Constraints in ER Models

CS 317, Fall 2007

Types of Constraints

- ***Keys*** are attributes or sets of attributes that uniquely identify an entity within its entity set.
- ***Single-value*** constraints require that a value be unique in certain contexts.
- ***Referential integrity*** constrains require that a value referred to actually exists in the database.
- ***Domain*** constraints specify what set of values an attribute can take.
- ***General*** constraints are arbitrary constraints that should hold in the database.

- Constraints are part of the schema of a database.

Keys

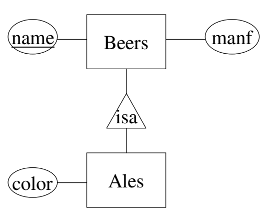
- A *key* is a set of attributes for one entity set such that no two entities in this set agree on all the attributes of the key.
  - In the case of multi attribute keys, it is allowed for two entities to agree on some, but not all, of the key attributes.

- We **must** designate a key for every entity set.

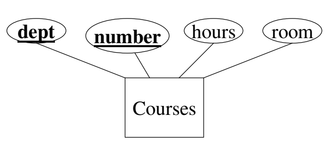
Keys in E/R Diagrams

- Underline the key attribute(s).
- An Entity set E can have multiple keys. We usually designate one as the *primary key*.
- In an *subclass* hierarchy, only the root entity set has a key, and it must serve as the key for ***all*** entities in the hierarchy.

Example: name is Key for Beers

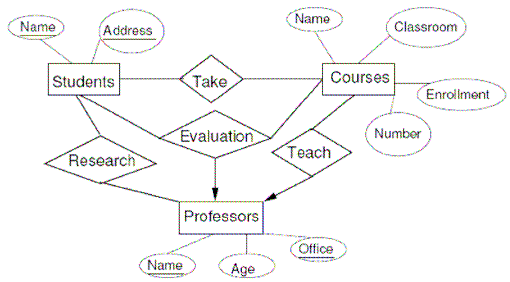


Example: a Multi-attribute Key



- Note that hours and room could also serve as a key, but we must select only one key.

Our Academic Example



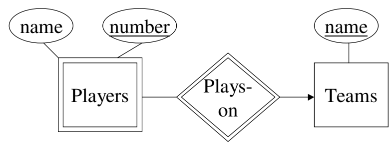
Weak Entity Sets

- Occasionally, entities of an entity set need "help" to identify them uniquely.
- Entity set *E* is said to be *weak* if in order to identify entities of *E* uniquely, we need to follow one or more *many-one* relationships from *E* and include the key of the related entities from the connected entity sets.

Example

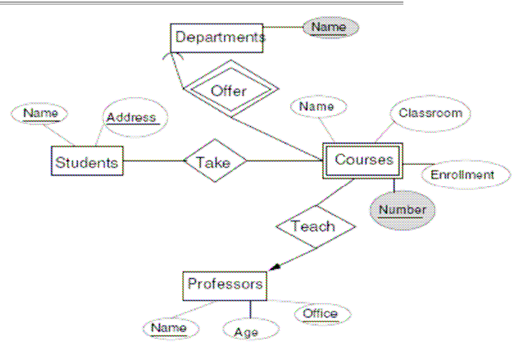
- name is almost a key for football players, but there might be two with the same name.
- number is certainly not a key, since players on two teams could have the same number.
- But number, together with the team name related to the player by Plays-on should be unique.

In E/R Diagrams



- Double diamond for *supporting many-one relationship*.
- Double rectangle for the weak entity set.

Another Example



Weak Entity-Set Rules

- A weak entity set has one or more many-one relationships to other (supporting) entity sets.
  - Not every many-one relationship from a weak entity set need be *supporting*.
- The key for a weak entity set is its own underlined attributes and the keys for the supporting entity sets.
  - E.g., (player) number and (team) name is a key for Players in the previous example.

Summarizing Weak Entity Sets

- If E is a weak entity set, its key consists of
  1. Zero or more of its own attributes and
  2. Key attributes from supporting relationships for E.
- A relationship R from a weak entity set E to F is supporting if
  1. R is a binary, many-one relationship from E to F,
  2. R has referential integrity from E to F.

How does F help E?

- F supplies its key attributes to define E’s key.
- If F is itself a weak entity set, some of its key attributes come to from entity sets to which F is connected by supporting relationships.

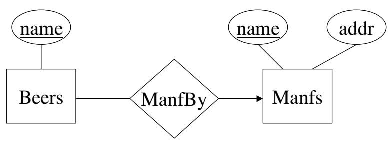
Design Tips for E/R Modeling

1. Avoid redundancy.
2. Limit the use of weak entity sets.
3. Don’t use an entity set when an attribute will do.
4. Confirm the correct cardinality and optionality of a relationship
5. ….

