELECT name FROM hotel
2 UNION SELECT name FROM see;
```

Homework 4: Get started creating your database!

Practical tips:

Use the ’arrow up’ key to repeat a query from earlier (from your history of queries, keep using the key to go further up).

Create more complex commands in a simple text editor first and save the file: this allows you to modify and gradually expand your commands without having a lot of hassle retyping everything or even re-create the database from scratch if you made a mistake which is difficult to fix.

Old-school computer things (like the terminal or SQL) may be less user-friendly than you are used to: if you delete something, there is no ‘undo’. Thus, always display what you want to delete first with a `SELECT` command before actually deleting (maybe more than you had intended).

Querying over multiple tables → Joins

SQL as a query language: `JOINS` i

In the *CREATE* command, we defined a table and set up a relational database using keys (SQL's data definition language). Now we want to analyze (i.e. query) our database to learn about the contents.

Remember: Foreign Keys

Foreign keys (*Fremdschlüssel*) are table cells containing ID number which uniquely reference the primary keys (*Primärschlüssel*) of another table.
For example the table ***books*** contains a column ***publisher_id*** which references one particular entry of the table ***publishers*** via its ***publisher_id***.

```
1 SELECT title, publisher_id FROM books ;
```

Attention: The table columns you link up don't necessarily share the same name (unless you made it so!)
Attention: Because foreign keys don't have to be unique they're no true keys – they are still named that way.

SQL as a query language: JOINS ii

Linking tables in queries

If you link up the table *books* to the table *publishers* using the *books*' tables foreign key *publisher_id*, you get a new relation containg all contents of both tables.

```
1 SELECT *
2 FROM books, publishers
3 WHERE books.publisher_id = publishers.publisher_id ;
```

```
+-----------------------------------------+---------------+
|                  from                   |      from     |
|                  books                  |   publishers  |
+----+-------------------+ ... +------+----+----+----------+
|  2 | XML in a Nutshell | ... | 40.0 |  2 |  2 | O'Reilly |
+----+-------------------+ ... +------+--- +----+----------+
| 48 | XML in der Praxis | ... | 34.9 | 10 | 10 | Addison- |
+----+-------------------+ ... +------+----+----+----------+
                                      /     \
                                     /       \
                            foreign key       primary key
```

SQL as a query language: JOINS iii

Table-specific columns I

In case we're only interested in specific columns of the linked tables, we can pick just those in the *SELECT* statement. Be mindful of potential name conflicts due to ambiguous column names. In those cases, you need to specifically adress them like so:

```
1 SELECT title, publisher_name,
2   books.publisher_id,
3   publishers.publisher_id
4 FROM books, publishers
5 WHERE books.publisher_id
6   = publishers.publisher_id;
```

Table-specific columns II

Because the column name *publisher_id* exists both in *books* as well as in *publishers* we need to tell the DBMS which one we mean by prefacing it with its parent table: *books.publisher_id*.
In longer queries it cam make sense to abbreviate table names to save some typing and keep a better overview:

```
1 SELECT title, publisher_name,
2   b.publisher_id,
3   p.publisher_id
4 FROM books b, publishers p
5 WHERE b.publisher_id
6   = p.publisher_id;
```

SQL as a query language: JOINS iv

Attention: Avoid the cartesian product!

When joining multiple tables, it is essential that you specify conditions for it:

```
1 SELECT count(*) FROM books ;
2 SELECT count(*) FROM publishers ;
3 SELECT count(*) FROM books, publishers ;
4 SELECT count(*) FROM books, publishers
5 WHERE books.publisher_id = publishers.publisher_id ;
```

These queries give the following result counts: 353, 82, 28946, 350.
The number of result lines in joins equals the product of the lines of all the joined tables. That way, we can easily generate results with millions (!) of lines which excede your database servers capacity and thus, might crash it.
Thus: Careful not to accidentally produce the cartesian product of multiple columns!

Linking tables using WHERE

The *WHERE* clause compares if values from one table are present in another table:

```
1 SELECT tab1.* FROM tab1, tab2
2 WHERE tab1.tab2_fk = tab2.id ;
```

Linking tables using JOIN

In the *FROM* clause, tables are linked using the *JOIN* statement:

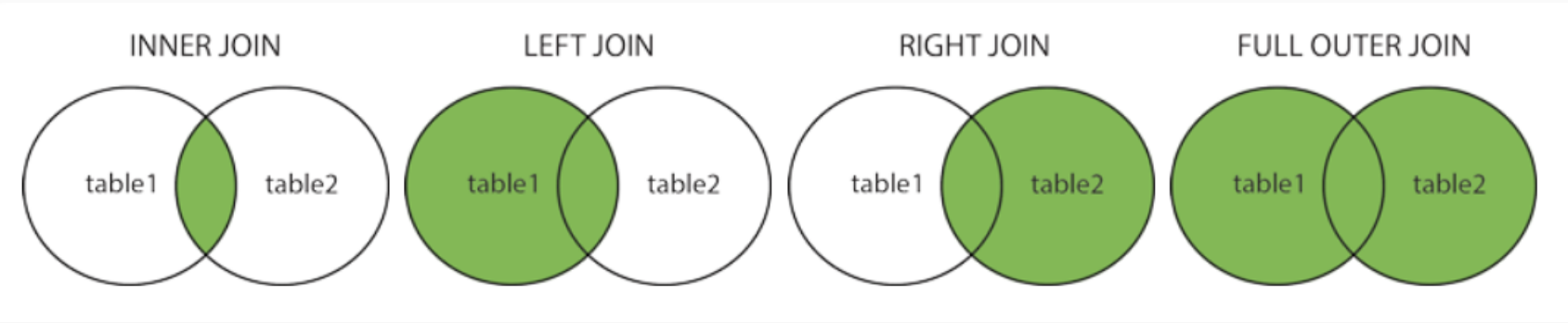
```
1 SELECT tab1.* FROM tab1
2 JOIN tab2
3 ON tab1.tab2_fk = tab2.id;
```

This syntax allows us to decide what happens if one of the tables contains no relevant values (e.g. ‘Give me the names of all course participants and their Bachelor field of origin, in case they already have a Bachelor’s degree.’).

JOINs ii

Show me the distance to each of the lakes in the database from the hotel with the name „Schlossberghotel“:

```sql
SELECT see.name, strasse.distanz
FROM strasse, see, hotel
WHERE hotel.name = 'Schlossberghotel'
    AND strasse.hotel_fk = hotel.hotel_id
    AND strasse.see_fk = see.see_id ;
```



Source: W3Schools Joins

JOINs iii

JOINs

Linking tables using conditions is called ‘joining them’. Joins are so common in database applicatoins that they have their own keyword in SQL: *JOIN*.
The example from before:

```
SELECT title, publisher_name
FROM books, publishers
WHERE books.publisher_id = publishers.publisher_id ;
```

...can also be written like this:

```
SELECT title, publisher_name
FROM books JOIN publishers
ON books.publisher_id = publishers.publisher_id ;
```

JOINs iv

JOIN and USING

If both tables linked by the join *JOIN* share the same names, you can shorten the command like so:

```
SELECT title, publisher_name
FROM books JOIN publishers USING ( publisher_id );
```

NATURAL JOIN I

If two tables share the same column names, one can use *NATURAL JOIN* to link them. Be careful – the DBMS will try to use *all* common column names for the comparison!

