COMS 4111
Introduction to Databases

Alex Biliris
Computer Science Department
Columbia University

My background and interests

- 30+ years experience in data management as a researcher and entrepreneur and
- Currently:
  - Early-stage advisor and angel investor, Member at New York Angels
  - Founder and managing member at JCM
  - Computer science adjunct professor at Columbia U.
- Past:
  - Bell Labs and AT&T Labs Research (91-02)
  - Computer science professor at Boston U. (85-91)
  - PhD in computer science, George Washington U.

What Is a DBMS?

- *Database*: a very large, integrated collection of shared data.
- *Database Management System (DBMS):* the software that stores and manages databases.
- Models real-world enterprise.
  - Entities (e.g., students, courses)
  - Relationships (e.g., student X is taking CS4111)

Why Use a DBMS?

- Data independence and efficient access.
- Reduced application development time
  - Uniform organization of data
  - Significant number of constraints enforced by the DBMS – not at each application, individually.
- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes.

Why Study Databases??

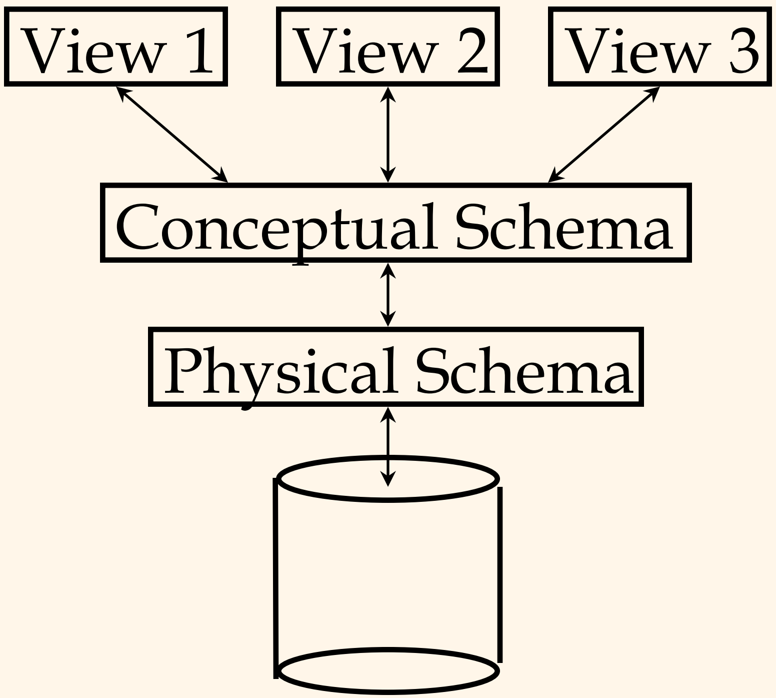
- Datasets increasing in diversity and volume.
  - The Web (transactions, social nets, etc), digital libraries, video & photos, financial applications, payroll, …
  - ... need for DBMS exploding
- DBMS encompasses most of CS areas
  - OS, languages, theory, AI, multimedia, logic

Data Models

- A *data model* is a collection of concepts for describing data; ie, how we view data (eg, Trees? Graph?)
- A *schema* is a description of a particular collection of data, using the given data model.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the columns, or fields.

Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema (or just, schema) defines logical structure
  - Physical schema describes the files and indexes used.



Example: University Database

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course_info(cid: string, enrollment: integer)*

Data Independence

- Applications insulated from how data is structured and stored.
- *Logical data independence*: Protection from changes in *logical* structure of data.
- *Physical data independence*: Protection from changes in *physical* structure of data.
- Data independence is one of the most important benefits of using a DBMS.

Concurrency Control

- DBMS allow concurrent execution of programs.
  - Keeps the CPU humming while disks are accessed (a frequent and slow operation)
  - But interleaving actions of different user programs can lead to inconsistency.
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system.
  - Achieved using Strict 2-Phase Locking Protocol, which ensures that a concurrent execution is equivalent to some serial execution.

Transaction: An Execution of a DB Program

- *Transaction*: an *atomic* sequence of database actions (reads/writes).
- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
  - A *log* (history) of all actions carried out by the DBMS is kept so that the DBMS can undo or redo them
  - When a transaction ends, it either commits (all its updates in DB) or aborts (none in DB)
- Each committed transaction must leave the DB in a *consistent state*. Users specify simple *integrity constraints* on the data, enforced by the DBMS.

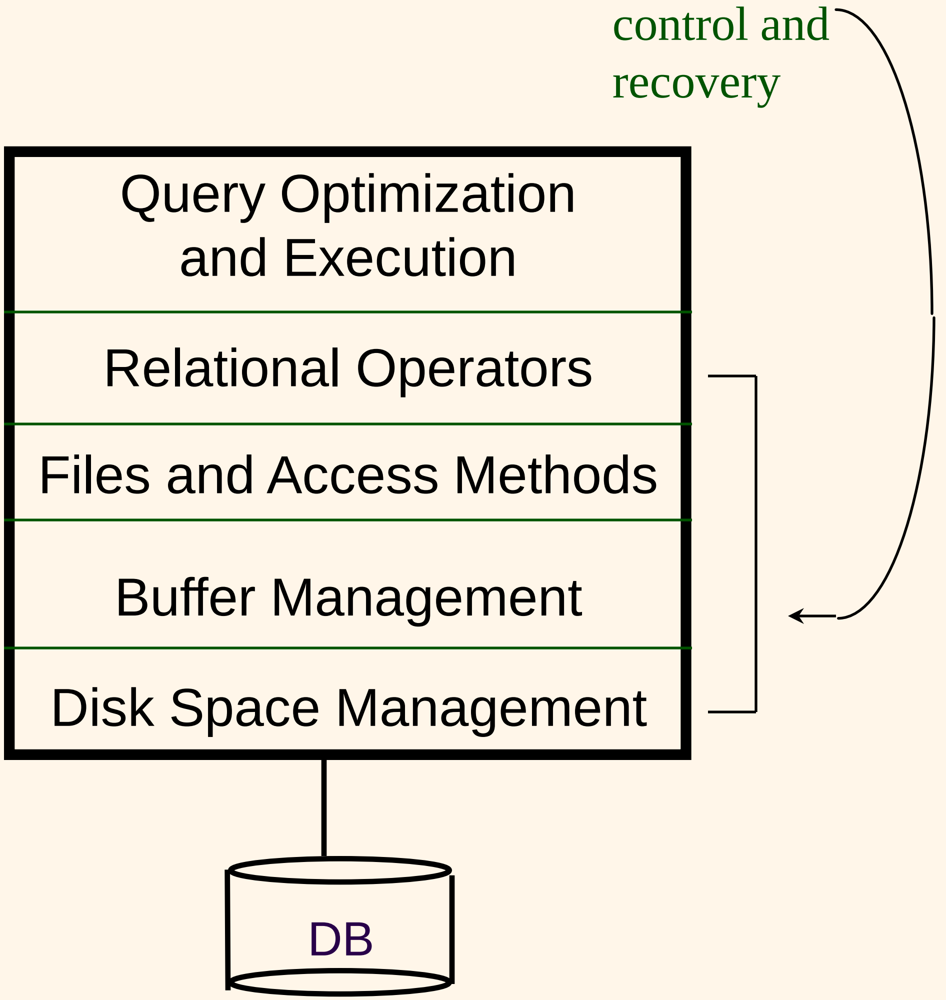
Kind of People Interacting with Databases

