UNIT 2
Conceptual Database Design

Read Text, Chapter 2

Learning Goals

Given a problem description,

- create an Entity-relationship (ER) diagram with the correct entities and relationships that accurately model the problem's data
- Accurately represent entities that depend on other entities, or inherit from other entities or have n-ary relationships with other entities, in the ER model created.
- identify alternative representations of the problem concepts and evaluate the choices
  - compare alternative ER models for the same domain and identify their strengths and weaknesses

Database Design

Consists of 6 steps:

- Requirements Elicitation & Analysis
  - common to all application designs; covered in CPSC 310
- Conceptual Database Design
  - produces a high level description of data (***semantic models***)
- Logical Database Design
  - converts conceptual design into a DB schema (***logical schema***)
- Schema Refinement
  - restructuring to satisfy desirable properties (***normalization***)
- Physical Database Design
  - deals with how the instances (actual data) will be stored on disk
- Application & Security Design
  - general application design
  - applications are restricted to only access data that they need

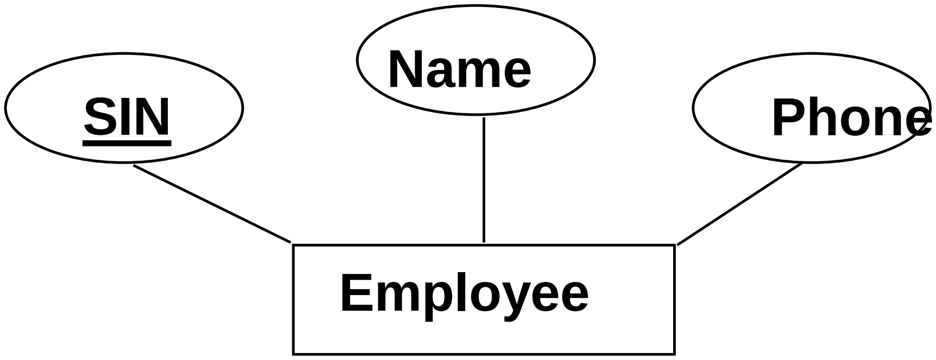
Conceptual Database Design

Main goal: Given the requirements for an application, identify:

1. Entities and relationships in the application's domain
   - Entities are usually nouns, but avoid irrelevant nouns
   - e.g., *At UBC, we have full-time and part-time students. We store such information in our computer. We have 1000's of courses…*; *Spielberg is a director who lives in Beverly Hills, CA.*
   - Relationships are statements about 2 or more objects. Often, verbs.
   - e.g., *Hopkins stars in Fracture; Mary took CPSC404 and earned an A+; Tom rated Sky Fall at five stars.*
2. Information about these entities and relationships that should be stored in the database
3. Integrity constraints or business rules that hold
   - For relational databases, the ER Model is used at this stage.
   - The enterprise data is usually represented by an ER diagram.

ER Model Basics: Entities

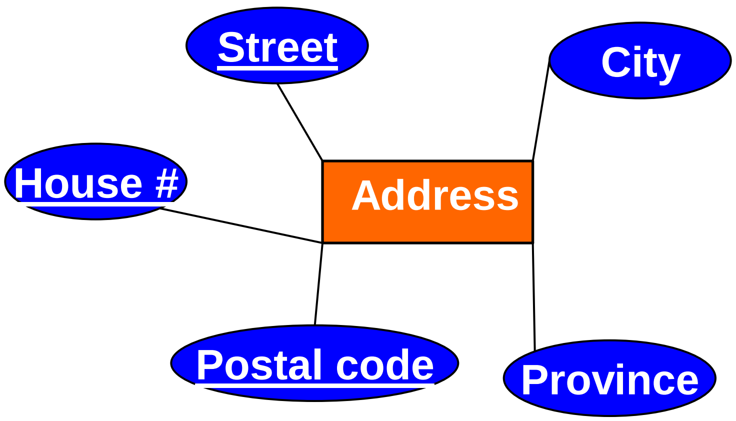
- ***Entity*:** Real-world object *distinguishable* from other objects, whose data is important for the application.
- An entity is described using a set of properties, called ***attributes***.



- ***Entity Set:*** A collection of similar entities. E.g., all employees of an organization.
  - All entities in an entity set have the same set of attributes, i.e., same schema. (Until we consider ISA hierarchies, anyway!)
  - Each attribute has a ***domain***. *(e.g.., float, string, date, int, color)*
  - Each entity set has a ***key.***

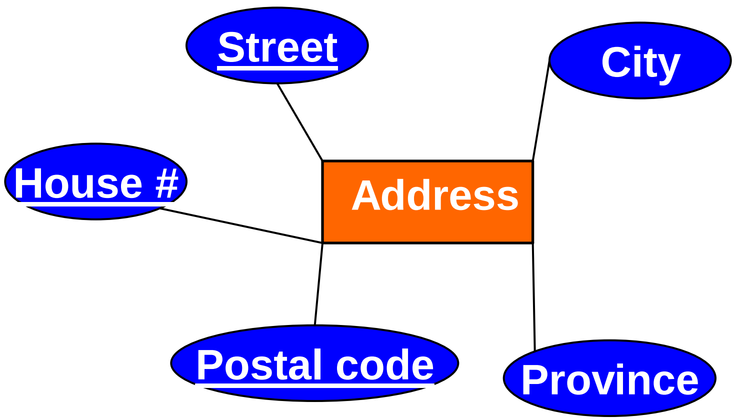
Keys

- Distinguish entities
- A ***key*** is the minimal set of one or more attributes which, taken collectively, identify uniquely an entity in an entity set.
- A ***superkey*** is a key plus zero or more other attributes in the entity set
- A ***primary key*** is the key chosen as the principal means to identify entities in an entity set – it's a matter of designation
- Only primary keys are shown in ER diagrams (underlined attributes)
- We'll discuss superkeys when we consider normal forms (for now, don't worry about them)



Keys

- Distinguish entities
- A ***key*** is the minimal set of one or more attributes which, taken collectively, identify uniquely an entity in an entity set.
- A ***superkey*** is a key plus zero or more other attributes in the entity set
- A ***primary key*** is the key chosen as the principal means to identify entities in an entity set – it's a matter of designation
- Only primary keys are shown in ER diagrams (underlined attributes)
- We'll discuss superkeys when we consider normal forms (for now, don't worry about them)

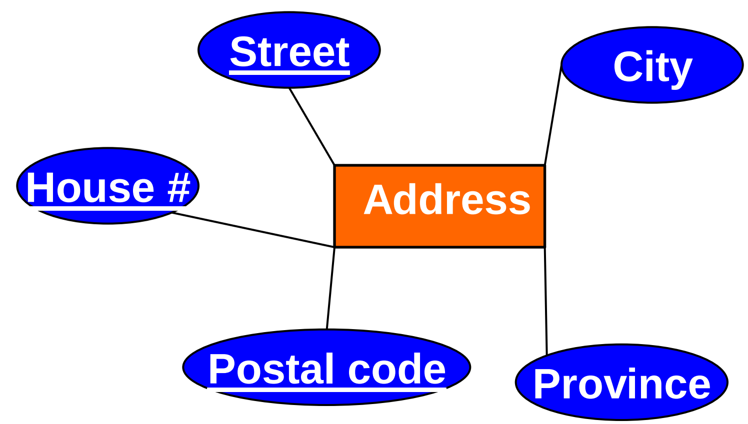


Key Constraint: E – entity Set; K set of key attributes; says

$$\forall e_i, e_j \in E: e_i.K = e_j.K \rightarrow e_i = e_j.$$

Keys

- Distinguish entities
- A ***key*** is the <u>minimal set</u> of one or more attributes which, taken collectively, identify uniquely an entity in an entity set.
- A ***superkey*** is a key plus zero or more other attributes in the entity set
- A ***primary key*** is the key chosen as the principal means to identify entities in an entity set – it's a matter of designation
- Only primary keys are shown in ER diagrams (underlined attributes)
- We'll discuss superkeys when we consider normal forms (for now, don't worry about them)

