Relational Database Systems I

Christoph Lofi
Simon Barthel

Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de
Data models define the structural constrains and possible manipulations of data
  – Instances of data models are called schemas
    • Careful: Often, sloppy language is used where people call a schema also a model

We have three types of schemas:
  – Conceptual Schemas
  – Logical Schemas
  – Physical Schemas

We can use ER modeling for conceptual and logical schemas
Summary last week

- Entity Type
- Weak Entity Type
- Attribute
- Key Attribute
- Multi-valued Attribute
- Composite Attribute
- Derived Attribute
- Relationship Type
- Identifying Relationship Type
• Total participation of E2 in R

• Cardinality
  – an instance of E1 may relate to multiple instances of E2

• Specific cardinality with min and max
  – an instance of E1 may relate to multiple instances of E2
3 Extended Data Modeling

• Alternative ER Notations
• Extended ER
  – Inheritance
  – Complex Relationships
• Taxonomies & Ontologies
• UML
3.1 ER – Alternative Notations

• There is a plethora of alternative notations for ER diagrams
  – different styles for entities, relationships and attributes
  – no standardization among them
  – also, notations are often freely mixed
    • ER diagrams can look completely different depending on the used tool / book

• In the following, we introduce the (somewhat popular) crow’s foot notation
• **Crow’s foot** notation was initially developed by Gordon Everest
  – derivate of 3.1 ERD notation
  – main Goal
    • consolidate graphical representation
    • provide a better and faster overview
    • allow for easier layouting
  – widespread use in many current tools and documentations
### 3.1 ER – Crow’s Foot Notation

- **Entity Types**
  - entity types are modeled with a named box
  - attribute names are written inside the box separated by a line
    - key attributes are marked with a leading asterisk
    - composite attributes are represented with indentation

![Entity Types Diagram]

<table>
<thead>
<tr>
<th>Book</th>
<th>isbn</th>
<th>first name</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* isbn</td>
<td>{ author }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>first name</td>
<td>last name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>title</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

- Book
  - isbn
  - first name
  - last name
  - { author }
  - title
3.1 ER – Crow’s Foot Notation

• **Relationship Types**
  – relationship types are modeled by lines connecting the entities
  – line is annotated with the name of the relationship which is a verb
  – cardinalities are represented graphically
    - 
      - 
    - 
      - 
    - 
      - (0, 1): zero or one
      - (1, 1): exactly one
      - (0, *): zero or more
      - (1, *): one or more
    - **Attention:** Cardinalities are written on the opposite side of the relationship (in contrast to Chen notation)
3.1 ER – Crow’s Foot Notation

- What happens to n-ary relationships or relationship attributes?
• **Problem**
  – N-ary relationship types **are not supported** by crow’s foot notation, neither are relationship attributes

• **Workaround solution:**
  – **intermediate entities** must be used
    • N-ary relationships are broken down in a series of **binary** relationship types anchoring on the intermediate entity
3.1 ER – Crow’s Foot Notation

This schema is NOT the same!
Originally, ER diagrams were intended to be used on a **conceptual** level

- model data in an abstract fashion **independent** of implementation

Crow’s foot notation sacrifices some conceptual expressiveness

- model is closer to the **logical** model (i.e. the way the data is later really stored in a system)
- this is **not** always **desirable** and may obfuscate the intended semantics of the model
• Barker’s notation
  – based on Crow’s Foot Notation
  – developed by Richard Barker for Oracle’s CASE modeling books and tools in 1986
  – cardinalities are represented differently
    • (0, 1): zero or one
    • (1, 1): exactly one
    • (0, N): zero or more
    • (1, N): one or more
  • cardinalities position similar to Crow’s Foot notation and opposite to classic ER
  – different notation of subtypes
3.1 ER – Even more notations...

