Lecture 5: Advanced Issues in Data Modeling

Database and SQL: Course Notes | Brendan Shea, Ph.D. (Brendan.Shea@rctc.edu)

1 CONTENTS

2 ENTITY TYPE HIERARCHIES AND "ISA"

In this lesson, we'll be answering the following questions:

1. What is the **enhanced entity-relationship model (EERM)?**
2. What is the **isA** relationship between **subtypes** and **supertypes**?
3. How do you draw and interpret **enhanced entity-relationship diagrams (EERDs)?**
4. What are the types of constraints that can be placed on parent entities?

In previous lectures, you learned about the Entity-Relationship (E-R) data model, and the closely related Entity-Relationship Diagrams (ERD). These models are instrumental in the early stages of database design, and can help facilitate communication between clients (or future database end-users), application programmers, database administrators, and others. In this lecture, we'll examine the **enhanced entity-relationship model (EERM)** and the related **enhanced entity-relationship diagrams (EERDs),** which provide ways to capture additional "semantic content" (that is, they help us capture more about what is *meaningful* about our data).

We'll specifically focus on how the EERM allows us to capture and describe "specialization hierarchies" and "inheritance" relationships between different entity types. This is closely related to both the object-oriented data model (for databases) and the object-oriented programming paradigm that underlines languages like Java or C++ (so, some of you may have seen similar ideas before!).

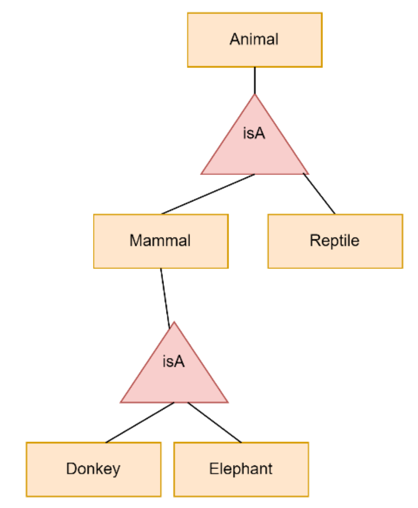
2.1 INTRODUCING ISA AND SPECIALIZATION HIERARCHIES

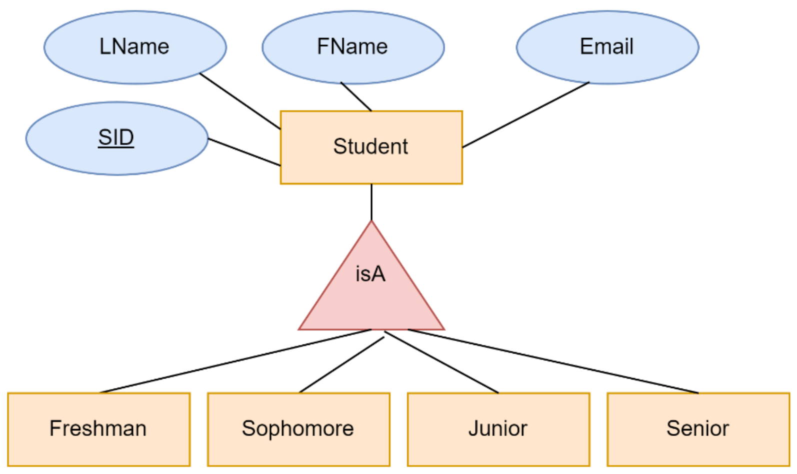
What is isA? One central addition of the EERM is the explicit inclusion of the **isA** relationship, which allows us to capture the logical connections between **subtypes**("children") and **supertypes** ("parents")**.** The isA relationship has a few essential characteristics:

