

Database Design

Instructor: Peter Baumann

email: pbaumann@constructor.university

office: room 88, Research 1

Core Database Design Steps

- Conceptual design ← Our focus in this Chapter
 - Construct a description of the information used in an enterprise
 - *Focus on documenting customer intention, disregard technology*
- Logical design
 - Construct a description based on a specific data model (e.g., relational)
 - *Focus on abstract tech, disregard implementation*
- Physical design
 - Describe implementation using a particular DBMS, file structures, indexes, security, ...

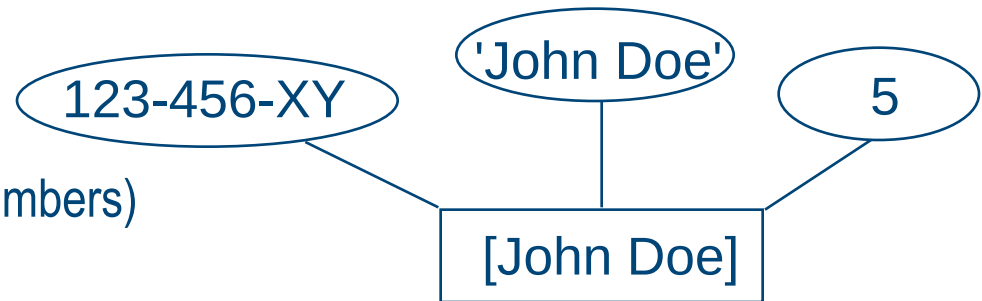
Issues in Conceptual Design

- Conceptual design: (we use ER Model at this stage)
 - What are the **entities** and **relationships** in the enterprise?
 - What **information about** these entities and relationships should we store in the database?
 - What are the **integrity constraints** or **business rules** that hold?
- database `schema' in the ER Model represented pictorially = **ER diagrams**
 - Can map an ER diagram into a relational schema
 - Actually lack of textual equivalent is shortcoming
 - ... also: no formal semantics (originally)

Entity-Relationship Model: Basics

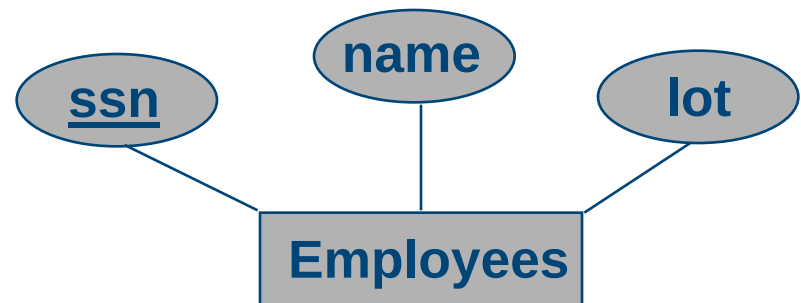
- **Entity**: Real-world object distinguishable from other objects

- entity described (in DB) using a set of attributes
- Simple **attribute values** (strings, numbers)



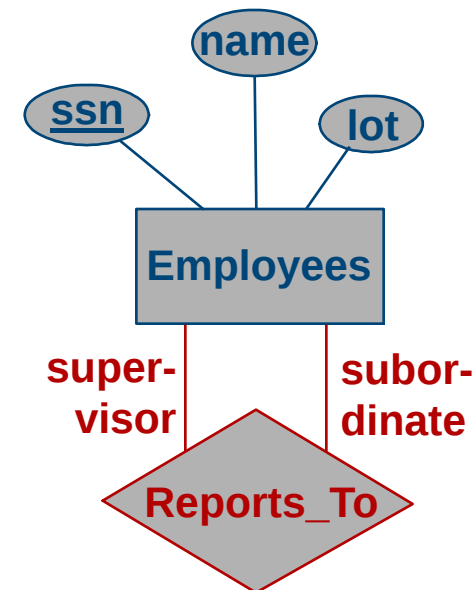
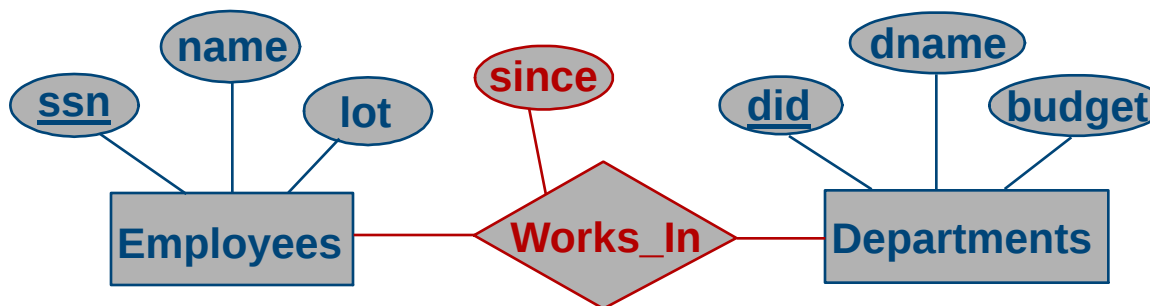
- **Entity set**: collection of similar entities

- E.g., all employees
- All entities in an entity set have the same set of attributes
 - *Until we consider ISA hierarchies, anyway!*
- Each entity set has a **key**
- Each attribute has a **domain** = data type



