Relational Database Systems I

Christoph Lofi
Simon Barthel

Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de
Summary last week

• Databases
  – are **logical interfaces**
  – support **declarative querying**
  – are **well-structured**
  – aim at **efficient manipulation of data**
  – support **control redundancy**
  – support **multiple views** of the data
  – support **atomic multi-user transactions**
  – support **persistence and recovery** of data
• Phases of DB Design
• Data Models
• Basic ER Modeling
  – Chen Notation
  – Mathematical Model
• Example
2.1 Database Applications

- **Database applications** consist of
  - database instances with their respective **DBMS**
  - associated **application programs** interfacing with the users
2.1 Database Applications

• Planning and developing application programs traditionally is a **software engineering** problem
  – Requirements Engineering
  – Conceptual Design
  – Application Design
  – …

• Software engineers and **data engineers** cooperate tightly in planning the need, use and flow of data
  – **Data Modeling**
  – **Database Design**
2.1 Universe of Discourse

- DB Design models a **miniworld** (also called universe of discourse) into a formal representation
  - restricted view on the real world with respect to the problems that the current application should solve
2.1 Phases of DB Design

Miniworld

- Requirements Analysis
  - Functional Requirements
  - Data Requirements
  - Conceptual Schema
  - Logical Schema

- Conceptual Design
  - High Level Transaction Specification
  - Logical Design
    - Application Program Design
      - Transaction Implementation
      - Internal Schema
    - Physical Design
      - Application Programs
      - DBMS independent
      - DBMS dependent

this lecture
2.1 Phases of DB Design

• **Requirements Analysis**
  – database designers interview prospective **users** and **stakeholders**
  – **Data Requirements** describe what kind of data is needed
  – **Functional Requirements** describe the operations performed on the data

• **Functional Analysis**
  – concentrates on describing **high-level** user operations and transactions
    • does not yet contain implementation details
2.1 Phases of DB Design

- **Conceptual Design**
  - transforms Data Requirements to *conceptual model*
  - describes high-level data entities, relationships, constraints, etc.
    - does not contain any implementation details
    - independent of used software and hardware

- **Logical Design**
  - maps the conceptual data model to the logical data model used by the DBMS
    - e.g. relational model, hierarchical model
    - technology independent conceptual model is adapted to the used DBMS software

- **Physical Design**
  - creates internal structures needed to efficiently store/manage data
    - e.g. table spaces, indexes, access paths
    - depends on used hardware and DBMS software
Modeling the data involves three design phases:

- Result of one phase is input of the next phase.
- Often, automatic transition is possible with some additional designer feedback.
2 Data Modeling

• Phases of DB Design
• Data Models
• Basic ER Modeling
  – Chen Notation
  – Mathematical Model
• Example
2.2 Data Semantics

• In databases, the data’s specific **semantics** are very important
  – what is described?
  – what values are reasonable/correct?
  – what data belongs together?
  – what data is often/rarely accessed?
2.2 Data Semantics

• Example: Describe the age of a person
  – semantic definition:
    *The number of years elapsed since a person’s birthday.*
  – integer data type
  – always: $0 \leq \text{age} \leq 150$
  – connected to the person’s name, passport id, etc.
  – may often be retrieved, but should be protected
  – …
2.2 Data Models

• A data model is an abstract model that describes how data is represented, accessed, and reasoned about

  – e.g. network model, relational model, object-oriented model

  – warning: The term data model is ambiguous

    • a data model theory is a formal description of how data may be structured and accessed, and is independent of a specific software or hardware

    • a data model instance or schema applies a data model theory to create an instance for some particular application (e.g., data models in MySQL Workbench designer refer to a logical model adapted to the MySQL database)
2.2 Data Models

• A data model consists of three parts

  – Structure
    • **data structures** are used to create databases representing the modeled objects

  – Integrity
    • rules expressing the **constraints** placed on these data structures to ensure structural integrity

  – Manipulation
    • operators that can be applied to the data structures, to **update** and **query** the data contained in the database
2.2 Data Models