- End users
- DB application programmers/developers
- *Database administrator (DBA)*
  - Designs the database (logical/physical schemas)
  - Handles security, authorization, data availability, crash recovery, database tuning as needs evolve
  - Understands not just what a DBMS does but how it does it
- Business analyst (during the database design state)
  - The go-to person to get details of the application for which a database is designed

Structure of a DBMS

These layers must consider concurrency control and recovery

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



Finally …

- Tour of the course website:

  http://www.cs.columbia.edu/~biliris/4111/

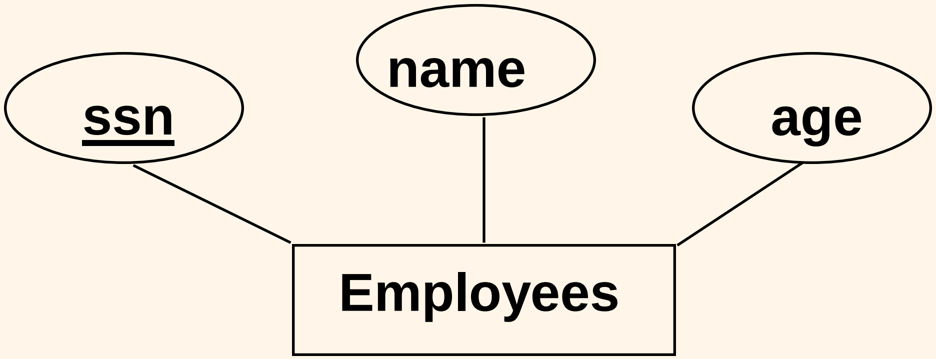
The Entity-Relationship Model

Computer Science Department
Columbia University

Overview of Database Design

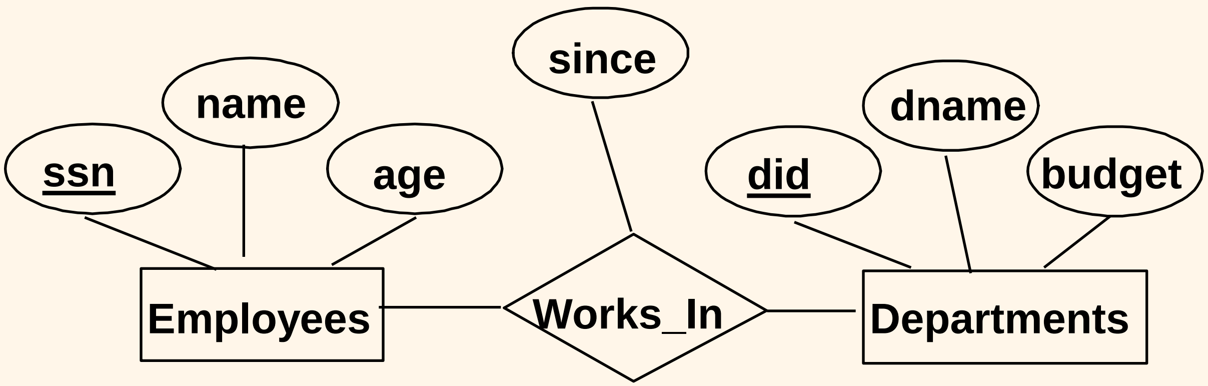
- *Conceptual design*: (*ER Model is used at this stage.)*
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - A database `schema' in the ER Model can be represented pictorially (*ER diagrams*).
  - Can map an ER diagram into a relational schema.
- *Schema Refinement (Normalization):* Check relational schema for redundancies and related anomalies.
- *Physical Database Design and Tuning*: Consider typical workloads and further refine the database design

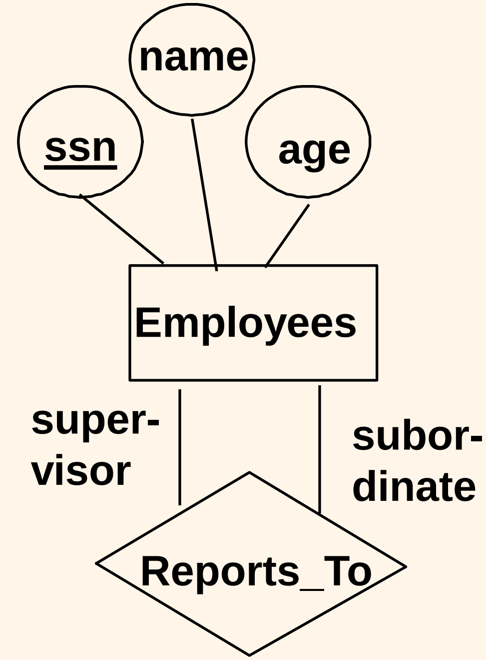
ER Model Basics



- *Entity:* Real-world object distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.
  - Each attribute holds a *single* value of some *domain/type* (no arrays, sets, or any other structure)
- *Entity Set*: A collection of similar entities.
  - All entities in an entity set have the same set of attributes.
  - Each entity set has a *key*.

ER Model Basics (Contd.)