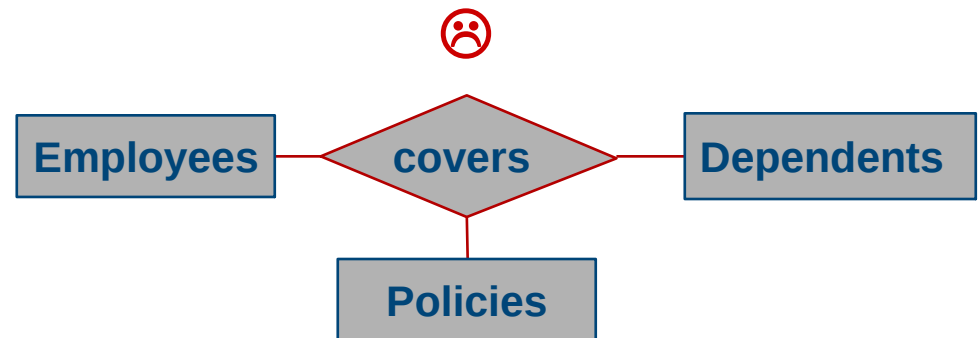
ER Model Basics (Contd.)

- **Relationship:** (unique!) association among two or more entities
 - E.g., Attishoo **works_in** Pharmacy department
- **Relationship Set:** Collection of similar relationships
 - An **n-ary** (binary, ternary, ...) relationship set R relates n entity sets $E_1 \dots E_n$
 - each relationship in R involves entities $e_1 \in E_1, \dots, e_n \in E_n$
 - Same entity set can participate in different relationship sets, or even in the same set (but then in different **roles**)



Binary vs. Ternary Relationships

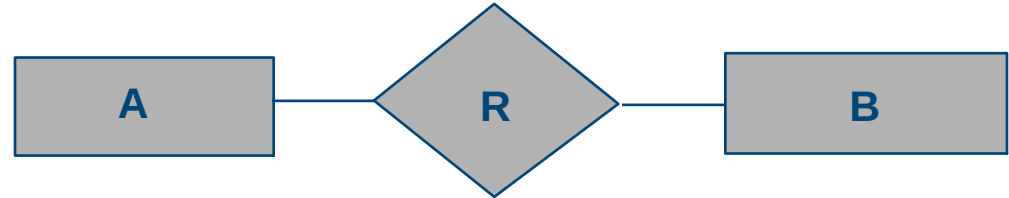
- Naming:
 - Binary relationship has 2 legs
 - Ternary relationship has 3 legs
 - N-ary relationship has N legs
- Rule of thumb:
use only binary, nothing else
 - Can model n-ary through binary + additional constraints



Constraints

- Used to capture more application semantics
- ...on relationship sets:
 - Key constraints (multiplicities)
- ...on entity sets:
 - Participation constraints

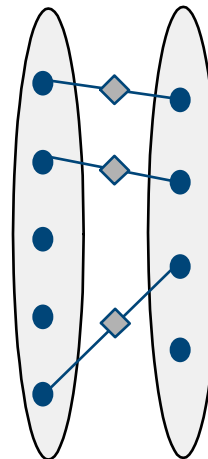
Key Constraints: Multiplicity (contd)



■ Multiplicity classification:

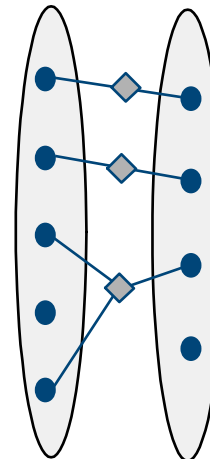
■ One-to-one

"1:1"



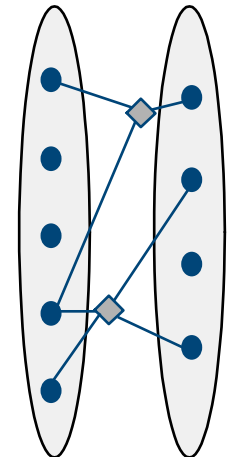
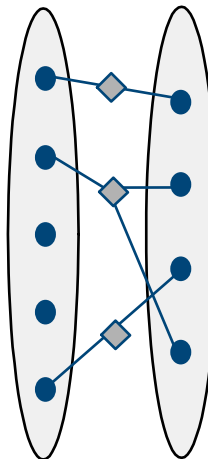
■ One-to-many

"1:n"