Avoiding Redundancy

- *Redundancy* occurs when we say the same thing in two or more different ways.
- Redundancy wastes space and (more importantly) encourages inconsistency.
  - The two instances of the same fact may become inconsistent if we change one and forget to change the other.

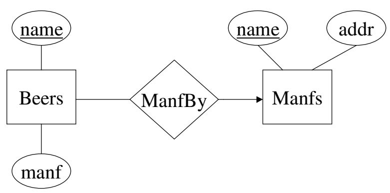
Example



Good or bad?

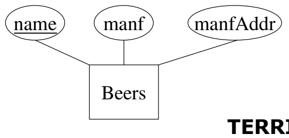
GOOD! This design gives the address of each manufacturer exactly once.

How about now?



Bad! This design states the manufacturer of a beer twice: as an attribute and as a related entity.

And now?



TERRIBLE DESIGN!

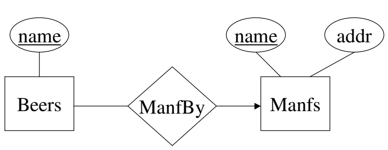
This design repeats the manufacturer's address once for each beer and loses the address if there are temporarily no beers for a manufacturer => TERRIBLE

Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
- It is more than the name of something; it has at least one non-key attribute.

  or
- It is the "many" in a many-one or many-many relationship.

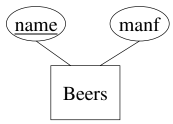
Example



•**GOOD!** Manfs deserves to be an entity set because of the non-key attribute addr.

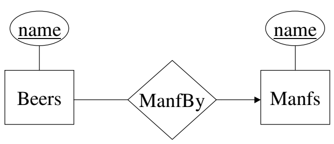
•Beers deserves to be an entity set because it is the "many" of the many-one relationship ManfBy.

Example



GOOD Design. There is no need to make the manufacturer an entity set, because we record nothing about manufacturers besides their name.

Example



BAD! Since the manufacturer is nothing but a name, and is not at the "many" end of any relationship, it should not be an entity set.

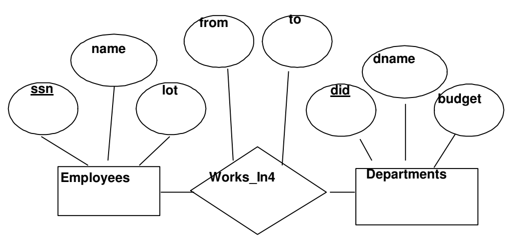
Entity vs. Attribute

- Should address be an attribute of Employees or an entity (connected to Employees by a relationship)?
- 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 - Warning

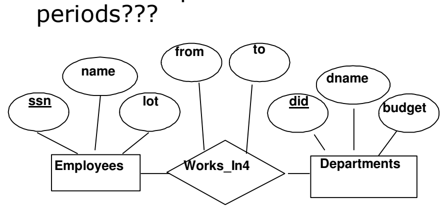
- *Do not introduce un-necessary entities (and complexity) if not needed for your application !*

Entity vs. Attribute



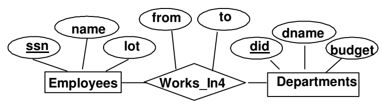
Entity vs. Attribute

- Does Works_In4 allow an employee to work in a department for two or more periods???



Entity vs. Attribute (Contd.)

- Works_In4 does not allow an employee to work in a department for two or more periods. If we are not allowed to use multi-valued attributes?

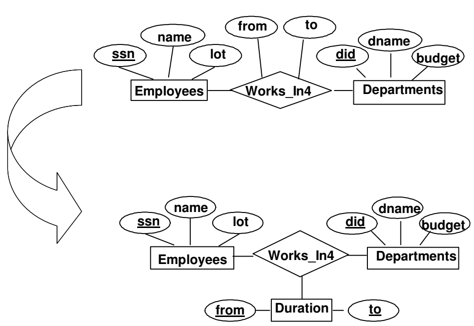


Entity vs. Attribute (Contd.)

- What do we do ?

- 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. Attribute



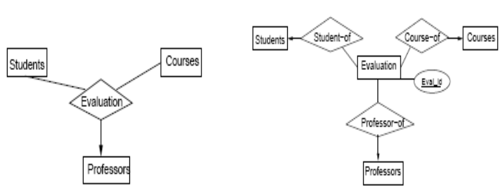
Don’t Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself.
  - They make all entity sets weak, supported by all other entity sets to which they are linked.
- In reality, we usually create unique ID’s for entity sets.
  - Examples include social-security numbers, automobile VIN’s etc.

When Do We <u>Need</u> Weak Entity Sets?