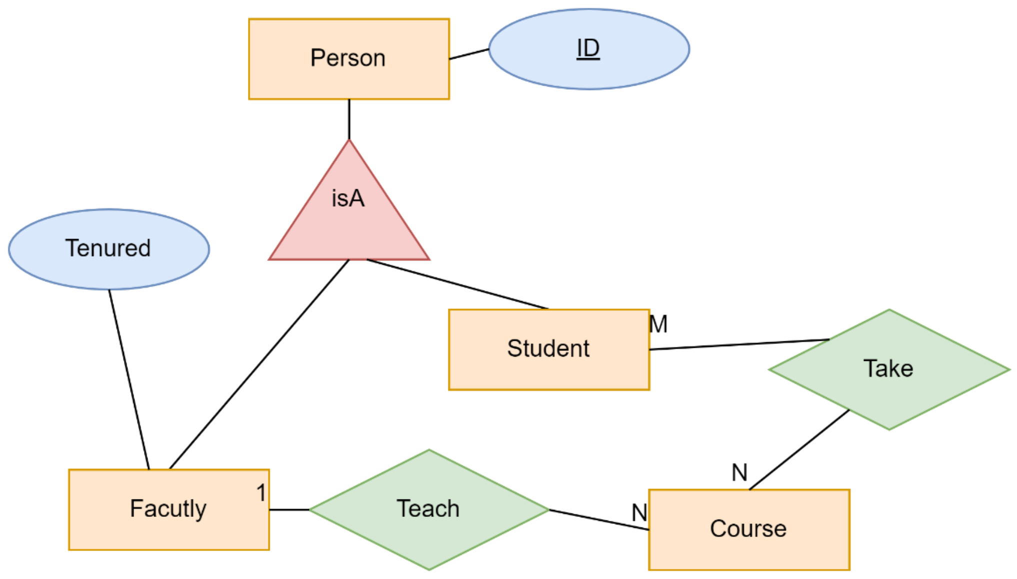
1. It has the form "subtype isA supertype." For example, "Elephant isA Mammal" or "Mammal isA Animal." As you might guess, isA is **transitive,** meaning that entities not only inherit properties from the direct "parents" but from the parents of these parents. So, we can infer that "Elephant isA Animal."
2. A subtype **inherits** all of the attributes and relationships of the supertype. So, for example, if we define "Freshmen" and "Sophomores" to be the subtypes of the more general class "Student", and all students have the attribute "Email address" and a relationship with a "Guardian" then Freshmen and Sophomores will have these attributes/relationships as well.
    a. Subtypes ALWAYS inherit their primary key from their supertype. So, if Student has a primary key of StudentID (or something of the type), this will also be the primary key for any subtypes of students.
    b. At the implementation level, subtypes have a "one-to-one" relationship with their parent entity. This means, for example, that any given row in the supertype table (for example, we have an entry for a Person named "Mario") should correspond to at most *one* row in a subtype table (for example, in the Contractor table, which is a subtype of Parent, Mario should appear at most once, if he is in fact a contractor).
3. The parent entity will have a **subtype discriminator** identifying the subtype of instance. So, for example, if our supertype is Animal, it may have an attribute named Species. The *value* of this attribute will then serve to identify the relevant subtype. That is, if Species = "Elephant," this indicates that this is of the subtype Elephant.

4. A subtype may ALSO have attributes and relationships that are unique to it (and are not inherited from its parent). So, for example, if a university database defines "Faculty" as a subtype of "People" it may nevertheless be that Faculty have unique attributes or relationships (for example, they are in a "Teach" relationship with Courses, and have a "tenured" attribute).

Example of EERM Diagrams (Chen Style)







2.2 Specialization Hierarchies: Why and How?

A **specialization hierarchy** is a group of related supertypes and subtypes (such as those illustrated above). We create such hierarchies for the following reasons:

1. "Supertypes" allow us to avoid duplicating information, including metadata. So, for example, if we want to store the first and last names for all Persons, it makes sense to have some sort of common database table, rather than recreating these exact same attributes for each and every "type" of Person.

2. "Subtypes" allow us to avoid the **proliferation of nulls** that would occur if some attributes of Person (or whatever the supertype happens) to be are only relevant to certain subtypes. So, for example, it would be wasteful to create *an entirely new entity type* for Faculty (they are just People, after all!). However, Faculty have at least *some* attributes or relationships (such as Teaches or Tenured) which we don't want to encode for each Person, as these will be irrelevant to most people (so, they would be "null" for every person that wasn't a faculty member).

In keeping with this, we should only create subtypes when we have good reason to: namely, that two or more entities share *some but not all* of the relevant attributes or relationships we'd like to capture in our data model. We can then model the shared attributes or relationships as belonging to the supertype, and the unique attributes/relationships as belonging to the subtype.

Entity Clusters. In the design phase (before implementing the database on a particular DBMS), database designers may also include **entity clusters—**abstract/virtual types meant to "stand in" for many subtypes. The entity cluster won't be entered in the implemented database (just the subtypes) will, but it can make the modeling/communication process much easier.

Building Hierarchies: Generalization vs. Specification. To build a specialization hierarchy, one can adopt several methods:

- **Generalization** proceeds from the "lowest," most specific entity types and then generalizes on these by identifying the *shared attributes and relationships* among these entity types. So, for example, you might "generalize" by creating the entity types for Student and Professor and then generalize to a Person supertype that includes their shared attributes/relationships (such as email address, first name, last name, etc.).
- **Specification** proceeds from the "highest," most general entity types and becomes more specific by determining which attributes/relationships are *unique* to different subtypes. So, we might BEGIN with a Person entity, and then create the subtypes Student and Professor by identifying how these differ from one another (e.g., Professor have Salaries, Students have GradePointAvg). Thi process is essentially the inverse of generalization.

In the specialization hierarchy, a **subtype discriminator** is the attribute of the supertype entity that determines the subtype of lower-level entities.