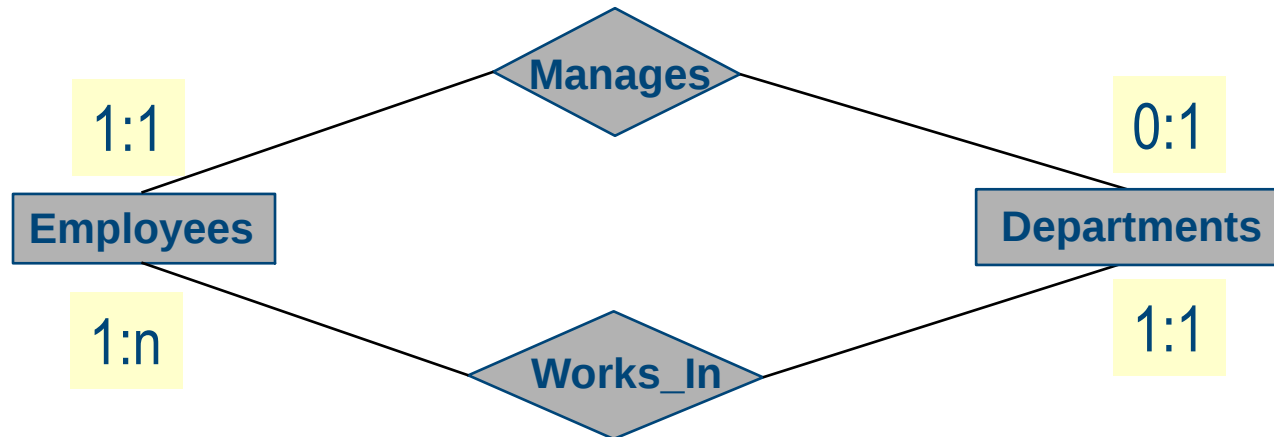
■ Many-to-many

"m:n"



More Detail Wanted!

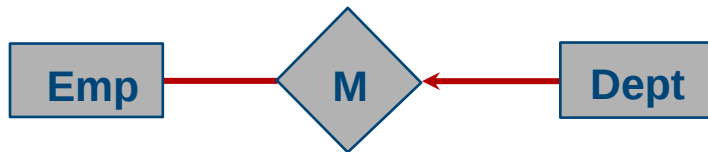
- Want to refine further: *how many connections on each leg of relship?*
- Attach intervals to leg:



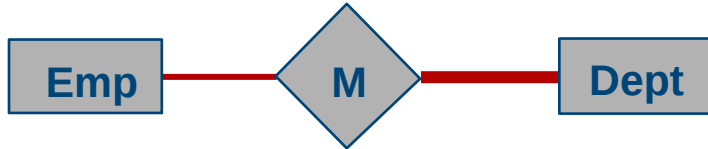
- Read as:
 - „an Employee sees, through its Manages tunnel, none or one Department“
 - „a Department sees, through its Works_In tunnel, at least one Employee“

Notation Variants: Multiplicity

forget this slide!

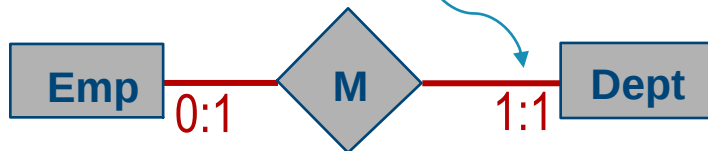


x:1 a la Ramakrishnan/Gehrke



1:x a la Ramakrishnan/Gehrke

My personal preference – allows for more details

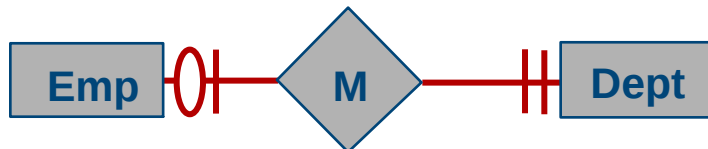


0:1

1:1

0:n

1:n



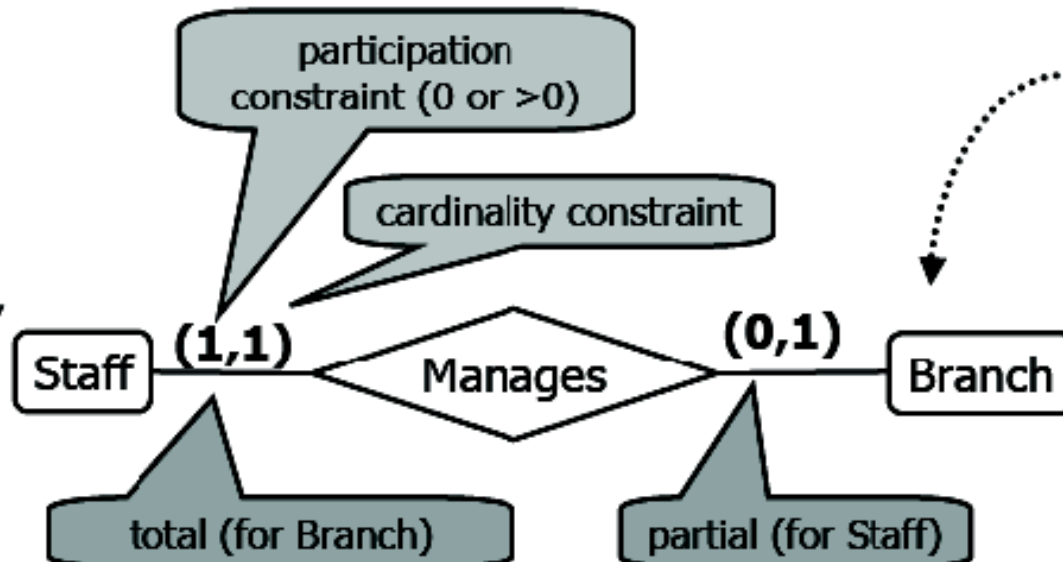
...plus many more

Citing a Similar Discussion by Bernhard Reus (U of Sussex)

forget this slide!