- The usual reason is that there is <u>no global authority</u> capable of creating unique ID’s.
- <u>Football Example</u>: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world.

Converting Multi-way to Binary

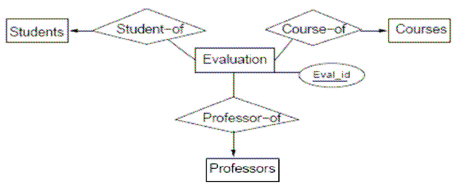


Details

•Create an entity in the new Evaluation entity set for each instance (row) in the ternary Evaluation relationship.

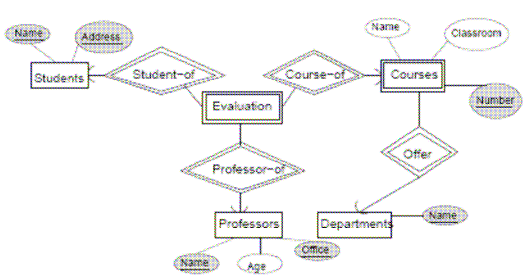
•In the Student of relationship, relate each entity in the Evaluation entity set with the corresponding student entity.

•The multiplicity of Student of is many-to-one from Evaluation to Student.



Without new attribute

- Can we perform the conversion without introducing a new, artificial attribute?
- *Evaluation* must be a weak entity set.
- What is/are the supporting relationships?
- All the relationships connected to Evaluation.
- The key for evaluation is composed of the key attributes of Students, Courses, and Professors.



Single Value Constraints

- There is at most one value in a given role.

1. Each attribute of an entity set has a single value.
   - If the value is missing, we can invent a "null" value.
   - E/R models cannot represent the requirement that an attribute cannot have a null value.
2. A many-one relationship implies a single value constraint.

Referential Integrity Constraint

- Asserts that a value must exist in a given context.
- Usually used in the context of relationships.
- Example: Many-one Advises relationship between Students and Professors.
  - Many-one requirement says that no student may have more than advising professor.
  - Referential integrity constraint says that each student must have exactly one advising professor and that professor must be present in the database.

More examples

- Each department has at most one chairperson who is its head (there are times when a department may not have a chairperson).

- Each chairperson can be the head of at most one department and this department must exist in the database.

Enforcing Referential Integrity

- We *forbid the deletion of a referenced entity* (e.g., a professor) until the professor advises no students.
- Conversely, we require that *if we delete a referenced entity, we delete all entities that reference it.*
- When we *insert* a student entity, we must specify an existing professor entity connected to the student by the Advises relationship.

Deriving Relationship Parameters

One course is studied by how many students? Answer = `zero or more'.

- This gives us the degree at the `student' end.
- The answer `zero or more' needs to be split into two parts.
- The `more' part means that the cardinality is `many'.
- The `zero' part means that the relationship is `optional'.
- If the answer was `one or more', then the relationship would be `mandatory'.

Contd.

One student studies how many courses? Answer = \`One'

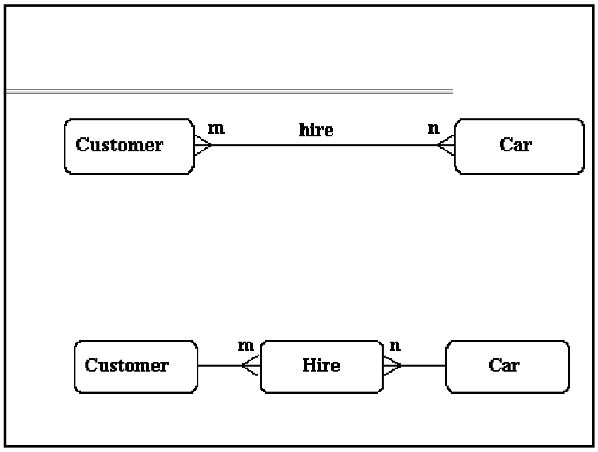
- This gives us the degree at the \`course' end of the relationship.
- The answer \`one' means that the cardinality of this relationship is 1, and is \`mandatory'
- If the answer had been \`zero or one', then the cardinality of the relationship would have been 1, and be \`optional'.

Redundant relationships

- Some ER diagrams end up with a relationship loop -check to see if it is possible to break the loop without losing info
- Given three entities A, B, C, where there are relations A-B, B-C, and C-A, check if it is possible to navigate between A and C via B. If it is possible, then A-C was a redundant relationship.

Splitting n:m Relationships

- A many to many relationship in an ER model is not necessarily incorrect.
- But they can be replaced using an intermediate entity. This should only be done where:
  - the m:n relationship hides an entity
  - the resulting ER diagram is easier to understand.


Customer
m
hire
n
Car
Customer
m
Hire
n
Car