```
SELECT title, publisher_name
FROM books NATURAL JOIN publishers ;
```

NATURAL JOIN II

Will give the same result as the last query. However, a *NATURAL JOIN* will compare **all** columns sharing the same name in both tables. If, for example, the table *publishers* had another column called *title*, the *NATURAL JOIN* would deliver an empty result because the query would correspond to this:

```
SELECT title, publisher_name
FROM books, publishers
WHERE books.publisher_id = publishers.publisher_id
AND books.title = publishers.title ;
```

The DMBS will try to link **all** columns with the same name in a *NATURAL JOIN*!

JOINS over multiple tables

In practice you often need to link more than two tables to achieve the desired result. That's why *JOIN* can be used as many times as needed per query:

```
SELECT title, publisher_name, binding
FROM books
JOIN publishers USING ( publisher_id )
JOIN bindings ON books.binding_id = bindings.id ;
```

JOINs for n:m relationships

Tables used n:m relationships function the same way as all others. To query all authors of a book, you would use the following command:

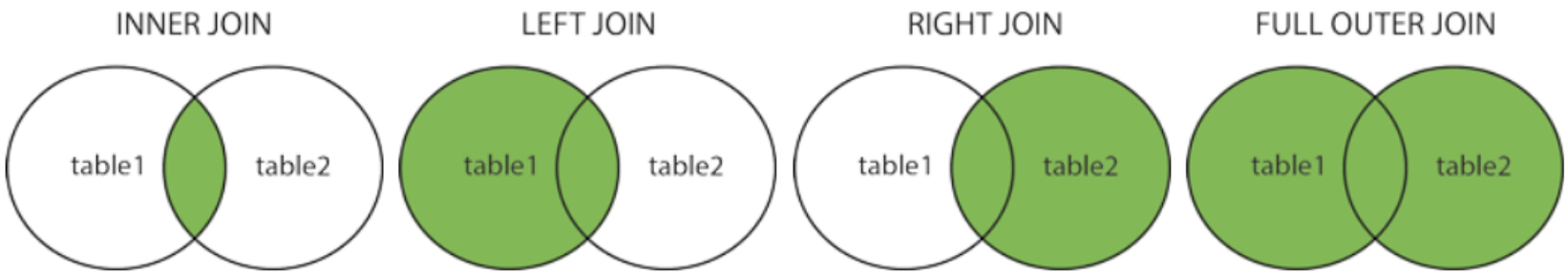
```
SELECT books.title , authors.firstname , authors.lastname
FROM books
JOIN authors_books USING ( book_id )
JOIN authors USING ( author_id ) ;
```

Each combination will be outputted in a single line. Thus, if a book has three authors, for example, you would get three a three-line result.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



Source: W3Schools Joins

INNER JOIN

Until now, we have been using *JOIN* to link tables. However, strictly speaking, these were all *INNER JOIN*s. Because *INNER JOIN*s are so frequent, you can use the shorthand *JOIN* instead of the long form.
But if there is an *INNER JOIN*, there must be an *OUTER JOIN* as well. The difference is in the way it handles missing values: An *INNER JOIN* would display nothing if there is no value corresponding to the foreign key, whereas the *OUTER JOIN* would.
Let's check if there are books with no publisher:

```
1 SELECT book_id, title FROM books WHERE publisher_id IS NULL ;
```

INNER versus OUTER JOIN: INNER

What happens in an *INNER JOIN* when we come across a book without a value for the *publisher* field?

```
1 SELECT books.title, publishers.publisher_name
2 FROM books
3 INNER JOIN publishers USING ( publisher_id )
4 WHERE books.book_id IN (3199, 3209, 3210) ;
```

Since there is no corresponding entry in *publishers* the data from *books* isn't printed either.

(INNER) JOIN

Only data points having a correspondent on both sides are printed:

```
1 SELECT studis.name, titel.abk
2 FROM studis
3 JOIN titel
4 ON studis.titel = titel.abk ;
```

LEFT JOIN

All entries in the table before *LEFT JOIN* are printed and the ones from the one on the right only if they correspond to the *ON* clause.

```
1 SELECT studis.name, titel.abk
2 FROM studis
3 LEFT JOIN titel
4 ON studis.titel = titel.abk ;
```

JOINs xii

```
1 SELECT distanz, see.name, hotel.name
2 FROM strasse
3 INNER JOIN see ON see.see_id = strasse.see_fk
4 INNER JOIN hotel
5 ON hotel.hotel_id = strasse.hotel_fk;
```

JOIN (FROM)

Querying entities or relationships (entries from multiple tables) which are connected using foreign keys.

INNER JOIN

Indicates which tables (or their entries) are linked up, i.e. represented in a new view together.

ON

Defines the criterion on which to perform the join.

INNER versus OUTER JOIN: OUTER

In an *OUTER JOIN* the search result looks like the following:

```
1 SELECT books.title, publishers.publisher_name
2 FROM books
3 LEFT OUTER JOIN publishers USING ( publisher_id )
4 WHERE books.book_id IN (3199, 3209, 3210) ;
```

```
+----------------------------------------+----------------+
| title                                  | publisher_name |
+----------------------------------------+----------------+
| Linked Data                            | NULL           |
| Writing Ideomatic Python for Python 2.7 | NULL           |
| Writing Ideomatic Python for Python 3.3 | NULL           |
+----------------------------------------+----------------+
2 rows in set (0.00 sec)
```

These are the three entries from earlier which were missing in our first *COUNT(*)* example: *books* contains 353 entries but the *JOIN* only resulted in 350 because these three entries have missing values in the *publisher* field.

LEFT OUTER JOIN

OUTER JOIN can't be used on it's own. You need to specify the directionality, indicating in which table values can be missing.