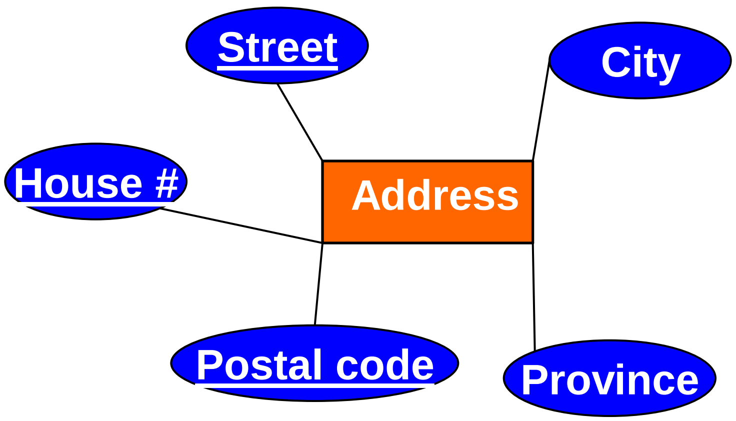


Key Constraint: E – entity Set; K set of key attributes; says

$\forall e_i, e_j \in E: e_i.K = e_j.K \rightarrow e_i.A = e_j.A$, for all attributes A of E.

Keys

- Distinguish entities
- A ***key*** is the minimal set of one or more attributes which, taken collectively, identify uniquely an entity in an entity set.
- A ***superkey*** is a key plus zero or more other attributes in the entity set
- A ***primary key*** is the key chosen as the principal means to identify entities in an entity set – it's a matter of designation
- Only primary keys are shown in ER diagrams (underlined attributes)
- We'll discuss superkeys when we consider normal forms (for now, don't worry about them)



E.g., $\forall t_i, t_j \in Address$:

$$t_i.House\# = t_j.House\# \ \& \ t_i.PCode = t_j.PCode \ \& \ t_i.Street = t_j.Street \rightarrow t_i.City = t_j.City \ \& \ t_i.Prov = t_j.Prov.$$

Keys in Datalog-like Notation

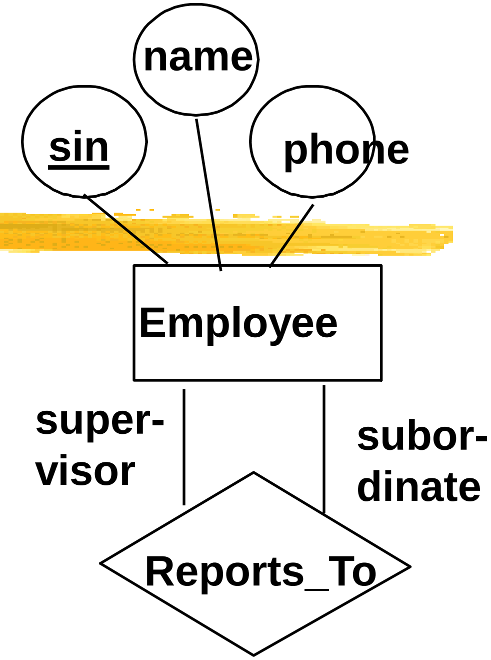
- Emp(<u>SIN</u>, Name, Phone).
- $Emp(S, N, P), Emp(S, N', P') \rightarrow N = N' \,\&\, P = P'.$
- This is an example of an *Integrity Constraint*.
- We can write the above rule equivalently as two rules:
- $Emp(S, N, P), Emp(S, N', P') \rightarrow N = N'$ and
- $Emp(S, N, P), Emp(S, N', P') \rightarrow P = P'.$

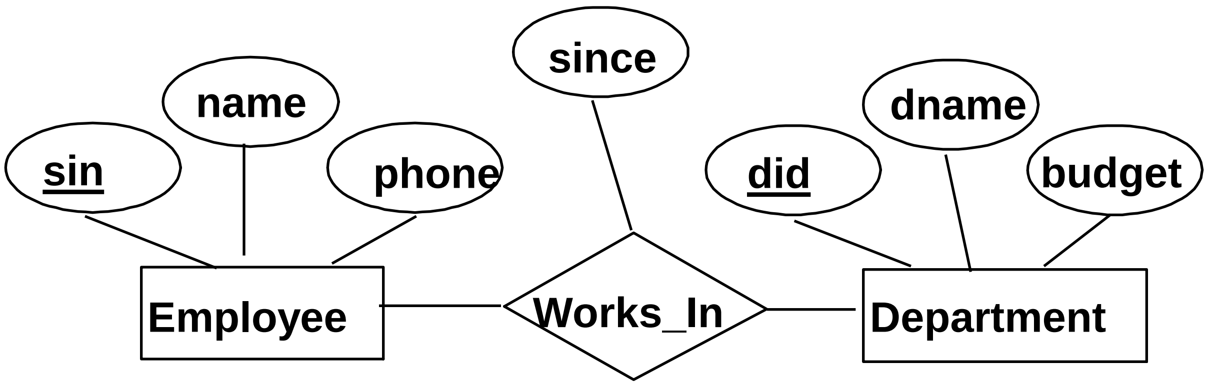
More examples of Key Constraints in Datalog-like Notation

- Address(<u>House#</u>, <u>Street</u>, City, Province, <u>PostalCode</u>).
- In Datalog-like rule syntax:
- $Address(H,S,C,P,PC) \& Address(H,S,C',P',PC) \rightarrow C = C'$.
- $Address(H,S,C,P,PC) \& Address(H,S,C',P',PC) \rightarrow P = P'$.

- Note: The two rules above *together* say that {House#, Street, PostalCode} is a key of Address.
- Observe which variables are shared (i.e., repeated) in the LHS of the key constraint rules.
- Key constraints are a special case of *Functional Dependencies* (FDs).
- FDs are a special case of *Equality Generating Dependencies* (EGDs). [We'll see both in detail later.]

ER Model Basics (Cont.)