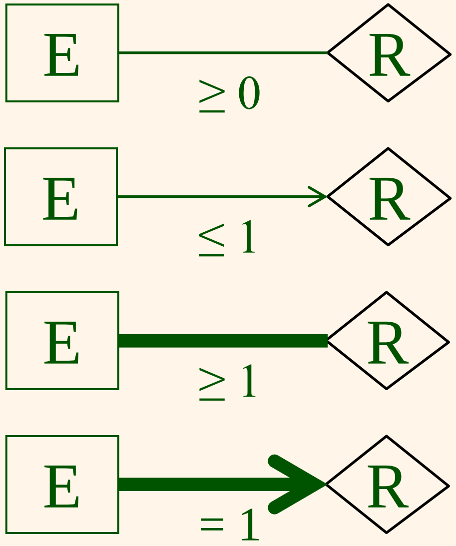


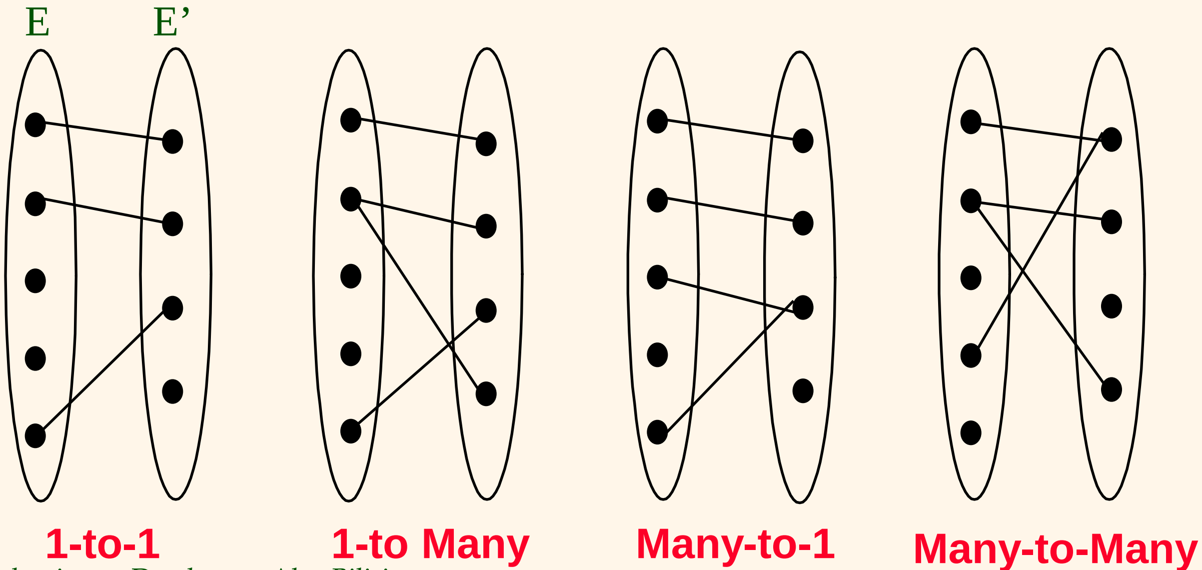


- ***Relationship***: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
- ***Relationship Set***: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 ϵ E1, ..., en ϵ En
    - Same entity set could participate in different relationship sets, or in different "roles" in same set.

Key and Participation Constraints

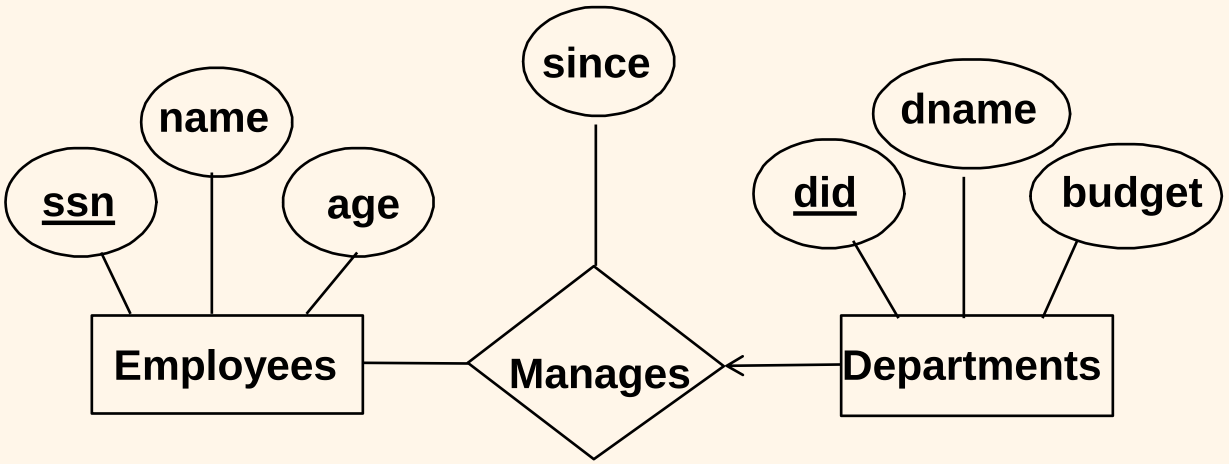
- We use the diagrams to the right to represent how many times an entity ***e*** in E may participate in a relationship R with some other entity in E′ (i.e, the number of E′ entities, ***e*** may be associated with)