```
1 SELECT books.title, publishers.publisher_name
2 FROM books
3 RIGHT OUTER JOIN publishers USING ( publisher_id )
4 WHERE books.book_id IN (3199, 3209, 3210) ;
```

```
    Empty set (0.00 sec )
```

RIGHT OUTER JOIN

If we invert the order of the tables in the query, the *RIGHT OUTER JOIN* works:

```
SELECT books.title, publishers.publisher_name
FROM publishers
RIGHT OUTER JOIN books USING ( publisher_id )
WHERE books.book_id IN (3199, 3209, 3210) ;
```

FULL (OUTER) JOIN

FULL OUTER JOIN is equivalent to *FULL JOIN*. It will give you all entries of the database, irrespective of missing values. Be careful, the result may be large!

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

By comparison, the *INNER JOIN* that is our most common, go-to normal *JOIN* will handle missing values very differently: It will only print complete records! If one value is missing, the whole record will not be printed, so make sure to use *COUNT(*)* for the result of different types of joins (full outer or normal inner join) to check if the numbers differs!

Exercise using `urlaub.db`

Exercise using `urlaub.db`

Exercise: Overview over the DB

Open the example database *`urlaub.db`* and construct SQL queries for the following tasks:

1. You want to get an overview of the attributes of the lakes in the table *`see`*. Look up the structure of the table using *`.schema`* and then, query all information contained in the table for maximum two lakes.
2. You're interested mainly in the names and depths of the lakes. Query just those two attributes!
3. You're scared of lakes deeper than 10 meters, thus, query only those which aren't deeper than 10m.

Getting an overview

First find out which info (table names) the database even contains using *`.schema`*, then try...

```
SELECT * FROM table;
etc.
```

Exercise using `urlaub.db`

Exercise 1 continued...

You like your creature comforts but would like to save money still. Thus, query the names and hotel IDs for all the hotels which have more than three stars (*austattung*) and are priced below 100€.

Advanced: Linking tables

1. You want the hotels and lakes which fit your requirements *and* are connected by a road. First, find out which hotels and lakes are connected by a direct path by inspecting the *straße* table and including **hotel** and *see* using a join.
2. Only query the connecting roads for hotels and lakes which meet your requirements and count the number of results.
3. Your ideal lake-hotel combination is the one where the length of the road/connection is as short as possible: Sort the result in ascending order highlighting the shortest distance.
4. Find out the average price of hotels per star rating.
5. **Bonus:** What is the name of the place in which the hotel with the shortest distance to your preferred lake is located?

Sample solution for the `urlaub.db` exercise

Sample solution for the `urlaub.db` exercise i

At first, we need to find out which tables even exist in the database (what are their names so we can use the *.schema* command on them): What is the structure of the *see* table?

```
.tables
SELECT * FROM see;
.schema see
.mode column
```

Only show two results (as a preview):

```
SELECT * FROM see
LIMIT 2;
```

There is a column *tiefe_in_m*:

```
SELECT name, tiefe_in_m FROM see;
```

Now we want to select lakes whose depth is smaller than 10m:

```
SELECT name, tiefe_in_m FROM see
WHERE tiefe_in_m < 10;
```

Sample solution for the `urlaub.db` exercise ii

Concerning the hotels:

```
.tables
SELECT * FROM hotel;
.schema hotel
```

Show only *name* and *hotel_id* for hotels with more than 3 stars which are under 100€:

```
SELECT name, hotel_id FROM hotel
WHERE zimmerpreis < 100
AND ausstattung >= 3;
```

For each of these examples, reflect whether you mean 'equals' or '>=/<='.
Do we need to use parentheses so the query works?
Hint: It's usually more of a big deal with *OR*...

Sample solution for the `urlaub.db` exercise iii

Find all hotels and lakes conected by a road:

```
SELECT see.name, strasse.distanz, hotel.name
FROM strasse, see, hotel
WHERE strasse.hotel_fk = hotel.hotel_id
    AND strasse.see_fk = see.see_id ;
```

Or using *INNER JOIN*:

```
SELECT distanz, see.name, hotel.name
FROM strasse
INNER JOIN see ON see.see_id = strasse.see_fk
INNER JOIN hotel
ON hotel.hotel_id = strasse.hotel_fk;
```

Sample solution for the `urlaub.db` exercise iv

Only show those which meet the criteria we defined earlier:

```
SELECT see.name, strasse.distanz, hotel.name
FROM strasse, see, hotel
WHERE strasse.hotel_fk = hotel.hotel_id
    AND strasse.see_fk = see.see_id
    AND see.tiefe_in_m < 10
    AND (hotel.zimmerpreis < 100 AND hotel.ausstattung >= 3);
```

Now we want to sort by minimum distance, showing the results in ascending order so that the shortest distance comes first. Modify the previous query by adding this at the end:

```
ORDER BY strasse.distanz ASC;
```

Sample solution for the `urlaub.db` exercise v

Average price of the hotels:

```
SELECT AVG(zimmerpreis) FROM hotel;
```

Or for all with less than 3 stars:

```
SELECT AVG(zimmerpreis) FROM hotel
WHERE ausstattung < 3;
```

For hotels with 5 stars:

```
SELECT AVG(zimmerpreis)
FROM hotel
WHERE ausstattung = 5;
```

Where is the hotel with the shortest distance to the lake corresponding to your criteria?

```
SELECT hotel.ort_id ort, hotel.name,
       see.name, MIN(strasse.distanz)
FROM strasse, see, hotel
WHERE see.tiefe_in_m < 10  AND strasse.see_fk = see.see_id
    AND strasse.hotel_fk = hotel.hotel_id
    AND (hotel.zimmerpreis < 100 AND hotel.ausstattung >= 3);
```

SQL addenda (auxiliary tables)

Adding multiple values using auxiliary tables i

Example for the use of auxiliary tables (from StackOverflow)

- Each cell in a SQL table can only have one value.
- Problem: Some cells should actually be able to have more than one value.
- Solution: Use auxiliary tables.

```sql
CREATE TABLE Product (
  ProductID INTEGER PRIMARY KEY,
  ProductName TEXT UNIQUE
);

CREATE TABLE Category (
  CategoryID INTEGER PRIMARY KEY,
  CategoryName TEXT UNIQUE
);

CREATE TABLE Product_Category (
  RecordID INT AUTO_INCREMENT PRIMARY KEY,
  CategoryID INT,
  ProductID INT
);
```

Adding multiple values using auxiliary tables ii

Example data record for the table:

```
INSERT INTO Category VALUES (1, 'Fruit');
INSERT INTO Category VALUES (2, 'Vegetable');

INSERT INTO Product
VALUES (1, 'Apple'),
       (2, 'Banana'),
       (3, 'Cabbage'),
       (4, 'Squash'),
       (5, 'Tomato');

INSERT INTO Product_Category (CategoryID, ProductID)
VALUES (1,1), (1,2), (2,3), (2,4), (1,5), (2,5);
```

SQL addenda (special queries)

Querying multiple values i

- In a common query, I can only query once from each table.
- Problem: Assuming I want to query sender and recipient which are listed in the *Briefe* table with Foreign Key but both come from the *Personen* table.
- this db setup makes sense to avoid redundant data, so it's a legitimate problem you might encounter.
- Solution: Include a subquery (cf. StackOverflow).