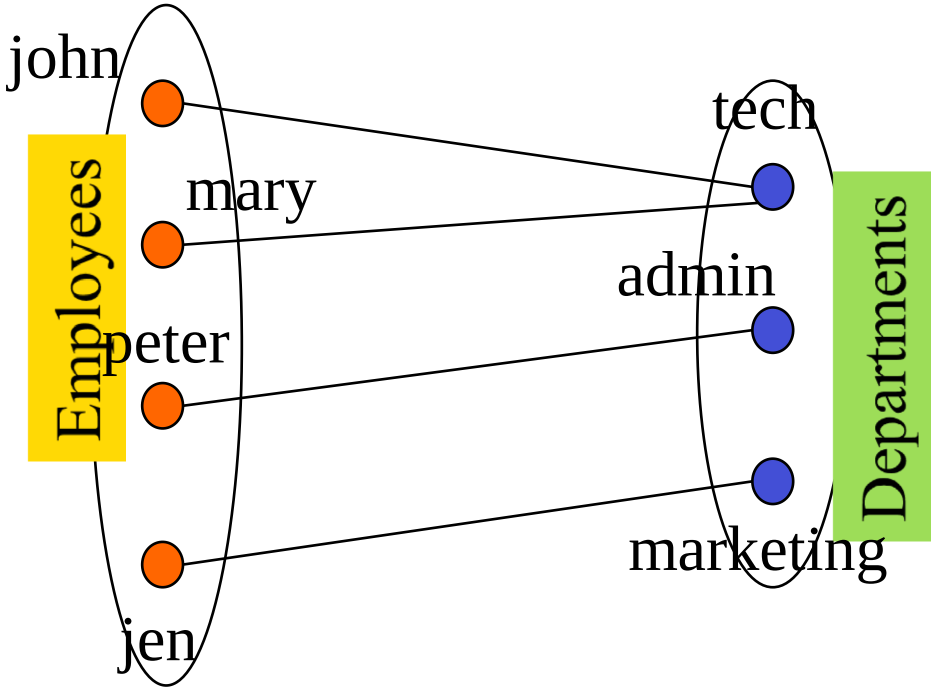


- ***Relationship:*** Association among two or more entities.
  - E.g., *John works in Marketing department.*
- ***Relationship Set:*** Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R connects entities e1 ∈ E1, ..., en ∈ En
    - Same entity set could participate in different relationship sets, or in different "roles" in the same relationship set.
  - A relationship set may have ***descriptive attributes*** (like **since**).
  - ***Degree*** or ***arity:*** # of entity sets in the relationship (binary, ternary, etc.)

What does a relationship really say?

- Suppose R is a relationship set between entity sets $E_1, \ldots, E_n$.
- $E_1 \times E_2 \ldots \times E_n$ – all possible combinations of those entities.
- Only some combinations are related via relationship R.

Relationship *worksIn* between *Employees* and *Departments*



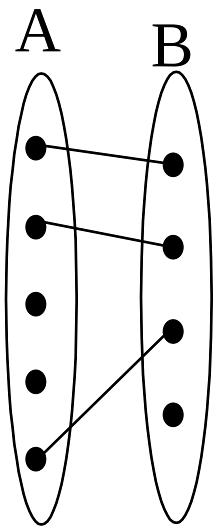
Cardinality Constraints (Mapping Cardinalities)

- A **cardinality ratio** for a relationship set specifies the number of relationships in the set that an entity can participate in.

Let R be a relationship set between entity sets A and B.

R can be :

- ***one-to-one*** from A to B:
  - an entity in A is associated with ***at most one*** entity in B and vice versa
  - e.g., A: driver, B: driver's license
  - Each country has a unique capital and each city is the capital of at most one country.
  - each movie has one lead actor. But does the converse hold?



One-to-One made precise

- More precisely, a one-to-one relationship R between A and B says:

$$(\forall a_i \in A)\big(\forall b_j \in B\big)(\forall b_k \in B)[R\big(a_i, b_j\big)\&R(a_i, b_k\,) \rightarrow b_j = b_k].$$

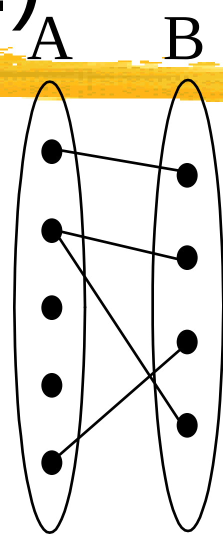
AND

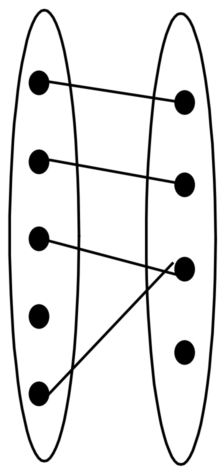
a similar statement with A and B interchanged.

- We'll revisit this with examples, where we'll use the simpler and more intuitive Datalog-like notation.
- Meanwhile, observe the similarity between the above constraint and EGD!

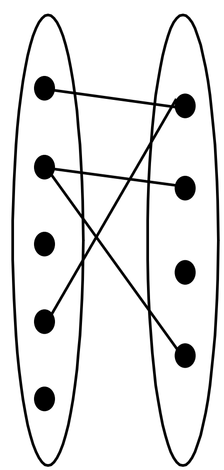
Cardinality Constraints (contd.)

- ***one-to-many*** from A to B:
  - an entity in A is associated with any number of entities in B
  - an entity in B is associated with at most one entity in A
  - e.g. A: mother, B: children
  - e.g. A: boss, B: employee

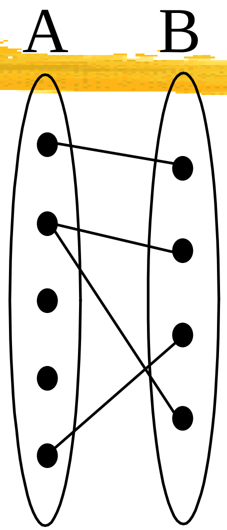




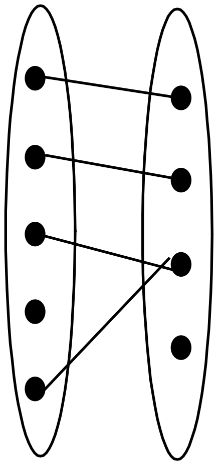
- ***many-to-one*** *from A to B: switch A and B above*
- ***many-to-many*** *from A to B:*
  - *an entity in A is associated with any number of entities in B and vice versa*
  - *e.g. A: students, B: courses*
  - An actor may star in any #movies and a movie has many actors.



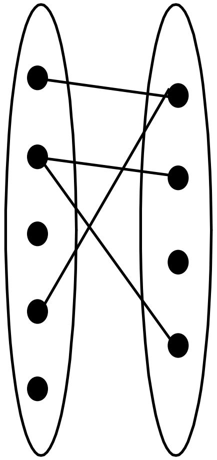
Cardinality Ratios (contd.)



- ***one-to-many*** from A to B:
  - $(\forall b_i \in B)\big(\forall a_j, a_k \in A\big)\big[R\big(a_j, b_i\big) \& R(a_k, b_i) \rightarrow a_j = a_k\big]$.
  - Do we need to say anything about the "A-to-B" direction?
- ***many-to-one*** *from A to B: switch A and B above*