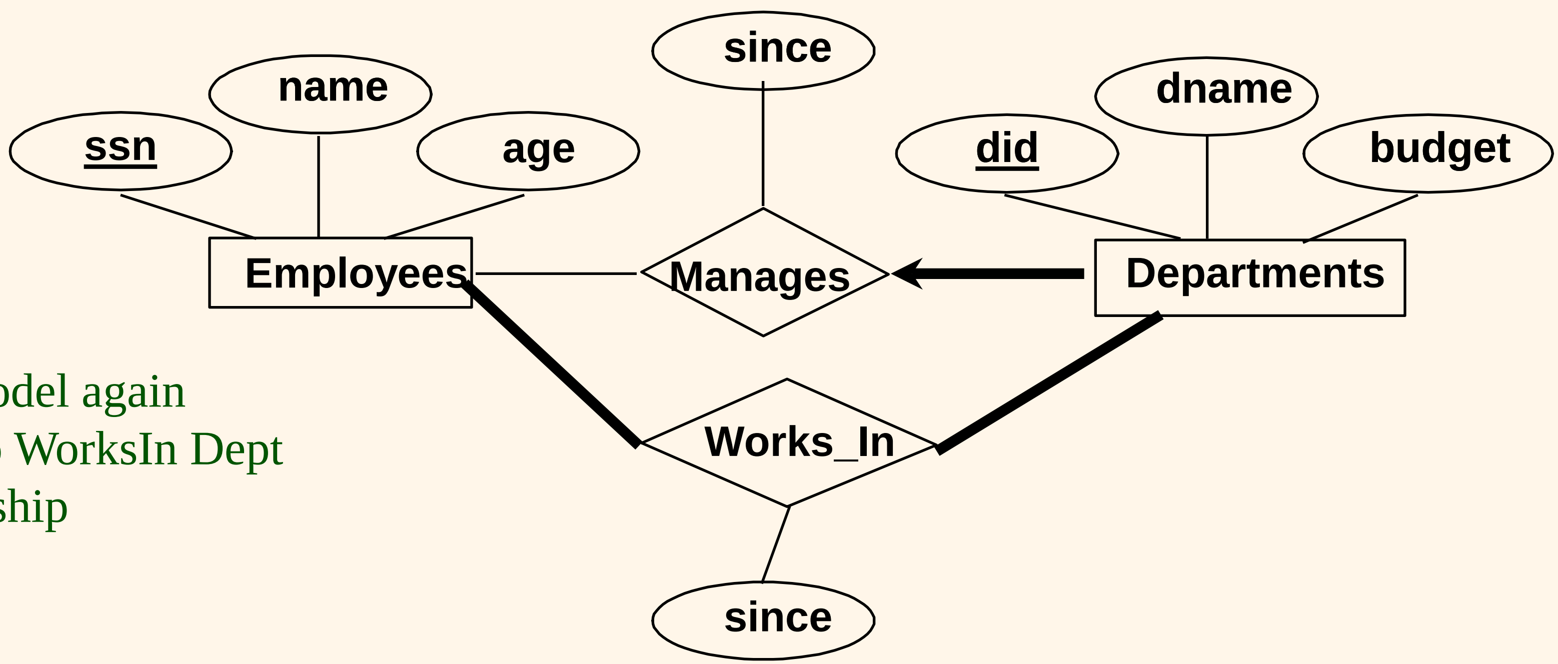
Key Constraints

- In this example, each department has at most one manager, according to the *key constraint* on Manages.



Participation Constraints

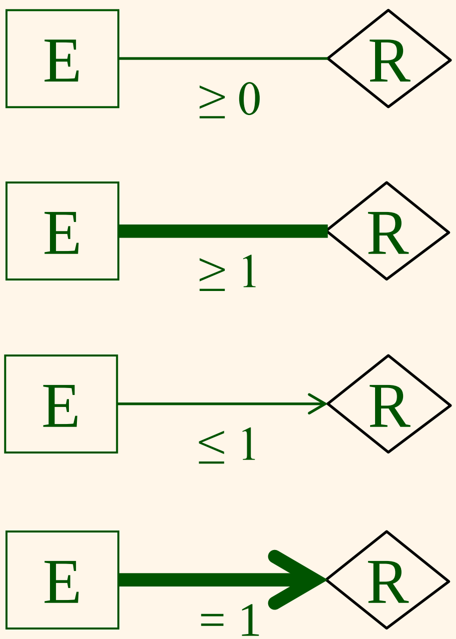
- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must participate in the Manages relationship (associated with a non-null *ssn* value)



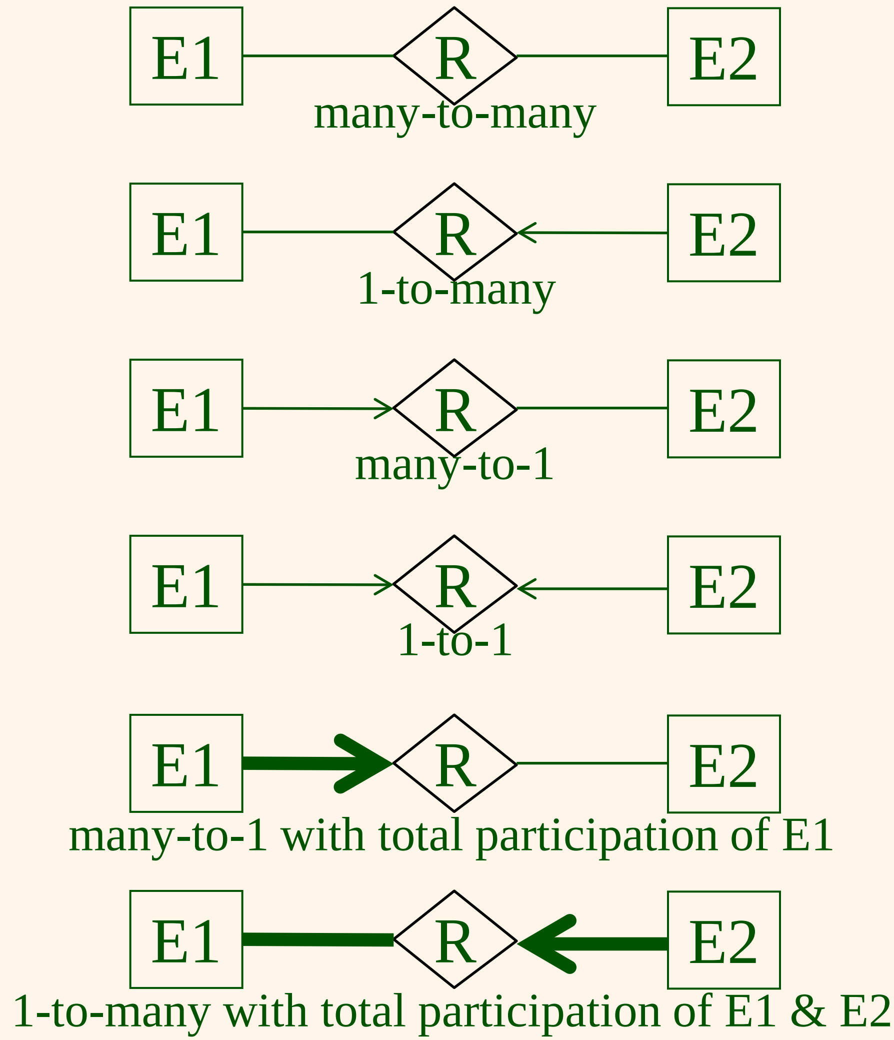
Let's model again the Emp WorksIn Dept relationship

Summary So Far

The number below the connection between E and R indicates the number of times an entity in E may participate in relationship R with some other entity