```sql
SELECT
    e.ID,
    e.Name,
    e.Boss,
    (SELECT Name FROM Employees b
     WHERE b.ID = e.Boss) as BossName
FROM Employees e;
```

Example by a previous student:

```sql
SELECT l.Signatur_FK as Signatur,
 (SELECT AutorName FROM Personen a
  WHERE a.ID_Person = l.ID_Empfaenger_FK)
  as Empfaenger,
 (SELECT AutorName FROM Personen b
  WHERE b.ID_Person = l.ID_Absender_FK)
  as Absender
FROM Abse_Empf l; /* l for the list */
```

Markup & annotation

Annotating in the Humanities

→ Typical practice for the Humanities. Can be done on paper.
You can go about it in many ways:

- collect specific metadata to enrich your primary data with (*research-driven modelling*)
- provide metadata as general as possible to facilitate reuse (*curation-driven modelling*)
- we can provide administrative or technical metadata but also encode semantic information (implicit structures) for the machine.
- Often: describe logical structure of documents for computers ('This is a heading. This is a paragraph.')

Example:

```
<name type="person">J. W. v. Goethe</name>
```

In the digital realm

Adding additional data to source document in a machine-readable format.

Formal models

If you model is machine-processable, it's a formal model – i.e. markup creates a formal model of your data.

Many names for the same thing

markup, encoding, annotating, ...

Different levels of annotation

Base annotation Encode formal criteria of the text structure (i.e. headings, paragraphs), adding some metadata. Mostly *presentational*.

Data enrichment Going from formal aspects to semantics: Annotating personal names, places (*Named Entities*), linking those to norm data, thus creating *Linked (Open) Data* (LOD).

Examples for norm data

- GND (*Gemeinsame Normdatei*, Integrated Authority File)
- GeoNames
- ...

Markup languages i

Annotation is also called mark-up.

> **Markup** refers to data included in an electronic document which is distinct from the document's content in that it is typically not included in representations of the document for end users, for example on paper or a computer screen, or in an audio stream. **Markup is often used to control the display of the document or to enrich its content to facilitate automated processing.**
>
> Older markup languages [...] typically **focus on typography and presentation**, [...] most modern markup languages, for example XML, **identify document components** (for example headings, paragraphs, and tables), with the expectation that technology such as stylesheets will be used to apply formatting or other processing.
>
> Some markup languages, such as the widely used HTML, have **pre-defined presentation semantics**, meaning that their specification prescribes some aspects of how to present the structured data on particular media. (Wikipedia)

Markup languages ii

machine readable

SMGL HTML XML ...

presentational vs. descriptive / semantic:

- e.g. ‘font size 14pt’ vs. ‘heading’ (=type). (explicit vs. implicit)
- text formatting vs. meaning of the text
- procedural, representative, descriptive / conceptual (semantic)
- WYSIWYG text processing vs. WYSIWYM
- Advantages of using macros in WS Word: change the settings once for the whole document
- this is achieved when we separate content from its presentation
- there are different ‘views’ on markup documents
- browser ‘renders’ HTML: I can see it in two different ways – rendered or as the HTML code

Markup languages iii

- There are tools to switch documents between different types of markup! (not all formats work equally well): **Pandoc** or **OxGarage** (for TEI mostly).
- Binary document formats, such as *.doc*, *.pdf*, *.dvi* (TEX output format) $\neq$ markup: You can tell by the fact that you cannot look at their 'code view'.
- A `.docx` file is a `.zip` archive (try unzipping it!) which contains XML files (but it's complicated).
- **Goal of markup:** Make the implicit (you know it's a heading) explicit for the computer (doesn't know otherwise).

Markup is all around you!

SGML

Standard Generalized Markup Language

.SGML

Introduces the principles of

the separation of form and content

Is a metalanguage (like XML)

HTML & XML are both derived from the older SGML (thus the similarity) – but HTML has a fixed tag set whereas XML is by definition *extensible*.

RTF

Rich Text Format Microsoft 1987 ● exchange format between text editors (different operating systems, such as Mac and Windows, didn't create interchangeable output formats).

Unlike plain text, RTF contains markup for formatting text which can then be 'retranslated' into the native editors.

.RTF

```
{\rtf1
Hello!
\line
{\i This} is \b{\i a
\i0 formattted \b0text}.
\par
\b THE \b0END.
}
```

JSON

.JSON JavaScript Object Notation pronounced like 'Jason' ● compact data format ● human readable ● set up in key-value pairs ● nested (like XML)

JSON vs. XML differences?

XML = describes structure, JSON = non-declarative syntax convention ● JSON: defines instances of structured data ● very flexible ● 'lightweight': little overhead, easier to read for data in key-value format ● valid javascript: can be instantiated into a javascript object via the *eval()* function.

Conclusion

JSON has advantages if you just need simple key-value pairs (**'simplicity'**). XML has more and allows for more **complexity**.

XML = mark up language

JSON = data exchange format

```
{
  "publisher": "Xema",
  "number": "1234-5678-9012-3456",
  "owner":
  {
    "Name": "Mustermann",
    "Vorname": "Max",
    "male": true,
    "hobbies": ["surfing", "chess"],
    "age": 42,
    "kids": [],
    "spouse": null
  }
}
```

PostScript

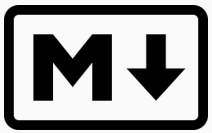
Page description language ● 1980s (Adobe Systems) ● vector graphic format for printers ● but also: Turing-complete, stack-oriented programming language ● used to be the standard in the printing industry ● today PDF (*Portable Document Format*) (also by Adobe, developed from PS) has become the standard ● could be generated via postscript printer drivers from all sorts of documents ● processed via 'Ghostscript' in UNIX ● decribes documents as scalable vector graphics which allows for loss-less zooming / scaling .ps // presentational

Example

This example program writes 'Hello World!' to position 50,50. By default, the PS coorindate system starts from the bottom left corner.