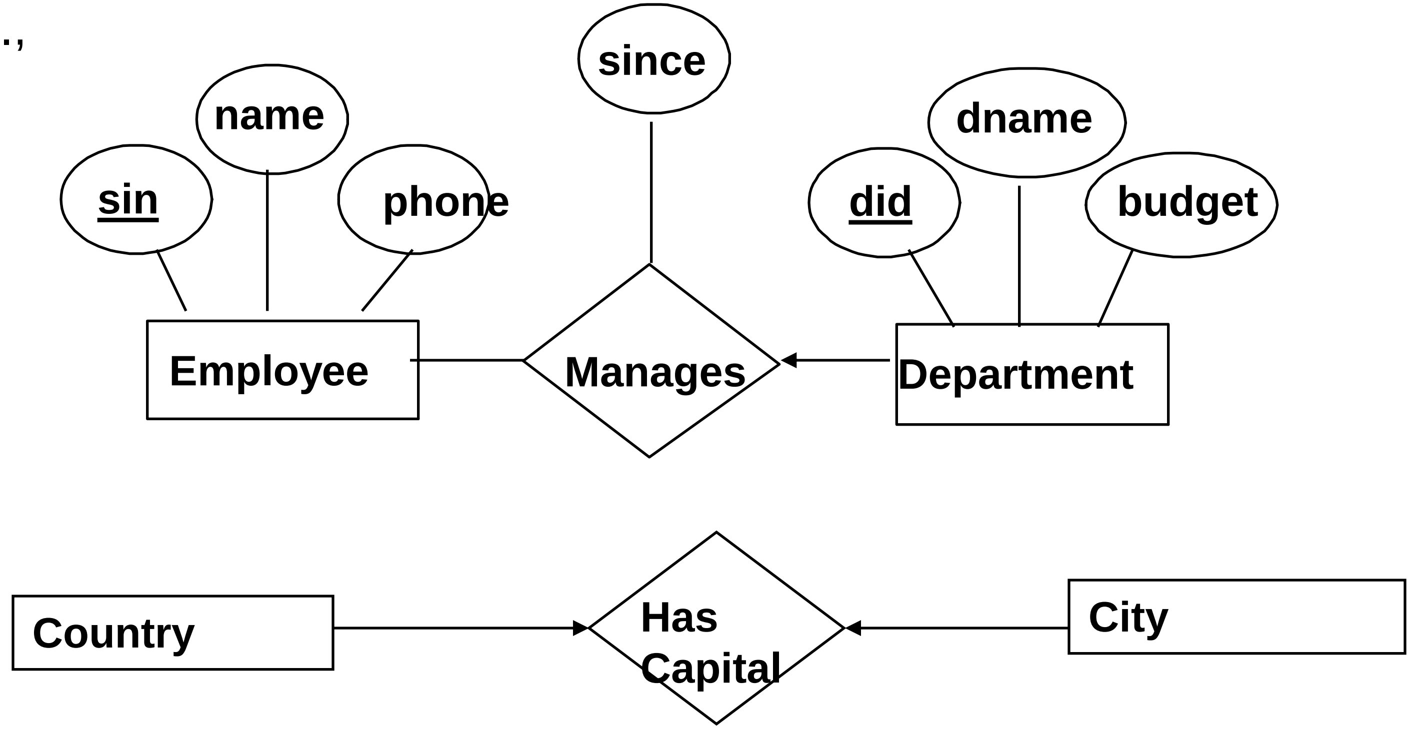


- ***many-to-many*** *from A to B:*
  - Do we need to say anything at all?



Key Constraints for Relationships

- The restriction imposed by a 1-to-1 and many-to-1 ratios are examples of ***key constraints,*** expressed at a relationship set level.
- A key constraint (for a relationship set) is shown with an ***arrow*** in the ER diagram.
- **DBMS should enforce these upon inserts.**
- E.g.,



Key Constraints for Relationships Made Precise

The many-to-one relationship **Manages** from **Department** to **Employee** really says:

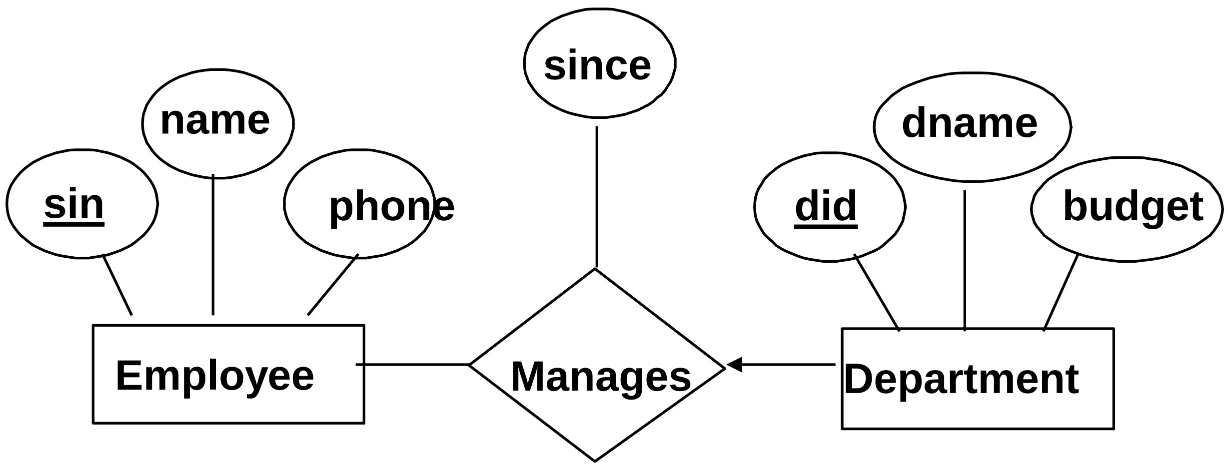
$$Dept(Did,\_,\_)\&Emp(Sin,\_,\_)\&Emp(Sin',\_,\_)\&$$
$$Manages(Did,Sin)\&Manages(Did,Sin') \rightarrow Sin = Sin'.$$

I.e., *each department is managed by at most one employee (aka manager). OR*
no dept. is managed by more than one employee.

The one-to-one relationship **HasCapital** between **Country** and **City** says no country has more than one capital and no city is the capital of more than one country.
Exercise *(on your own):* Write down this constraint in Datalog-like rule notation.

A brief note on notation

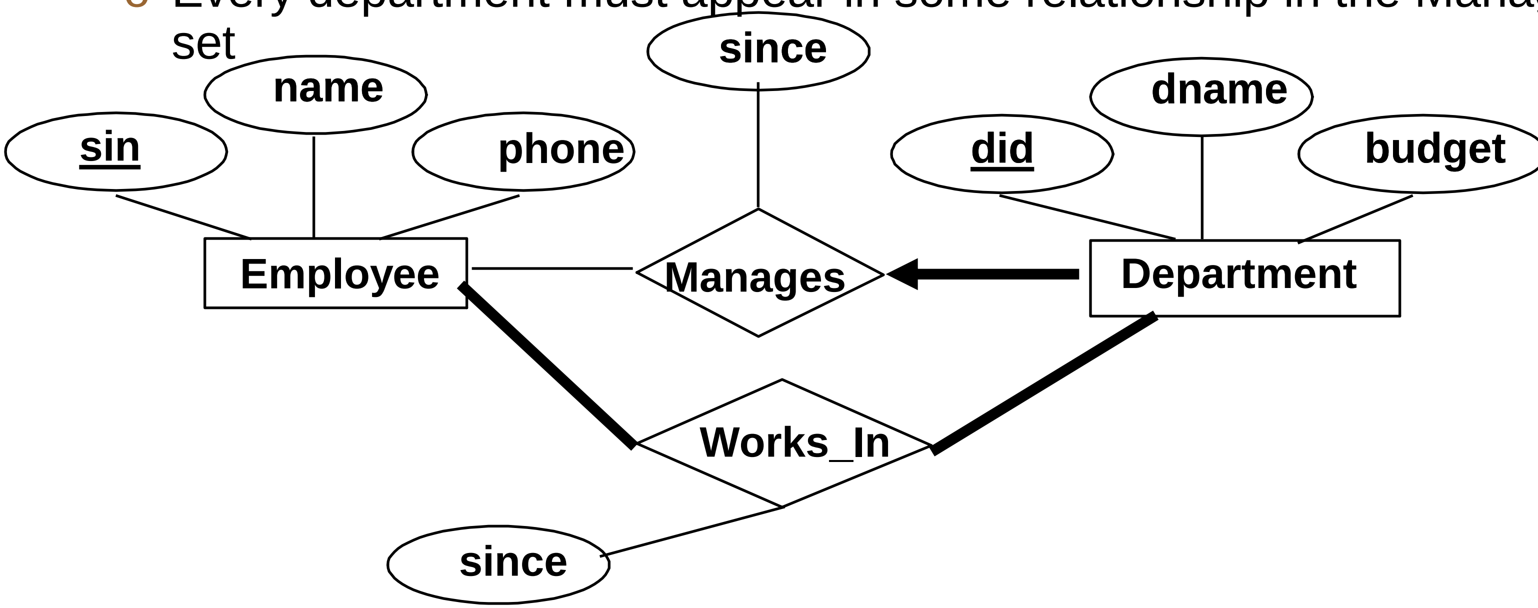
- The ER notation we use can be read: “if you know the entity at the tail of the arrow, then you know the relationship (and the other entities involved)”



- Other notations put the arrows in other places; be careful when using other sources!