• Different categories of data models exist
  – **conceptual** data models (**high-level**)  
    • represent structure in a way that is close to the users’ perception of data
      – e.g., the relational model, network models, etc.
  – **representational** or **logical** data models
    • represent structure in a way that is still perceivable for users but that is also close to the physical organization of data on the computer
  – **physical** data models (**low-level**)  
    • represent structure that describe the details of how data is stored from the computer
Concrete instances of data models are called schemas

- A **conceptual schema** describes the data semantics of a certain domain
  - what facts or propositions hold in this domain?
- A **logical schema** describes the data semantics, as needed by a particular data manipulation technology
  - e.g. tables and columns, object-oriented classes, XML elements
- A **physical schema** describes the physical means by which the data is stored
  - e.g. partitions, tablespaces, indexes
2.2 Three-layer Architecture

- Also called ANSI-SPARC Architecture

Presentation Layer
- External View
- External/Logical Mapping

Logical Layer
- Logical Schema
- Logical/Internal Mapping

Physical Layer
- Physical Schema
- Stored Database

End Users
- External View

Conceptual Schema
- Defines

DB Designer
2.2 Three-layer Architecture

• Also called ANSI-SPARC Architecture
  – Careful: A lot of ambiguous naming is going on!
  – the **logical layer** is often referred to as the **conceptual layer**
    • usually **logical** or **representational** data model
      – e.g., lower level ER schemas
    • but often based on a **conceptual schema design** in a high-level data model
      – e.g., high level Extended ER schemas
  – **external views**
    • typically implemented using a **logical** data model
    • but often based on a **conceptual schema design** in a high-level data model
2.2 Three-layer Architecture

Why do we need layers?

- they provide independence
- physical independence
  - storage design can be altered without affecting logical or conceptual schemas
  - e.g. regardless on which hard drive a person’s age is stored, it remains the same data
- logical independence
  - logical design can be altered without affecting the data semantics
  - e.g. it does not matter whether a person’s age is directly stored or computed from the person’s birth date
• **Problems**: Persons designing a data model for the **same domain** will often come up with very different schemas
  
  – each schema can be a correct representation of the domain
  
  – but merging and mapping them is difficult due to their differences
  
  – **exchanging** and **integrating data** between organizations with incompatible schemas is tough
2.2 Data Models

– often **different levels of abstraction** are used
  • the semantic expressiveness of schemas is different
  • e.g. one schema may contain *Cows* and *Dolphins* while another only contains the higher-level concept *Animals*

– **extending** a schema is often necessary
  • e.g. when the focus changes or new information about the domain becomes available
  • schemas limit what can be expressed about a domain
  • adjustments may result in a complete re-modeling of a schema
2.2 Generic Data Models

• **Generic data models** are generalizations of conventional data models
  – definition of standardized general relation types, together with the kinds of things that may be related by such a relation type
  – similar to the definition of a natural language
2.2 Generic Data Models

• Example: A generic data model may define relation types such as

  – **classification relation** – as a binary relation between an individual thing and a kind of thing (i.e. a class)
    • e.g. Dolphin *is_a* Animal, Cat *is_a* Animal
      *is_a*: (Dolphin, Animal), (Cat, Animal), (Snowball, Cat)

  – **part-whole relation** – as a binary relation between two things: one with the part role and the other with the whole role
    • e.g. Wheel *is_part_of* Car, Branch *is_part_of* Tree
      *is_part_of*: (Wheel, Car), (Branch, Tree)
• **Current state of the art:**
  Usually data is structured best using *(relational) tables*!
  – modeling data in tables is very natural and efficient

• **Think:** *Index card*
  – all data about a certain object on a single card
  – ordered/sorted by a single attribute
• Sounds pretty obvious, huh?
  – we owe this belief to **Edgar F. Codd** (around 1970)
  – before that, people had a very **different perspective** on what data actually is
2 Data Modeling I

- Phases of DB Design
- Data Models
- **Basic ER Modeling**
  - Chen Notation
  - Mathematical Model
- Example
2.3 ER Modeling

• Traditional approach to **Conceptual Modeling**
  – **Entity-Relationship Models** (ER-Models)
    • also known as **Entity-Relationship Diagrams** (ERD)
    • introduced in 1976 by Peter Chen
    • graphical representation

• Top-Down-Approach for modeling
  – entities and attributes
  – relationships
  – constraints