```
%!
/Courier findfont     % font type
20 scalefont          % font size 20
setfont               % set it
50 50 moveto          % (50, 50)
                      % = writing pos
(Hello World!) show   % print text
showpage              % show page
```

Markdown



→ Markdown in 60s

simple text formatting

.md // presentational

`*Italic*`	*Italic*
`**Bold**`	**Bold**
`## Heading 1`	Heading 1
`#### Heading 2`	etc.
`[Link]{http://a.com}`	‘hidden’ link
`![Image][http://url/a.png`	image
`> Blockquote`	quote
`- List`	list (can be nested, 2 spaces)
`* List`	alternative list
`1. enumerate`	numbered list
`---`	separator line
`` `Inline code` ``	Code (‘backticks‘)
`` ```code block``` ````	code block

Practice!

Try Markdown in 60s or 10min exercise

A primer on data structures

Digital data representation

→ i.e. machine processable

Digital representations

- Images
  - raster graphics (*.png*, *.jpeg*)
  - vector graphics (*.svg*)
- Text
  - plain text (*.txt*)
  - formatted text (*.docx*, *.rtf*, *.xml*, *.tex*)
- Lists & tables (*.csv*, *.xlsx*)
- Sound (*.wav*, *.midi*)
- **Objects:** (Simulated) 3D view, abstracted representation by description and images

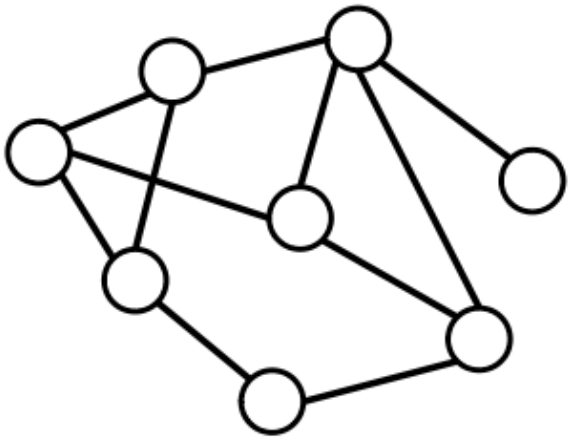
Further types

- markup languages (*.xml*, *.html*, etc.)
- data objects (*.json*, etc.)
- graphs / graph databases (*.rdf*, etc.)
- relational databases → SQL

Data structures: Graphs

Applications

- **Resource Description Framework (RDF):**
  - e.g. Blazegraph (Graph-DB)
  - Query: SPARQL
- Labeled Property Graphs
  - e.g. Neo4j (Graph-DB)
  - Query: Cypher



RDF/Turtle-Notation (`.ttl`)

```
@prefix ex: <http://example.com/#> .

ex:Graz a ex:city;
ex:name "Graz" ;
ex:inhabitants 288806 ;
ex:location [ ex:lat 47.4; ex:long 5.26 ] .

ex:Wien a ex:city;
ex:name "Wien" ;
ex:inhabitants 1897491 ;
ex:location [ ex:lat 47.12; ex:long 16.22 ] .
```

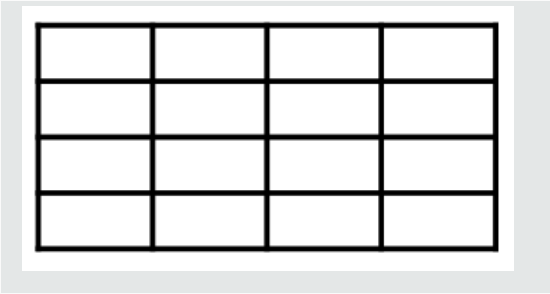
SPARQL query

```
@prefix ex: <http://example.com/#> .

SELECT ?name, ?population
WHERE {
    ?city ex:inhabitants ?population .
}
```

Data structures: tabular data / relational databases

Applications

- e.g. SQLite, MySQL (relational dbs)
- Query: SQL



SQL query

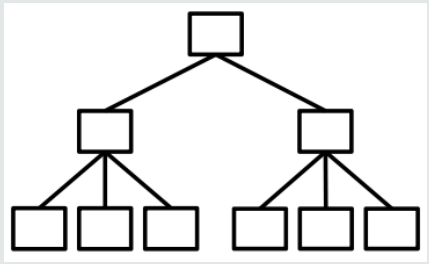
```sql
CREATE TABLE "places" (
 "name" TEXT,
 "population" INTEGER,
 "longitude" REAL,
 "latitude" REAL,
 PRIMARY KEY("name")
);

INSERT INTO places
VALUES
('Graz', 288806, 47.066667, 15.433333),
('Wien', 1897491, 48.208174, 16.373819);
```

(name, population, longitude, latitude)

Data structures: tree hierarchy 1 / JSON

Applications: **JavaScript Object Notation (JSON)**

- e.g. MongoDB
- no standardized query language, just JavaScript (*.js*)



JSON & JS (w3s)

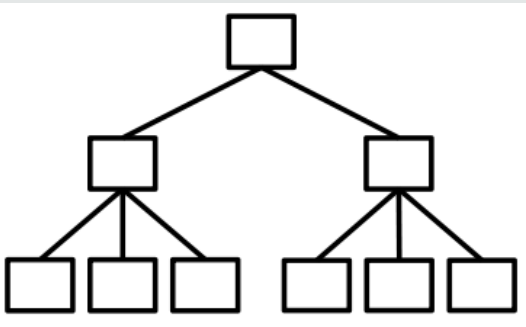
```
'{"name": "John", "age": 30, "car": null,
  "tree" : [
    "key": "value"
  ]
}'
const obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}');
obj.age = obj.age.toString();
```

Data structures: tree hierarchy 2 / XML

Applications:

eXtensible Markup Language (XML):

- DBs: eXist, BaseX
- Query: XPath (w3s) and XQuery (w3s)

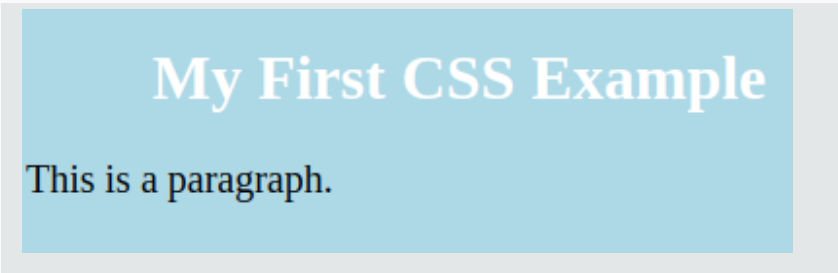


```
<!-- books.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>

<!-- XQuery -->
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title

<!-- XPath -->
//title[@lang='en']
/bookstore/book[price>35.00]
```

Data structures: tree hierarchy 3 / web pages (HTML)

HTML (w3s) – structure

```
<!DOCTYPE html>
<html>
 <head>
   <title>Page Title</title>
 </head>
 <body>
  <h1>This is a Heading</h1>
  <p>This is a paragraph.</p>
 </body>
</html>
```



CSS in HTML (w3s) – rendering

```
<!DOCTYPE html>
<html>
  <head>
    <style>
body {
  background-color: lightblue;
}
h1 {
  color: white;
  text-align: center;
}
p {
  font-family: verdana;
  font-size: 20px;
}
    </style>
  </head>
  <body>
    <h1>My First CSS Example</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

Why so many data formats?