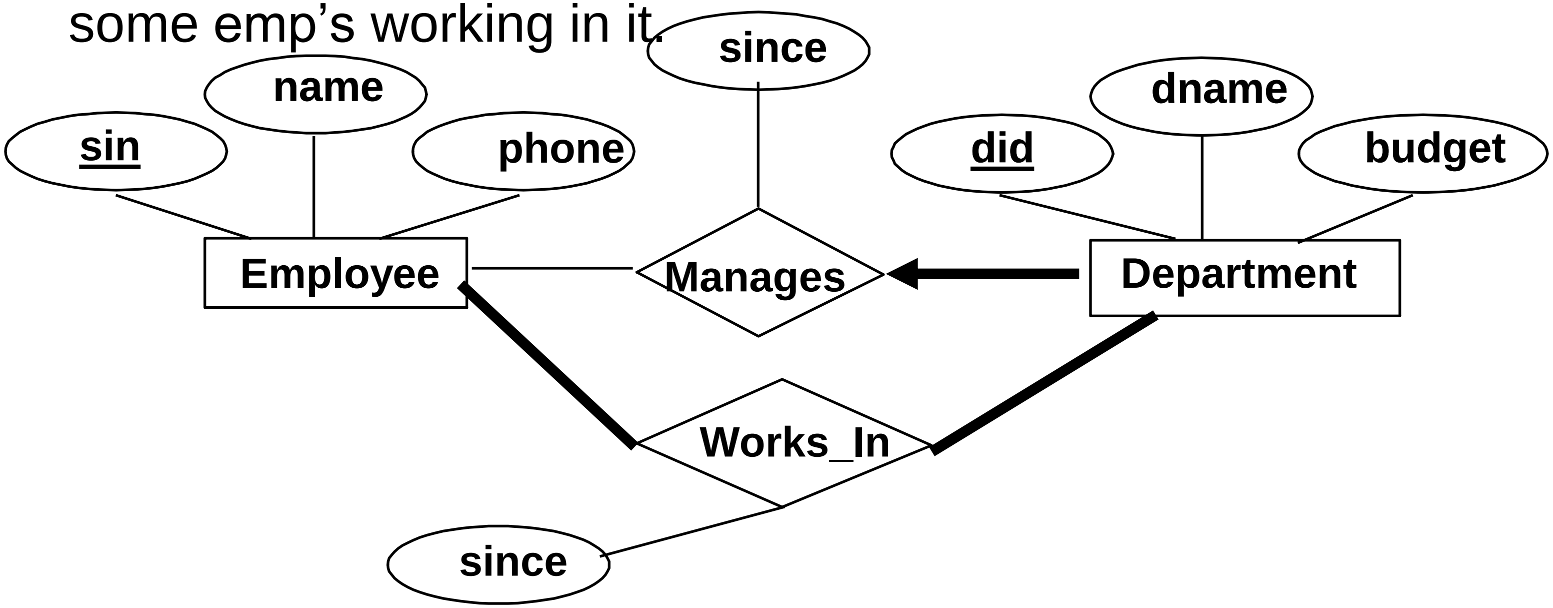
Participation Constraints

- Indicate if all entities must participate in the relationship.
- An entity set’s participation can be ***total*** (everyone participates) or ***partial*** (some may not participate)
- A total participation is a ***participation constraint*** and it is shown with a thick line in ER diagrams
  - **A DBMS must enforce these upon deletes.**
  - i.e. participation of Departments in Manages is total (thick line)
    - Every department must appear in some relationship in the Manages set



Participation Constraints

- Focusing only on thick lines (i.e., not arrows):
- $(\forall D, Dn, B)[Dept(D, Dn, B) \rightarrow (\exists Sin, S)Man(Sin, D, S)]$.
  - Every dept must have a manager.
- $(\forall D, Dn, B)[Dept(D, Dn, B) \rightarrow (\exists Sin, S)WI(Sin, D, S)]$ **and** $(\forall Sin, N, P)[Emp(Sin, N, P) \rightarrow (\exists D, S)WI(Sin, D, S)]$.
  - Every emp must work in some dept and every dept must have some emp's working in it.



Participation Constraints

- (Total) participation constraints induce a special kind of integrity constraints called *Tuple Generaing Dependencies* (TGDs).
  - See the red statements on previous slide.
- In general, Key constraints for relationships and for entities induce EGDs and participation constraints for relationships induce TGDs.
- EGDs and TGDs precisely tell us the meaning of these constraints, so a DBMs knows how to enforce them upon updates.
- ~~We'll revisit EGDs and TGDs when discussing data integration (@ end of course).~~

Primary Keys of Relationship Sets

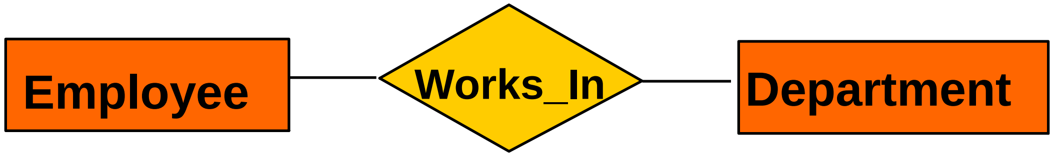
- Let R be a relationship set between entity sets A and B. R's primary key is:

TYPE OF R	PRIMARY KEY OF R
one-to-one	primary key of A **or** primary key of B
one-to-many from A to B	primary key of B
many-to-many	primary key of A + primary key of B

- R may have its own attributes, in addition to the key(s) that it inherits from the entities.

Line Types Review

- Thin lines mean many to many and partial participation:

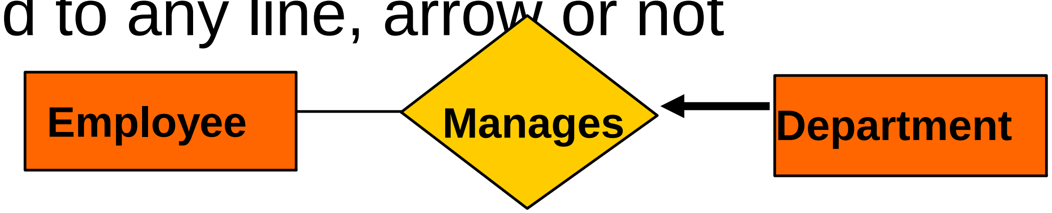


(many to many)

- Arrows mean the arrow side has a cardinality of one

(one to many)

- Thick line requires total participation and thickness can be added to any line, arrow or not



(one to many)

4, … 8

So what kind of constraints have we seen so far?