Multiplicity (Example)

*Each branch is managed by (exactly) one member of staff.
A member of staff can manage (at least) zero or (at most) one branch.*

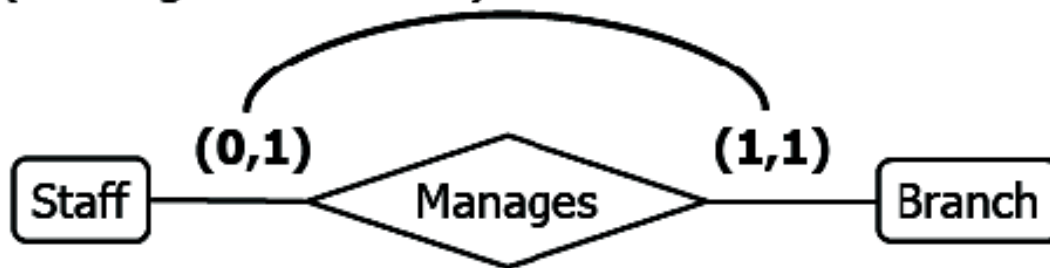


Citing a Similar Discussion by Bernhard Reus (U of Sussex)

forget this slide!

Which side are you on?

- **Attention:** In the literature the multiplicity constraints are sometimes swapped like that: (don't get confused)



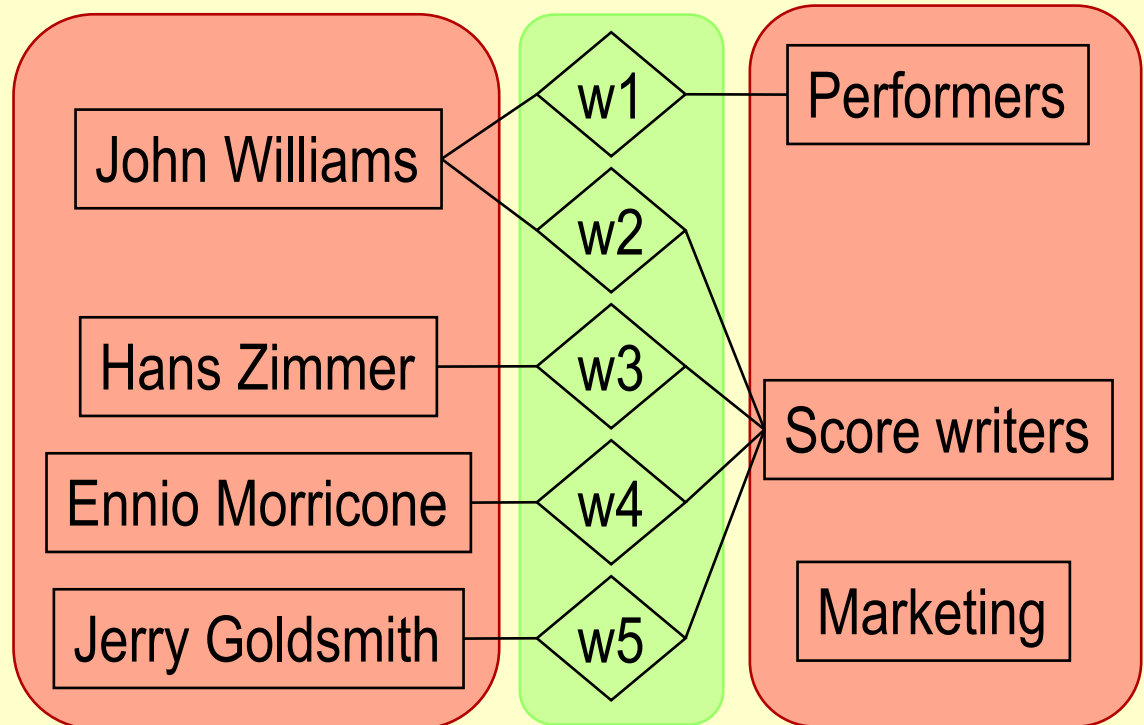
We stick to convention on previous slide (UML).

Relationships Example

- Schema:



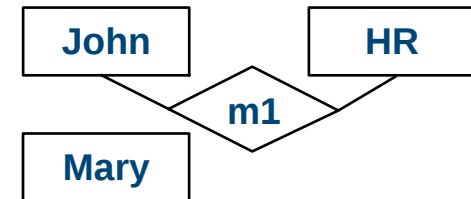
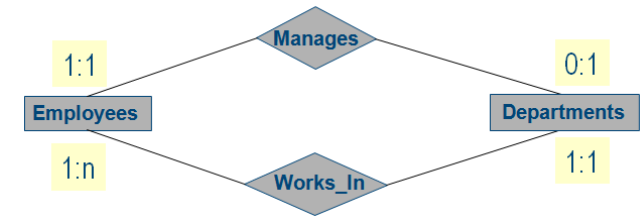
- Instances:



- Uniqueness?
- Multiplicities?
- Participation?