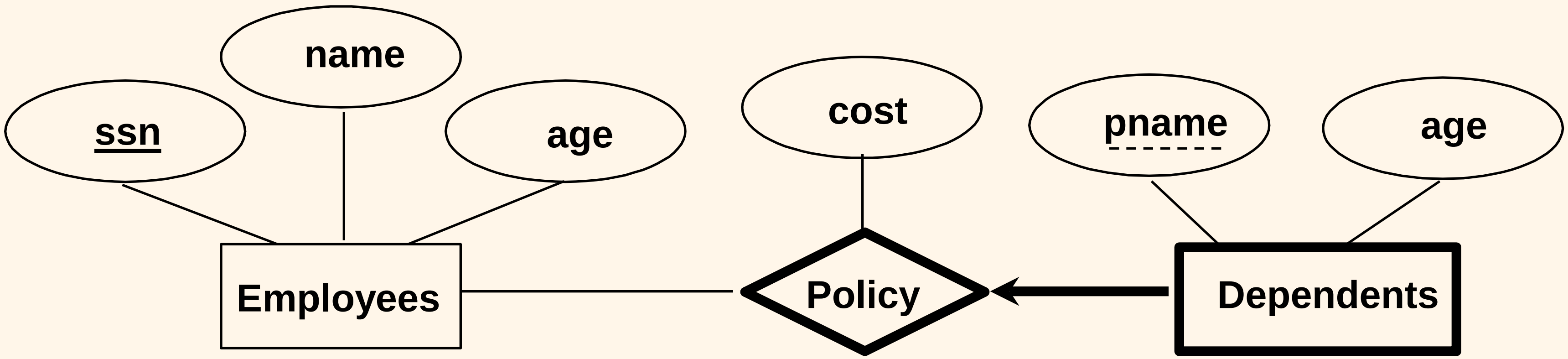


Examples of types of associations (relationships) between entity sets E1 and E2

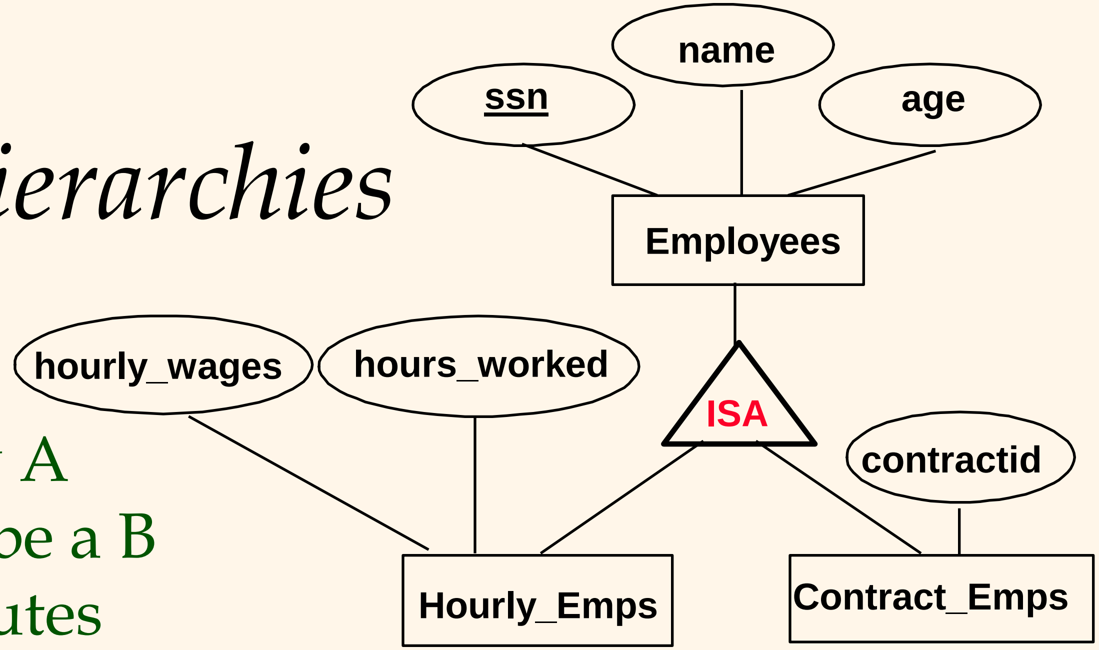


Weak Entities

- A *weak entity* can be 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-one or one-to–many relationship set (one owner, one or many weak entities; each weak entity has a single owner).
  - Weak entity set must have total participation in this *identifying* relationship set.