Different data formats (& standards) focus on different aspects & have different goals:

1. **text-based**
   - Text Encoding Initiative (**TEI**)
   - Extensible Hypertext Markup Language (**XHTML** = XML-compliant HTML)
   - Open Document Format for Office Applications (**ODF**)
2. **page-based**
   - TEX / LATEX
   - **XSL-FO** (XSL Formatting Objects, discontinued)
3. **ontology-based**
   - Resource Description Framework (**RDF**) & RDF Schema (**RDFS**)
   - Web Ontologie Language (**OWL**)
   - Simple Knowledge Organisation System (**SKOS**)
   - Conceptual Reference Model (**CIDOC-CRM**)
4. **digital archiving / digital objects**
   - Dublin Core Metadata Initiative (**DCMI**), known as Dublin Core (**DC**)
   - Metadata Encoding and Transmission Standard (**METS**)
   - Metadata Object Description Schema (**MODS**)
   - Encoded Archival Description (**EAD**)
   - Charters Encoding Initiative (**CEI**)

A primer on metadata formats

A primer on metadata

What are metadata?

- „data about data“
  1. data about containers of data = **structural metadata**
  2. data about the content represented by data = **descriptive metadata**
- functions:
  1. descriptive
  2. administrative
  3. technical
  4. use

There are standards for the description of metadata (and many are XML-based), e.g.

- Machine-Readable Cataloging (**MARC**)
- Metadata Object Description Schema (**MODS**)
- Encoded Archival Description (**EAD**)
- Lightweight Information Describing Objects (**LIDO**)
- Collective Description of Works of Art (**CDWA**) / Visual Research Association (**VRA**)
- Europeana Metadata Model (**EDM**)
- Resource Description Framework (**RDF**)
- Metadata Encoding & Transmission Standard (**METS**)
- Dublin Core (**DC**)
- Functional Requirements for Bibliographic Records (**FRBR**)
- the *<teiHeader>* has metadata…

Slides on metadata in DH with more information

Dublin Core (DC) i

What is the DC?

- founded in Dublin (Ohio) in 1995
- **two levels:** simple (15 elements) & qualified (additional *`Audience`*, *`Provenance`* and *`RightsHolder`*)
- **classes of terms:** elements (nouns) & qualifiers (adjectives).
- can be expressed in RDF/XML
- each element is optional & can be repeated
- also: dc:terms

> **The Dublin Core™ metadata standard** is a simple yet effective **element set for describing a wide range of networked resources.** [...] Another way to look at Dublin Core™ is as a "small language for making a particular class of statements about resources". In this language, there are two classes of terms – *elements* (nouns) and *qualifiers* (adjectives) – which can be arranged into a simple pattern of statements.
> (source)

Dublin Core (DC) ii

The Core

i.e. the elements:

1. title
2. subject
3. description
4. type
5. source
6. relation
7. coverage
8. creator
9. publisher
10. contributor
11. rights
12. date
13. format
14. identifier
15. language

Qualified

1. (audience)
2. (provenance)
3. (rights holder)

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="http://media.example.com
                               /audio/guide.ra">
      <dc:creator>Rose Bush</dc:creator>
      <dc:title>A Guide to Growing Roses</dc:title>
      <dc:description>Describes process for
        planting and nurturing different kinds
        of rose bushes.</dc:description>
      <dc:date>2001-01-20</dc:date>
   </rdf:Description>
</rdf:RDF>
```

Note the two namespaces *rdf:* and *dc:*.

Metadata Encoding and Transmission Standard (METS)

METS

- tool for encoding digital library objects
- container format for documents in which contents of different formats can be integrated
- also describes relationships between objects
- describes logical and physical structure of an object
- also contains descriptive (bibliographical) and administrative metadata
- relatively simple and straightforward
- supports a wide range of materials
- <website> & more info here
- DFG-Viewer: *http://dfg-viewer.de/*

Only structural map is required.

```
<mets>
  <metsHdr/>
  <dmdSec/>
  <amdSec/>
  <fileSec/>
  <structMap/>
  <structLink/>
  <behaviorSec/>
</mets>
```

Goals

- link/summarize related metadata
- e.g. link related images to text
- organize data
- provide usage metadata

Metadata Object Description Schema (MODS)

MODS

- can represent the major elements from a MARC record

→ represents key bibliographic data in easily understandable names

→ easier to understand than MARC (for the uninitiated)

- bridges the gap between library application and bibliographic source that don't make use of cataloging metadata formats
- richer than the Dublin Core (DC) but less detailed than MARC
- partially backwards compatible with MARC

```
<mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
  <mods:titleInfo>
    <mods:nonSort>The </mods:nonSort>
    <mods:title>
      1946 Library of Congress recital
    </mods:title>
  </mods:titleInfo>
  <mods:relatedItem type="constituent"
                    ID="DMD_disc01_tr001">
    <mods:titleInfo type="uniform">
      <mods:partName>Chaconne von Vitali
        </mods:partName>
    </mods:titleInfo>
  </mods:relatedItem>
  <mods:identifier type="lccn">99594334
    </mods:identifier>
</mods:mods>
```

Code example from here.

Encoded Archival Description (EAD)

EAD

is a standard for encoding descriptive information regarding archival records

example EAD & Wikipedia (source of the example)

```
<eadheader>
   <eadid countrycode="us" identifier="bachrach_lf">
     bachrach_lf</eadid>
   <filedesc>
      <titlestmt>
         <titleproper encodinganalog="Title">
          Louis Fabian Bachrach Papers</titleproper>
         <subtitle>An inventory of his papers at
           Blank University</subtitle>
         <author encodinganalog="Creator">Mary Smith</author>
      </titlestmt>
      <publicationstmt>
         <publisher encodinganalog="Publisher">
           Blank University</publisher>
         <date encodinganalog="Date" normal="1981">
           1981</date>
      </publicationstmt>
   </filedesc>
   <profiledesc>
      <creation>John Jones
         <date normal="2006-09-13">13 Sep 2006</date>
      </creation>
   </profiledesc>
</eadheader>
```

Charters Encoding Initiative (CEI)

CEI

considers the possibilities of a standard to encode medieval and early modern charters with XML.

- implements the *Vocabulaire Internationale de Diplomatique*
- founded in 2004
- MOM-CA, the collaborative charter archive of Monasterium.net works with CEI
- <website> & an example. Also: the example below.