Participation Constraints

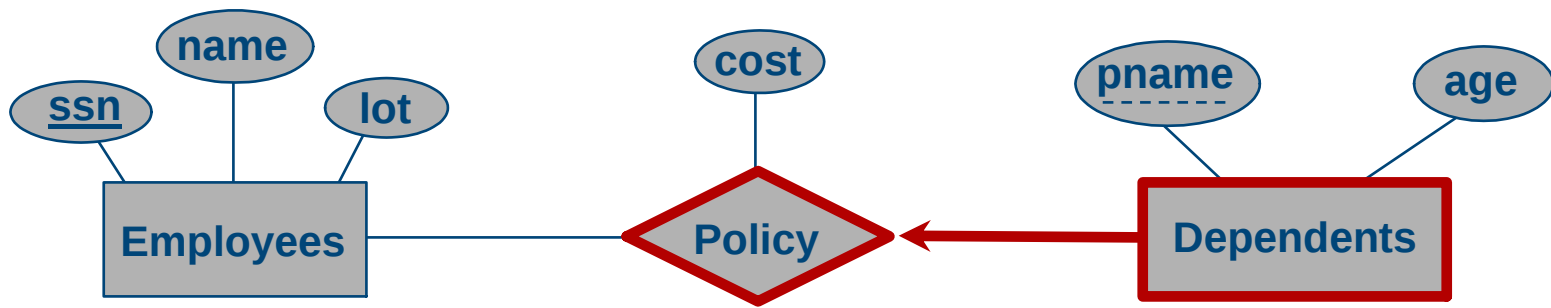
- Does *every* department have a manager?
- Entity set E is **total** wrt. relationship set R
:⇔ all E entities participate in R
- Entity set E is **partial** wrt. relationship set R
:⇔ some E entities do not participate in R



Weak Entities

forget this slide!

- **weak entity**: identified uniquely only by considering the primary key of another (**owner**) entity
- Owner entity set and weak entity set must participate in a **one-to-many** relationship set (one owner, many weak entities)
- Weak entity set must have **total participation** in identifying relationship set (no identification of its own!)



ISA ('is a') Hierarchies

- **A ISA B**: every A entity is also a B entity ("A inherits from B")

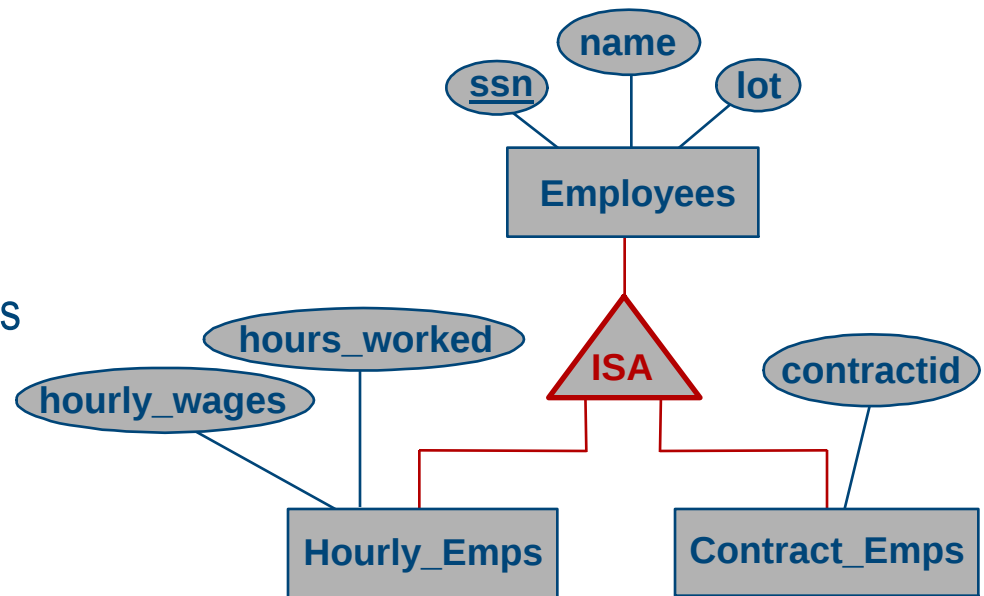
- A entities have attributes like B entities have, plus maybe more
- A is called **subclass**, B **superclass**

- Purpose:

- add attributes specific to a subclass
- identify specific entities that participate in a relationship

- Constraints:

- **Overlap constraints**: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (Allowed/disallowed)
- **Covering constraints**: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? (Yes/no)