• Some derivates became popular
  – **ER Crow’s Foot Notation** (Bachman Notation)
  – **ER Baker Notation**
  – later: **Unified Modeling Language** (UML)
• Entities

– an entity represents a *thing* in the real world with an independent existence
  • an entity has an own identity and represents just one thing

– e.g. *a car*, *a savings account*, *my neighbor’s house*, *the cat Snowflake*, *a product*
• Attributes

– a **property** of an entity, entity type or a relationship type

– e.g. *name* of an employee, *color* of a car, *balance* of an account, *location* of a house

– attributes can be classified as being:
  
  • **simple** or **composite**
  
  • **single-valued** or **multi-valued**
  
  • **stored** or **derived**

  • e.g. name of a cat is simple, single-valued, and stored
2.3 ER – Entity Types

• Entity types
  – sets of entities sharing the same characteristics or attributes
    • each entity within the set has its own attribute values
  – each entity type is described by its name and attributes
    • each entity is an instance of an entity type
  – describes the so called schema or intension of a set of similar entities
• **Entity Set** *(of a given entity type)*
  
  – collection of all stored entities of a given entity type
  
  – entity sets often have the same name as the entity type
    
    • *Cat* may refer to the entity type as well as to the set of all *Cat* entities (sometimes also plural for the set: *Cats*)
  
  – also called the *extension* of an entity type
    
    (or *instance*)
2.3 ER Diagrams

- ER diagrams represent **entity types** and **relationships** among them, not single entities.

- Graphical Representation
  - **entity type**
    - Rectangle labeled with the name of the entity
    - Usually, name starts with capital letters
  
  - **attributes**
    - Oval labeled with the name of the attribute
    - Usually, name starts with lower case letters
2.3 ER Diagrams

- Textual Representation
  - entity types
    - written: entity_type_name(attribute_1, …, attribute_n)
  - entity
    - written: (value of attribute_1, …, value of attribute_n)

- Example
  - **Entity Type** Cat
    - Cat(name, color)
  - **Entity Set** Cats
    - (Fluffy, black-white)
    - (Snowflake, white)
    - (Captain Hook, red)
    - (Garfield, orange)
2.3 ER – Composite Attributes

- **Simple Attribute:**
  - attribute composed of a single component with an independent existence
  - e.g. name of a cat, salary of an employee
    - Cat(name), Employee(salary)

- **Composite Attribute:**
  - Attribute composed of multiple components, each with an independent existence
    - graphically represented by connecting sub-attributes to main attribute
    - textually represented by grouping sub-attributes in ()
  - e.g. address attribute of a company (is composed of street, house number, ZIP, and city)
    - Company(address(street, house_no, ZIP, city))
2.3 ER Multi-Valued Attributes

- **Single-Valued Attribute**
  - attribute holding a *single* value for each occurrence of an entity type
  - e.g. *name* of a cat, *registration number* of a student

- **Multi-Valued Attributes (lists)**
  - attribute holding (possibly) *multiple* values for each occurrence of an entity type.
    - graphically indicated by a double-bordered oval
    - textually represented by enclosing in {}
  - e.g. *telephone number* of a student
    - Student({*telephone_no*})
2.3 ER – Derived Attributes

- **Stored Attribute**
  - the attribute is directly stored in the database

- **Derived Attribute**
  - the attribute is (usually) not stored in the DB but derived from an other, stored attribute
    - On a logical schema, it’s a design decision if an attribute should really be derived or stored (redundantly)
    - Redundant storage might lead to better performance, but requires dealing with consistency of updates
  - indicated by dashed oval
  - e.g. *age* can be derived from *birth date*, *average grade* can be derived by aggregating all stored *grades*
2.3 ER – Keys

• Entities are **only** described by attribute values
  – two entities with identical values cannot be distinguished
    • Later, we might introduce OIDs, row IDs, etc. to fix this problem in a logical schema

• Entities (usually) must be distinguishable

• Identification of entities with **key attributes**
  – value combination of key attributes is **unique** within all possible extensions of the entity types
  – key attributes are indicated by underlining the attribute name
2.3 ER – Keys