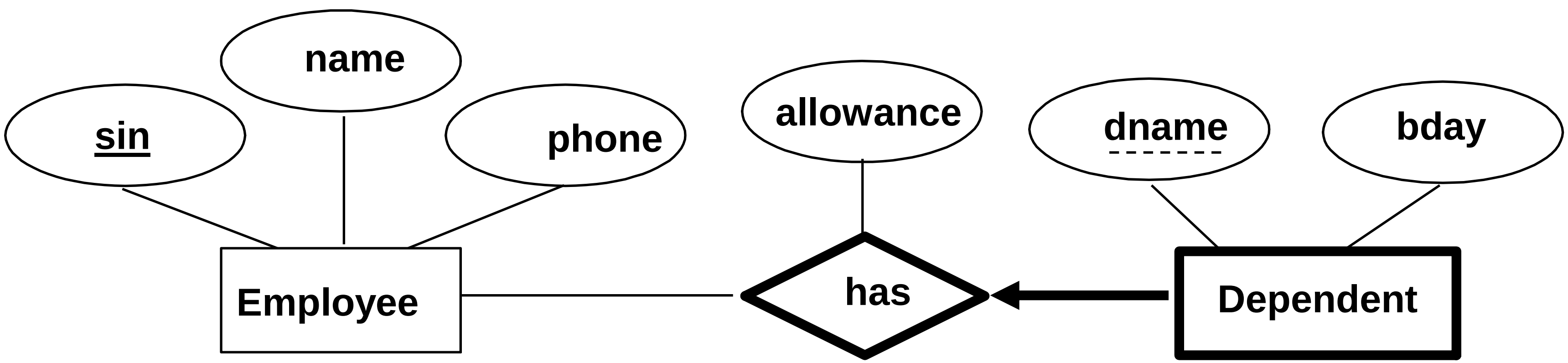
- Key constraints:
  - apply to both entities and relationships.
  - Are a special case of functional dependencies.
  - Which are a special case of EGDs.
- Cardinality constraints:
  - Apply to relationships.
  - *-to-one and one-to-* correspond to EGDs.
- Participation constraints:
  - Apply to relationships.
  - Only total participation needs to be asserted (i.e., stated as a constraint).
  - Correspond to TGDs (tuple generating dependencies).

“Shapes” of constraints

- EGDs: Condition & Condition & … & Condition → X=Y.
- TGDs: Condition & Condition & … & Condition → $\exists X, Y, \ldots : relation(X, Y, \ldots)$.

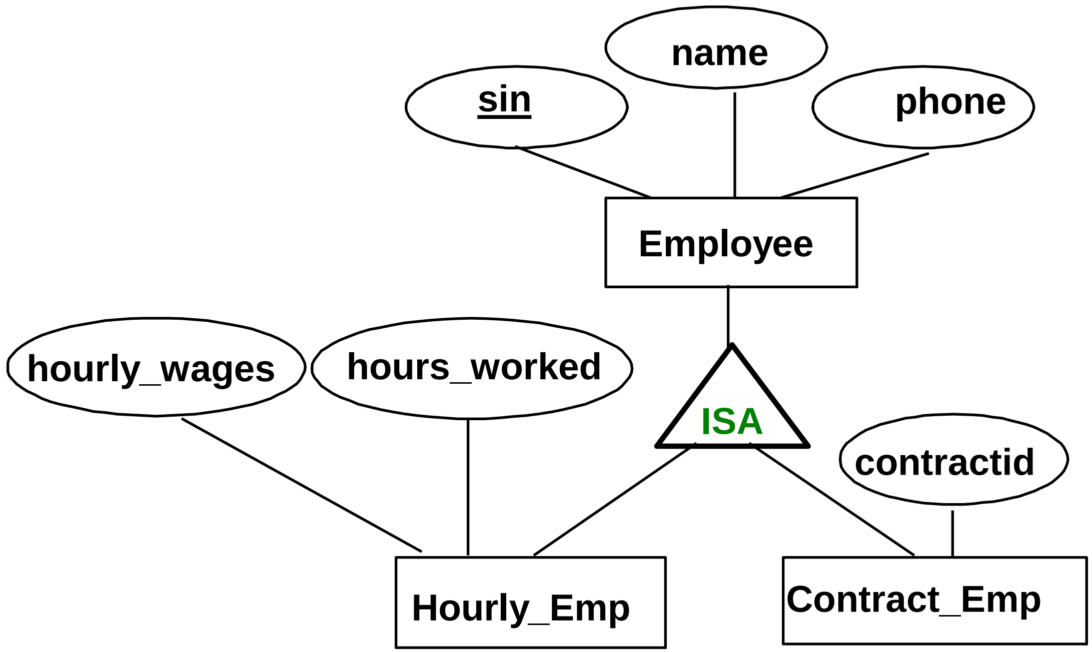
Weak Entities

- A ***weak entity*** can be identified uniquely only by considering the primary key of another (*owner*) entity. (Some books call it *dominant* 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 this ***identifying*** relationship set.
  - Think of this as a “belongs to” relationship.
- Weak entity sets and their identifying relationship sets are shown with thick lines.



Generalization/Specialization (ISA relationships)

- As in C++, or other PLs, attributes can be inherited.
- If we declare A ***ISA*** B, every A entity is also considered to be a B entity, i.e., A is a "subclass" of B.



Point to ponder:
What is the primary Key of H_E and of C_E?

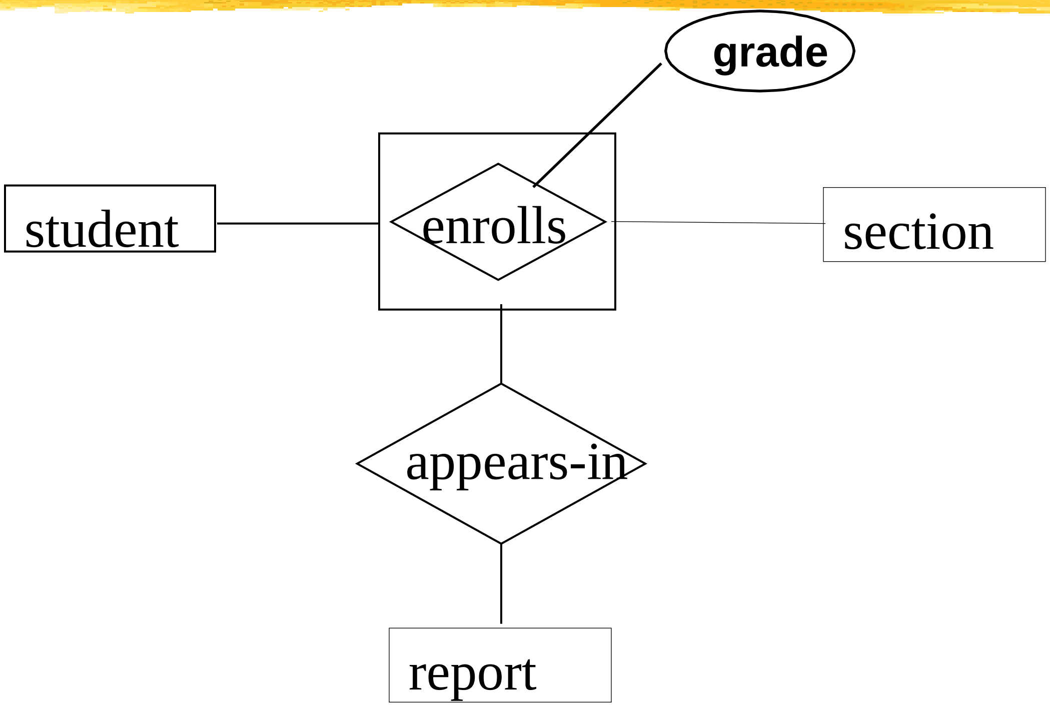
Specialization Types

- ***Overlap constraints***: Specializations can be:
  - ***Disjoint :*** a superclass entity may not belong to more than a single subclass
  - ***Overlapping :*** subclasses may overlap
- ***Covering constraints***: Specializations can be:
  - ***Total :*** a superclass entity must belong to some subclass
  - ***Partial :*** some superclass entity may not be in any subclass

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To restrict the kind of entities that participate in a relationship.
  - Adds to the richness (and accuracy) of modeling.