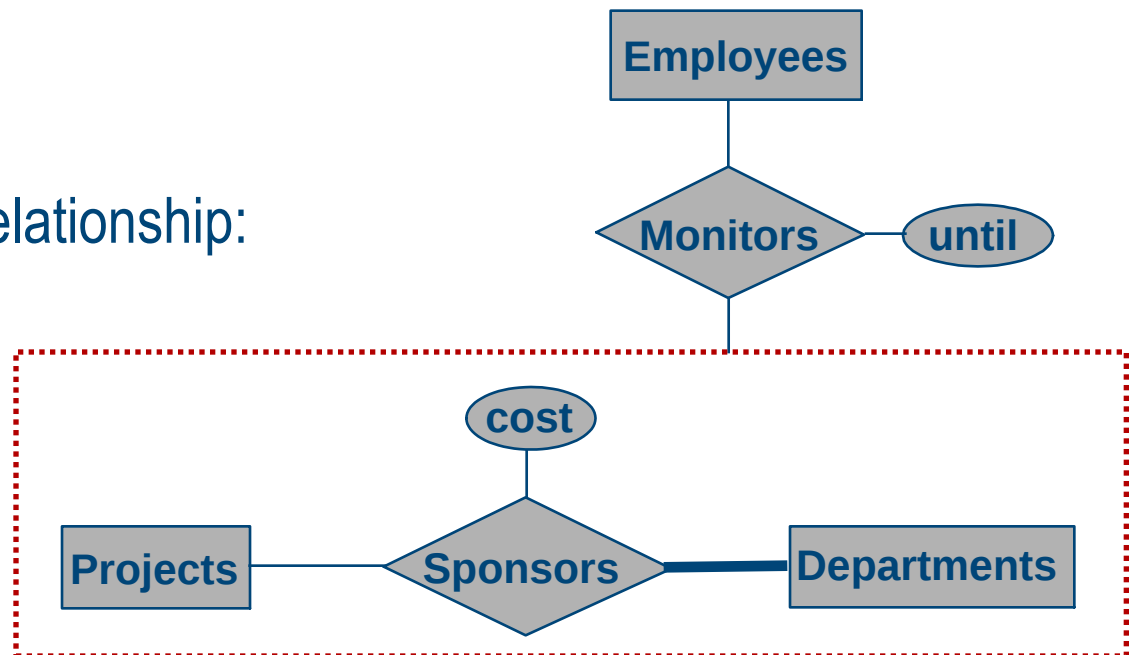


Aggregation

- **Aggregation** = relationship involving (entity sets and) a relationship set
- Aggregation allows us to **treat a relationship set as an entity set** for purposes of participation in (other) relationships

- Aggregation vs. ternary relationship:

- Monitors is a distinct relationship, with a descriptive attribute
- each sponsorship is monitored by at most one employee



Conceptual Design Using the ER Model

- Design choices:
 - concept modeled as entity or attribute?
 - concept modeled as entity or relationship?
 - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
 - A lot of data semantics can (and should) be captured
 - But some constraints cannot be captured in ER diagrams – *comment your design!*
- *Let's see...*

Summary of ER

- ER model popular for conceptual design
 - simple & expressive
 - close to the way people think about their applications
- Basic constructs:
entities and relationships, both with attributes
- Some additional constructs:
weak entities, ISA hierarchies, and aggregation
- Note: There are many variations on ER model

Summary of ER (Contd.)

- Several kinds of integrity constraints can be expressed in the ER model
 - key constraints
 - participation constraints
 - overlap/covering constraints for ISA hierarchies
- Some foreign key constraints implicit in definition of a relationship set
 - Some (actually: many) constraints cannot be expressed in the ER model
 - *notably, functional dependencies*
 - But: constraints play an important role in determining the best database design

Summary of ER (Contd.)

- ER design is subjective
 - often many ways to model a given scenario
 - When in doubt (and not only then), ask customer how they will query their data – this usually gives valuable insights
 - Analyzing alternatives can be tricky, esp. large schemas (SAP R/3: 15,000 tables!)
- Common choices include:
 - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship
 - whether or not to use ISA hierarchies, whether or not to use aggregation
- Ensuring good database design: resulting relational schema should be analyzed and refined further → logical design phase
 - Functional dependency information, normalization techniques

UML™

[not in DBMS book, see course website]

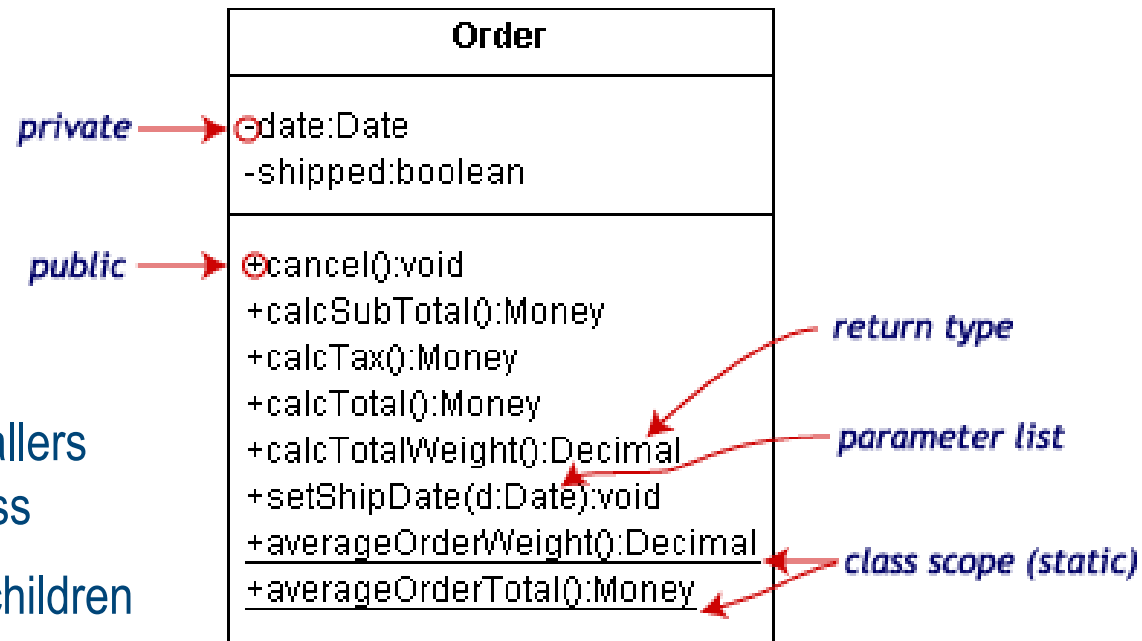
- UML = **Unified Modeling Language** [www.uml.org]
 - Issued by OMG [Object Management Group, www.omg.org]
- "UML is a **graphical language** for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system."
 - does not prescribe particular methodology or process
- Notation & semantics for domains:
 - Use Case Model; Communication Model; Dynamic Model; **Class Model**; **Physical Component Model**; **Physical Deployment Model**
- Much more comprehensive than ER!
 - For our purpose: overkill