• Key attribute examples
  – single key attribute
    • Student(registration_number, name)
    • (432451, Hans Müller)
  – composite key (multiple key attributes)
    • Car(brand, license_plate(district_id, letter_id, numeric_id), year)
    • (Mercedes,(BS,CL,797),1998)
    • please note that each key attribute itself does not need to be unique!
2.3 ER Modeling

- **Sample Entity Type**
  - Book(isbn, {author(firstName, lastName)}, title, publisher(name, city, country), {revision(no, year)})
  - (0321204484, {(Ramez, Elmasri), (Shamkant, Navathe)}, Fundamentals of Database Systems, (Pearson, Boston, US), {(4,2004), (2, 1994)})
Attributes cannot have arbitrary values: they are restricted by the attribute value sets (domains)

- *zip codes* may be restricted to integer values between 0 and 99999
- *names* may be restricted to character strings with maximum length of 120
- domains are not displayed in ER diagrams
- usually, popular **data types** are used to describe domains in data modeling
  - e.g. integer, float, string
2.3 ER – Domains

- Commonly used data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer</td>
<td>32/64-Bit signed integer values between $-2^{31/64}$ and $2^{31/64}$</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>64-Bit floating point values of approximate precision</td>
</tr>
<tr>
<td>numeric</td>
<td>numeric($p$, $s$)</td>
<td>A number with $p$ digit before the decimal and $s$ digits after the decimal (exact precision)</td>
</tr>
<tr>
<td>character</td>
<td>char($x$)</td>
<td>A textual string of the exact length $x$</td>
</tr>
<tr>
<td>varying character</td>
<td>varchar($x$)</td>
<td>A textual string of the maximum length $x$</td>
</tr>
<tr>
<td>date</td>
<td>date</td>
<td>Stores year, month, and day</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
<td>Stores hour, minute, and second values</td>
</tr>
</tbody>
</table>
• Using data types for modeling domains is actually a crutch
  – Some modern programming language are better in this way!
  – the original intention of domains was modeling all valid values for an attribute
    • color: \{Red, Blue, Green, Yellow\}
  – using data types is very coarse and more a convenient solution
    • color: varchar(6) ???
  – to compensate for the lacking precision, often restrictions are used
    • color: varchar(6) restricted to \{Red, Blue, Green, Yellow\}
2.3 ER – NULL Values

• Sometimes, an attribute value is **not known** or an attribute does **not apply** for an entity
  – this is denoted by the special value **NULL**
    • so called **NULL-value**
  – e.g. attribute *university_degree* of Entity *Heinz Müller* may be NULL, if he does not have a degree
  – NULL is usually always allowed for any domain or data type unless explicitly excluded
• What does it mean when you encounter a NULL-value?
  – attribute is not applicable
    • e.g. attribute maiden name when you don’t have one
  – value is not known
  – value will be filled in later
  – value is not important for the current entity
  – value was just forgotten to set

• Actually there are more than 30 possible interpretations…
2.3 ER – Relationships

• Entities are not enough to model a miniworld
  – the power to model dependencies and relationships is needed
• In ER, there can be relationships between entities
  – each relationship instance has a degree
    • i.e. the number of entities it relates to
  – a relationship instance may have attributes
2.3 ER – Relationships

• Similar to entities, ERDs do not model individual relationships, but **relationship types**

• **Relationship type**
  – named set of all similar relationships with the same attributes and relating to the same entity types

  ![Diamond labeled with the name of the relationship type](name)
  • Diamond labeled with the name of the relationship type
  • Usually, name starts with lower-case letters

• **Relationship set**
  – set of all **relationship instances** of a certain relationship type
2.3 ER – Relationships

- **Relationships** relate **entities** within the **entity sets** involved in the **relationship type** to each other.
• Example:
  – there is an ownership relation between cats and persons

– but more modeling detail is needed
  • does every person own a cat? Does every cat have an owner?
  • can a cat have multiple owners or a person own multiple cats?
  • since when does a person own some cat?
  • who owns whom?
Additionally, restrictions on the combinations of entities participating in an entity set are needed – e.g. relationship type *married to*

- unless living in Utah, a restriction should be modeled that each person can only be married to a single person at a time – i.e. each person entity may only appear once in the “married to” relationship set
- **cardinality** annotations are used for this
- relationship types referring to just one entity type are called recursive
2.3 ER – Relationship Cardinality