2.3 CONSTRAINTS: OR, AND, & (IN)COMPLETENESS

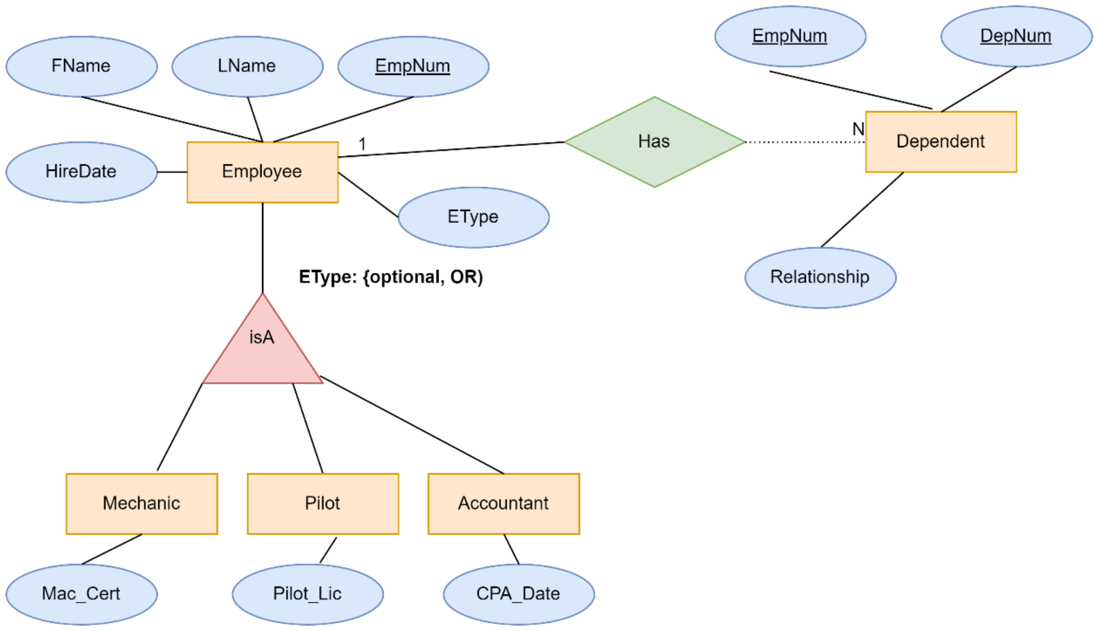
We said earlier that a *subtype discriminator* is an attribute of the supertype that serves to identify which subtype, if any, this entity is. As it turns out, depending on the *constraints* we place on this subtype discriminator, we can more fully capture the "nature" of the relationship between the supertype and subtype. The most important distinctions are as follows:

Disjoint (OR) vs Overlapping (AND) Subtypes. We say that a subtype discriminator is disjoint (or "OR") if each entity can only belong to ONE subtype. For example, each Animal can belong to only ONE Species. It is either a Lion or an Elephant but not both! By contrast, an Overlapping (AND) subtype discriminator allows for multiple subtypes. For example, a single Person might have various types of Jobs (the subtype discriminator).

Partial completeness ("Optional") vs. Total completeness ("Mandatory") Subtypes. Some supertypes require that an entity have an identified subtype, which cannot be left NULL (for example, you can't have an

Animal with no Species!). By contrast, some supertypes do NOT require an entry for subtype. For example, you can be a Person without having a Job.

Here is an example of an EERD showing a "partially complete, disjoint" supertype-subtype relationship. In this case, the parent entity is Employee, and the attribute that serves as a subtype discriminator is EType.



2.4 Review Questions

1. In your words, describe the following concepts: (a) supertypes, (b) subtypes, (c) isA relationship, (d) specialization hierarchy, (e) generalization/specification, (f) inheritance, (g) subtype discriminator.
2. Draw the EERD for a specialization hierarchy with at least ONE supertype and TWO subtypes. You should also include:
    a. At least two attributes for the supertype, including the subtype discriminator.
    b. At least one attribute per subtype which are DIFFERENT from either the supertype (and from each other).
3. Give an original example of a supertype/subtype combination with the following characters:
    a. Overlapping/Complete (must have at least one subtype, but can have multiple)
    b. Disjoint/Complete (must have precisely one subtype)
    c. Overlapping/Partial (may or may not have a subtype, can have many subtypes)
    d. Disjoint/Partial (may or may not have a subtype, but if it does, it has only one).
4. Draw the EERDs for your answers for 3. Include relevant attributes and relationships.

3 The keys to Good Keys

In this lesson, we'll be answering the following questions:

1. What makes for a good primary key? What are the different types of primary keys?
2. How can you choose an appropriate primary key for a given entity?
3. How do you model 1:1 relationships? What about "time-variant" data?

4. What is a **fan trap,** and how can you avoid them?

3.1 Picking Primary Keys: Some Rules of Thumb

In previous lessons, we've said that a *primary key* is an attribute (or set of attributes) that fully determines all of the entity's other attributes. If you know the primary key, you can get all the other information we know about the entity. More technically, we said that a primary key is a "minimal superkey", in that it (1) does not contain any "unnecessary" attributes other than those it *needs* and (2) uniquely determines all of the other attributes in the relation.

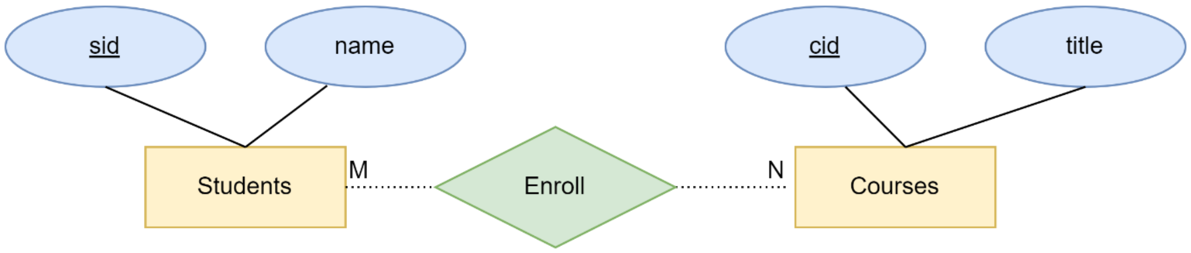
Part of the beauty of a relational DBMS is that it will not ALLOW you to create tables that lack primary keys or insert multiple entities with the same primary key. Yay! That being said, not all "legal" choices of primary keys are good ones. With this in mind, here are a few rules of thumb that must be followed:

