February 21, 2019

# ER Diagrams

Why ER (Entity Relationship) Diagrams?

* Helps communicate across disciplines
* Helps visualize aspects of the schema we’re trying to design

Terminology:

**Entity**

* Abstract oject with attributes

**Entity Set**

* Permuations of all entities of certain types

**Attributes**

* Properties of an entity

**Relationships**

* Connections between two or more entities
  + Have **names**
    - Two basic types: **has a** and **is a** 
      * Person **has a** phone number
      * Student **is a** person
        + Inheritance/Polymorphism: some databases support this sort of thing

**Rectangles** become an **entity set**

**Attributes** are **ovals connected to a rectangle** with *no* **arrows**

* Rectangles can be used when there are a lot of attributes to hold them all with the entity title

**Relationships** are depicted as **diamonds**

**Lines** connect **two entities** to the **relationship** to form an ER Diagram

Primary keys are identified with an underline

Multiplicity of Relations

* One to one, one to many, many to many relations
* This is important because these drive our functional dependencies

We also have to decide if “one” means zero or one

**Singleton Relationship**

* Connection between two instances of the same entity

**Multiway Relationships**

* Relations that connect many entities
  + Sometimes these attributes can have attributes of its own

How do we represent inheritance?

Our base class can be a movie

* Subclass -> Cartoon(type): traditional, cgi, Claymation
* Subclass -> Murdermystery(weapon): knife, gun, rope

**Created with triangles on the arrow**

**Weak Entity Sets**

* Some types of entities are not “stand alone”
  + Their keys consist of attributes from another entity

**Example:**

*Model Number*

A model number is only *locally* unique for a manufacturer

* It is not *globally* unique

**Some attribute is only unique within the context of some parent**

Once we have a semi perfected diagram:

* Each regular entity will become a table
* Each relationship will also become a table *if*:
  + It is a many-to-many relationship
  + If it had attributes attached to it that don’t make sense associated within one of the entities being related

Can make 1:1 and 1:N relations less complex

* Make the 1 to many an attribute of the many
* Sometimes you don’t want to do this though
  + If one attribute is complex, you might not want to condense

We **can’t** just create a table out of that:

* Primary key from the parent plus the key of the weak entity becomes the attributes

**Is A relationships**:

Central Question: is an instance of a subtype indivisible, or is it actually two sets of attributes – the ones from the base and the ones from the subtype?

Generally… choose between three options:

1. Subclass is its own table, each row has a key identifying the row in the base table that it is associated with
2. No base class table, just subclass tables tables with the base attributes repeated
   1. THIS IS A CRUMMY SOLUTION (Dr. Frees’s least favorite)
   2. Makes basic queries really difficult
3. One table for the entire hierarchy, with nullable columns corresponding to all subtype properties

EXAM:

Insert, select, insert, delete, update, create, modify etc.

Normalization:

* Trying to get rid of redundancy
* No lists as attributes (1st normal form)
* Functional dependencies
  + AB -> CD
    - **Each time AB are found, they correspond to exactly CD**
      * (The same values in AB are the same in CD)
  + Closure
    - All of the things we can get from knowing certain attributes
  + BCNF
    - Left side of all FDs are keys