- **Black Diamond Notation**
  - Cardinalities are represented differently
  - Cardinality annotation per relationship, not per relationship
    - 1:1
    - 1:N
    - N:M
  - Also, N-ary relationships possible
  - Ternary
3 Extended Data Modeling

• Alternative ER Notations
• Extended ER
  – Inheritance
  – Complex Relationships
• Taxonomies & Ontologies
• UML
3.2 Extended Data Modeling

• Traditional **ER modeling** proved to be very **successful** in classic **DB** domains:
  – accounting
  – banking
  – airlines
  – business and industry applications in general
  – …
However, in the late 70s, popularity of DBs extended into fields with more complicated data formats

- computer-aided design and manufacturing (CAD/CAM)
- geographic information systems (GIS)
- medical information systems
- ...

Expressiveness of ERD is not sufficient here
3.2 Extended Data Modeling

- Extended entity relationship (EER) models provide many additional **features** for more accurate **conceptual modeling**
  - refinement of relationship types
    - specialization and generalization
    - class, subclass, and inheritance
  - entity sets with existence dependencies
  - extended modeling of domains and constraints
- Extended ER contains all features of **classic ER**
3.2 Extended Data Modeling

• Problem

– model secret lairs to base highly secret research activities

– secret island and secret space station are special kinds of secret lairs, share many attributes, but still need some unique attributes
3.2 Subclasses / Superclasses

- **Solution:** subclasses and superclasses
- A **subclass** entity type **inherits** all attributes and constraints from its **superclass** entity type
  - subclasses may add additional attributes, constraints or relationship types
  - in EER, subclass relationship types are annotated with an open arc, which is opened to the super class (think of set inclusion)
  - describes an **is_a** relationship
• **Subclass entity types** represent subsets of the entity set of the superclass’ entity type
  – i.e. an entity which is contained in the subclass is also contained in the superclass
  – In particular, no entity can **only** exist in a subclass set
3.2 Subclasses / Superclasses

- Possible **implementation**: two distinct database entries that represent the same entity
  - the same instance appears as a database entry in the superclass and subclass sets, and they are related to each other
  - 1:1 relationship on **entity level**
    - linking two database entries of the same entity in a specialized role
  - often, this solution is easier and more flexible to implement
3.2 Specialization / Generalization

• The process of defining a set of subclasses for a superclass is called specialization
  – specialized entity types supplement additional attributes and relationships
  – Secret Lair can be specialized into Secret Space Station and Secret Island

• The inverse process is generalization
  – generalization suppresses differences among specialized subclasses
  – Secret Space Station and Secret Island are generalized to Secret Lair
• Specialization and generalization may result in the same model
  – however, the process of how to reach the model is different
  – specialization: top-down conceptual refinement
    • start with superclasses, find suitable subclasses
  – generalization: bottom-up conceptual synthesis
    • model subclasses, find proper generalized superclass
3.2 Constraints on Specialization

• Specializations can be constrained and modeled in further detail regarding two properties

– exclusiveness (indicated by a labeled circle)
  • disjoint: subclasses are mutually exclusive (default, label d)
  • overlapping: each entity may be contained in more than one subclass (label o)

– completeness
  • total: no entity is member of the superclass without being member of a subclass (denoted by double line)
  • partial: there are entities that are not contained in any subclass (default)
3.2 Constraints on Specialization

• Examples

  – **disjoint** and **total**:

A secret lair may either be a secret island or a secret space station (but nothing else).
3.2 Constraints on Specialization

• Examples

– **overlapping and partial:**

A villain is a mad scientist, or a super villain, any combination of both, or something else (just a villain).
3.2 Constraints on Specialization

- Specializations may be **predicate-defined**
  - a subclass is predicate-defined if there is a predicate (condition) that implies an entity’s membership
  - condition is added to the specialization line
  - predicate-defined specialization are not necessarily total
• Specializations may be **attribute-defined**
  – attribute-defined is a special case of predicate-defined, where the membership in subclasses depends on a **single attribute value**
  – attribute is added to line connecting circle and superclass, condition added to lines connecting circle and subclasses
3.2 Constraints on Specialization