Aggregation

- Used when we have to model a relationship involving another *relationship set*.
  - ***Aggregation*** allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.



Aggregation vs. ternary relationship:

- *enrolls* is a distinct relationship, with a descriptive attribute.
- we can restrict each enrollment (and grade) to appear in at most one report, if we want

Conceptual Design Using the ER Model

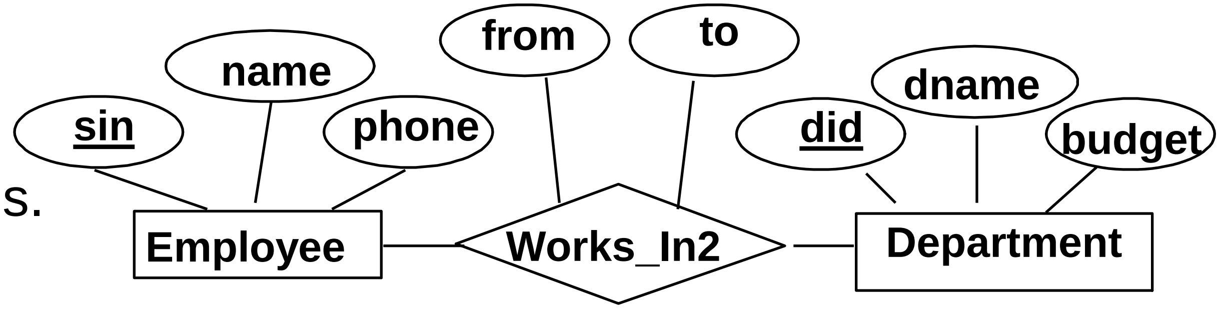
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured
  - But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

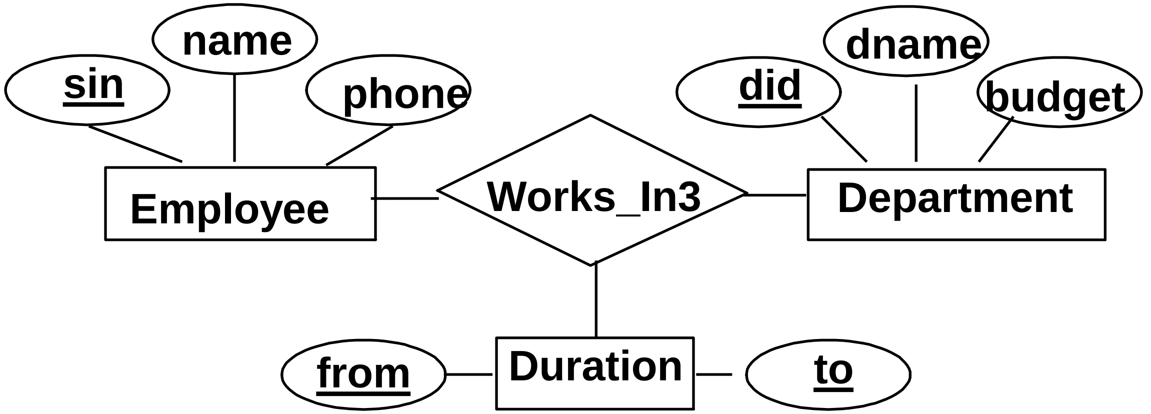
Entity vs. Attribute

- 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
  - the semantics of the data:
    - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
    - If an employee has only one street address, one city, one province, one postal code, etc. then these should simply be attributes.

Entity vs. Attribute (Cont.)

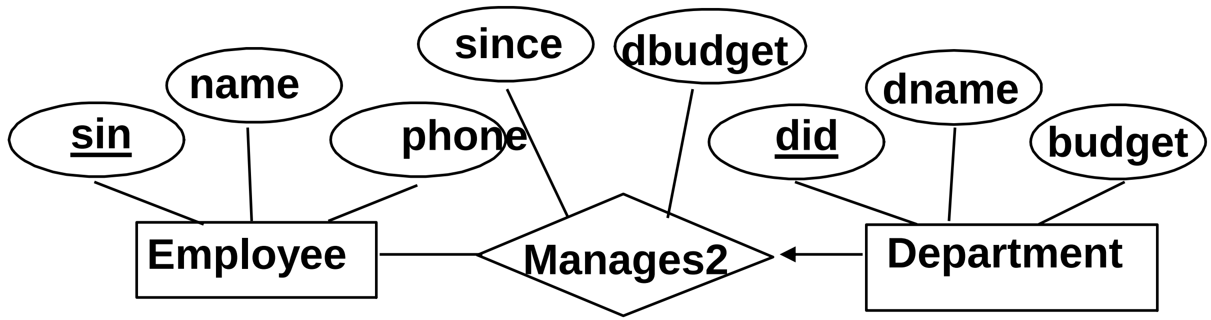
- Works_In2 does not allow an employee to work in a department for two or more periods.
- We want to associate the same pair (employee, department) with more than one set of values for the descriptive attributes.
- Solution: change descriptive attributes into entities.