- **Cardinality annotations**
  - one cardinality annotation per entity type / relationship end
    - minimum and maximum constrains possible
  - Common Cardinality Expressions
    - \((1, 1)\): each entity is bound exactly once
    - \((0, *)\): each entity may participate arbitrary often in the relationship
    - \((2, *)\): each entity may participate arbitrary often in the relationship, but at least twice
  - Convention you might see outside this lecture
    - no annotation is usually interpreted as \((0, *)\)
    - if only one symbol / number \(s\) is used, this is interpreted as \((0, s)\)
      - \(* = (0, *)\); \(4 = (0, 4)\)
    - sometimes, \(N\) or \(M\) are used instead of \(*\)
2.3 ER – Relationship Cardinality

• Cardinalities express how often a specific entity may appear within a relationship set
  – Please note: There are other notations which look similar but use different semantics (e.g., UML)

  – a specific entity of type A may appear up to once in the relationship set, an entity of type B appears at least once and at most twice
    • this means: Up to two entities of type A may relate to one entity of type B. Some entities in A are not related to any in B. All entities in B are related to at least one in A.
• To each entity of type B, one or two entities of type A are related
Example

- Each person can only be married to one other person.

- Each entity can only appear in one instance of the married to entity set
  - Still, could be married to oneself

ER - Relationship Cardinality

married to

(0,1)

Person

(0,1)
2.3 ER – Relationships

(0,1) Person

married to

(0,1)

Person

married to

Person

P1

P2

P3

P4

P5

P6

R1

R2
2.3 ER – Relationship Cardinality

• Example
  – A cat has up to 4 owners, but at least one. A person may own any number of cats.

  ![Relationship Diagram]

  - Lisa owns Snowball
  - Lisa owns Snowball II
• Example

  – A person may supervise any other number of persons.
    • Drake Mallard supervises Launchpad McQuack.
    • Drake Mallard supervises Gosaly Mallard.
2.3 ER – Relationship Cardinality

• Cardinalities for **binary** relationship types can be classified into common, more general **cardinality types**
  – these cardinality types are also often found in other modeling paradigms
    • **One-To-One (1:1)** – each entity of the first type can only relate to exactly one entity of the other type
    • **One-To-Many (1:N)** – each entity of the first type can relate to multiple entities of the other type
    • **Many-To-One (N:1)** – multiple entities of the first type can relate to exactly one entity of the second type
    • **Many-To-Many (N:M)** – any number of entities of first type may relate to any number of entities of second type (no restrictions)
  – As we will see later, these will have a direct impact on the logical database schema
Often, it is beneficial to clarify the role of an entity within a relationship – e.g. relationship *supervises*

- what is meant? Who is the supervisor? Who is the supervised person?
- roles can be annotated on the relationship lines
• Relationship instances involve multiple entities
– the number of entities in each relationship instance is called **relationship degree**

• degree = 2 – Binary Relation

- Person owns Cat

• degree = 3 – Ternary Relation

- Supplier supplies Customer Part
Similar to entities, relationship types may even have attributes.

Later, when designing the logical schema:

- for 1:1 relationships, the relationship attribute may be migrated to any of the participating attributes.
- for 1:N relationships, the attribute may be only migrated to the entity type on the N-side.
- for N:M relationships, no migration is possible.
To express that all entities of an entity type appear in a certain relationship set, the concept of **total participation** can be used

– the entity type which is totally participating is indicated by a double line

– e.g. *Each driver’s license must belong to an exact person.*
• Each entity needs to be identifiable by a set of key attributes

• Entities that exist independently of the context are called strong entities
  — a person exists whether it is married or not

• In contrast, there may be entities without a unique key called weak entities
2.3 ER – Weak Entities

• **Weak entities** are identified by being related to **Strong Entities**
  
  – the strong entities *own and define* the weak entities
    • the weak one cannot exist without the strong ones
  
  – the relationships relating the strong to the weak are called **identifying relationships**
    • weak entities are **totally participating** in that relationship
  
  – weak entities have **partial keys** which are unique within the identifying relationship sets of their strong entities
    • to be unique, the weak entity instance has to borrow the key values of the respective strong entity instances
2.3 ER – Weak Entities

– weak entity types and identifying relationship types are depicted by double-lined rectangles

– Example

  * An online shopping order contains several order items.