ISA (`is a') Hierarchies

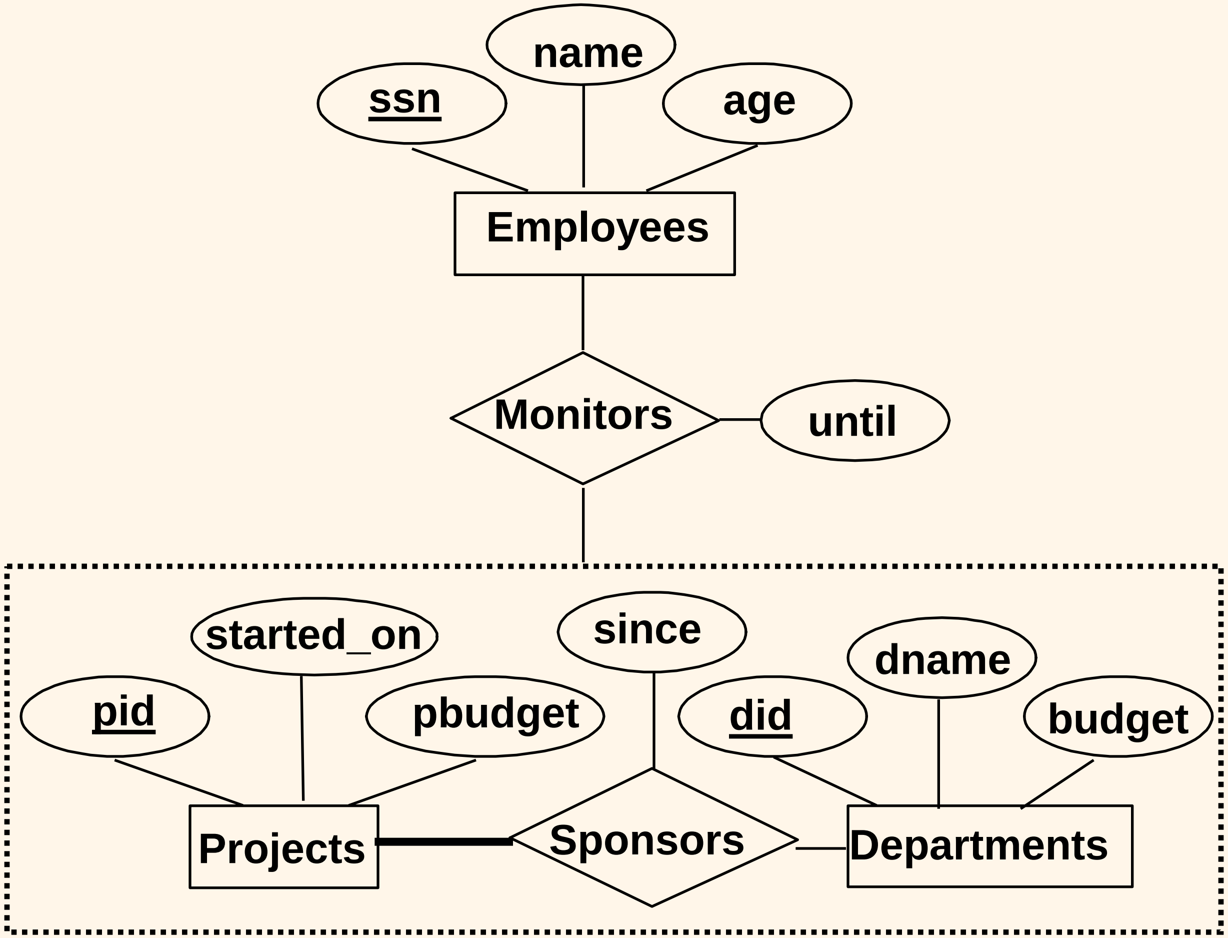


If we declare A **ISA** B, every A entity is also considered to be a B entity. As in OO PLs, attributes are inherited.

- *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*)
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entitities that participate in a relationship.

Aggregation

- Used when we have to model a relationship involving (entitity 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.
- Also, can say that each sponsorship is monitored by at most one employee.

Conceptual Design Using the ER Model

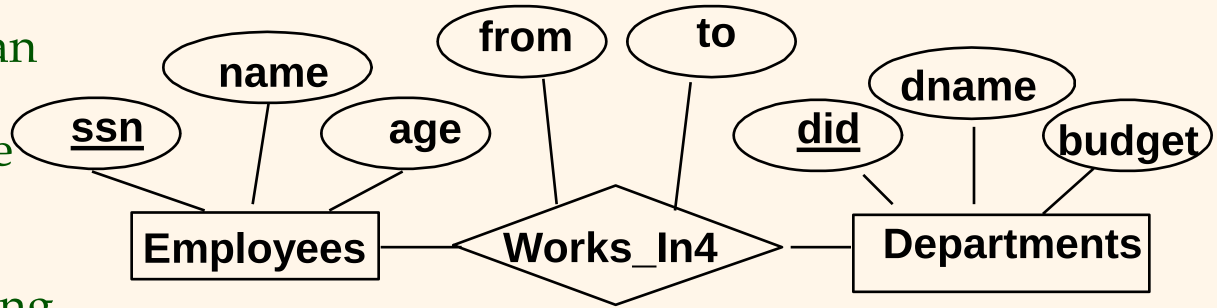
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

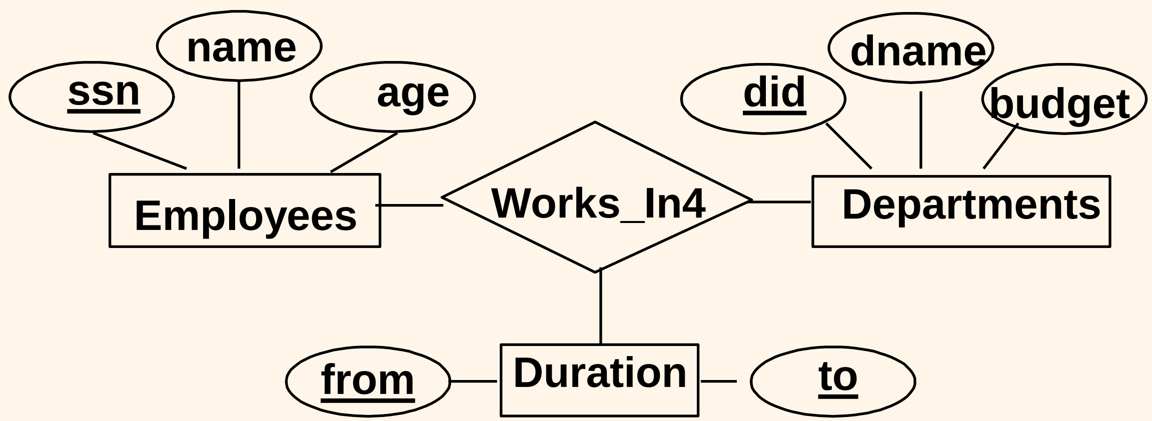
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute (Contd.)

- Works_In4 does not allow an employee to work in a department for two or more periods. (Because a relationship is uniquely identified by the participating entities.)