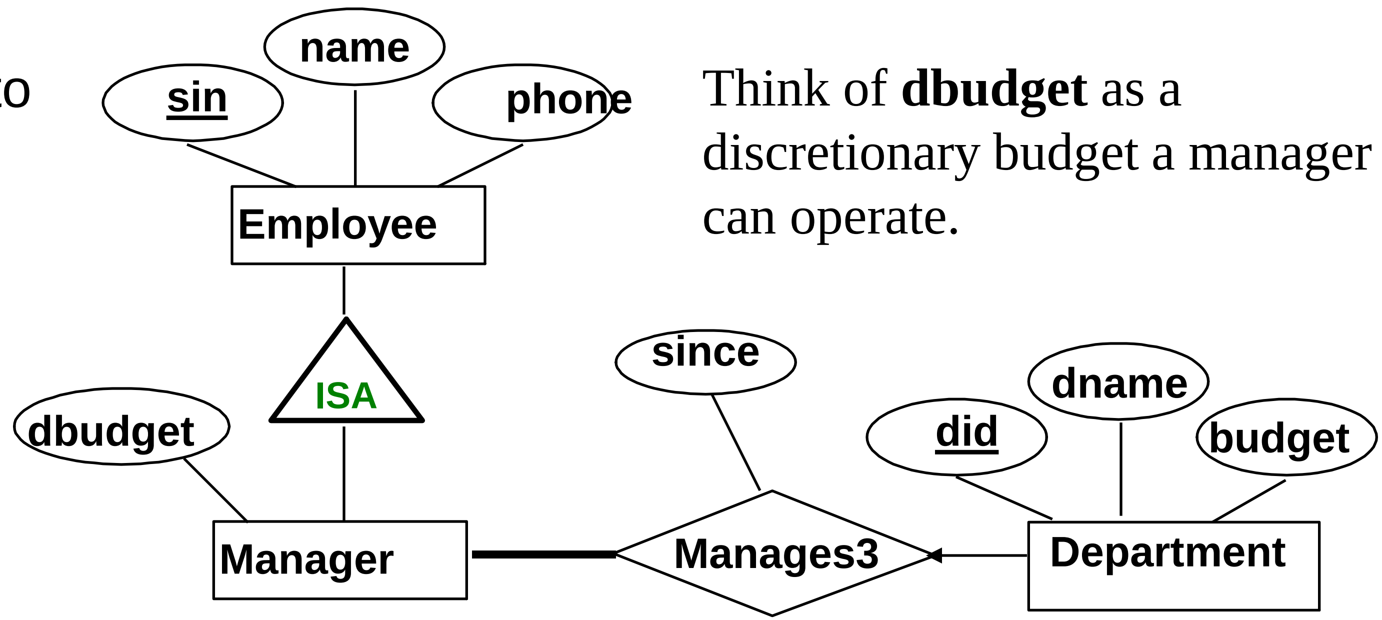




Entity vs. Relationship

- First ER diagram is OK if a manager gets a separate dbudget for each dept she manages.
- If a manager gets a dbudget that covers *all* managed depts (to be spent appropriately)?
  - redundancy
  - misleading
- Solution: Define a new entity for managers

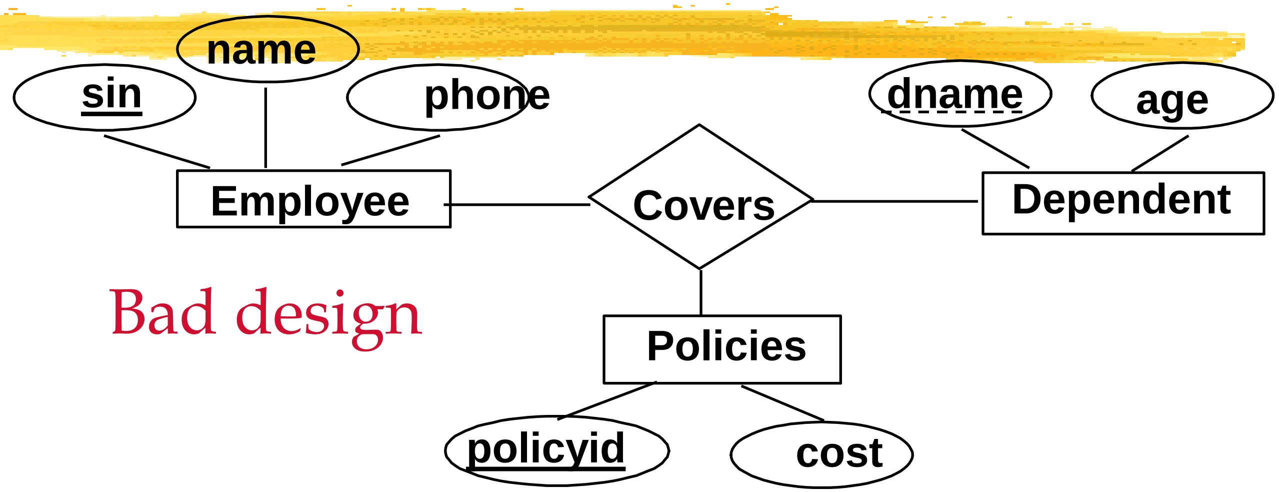


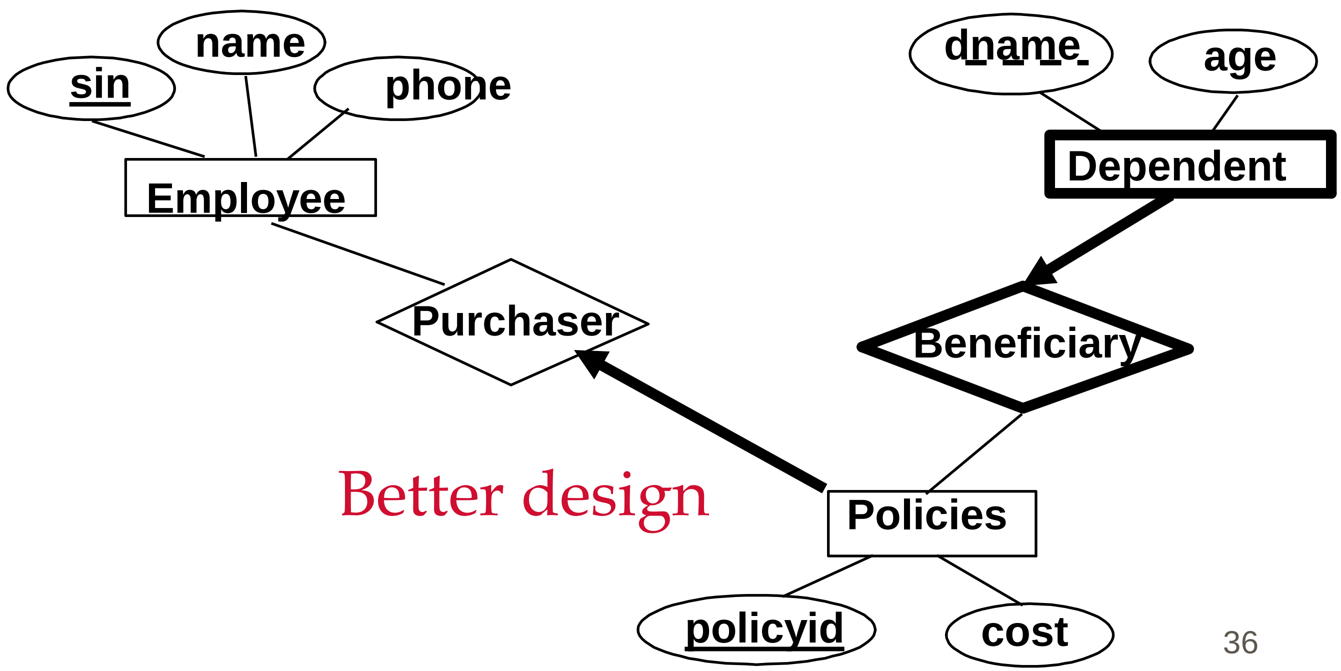


Think of **dbudget** as a discretionary budget a manager can operate.

Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee:
  - Key constraint on Policies would mean policy can only cover 1 dependent!
- What are the additional constraints in the 2nd diagram?





Binary vs. Ternary Relationships vs. Aggregation

- Previous example: two binary relationships were better than one ternary relationship.
- An example in the other direction: a ternary relation Drinks relates entity sets Person, Bar and Drink, and has descriptive attribute *date*. No combination of binary relationships is an adequate substitute:
  - P "likes" D, P "visits" B, and B "serves" D does not imply that P drinks D in B.
  - Also, how would we record *date*?
- Aggregation can be used instead of a ternary relation if we need to impose additional constraints:
  - i.e., a person cannot have more than one drink in the same bar.

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 relationships*, and *aggregation*.
- Note: There are many variations on ER model.

Summary of ER (Cont.)

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *participation constraints*, and *overlap/covering constraints* for ISA relationships. Some *foreign key constraints* are also implicit in the definition of a relationship set.
  - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
    - Functional dependencies (FDs) are more general than key constraints.
  - Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Cont.)

- 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
  - whether or not to use aggregation
- Ensuring good database design: resulting relational schema should be analyzed and refined further.
- Functional dependencies and normalization techniques are especially useful (see later).