- **Consequences of specialization**
  - **deleting** an entity from the superclass also deletes it from all subclasses
    - Deleting only from subclass has no clear semantics
  - **inserting** an entity in a superclass automatically inserts it into all matching *predicate-defined* subclasses
  - in a **total** specialization, inserting one entity into a superclass implies that it has to be inserted into **at least one** subclass, too
3.2 Hierarchies and Lattices

• A subclass may be further specialized
• If every subclass has just one superclass, the inheritance structure is a specialization hierarchy
• If there are subclasses having more than one superclass at the same time, the structure is a specialization lattice
  – shared subclasses possible with multiple inheritance
• Subclasses recursively inherit all attributes and relationships of their superclasses up to the root
3.2 Polymorphism

- **Inheritance** may lead to two special problems
  - polymorphism
  - multiple inheritance

- **Polymorphism**
  - usually, subclasses inherit all attributes and relationships of their supertypes
  - subtypes may define additional attributes/relationships
  - what happens if an attribute in the subtype means something different?
  - what happens if an attribute is not needed at all?
  - what if some attribute should have a different name?
3.2 Polymorphism

- Example
  - sovereign territory just doesn’t make sense for a space station
    - should be removed
  - geo coordinates are also useless
    - but: Orbital trajectory somehow represents the same concept (location)
  - unfortunately, relational databases and ER don’t provide any useful support for polymorphism
    - avoid schemas where you need it!
    - if it is really necessary, constraints and null-values may be used to help out…
3.2 Multiple Inheritance

• Multiple inheritance
  – a subclass may have multiple superclasses
    • inheritance lattice instead of inheritance hierarchy
  – **but:** what happens if superclasses define the same attribute/relationship differently
    • which one should be inherited?
    • are both needed?
    • ER provides no support for conflicting multi-inheritance
      – avoid models with such conflicts
3.2 Union Types

• In a superclass-subclass relationship, the subclass inherits all attributes and relationships of the superclass(es)

• However, sometimes it is beneficial that a subclass inherits from only one superclass (chosen from a set of potential distinct superclasses)
  – every space station has an owner
  – a space station owner is either a space agency or a super villain
3.2 Union Types

• Solution: union types
  – Denoted by a $u$ in a circle
  – Space Agency and Super Villain are neither related, nor of the same type
  – an Owner is either a Space Agency or a Super Villain
Quick Exercise

• Another super hero database
  – We have people with a first name and last name
  – People can also be super heroes, which can have any number of aliases and any number of super powers
  – Super powers have a name, and can be of magical origin, of technological origin, or can be due to mutation
    • …and any combination of it

Charles Xavier, aka. “Professor X”, “Onslaught”
Quick Exercise
3 Extended Data Modeling

• Alternative ER Notations
• Extended ER
  – Inheritance
  – Complex Relationships
• Taxonomies & Ontologies
• UML
Science and philosophy always strived to explain the world and the nature of being

- first formal school of studies: Aristotle’s metaphysics (beyond the physical, around 360 BC)
- traditional branches of metaphysics
  - ontology
    - study of being and existence
  - natural theology
    - study of god, nature and creation
  - universal science
    - First Principles and logics
Ontology tries to describe everything which is (exists), and its relation and categorization with respect to other things in existence

- What is existence? Which things exists? Which are entities?
- Is existence a property?
- Which entities are fundamental?
- What is a physical object?
- How do the properties of an object relate to the object itself? What features are the essence?
- What does it means when a physical object exists?
- What constitutes the identity of an object?
- When does an object go out of existence, as opposed to merely change?
- Why does anything exist rather than nothing?
• Parts of metaphysics evolved into natural philosophy
  – study of **nature** and the **physical universe**
  – in the late 18\textsuperscript{th} century, it became just **science**
  – ontology is still a dominant concept in science
    • representation of all knowledge about things
• **Ars Generalis Ultima**
  