1. **Be Something Unique.** Ok, so this one isn't really a "rule of thumb", but a requirement of the relational model. Primary keys are not allowed to be NULL (they must be "something"), and you can't have two entities with the same primary key ("unique").
2. **Live Naturally (When You Can).** If the entity type you are modeling has a **"natural key"** that is used to identify it in the real world, you can SOMETIMES use this as a primary key. For example, if you are creating a database for a company where employees have already been assigned employee id numbers, these could also make great primary keys.
    - You should only use natural keys as primary keys if doing so doesn't break any of the other rules. If there isn't a natural key (or if the natural key won't work), you'll need to create an artificial **surrogate key** to use as a primary key. For example, we might *create* an attribute "id_number" even if no such thing exists right now (and even if it has no meaning/use outside of the database itself).
3. **You Mean Nothing to Me, and This Will Never Change.** While it may seem odd, we generally do NOT want primary keys that have any "meaning" (or **"semantic content"**) in the real world. So, for example, things such as email addresses, full legal names, date of birth, etc. would generally NOT be good primary keys, even if we were somehow assured they were unique. The reason is because data that has meaning might *change*. You might change your email address or name or even (more rarely) discover you were born on a different day than you initially thought.
4. **Numbers (and GUIDs) are the Rule.** Where possible, it is generally good practice to use NUMBERS (and more specifically, integers) as primary keys, as these can be assigned "serially" (in numerical order: 0, 1, 2, …) by the RDBMS and can be easily indexed. In distributed NoSQL/JSON databases, you'll sometimes have to use **Globally Unique IDs (GUIDs)** instead, which are collections of characters and numbers. Both of these options are much better than using long blocks of text, which waste space and, more importantly, slow down the operation of the database.
5. **Keep it "Simple" Stupid.** Generally, it is best to use as *few attributes as possible* for the primary key. Ideally, we would like to use a **simple key** of exactly ONE attribute. For example, using the single attribute "id_number" as a primary key should be preferred to a combination of 2+ attributes.
6. **Security: It Shouldn't Be a Big Deal if You Know.** Finally, you should NOT choose an attribute that needs to remain private as a primary key, as primary keys are easily accessible to many database users. For example, using a social security number is generally a bad idea.

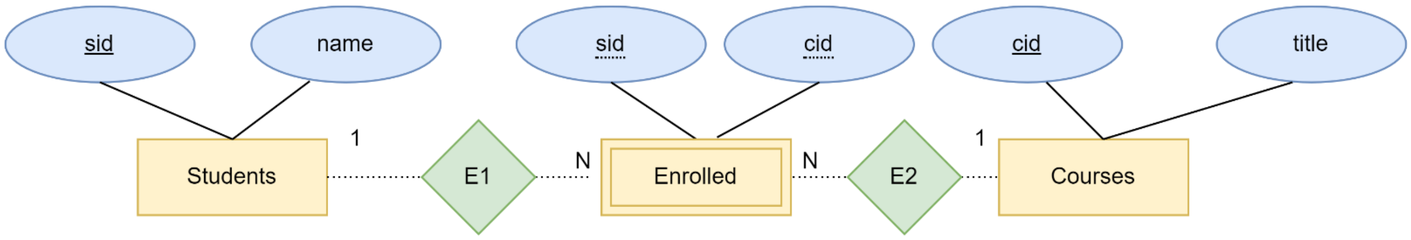
3.2 Composite Keys

According to the rules laid out in the previous section, we should generally prefer simple keys. However, in some cases, it can make sense to use **composite keys,** which are made up of two or more attributes. In particular, composite keys should often be used for the following sorts of entities:

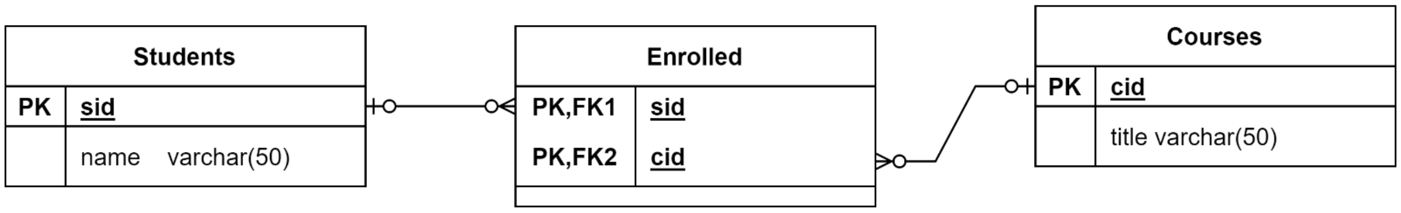
Case 1: "Bridge" or "Linking" Entities. As we've previously discussed, the relational model can't "natively" many-to-many (M:N) relationships, such as those between Student and Course (where many students may register in a particular course AND a given student may register in many courses). In this case, we create a linking entity Enrolled, with a composite primary of key made up of the two *foreign keys* that correspond to the primary keys of the entities on each "side." So, for example, consider the following E-R diagram, which contains an optional M:N relationship:



In order to make this relational-model friendly, we'll need to "break up" this M:N relationship into two distinct 1:M relationships. This, in turn, requires that we model the relationship "Enroll" as an entity. In particular, it will be a weak entity, whose existence depends on Students and Courses. Here E1 and E2 are simply generic names for the relationships that Students and Courses have with this "new" entity:

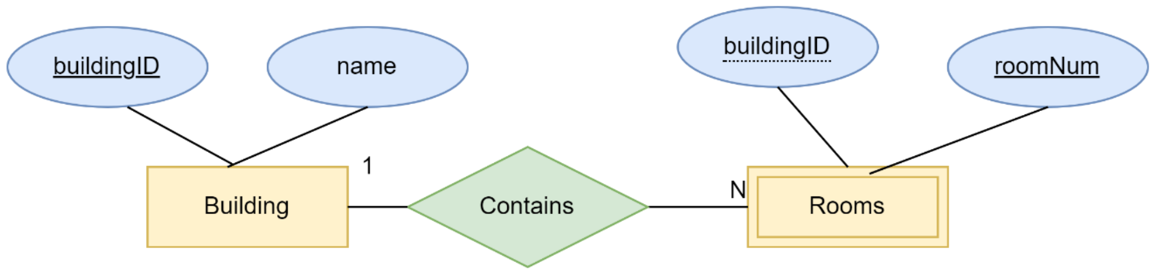


Notice that Enrolled here has a **composite key** that is built out of two foreign keys. Here's a modified crow's foot showing the full **relational schema:**



Case 2: Any weak entity. This is a more general version of case 1. As you'll recall, a weak entity is (by definition) an entity whose existence depends on some other entity (to which it has a "strong" relationship) AND it derives part/all of its primary key from this entity (or entities). In the previous example, Enrolled provides an example of this. So, for instance, in the following diagram, the primary key for Room (a weak

entity) is defined, in part, by the foreign key from the Building entity. Notice that roomNum *would not be an adequate primary key on its own.* After all, there may be rooms with the same number in different buildings!



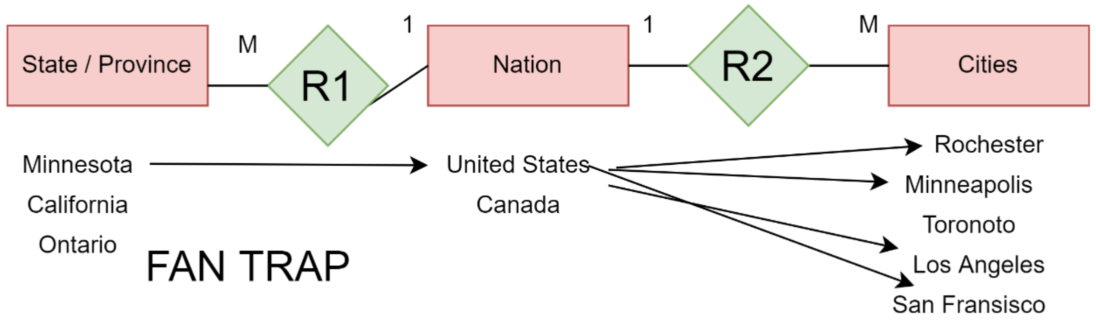
3.3 Design Patterns: Dos and Don'ts

Data modeling is somewhere between an art and a science, making it impossible to specify rules that apply to every case. That being said, there are some common "patterns" in real-world data that occur pretty frequently. Learning to recognize these patterns, and their impact on modeling, can save you quite a bit of work:

(Reminder) Many-to-many relationships need to be broken up. There are many, many examples of many-to-many relationships in the real world. However, the relational model can't handle these. So, if A and B are in M:N relationship, we "model" this by creating a new "bridge" entity E to capture the relationship between A and B. Both A and B will be in a 1:M relationship with E. In general, the primary key for E will just be a combination of the keys for A and B (see Enrolled, above).

The "Middle Term" and the Fan Trap. It's important to remember that many perfectly "consistent" data models (which satisfy the "rules" of the relational model) can fail to capture those aspects of the real world that they are intended to be "about." These might be called **semantic errors** (in that they concern the "meaning" of the data) as opposed to **syntactic errors** (the sort that produces error messages from the DBMS). One way that this can occur is a so-called **"fan trap,"** which occurs when entities are "linked" together (such as Students, Enrolled, and Courses), but the "middle" term is chosen incorrectly during data modeling. This means the model has failed to capture something about the real-world relationship and can cause seemingly reasonable queries to "fan out" and produce numerous, nonsensical results:

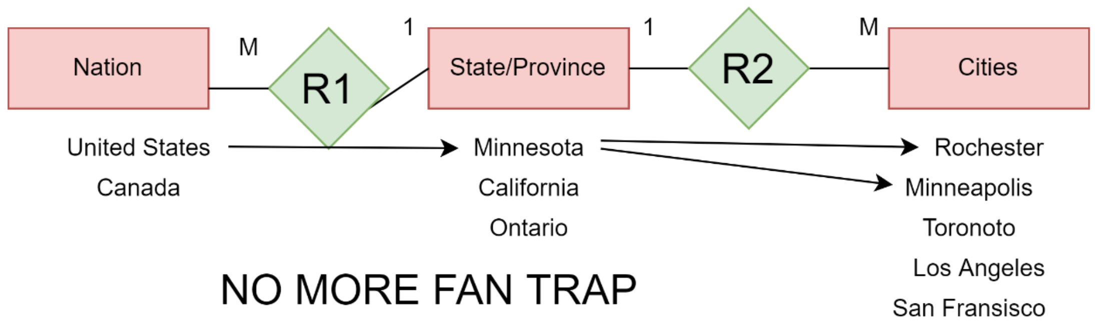
So, for example, suppose that we model the relationship between Cities, States, and Nations in terms of the business rules that "states are in nations AND cities are in nations." When we try to use relational algebra (or SQL) to ask questions like "Tell me which cities are in Minnesota" we INCORRECTLY are told that this includes every city in the US!



Fixing Fan Traps. We can fix fan traps in one of two ways. We can either

(1) recognize that they exist within our data model (after all, every model has gaps!), and carefully avoid asking the sorts of questions that cause problems or
(2) fix our data model to more accurately capture the relationships in question (this is often the better choice).

In the case above, this means recognizing we need to model the (better) business rules that "Cities are in states AND states are in nations.". The problem is solved!

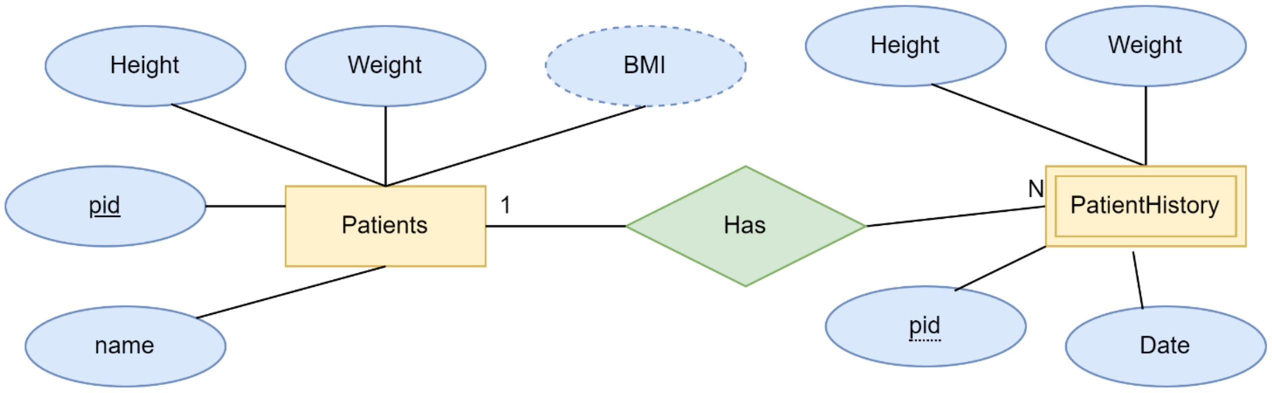


One-to-one relationships (sometimes) should be modeled as two distinct entities. As a rule of thumb, if entities A and B are in a one-to-one relationship (for example, a citizen and their social security number), our model only needs to contain a table for ONE of these entities, while the other can be modeled as an attribute (Person=table; social security number = attribute). In some cases, however, we have good reasons for wanting to keep the entities "logically" separate. For example, a person employed at a medical facility might also be a patient at that same facility (that is, there is a one-to-one relationship: it is the exact same person!). However, there are many good reasons for keeping our Employee and Patient data separately. We can, of course, keep track of links between these (for example, with an optional foreign key in Patient linking to the primary key of Employee).

- **Other examples:** A subtype-supertype relationship (such as Student: Person) is, by definition, a 1:1 relationship (the same person is both a Student and a Person). In many cases, our relational model will end up storing data about the same person in two separate tables. Here, since a Student MUST be a Person (while a Person need not be a Student), we'd want to store a foreign key reference to the corresponding Person entity within the Student table (and not vice versa).

3.4 IT'S TIME(1), TIME(2), TIME(3)

Many real-world problems require that we keep track of how some data set changes over time. For example, in a medical database we want to know not just what a patient's weight and height and height are *right now,* but how this has changed over time. Such **time-variant data** can generally be modeled along the lines of a 1:M relationship between the entity and its history. The weak "historical" entity will nearly always have a composite primary key made up of the primary key of the original entity plus the DATE. So, for example:



When implementing this in a relational database, we could simply collapse the Patients table into the PatientHistory table (since there's no need to store the most recent height and weight *twice*). In other cases, storing "current" values separately can make sense, especially if they represent the result of a calculation (such as BMI) that users will frequently access.