```
<text type="charter">
  <idno>600202b</idno>
  <chDesc id="a">
    <head>Prag, 1360 Febr. 2.</head>
      <issued>
         <placeName>Prag</placeName>
         <date>1360-02-02</date>
      </issued>
      <abstract>
         <p> Karl verspricht Ludwig ...
         </p>
      </abstract>
      <witList>
         <witness sigil="B">
            Brandenburgisches LHA Potsdam
            “Rep. 37 Hohennauen Nr. 683,
            fol. 225” (18. Jh.)
         </witness>
      </witList>
      <diplomaticAnalysis>
         <bibl type="D">Fidicin, 42...</bibl>
      </diplomaticAnalysis>
   </chDesc> <tenor> </tenor>
</text>
```

Resource Description Framework (RDF)

RDF

- framework for describing resources in the World Wide Web
- can contain metadata
- language of the ‘Semantic Web’ (web 3.0) – makes things machine-processable
- RDF Schema (RDFS) offers Classes and Properties

RDF/ turtle notation (`.ttl`) example from before

```
@prefix ex: <http://example.com/#> .

ex:Graz a ex:city;
ex:name "Graz" ;
ex:inhabitants 288806 ;
ex:location [ ex:lat 47.4; ex:long 5.26 ] .
```

Simple Knowledge Organization System (SKOS)

SKOS

RDF vocabulary for representing semi-formal *knowledge organization systems* (KOSs), such as thesauri, taxonomies, classification schemes and subject heading lists.
$\rightarrow$ less rigorous than the logical formalism of ontology languages such as OWL

(SKOS primer)

```
@prefix skos:
  <http://www.w3.org/2004/02/skos/core#> .
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://www.example.com/> .
@prefix ex1: <http://www.example.com/1/> .
@prefix ex2: <http://www.example.com/2/> .

ex:animals rdf:type skos:Concept;
  skos:prefLabel "animals"@en;
  skos:narrower ex:mammals.

ex:mammals rdf:type skos:Concept;
  skos:prefLabel "mammals"@en;
  skos:broader ex:animals.
```

Europeana Data Model (EDM) i

EDM

Model to integrate data sources from different providers and thus improve interoperability:

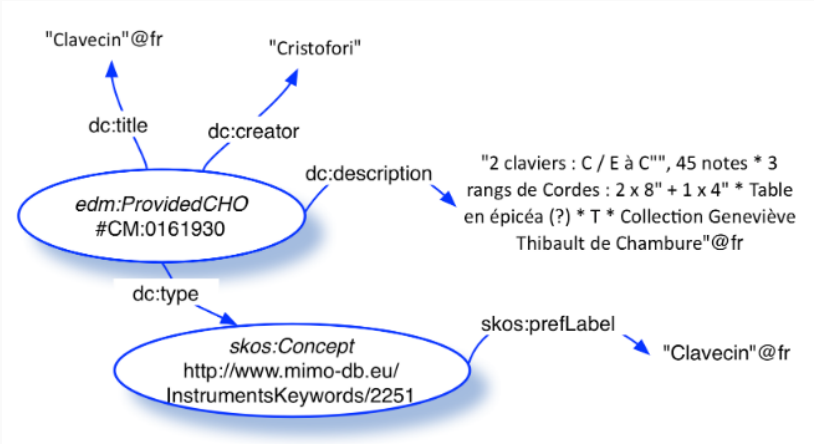
> "EDM transcends domain-specific metadata standards, yet accommodates the range and richness of community standards such as LIDO for museums, EAD for archives or METS for digital libraries." (EDM Factsheet)

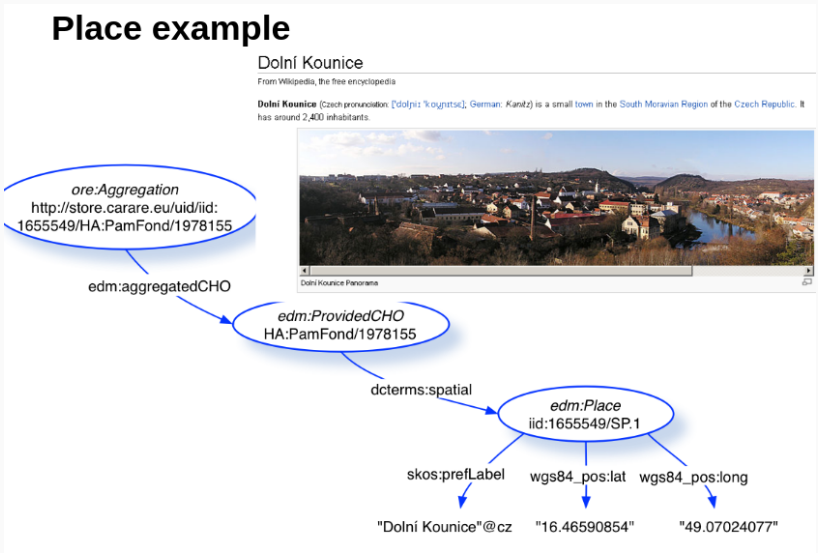
...integrates the following standards:

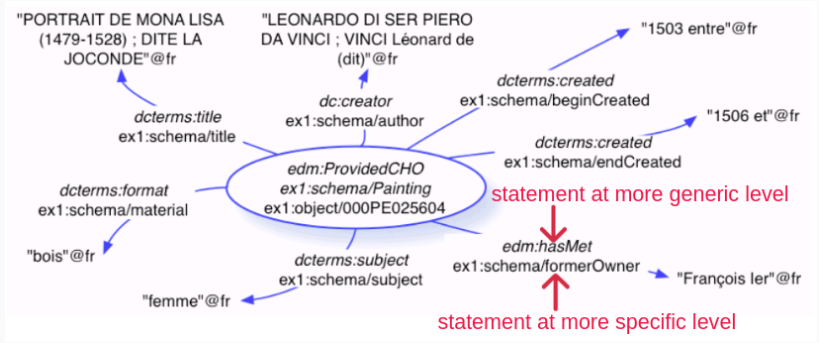
1. **OAI ORE** (Open Archives Initiative Object Reuse & Exchange) for organizing an object's metadata and digital representation(s)
2. **Dublin Core** for descriptive metadata
3. **SKOS** (Simple Knowledge Organization System) for conceptual vocabulary representation
4. **CIDOC-CRM** for event and relationships between objects

This is achieved in RDF (can be written as XML) → RDF uses the Semantic Web principles to integrate those data sources → Metadata standards aren't exclusive, they can be combined!

Europeana Data Model (EDM) ii







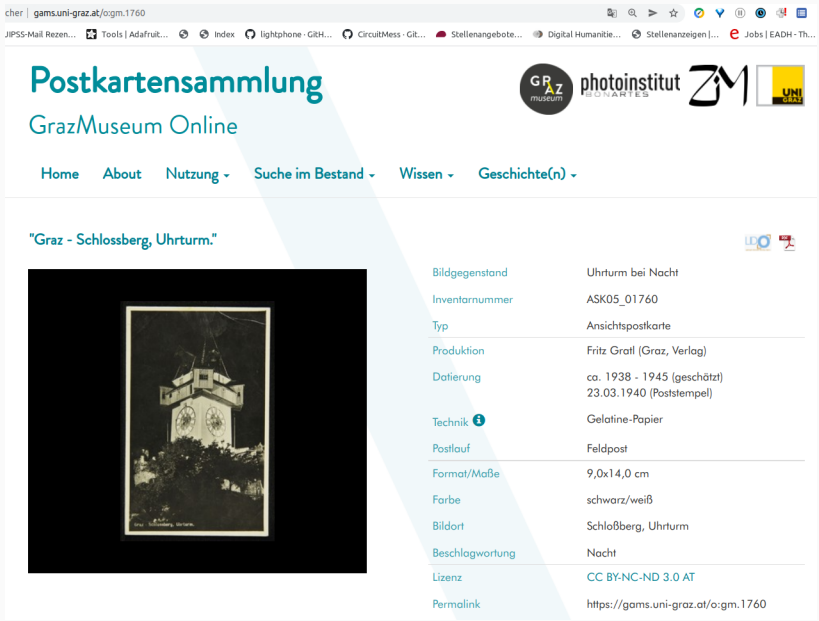
(More info.)

Lightweight Information Describing Objects (LIDO)

LIDO

> Lightweight Information Describing Objects (LIDO) is an **XML schema for describing museum or collection objects.** Memory institutions use LIDO for "exposing, sharing and connecting data on the web". It can be applied to all kind of disciplines in cultural heritage, e.g. art, natural history, technology, etc.
> LIDO is a specific application of CIDOC CRM. (Wikipedia)

A postcard (text-bearing object)