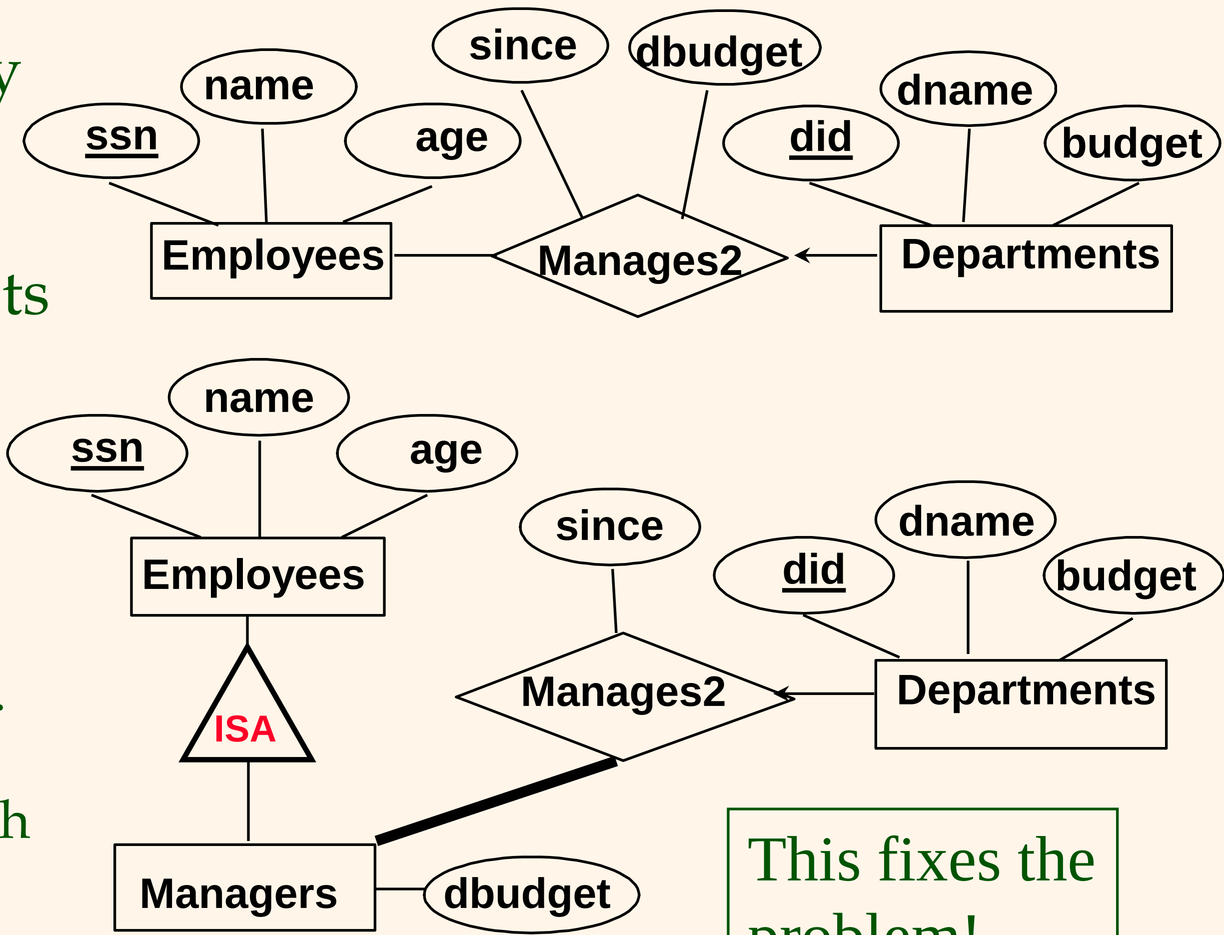


- Similar to the problem of wanting to record several addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship.* Accomplished by introducing new entity set, Duration.



Entity vs. Relationship

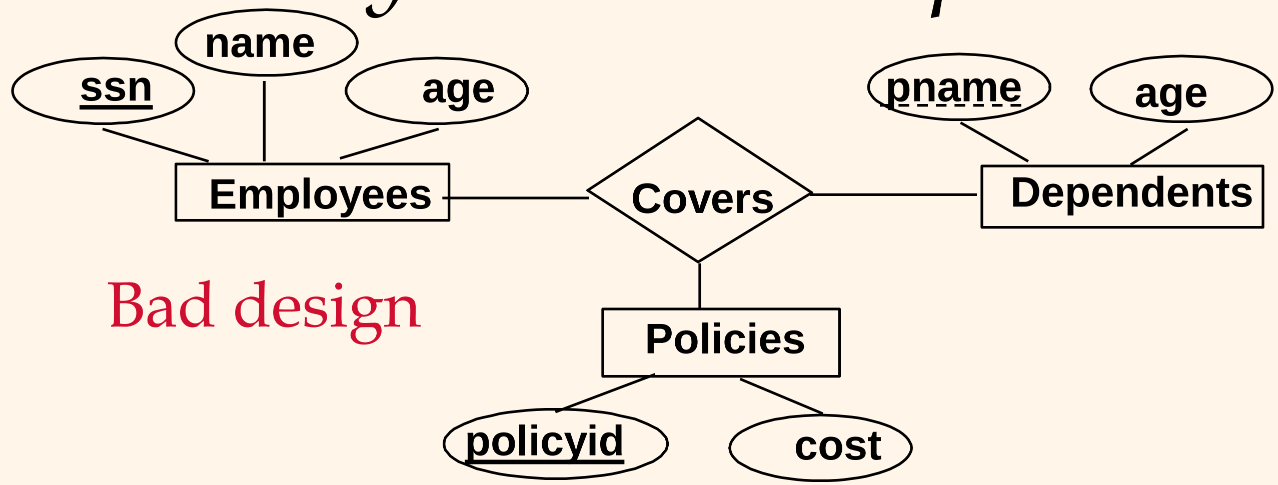
- A manager gets a separate discretionary budget for each dept. First ER diagram OK
- What if a manager gets a discretionary budget that covers *all* managed depts?
  - Redundancy: *dbudget* stored for each dept managed by manager.
  - Misleading: Suggests *dbudget* associated with department-mgr combination.



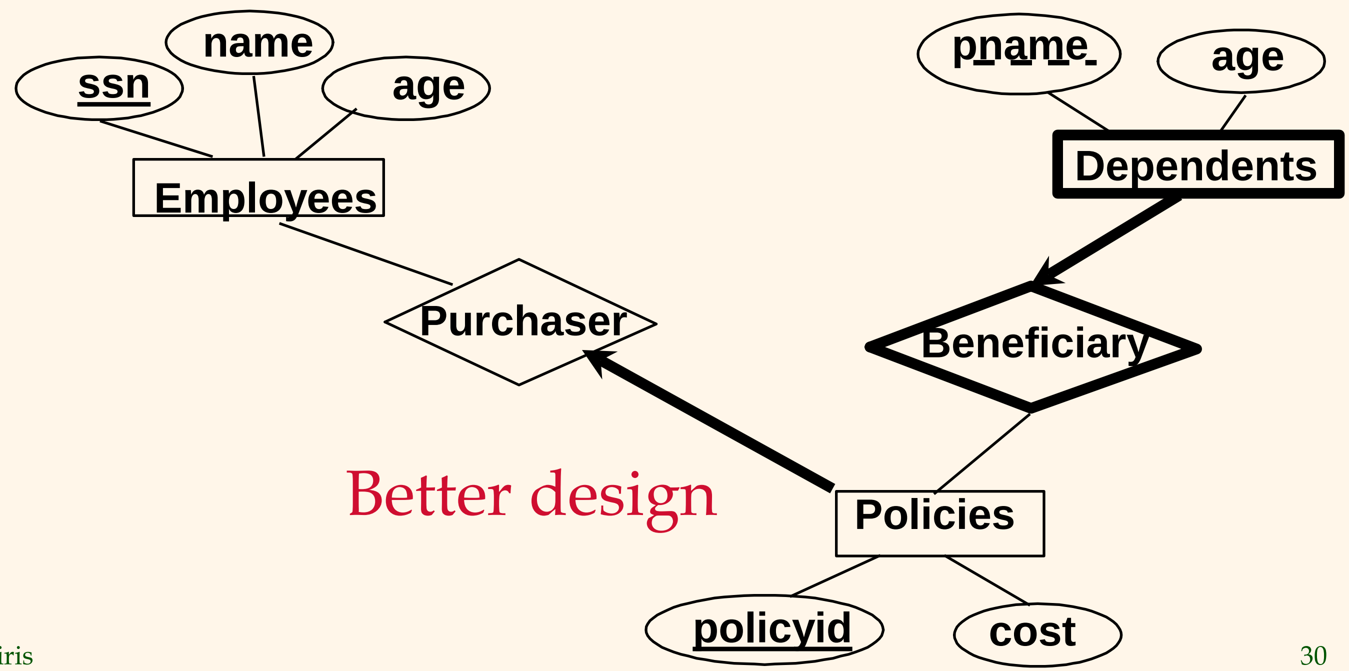
This fixes the problem!

Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate.



Bad design

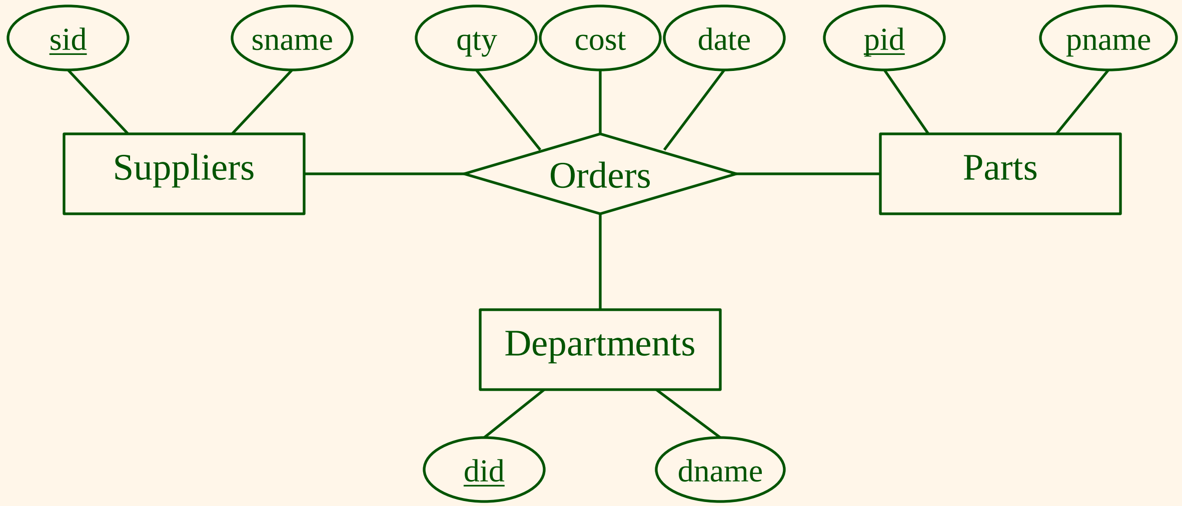


Better design

- What are the additional constraints in the 2nd diagram?

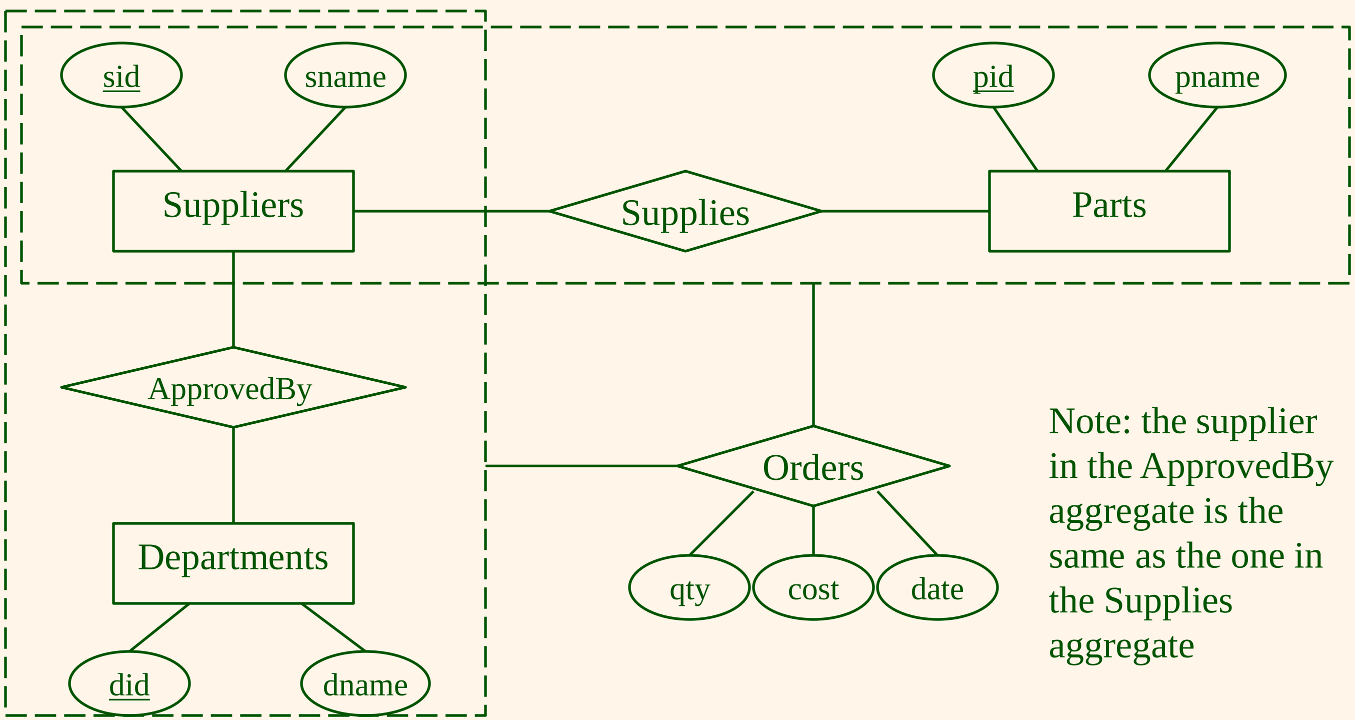
Binary vs. Ternary Relationships (Contd.)

- An example where a ternary relationship is better than two binary relationships: Orders relates entity sets Suppliers, Departments and Parts, and has descriptive attributes *qty*, *cost* and delivery *date*.
  - An order <S,D,P> indicates: a supplier S supplies to a department D *qty* of a part P at a certain *cost* on a certain *date*. Not possible to record *qty*/ *cost*/*date* with some combination of binary relationships



Same app, different constraints

- As before: orders between departments and suppliers for parts at certain qty/cost. And with these additional constraints:
- A department maintains a list of approved suppliers who are the only ones that can supply parts to the department
- Each supplier maintains a list of parts that can supply



Note: the supplier in the ApprovedBy aggregate is the same as the one in the Supplies aggregate

Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis,*
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities*, *ISA hierarchies*, and *aggregation*.
- Note: There are many variations on ER model.

Summary of ER (Contd.)

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, and whether or not to use aggregation.
- Ensuring good database design: resulting relational schema should be analyzed and refined further.