3.5 Review Questions

1. In your own words, describe THREE different criteria for choosing a good primary key. Now, give an original example of a primary key choice that would violate each of these criteria.
2. Give an example of a many-to-many relationship between two entities from the real world (other than those we've discussed above). Now:
    a. Describe some attributes of each of the two entities, and say what you would use as the primary key, and why.
    b. Draw an ER diagram for this, using the M:N relationship.
    c. Draw a new ER diagram, with the M:N relationship replaced by two 1:M relationships with a "bridge" entity.
    d. Identify the primary keys for the bridge entity.
    e. Explain why/how your model does NOT commit a fan trap.
3. Give three examples of areas where you might want to keep time-variant data.

4 Why can't you be Normal? Database Normal Forms[1]

Database normalization is the process of structuring and refining the data we want to store in such a way that we eliminate repeated information and represent as much connection between records as possible. It has its roots in the work of Edgar Codd (the creator of relational databases). When a database meets particular rules or features of normalization, it is usually referred to as being in a particular normal form. The collection of rules progress from the least restrictive (first normal, or 1NF) through the most restrictive (fifth normal, or 5NF) and beyond. Databases can still be useful and efficient at any level depending on their use, and anything beyond third normal form is more of a rarity in real-world practice.

Bear in mind, that normalization is a theory on data organization, not law. Your database can operate just fine without adhering to the following steps, but following the process of normalizing will make your life easier

[1] This is adapted, with significant modifications from: M. Mendez, 2021, *The Missing Link - An Introduction to Web Development.* Open Suny Publishing.

and improve the efficiency of your website. These rules are designed to help you eliminate repeated data, keep your overall database size as small as possible, and ensure data integrity. Not every set of circumstances will require all of these rules to be followed. This is especially true if they will make accessing your data more difficult for your particular application.

4.1 AN OVERVIEW OF THE PROCESS

The normalization process involves carefully thinking about each table's candidate keys (or prime attributes) and the *relationships* that non-key attributes (or **non-prime attributes)** bear to them.

Here's the process. To simplify, I've assumed there is only ONE candidate key (which may be simple or composite)

1. **Zero-Normal Form (0NF)** is our "starting" point of unnormalized data. There is often *no* candidate key, and some "cells" may contain many values. The lack of a key means the data can't be entered into a relational database.
2. **First Normal Form (1NF)** requires that we (1) create tables for related information so that we have no **repeating groups** (which occur when we have *more than one item in a single table cell*) and (2) assign an appropriate primary key for each table.
    a. Data in 1NF *can* be entered into a relational database (unlike 0NF).
    b. However, we still might have lots of data redundancy if some attributes are "determined" by attributes *besides* those we've chosen as the primary key.
3. **Second Normal Form (2NF)** requires that we (1) satisfy 1NF and (2) eliminate all **partial dependencies** (where attributes depend on only *part of* a primary key). We do this by moving the partially dependent data (and the attribute it depends on) to a new table, which we link to existing tables using a foreign key.
    a. Converting to 2NF reduces redundancy, since we no longer need to enter data of the form "A, B, and C" when we *know* that the values of B and C depend on A. (We can simply enter "A" instead!).
4. **Third Normal Form (3NF)** requires that we (1) meet second normal form and (2) move/eliminate attributes that depend on any attributes *besides* the primary key (called **transitive dependencies).**
    a. Similar to 2NF, this reduces redundancy. For most business/scientific purposes, we can stop here.
5. Higher levels of normalization include **Boyce-Codd Normal Form,** which requires that all "determinants" (any attribute that determines the value of another) be part of a candidate key. Finally, **Fourth Normal Form (4NF)** requires that we (1) meet third normal form and (2) have no non-trivial **multi-valued dependencies.** A multi-valued dependency occurs when the key fully determines independent attributes Y and Z, and both Y and Z can take many values.

4.2 NORMALIZING AN EXAMPLE TABLE[2]

These steps demonstrate the process of normalizing a fictitious student table.

4.2.1 Unnormalized table (Zero-Normal Form)

Student_id	**Advisor**	**Adv-Room**	**Class1**	**Class2**	**Class3**

[2] Adapted, with modifications, from https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description.

1022	Rincewind	412	101-07	143-01	159-02
4123	Weatherwax	216	101-07	143-01	179-04

4.2.2 First normal form: No repeating groups

Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, and Class3 in the above records are indications of design trouble.

Spreadsheets often use the third dimension, but tables should not. Another way to look at this problem is with a one-to-many relationship, do not put the one side and the many side in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class_id), as shown below:

FLAT FILE:

Student_id	**Advisor**	**Adv-Room**	**Class_id**
1022	Rincewind	412	101-07
1022	Rincewind	412	143-01
1022	Rincewind	412	159-02
4123	Weatherwax	216	101-07
4123	Weatherwax	216	143-01
4123	Weatherwax	216	179-04

4.2.3 Second normal form: Eliminate redundant data

Note the multiple Class_id values for each Student_id value in the above table. Class_id is not functionally dependent on Student_id (primary key), so this relationship is not in second normal form.