```
<lido:titleWrap>
    <lido:titleSet>
        <lido:appellationValue>
            Chickens and Ducks
        </lido:appellationValue>
    </lido:titleSet>
</lido:titleWrap>
```

LIDO example 1

Notice the two namespaces (*lido:* and *t:*)!

```
<lido:lido
  xmlns:lido="http://www.lido-schema.org"
  xmlns:t="http://www.tei-c.org/ns/1.0">
  <lido:lidoRecID lido:type="PID">o:gm.1760</lido:lidoRecID>
  <lido:category>
    <lido:conceptID lido:source="CIDOC"
                    lido:type="ID">E22</lido:conceptID>
    <lido:term xml:lang="eng">Man-Made Object</lido:term>
    <lido:term lido:label="info:fedora/context:gm">
      Postkartensammlung Online</lido:term>
  </lido:category>
  <lido:descriptiveMetadata xml:lang="deu">
    <lido:objectClassificationWrap>
      <lido:objectWorkTypeWrap>
        <lido:objectWorkType>
          <lido:conceptID lido:source="http://vocab.getty.edu/aat"
            lido:type="ID">300026819</lido:conceptID>
          <lido:term lido:label="info:fedora/context:gm-ansicht">
            Ansichtspostkarte</lido:term>
        </lido:objectWorkType>
      </lido:objectWorkTypeWrap>
    </lido:objectClassificationWrap>
    <!-- to be continued... LIDO is very verbose --->
```

LIDO example 2

```
<!-- continued... --->

    <lido:objectIdentificationWrap>
      <lido:titleWrap>
        <lido:titleSet>
          <lido:appellationValue>Graz - Schlossberg, Uhrturm.
             </lido:appellationValue>
        </lido:titleSet>
      </lido:titleWrap>
      <lido:repositoryWrap>
        <lido:repositorySet lido:type="orgname">
          <lido:repositoryName>
            <lido:legalBodyID lido:source="http://d-nb.info/gnd"
              lido:type="ID">2022740-1</lido:legalBodyID>
            <lido:legalBodyName>
              <lido:appellationValue>GrazMuseum</lido:appellationValue>
            </lido:legalBodyName>
          </lido:repositoryName>

    <!-- to be continued... LIDO is very verbose --->
```

Internat. Image Interoperability Framework (read: *Triple-I-F*)

IIIF

> The International Image Interoperability Framework (*https://iiif.io/*) defines several application programming interfaces that provide a standardised method of describing and delivering images over the web, as well as "presentation based metadata" about structured sequences of images. (Wikipedia)

The standard

- proposed in 2011
- 2012: Version 1.0
- **Image API:** URL for viewing the images
- **Presentation API:** standard for describing a sequence of canvases and the images they are represented by as a representation of an object (*manifest*, *manifest.json*)

{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}

https://fragmentarium.ms/loris/F-t38z/New_York_State_Library_recto_.jpg/full/full/0/default.jpg

https://fragmentarium.ms/loris/F-t38z/New_York_State_Library_recto_.jpg/**100,100,100,100**/full/0/default.jpg

https://fragmentarium.ms/loris/F-

IIIF Image API

region

defines the rectangular portion of the underlying image content to be returned

size

determines the dimensions to which the extracted region is to be scaled

rotation

otation parameter specifies mirroring and rotation

IIIF Presentation API

example 1, example 2

quality

determines whether the image is delivered in color, grayscale or black and white (values: color, gray, bitonal, default)

format

format of the returned image (e.g. jpg, tif, gif, jp2, pdf, webp)

Mirador Web Viewer

Fully featured IIIF Viewer:
https://projectmirador.org

{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}

What’s next?

What's next?

Connection to other Zim classes

Many of the contents this course only touched upon are treated in more detail in follow-up classes offered here.

X-Technologien (1+2) XML, XPath, XSLT, XQuery, XML databases, text modelling

Intro to Programming (1+2) Data structures and data types

Webentwicklung HTML, CSS, JavaScript

Informationsmodellierung 2 ER, UML; tables, graphs, trees; focus on graphs/RDF; queries in SQL and SPARQL

Textmining / Data Science analyzing text or structured data

Langzeitarchivierung/Datenmanagement long-term archiving

Grundfragen-Seminar What is DH? Current discourses; corpus and data criticism.

Thanks for taking part in this class!

Please don’t forget to evaluate in UGO.

References i

- Chen, Peter (1976). *The Entity-Relationship Model – Toward a Unified View of Data*.
- Jannidis, Fotis (2017). “Grundlagen der Datenmodellierung”. In: *Digital Humanities. Eine Einführung*. Ed. by Fotis Jannidis et al. Stuttgart, pp. 99–108.
- Stachowiak, Herbert (1973). *Allgemeine Modelltheorie*. Wien.