  – created in 1305 by Ramon Llull
  
  – *Ultimate* solution for the **Ars Magna (Great Art)**
    
    • mechanical combination of terms to create knowledge
    
    • base hope: all facts and truths can be created in such a way
  
  – heavy use of Arbor Scientiae (*Tree of Knowledge*)
    
    • tree structure showing an hierarchy of philosophical concepts
    
    • together with various *machines* (paper circles, charts, etc.) *reasoning* was possible
3.3 Taxonomies & Ontologies

- **Taxonomies** (τάξις : arrangement) are part of ontology
  - groups things with similar properties into **taxa**
  - taxa are put into an **hierarchical structure**
    - hierarchy represents supertype-subtype relationships
    - represents a **specialization** of taxa, starting with the most general one
  - taxonomies can be modeled with ER using specialization hierarchies
    - taxa are represented by entity types
3.3 Taxonomies

• **Example: Linnaean Taxonomy**
  
  – classification of all living things by Carl von Linné in 1738
  
  – classification into multiple hierarchy layers
    
    • domain, kingdom, phylum, subphylum, class, cohort, order, suborder, infraorder, superfamily, family, genus, species
  
  – each layer adds additional properties and restrictions
3.3 Taxonomies

- **Domain: Eukaryotes** – organisms having cell membranes

Sub-Domains

- Archaeplastida (Plantae)
- Opisthokonts
- Amoebozoa
- Chromalveolata
- Rhizaria
- Excavates

Kingdom

- Green plants (green algae, including prasinophytes, and land plants)
- Rhodophyta (red algae)
- Glaucophyta (eukaryotic algae with uniquely cyanobacteria-like chloroplasts; e.g., Cyanophora)
- Animals (Metazoa)
- Chytridiomycota (chimeric flagellates)
- Fungi (mushrooms, sac fungi, yeast, molds, rusts, smuts, etc.)
- Nucleomycota (filose amoebae; e.g., Nucleospora)
- Eumycota (e.g., Entamoeba)
- Protists (diatomophytes, ciliates, and apicomplexan parasites)
- Synurophyta (horny algae, diatoms, brown algae, chloromonads and relatives)
- Cryptophyta (haptophyceae-bearing microalgae, e.g., coccolithophorids)
- Cryptomonads (microalgae with a plastid-associated nucleomorph; e.g., Cryptomonas)
- Katharophyceae (heterotropic flagellates; e.g., Katablepharids)
- Prochlorophyta
- Cercozoa ( cercozoans; euglyphids, chlorarachniophytes and many other amoeboid flagellates)
- Foraminifera (complex cells with reticulopodia and a test/shell)
- Radiolaria (polycistrines and ancasharia)
- Malawimonads
- Euglenozoa (euglenids, dinoflagellates and kinetoplasts; e.g., Euglena and Trypanosoma)
- Heterokontophyta (amoeboid flagellates with discoidal mitochondrial cristae)
- Jakobida (free-living, heterotrophic flagellates)
- Parabasalida (trichomonads and hypomonad flagellates; e.g., Trichomonas and Trichonympha)
- Foraminifera (foraminifera and retortamonads; e.g., Giardia and Chilomastix)
- Prevarosyn (choanoflagellates + Trichostomatina)

**Domain**

- Archaea
- Euryarchaeota
- Crenarchaeota-Eocytes
- Eukaryotes

Animals Here

Detour
3.3 Taxonomies

• Example: Red Squirrel
  *(Binomial Name: Tamiasciurus hudsonicus)*
  - kingdom: Animals
  - phylum: Chordata (with *backbone*)
  - class: Mammalia (with backbone, *nursing its young*)
  - order: Rodentia (backbone, nursing its young, *sharp front teeth*)
  - suborder: Sciuromorpha (backbone, nursing its young, sharp front teeth, *like squirrel*)
  - family: Sciuridae (backbone, nursing its young, sharp front teeth, like squirrel, *bushy tail & lives on trees (i.e. real squirrel]*)
  - genus: Tamiasciurus (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, *from N-America*)
  - species: Hudsonicus (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, from N-America, *brown fur with white belly*)
3.3 Taxonomies