Classes

- **Class Model** at the core of object-oriented development and design
- Naming: **instance** (ER: entity) belongs to **class** (ER: entity set)

- Attributes and methods may be marked as:

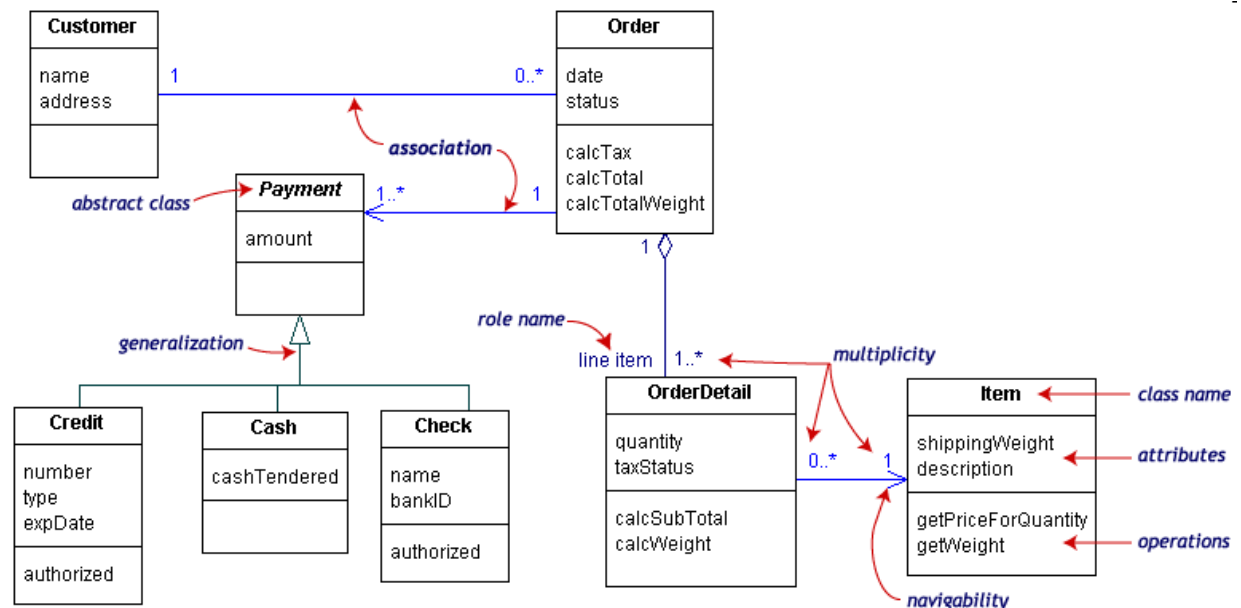
- **Private** -- not visible to callers outside the class
- **Protected** -- only visible to children of the class
- **Public** -- visible to all



Relationships & Class Diagrams

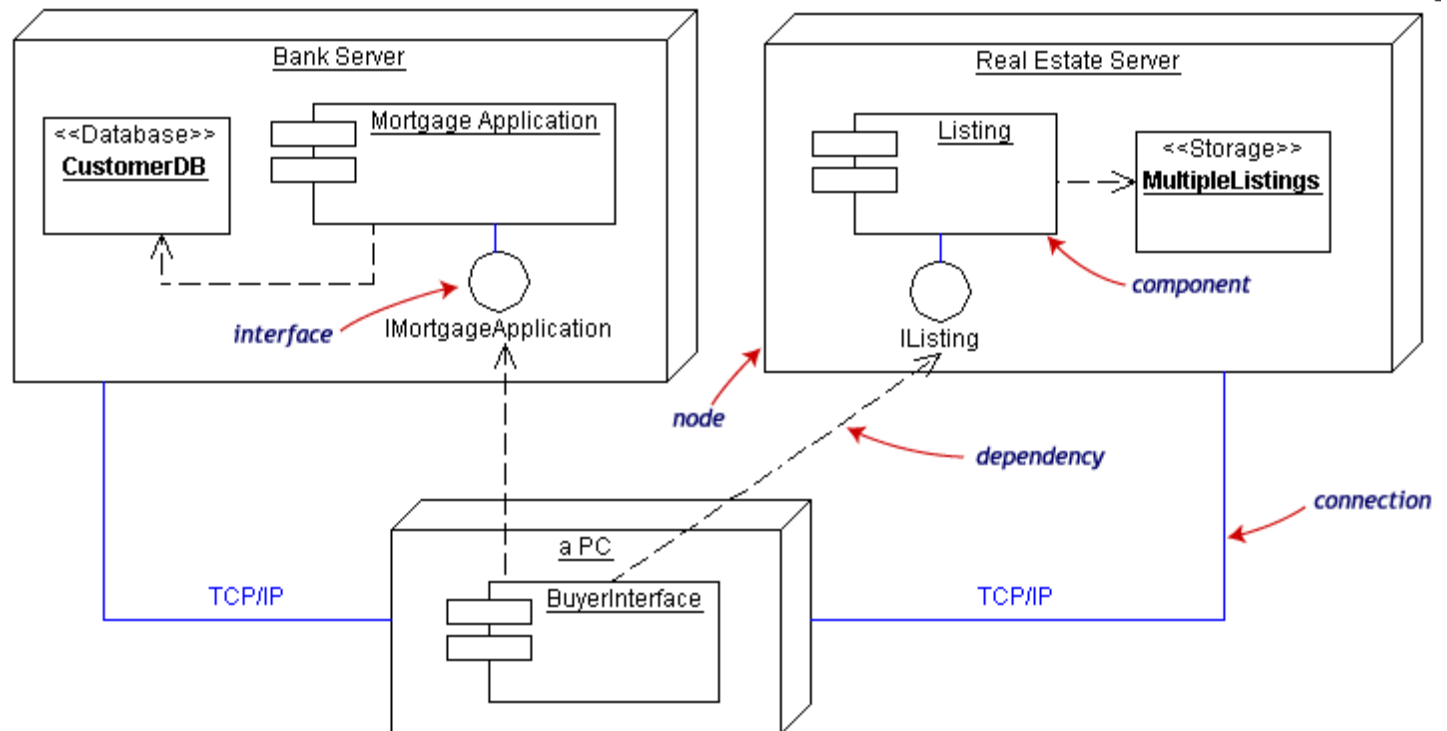
- Relationship types:
 - association ("must know about the other")
 - aggregation / composition (class belongs to a collection)
 - generalization (one class is a superclass of the other)