  • an order item can only exist within an order
  • each order item can be identified by the order no of its owning order and its item line
2.3 ER – Overview

- 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
Quick Exercise

- We want to build a database for super heroes
  - In a our database, we have heroes
  - Each hero has a real name, which consists of a first name and a last name. Also, each hero has an unique alias.
  - There are super hero teams with unique names. Each hero can belong to any number of teams.
  - For each hero which joins or leaves a team, the join and leave date needs to be stored.

James Howlett, aka. “Wolverine”
Teams: X-Men, Avangers
Quick Exercise

Diagram:

- **Hero**
  - First name
  - Last name
  - Name
  - Alias
- **Team**
  - Name
  - Join date
  - Leave date

Relationship:
- **Member of (0,*)**
  - **Hero** to **Team**

Note: The image includes a logo and some text related to knowledge-based systems and deductive databases.
2 Data Modeling 1

- Phases of DB Design
- Data Models
- Basic ER Modeling
  - Chen Notation
  - Mathematical Model
- Example
We want to model a simple university database

- In our database, we have students. They have a name, a registration number, and a course of study.
- The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have a unique ID.
- Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may register for that lecture.
- Professors have a name and are member of a specific department.
• How to start? What to do?
  – find the basic **entity types**
  – find the **attributes** of entities
    • decide to which entity an attribute should be assigned
    • which attributes are key attributes?
    • some attributes are better modeled as own entities, which ones?
  – define the **relationship types**
    • which role do entities play?
    • do relationships require additional entity types?
    • are the relationships total? Identifying? Are weak entities involved?
    • what are the cardinalities of the relationship type?
2.4 Example

• Which are our entity types?
  – In our database, we have students. They have a name, a registration number and a course of study.
  – The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have a unique ID.
  – Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may register for that lecture.
  – Professors have a name and are member of a specific department.
2.4 Example

• What attributes are there?
  – In our database, we have students. They have a name, a registration number and a course of study.
  – The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have unique ID.
  – Professors have a name and are member of a specific department.
• First try…
  – this model is **really crappy**!
  – *course of study* does not seem to be an attribute
    • used by student and lecture. Even worse, lecture refers to a course of study in a specific curriculum semester.
    • use additional entity type with relationships!
  – *prerequisite lecture* also is not a good attribute
    • prerequisite lectures are also lectures. Use a relationship instead!
  – *professor* does not have key attributes
2.4 Example

- **Second try**
  - professor uses a *surrogate* key now
    - key is automatically generated and has no meaning beside unique identification (*but must be present!*)
  - course of study is an entity type now

- **Which entity types are additionally related?**
  - Each year, some lectures of the pool of all lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may attend that lecture.
2.4 Example

Better?

- add cardinalities
- add total and identifying annotations
- *termwise lecture* has no key
2.4 Example

Student

- registration number
- name
- enrolls

Professor

- id
- name
- department

Lecture

- id
- title
- credits
- part of
- prerequisite
- curriculum
- semester

Lecture instance

- time
- day of week
- room
- semester

Course of Study

- name

registration number (0, *)
name (1, 1)
enrolls (0, *)
Professor (0, *)
Lecture (0, *)
Lecture instance (1, 1)
Student (1, 1)

Attends

Teaches

Instantiates

Part of

Prerequisite

Curriculum semester

Enrolls

Day of week

Room

Semester
2.4 Example

• In general, modeling is not that simple
• Many possible ways of modeling the same miniworld
  – some are more elegant, some are less elegant, but all may be valid!
• Models alone are not enough, they need to be documented
  – what do the attributes mean?
  – what do the relationships mean?
2 Summary

• Data models
  – 3 parts (structural, integrity, manipulation)
  – 3 categories (conceptual, logical, physical)
  – Schemas are instances of Data Models

• Database Applications
  – ANSI-SPARC architecture
    • 3 Layers (presentation, logical, physical)
      – Careful: these layers are defined slightly different than the 3 categories
    • Data Independence

• ER Modeling
  – Chen notation
Next week

- Alternative ER Notations
- Extended ER
  - Inheritance
  - Complex Relationships
- Taxonomies & Ontologies
- UML