• Example: Edible Dormouse
  (*Binomial Name:* *Glis Glis*)
  – kingdom: **Animals**
  – phylum: **Chordata** (with **backbone**)
  – class: **Mammalia** (with backbone, **nursing its young**)
  – order: **Rodentia** (backbone, nursing its young, **sharp front teeth**)
  – suborder: **Scruromorpha** (backbone, nursing its young, sharp front teeth, **like squirrel**)
  – family: **Gliradae** (backbone, nursing its young, sharp front teeth, like squirrel, **sleeps long**)
  – genus: **Glis** (backbone, nursing its young, sharp front teeth, bushy tail, like squirrel, **eaten by Romans**)
  – species: **Glis** (backbone, nursing its young, sharp front teeth, bushy tail, climbs trees, **nothing more to classify**)

Relational Database Systems 1 – Christoph Lofi – Institut für Informationssysteme – TU Braunschweig
3.3 Taxonomies

- Rodentia (Rodents)
  - Myomorpha (Mouse-like)
  - Castorimorpha (Beaver-like)
  - Sciuromorpha (Squirrel-like)
    - Sciuridae (Squirrel)
      - Sciurini (Tree Squirrel)
        - Tamiasciurus (Pine Squirrel)
          - Hudsonicus (Red Squirrel)
        - Pteromyini (Flying Squirrel)
      - Sciurillusae (Pygmy Squirrel)
    - Aplodontiidae (Mountain Beaver)
    - Gliridae (Dormouse)
      - Glirinae (Real Dormouse)
      - Leithiinae (Other Dormice)
      - Glirulus (Japanese DM)
        - Glis (Edible Dormouse)
        - Glis (Yummy)
      - Graphiurinae (African Dormouse)
  - Hystricomorpha (Hedgehog-like)
  - Anomaluromorpha (Springhare-like)

- Hystricomorpha (Hedgehog-like)
  - Hystricomorpha (Hedgehog-like)

- Anomaluromorpha (Springhare-like)
  - Anomaluromorpha (Springhare-like)

- Rodentia (Rodents)
  - Rodentia (Rodents)
  - Rodentia (Rodents)
Recently, creating **ontological models** became fashionable in CS

– so called **ontologies**
– widely used in e.g. medical informatics, bio-informatics, Semantic Web

In addition to *normal* data models, ontologies offer **reasoning capabilities**

– allow to classify instances automatically
– allow to extract additional facts from the model

In CS, ontologies are usually modeled using **special languages**

– e.g. OWL, DAML+OIL, IDEF
3 Extended Data Modeling

• Alternative ER Notations
• Extended ER
  – Inheritance
  – Complex Relationships
• Taxonomies & Ontologies
• UML
3.4 UML

- **UML (Unified Modeling Language)** is a set of multiple modeling languages and diagram types
  - first standardized in 1997
  - unification of dominating object-oriented software design methods
    - James Rumbaugh: OMT
    - Grady Booch: Boochs Method
    - Ivar Jacobsen: OOSE
3.4 UML

- UML provides support for various software modeling problems
  - Static structural diagrams
    - Class diagram
    - Component diagram
    - Deployment diagram
    - Composite structure diagram
    - Object diagram
    - Package diagram
  - Dynamic behavior diagrams
    - Activity diagram
    - State diagram
    - Use-case diagram
  - Interaction diagrams
    - Communication diagram
    - Sequence diagram
    - Timing diagram
    - Interaction overview diagram
For data modeling, only **class diagrams** are used
- closely related to ER diagrams in crow’s foot notation
  - additional notations for logical design and operations