- Navigability arrows
- Multiplicity
- Role names (optional)



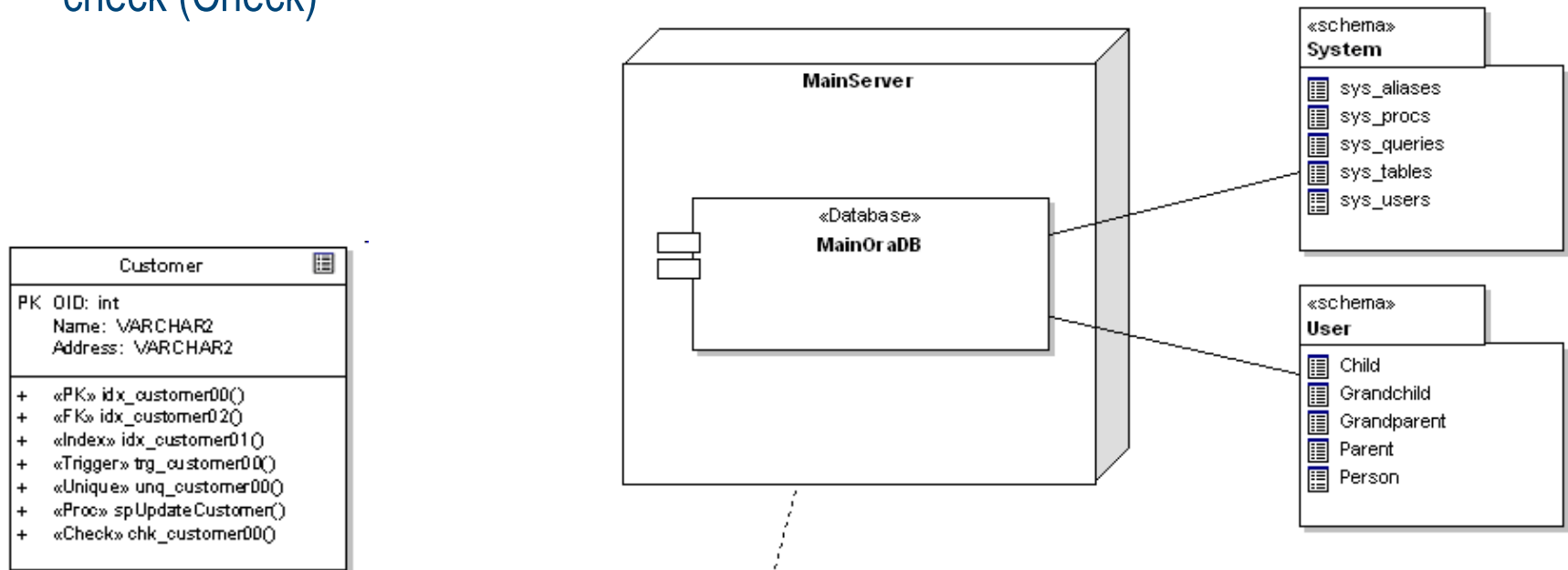
Components and Deployment Diagrams

- **Component** = code module
- **Deployment diagram** = physical configuration of software and hardware



Excursion: UML Physical DB Modelling

- Some relational constructs that can be expressed:
 - primary key constraint (PK), foreign key constraint (FK), index constraint (Index), trigger (Trigger), uniqueness constraint (Unique), stored procedure (Proc), validity check (Check)



A Node is a physical piece of hardware (such as a Unix server) on which components are deployed. The database component in this example is also mapped to two logical «schema», each of which contains a number of tables.