Entity type becomes **class**
- attributes written as in crow’s foot notation
  - usually, also domains are modeled
  - no composite or multivalued attributes
  - derived attributes are modeled as operations
  - key attributes are marked with a *
- operations are only needed for derived attributes in pure data models
- entity type instances are called **objects**
• In UML, relationship types are called *associations*

• Simplest case: just a plain **line**
  – although using just a line is valid, a good model should provide additional information
  • name
  • direction
  • multiplicity
  • order
  • navigability
  • special aggregation types

**Diagram:**

- **Super Hero**
- **Sidekick**
• Example

A *super hero may mentor multiple sidekicks.*

– **careful:** multiplicity in opposite direction to Chen ER

![Diagram of UML relationship showing roles, multiplicity, and association name.](attachment:image.png)
• UML does not allow to add attributes to associations directly
• Workaround: association classes
  – association classes belong to an association (indicated by dashed line)
  – they share the association name
  – each instance of the association creates an according class object
• Association classes cannot directly be replaced by a normal class

– introduces additional semantics
– the replacement model allows that a hero is assigned **twice** to the same super team!
• For *n*-ary associations (*n* > 2), the diamond returns
• **Aggregation**
  – the aggregation is a special association within UML
  – colloquial: *is_part_of* or *consist_of*
  – denoted by a small, empty diamond
  – aggregation just states that one class is part of another; it poses no further restrictions
    • objects may still exist independently of each other
    • objects may be part of several other objects
  – Example
    • A *plan to take over the world consists of several things that need to be done.*

![Diagram showing the relationship between Plan to take over the world and Stuff to do.](image)
• **Composition** (also called strong aggregation)
  
  – **stricter** version of aggregation
    
    • diagrammed by solid diamond
  
  – based on multiplicity of the part-side
    
    • **1**: an object is **always part** of just **one** other object. If the *main* object is deleted, the part needs to be assigned to another *master* or is deleted.
    
    • **0..1**: an object may be part of **at most one** other object. It may also exist alone.
    
    • ****: not allowed. Part of one object max.
  
  – **Example**
    
    • *A doomsday machine is made of multiple parts.*
• **Qualified associations**

  – associations may be qualified by an additional attribute
    
    • each association instance between objects is **classified** by this attribute

  – **Example**
    
    • *Von Doom Industries employs Victor von Doom as CEO.*
    • *Von Doom Industries employs all members of the Terrible Trio as henchmen.*
• **Weak entities** through qualified associations
  
a weak entity’s partial key is modeled by the classifying attribute of a qualified association

  – **Example**
  
  • *A lecture hall has many seats. A seat is identified by a number and the room number of its lecture hall.*
• Generalization
  – induces a class-subclass relationship (*is_a*)
    • diagrammed with an hollow arrow
  – by default, generalization is **disjoint**
    • **overlapping** is additionally annotated in curly brackets
  – by default, generalization is **partial** (*incomplete* in UML)
    • **total** (*complete*) is also annotated in curly brackets
• Classification attributes
  
  – similar to EER’s *attribute-defined* relationship types
  
  – denoted by `:attribute_name`
  
  – all objects of a given subtype have the **same value** for the classifier attribute
• Association **navigability**
  – denoted by an arrowhead and small cross
  – models how you can navigate among objects involved in the association
  – one-way association
  – Example
    • for each hero, you can navigate to the substances which may kill him
    • you cannot natively navigate from a substance to a hero
      – This may modify how the actual data structures implementing the model may look like
• **XOR restrictions on associations**
  
  – a class having multiple associations can be modeled in such a way that **only one of these associations can be active at a time**

  – **Example**

  • A villain lives either in a secret lair, or in a prison (but not in both).
3 Next Week

- View integration
- Resolving conceptual incompatibility
- Entity clustering for ER models
- Commercial dimension: The BEA story