The following tables demonstrate second normal form:

STUDENT table:

Student_id	**Advisor**	**Adv-Room**
1022	Rincewind	412
4123	Weatherwax	216

REGISTRATION table:

Student_id	**Class_id**
1022	101-07
1022	143-01
1022	159-02
4123	101-07

4123	143-01
4123	179-04

4.2.4 Third normal form: Eliminate data not dependent on key

In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute. The solution is to move that attribute from the Students table to the Faculty table, as shown below:

STUDENT table:

Student_id	**Advisor**
1022	Rincewind
4123	Weatherwax

FACULTY table:

Name	**Room**	**Dept**
Rincewind	412	42
Weatherwax	216	42

4.3 CODD'S MNEMONIC

One way of thinking about what we are doing is to remember the slogan "THE KEY, THE WHOLE KEY, AND NOTHING BUT THE KEY, SO HELP ME CODD." This slogan breaks down as follows

1. "The key" : Tables may not contain repeating groups, which prevent a table from having a primary key. (1NF)
2. "the whole key": Every attribute must be functionally dependent on the entire primary key, not merely on part of the key. (2NF)
3. "and nothing but the key," : The can be no transitive dependencies on a non-key field.(3NF)
4. " so help me, Codd." : The inventor of relational databases who gave us these rules.

4.4 REVIEW QUESTIONS

1. (Review) What does it mean for a set of attributes A to *fully determine* a set of attributes B? How does this relate to the concept of a candidate key?
2. What is the point of database normalization? Why would anyone bother to learn all of these rules?
3. What are the requirements for a data model to satisfy 1NF? Give an original example of a data model that is NOT in 1NF, and show how it could be transformed to meet the requirements.
4. What are the requirements for a data model to satisfy 2NF? Give an original example of a data model that is in 1NF but not 2NF, and show how it could be transformed to meet the requirements.
5. What are the requirements for a data model to satisfy 3NF? Give an original example of a data model that is in 2NF but not 3NF, and show how it could be transformed to meet the requirements.

5 Reading: PowerPoint Is Evil (By Edward Tufte)

Power Corrupts. PowerPoint Corrupts Absolutely.

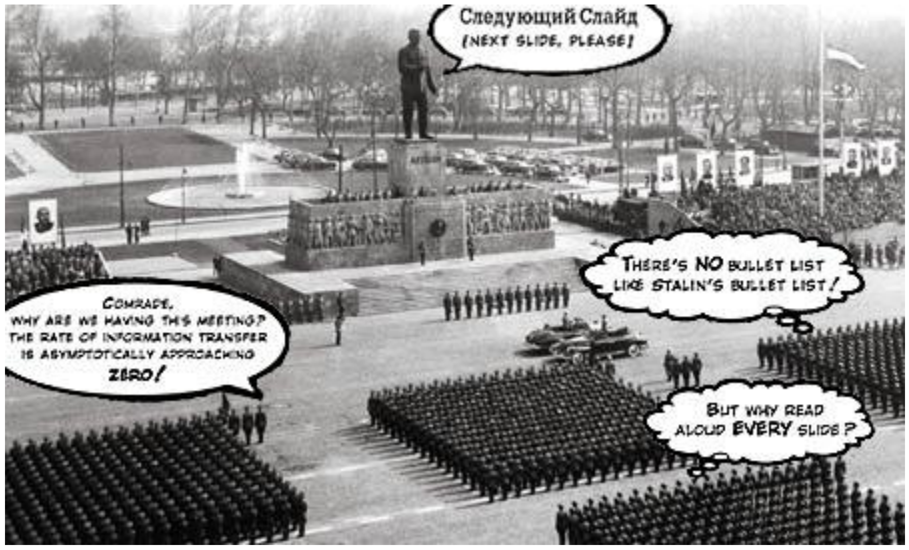
[Brendan's Note: Edward Tufte is a computer scientist and statistician who has written a lot about how to visually communicate complicated facts about data. Here, in a famous article, he argues that PowerPoint, which has become something of a business "standard", is a terrible way of doing this.

This article is from the early 2000s when PowerPoint use was just becoming widespread. Do you think things have gotten better since then? Worse?]

Imagine a widely used and expensive prescription drug that promised to make us beautiful but didn't. Instead the drug had frequent, serious side effects: It induced stupidity, turned everyone into bores, wasted time, and degraded the quality and credibility of communication. These side effects would rightly lead to a worldwide product recall.

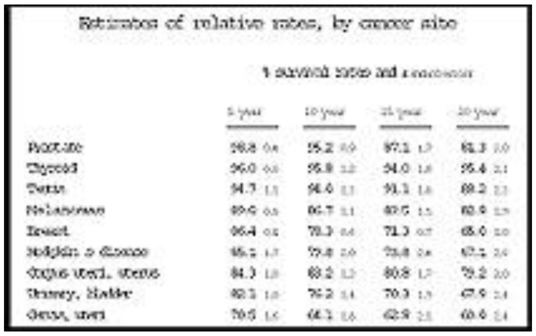
Yet slideware -computer programs for presentations -is everywhere: in corporate America, in government bureaucracies, even in our schools. Several hundred million copies of Microsoft PowerPoint are churning out trillions of slides each year. Slideware may help speakers outline their talks, but convenience for the speaker can be punishing to both content and audience. The standard PowerPoint presentation elevates format over content, betraying an attitude of commercialism that turns everything into a sales pitch.

Of course, data-driven meetings are nothing new. Years before today's slideware, presentations at companies such as IBM and in the military used bullet lists shown by overhead projectors. But the format has become ubiquitous under PowerPoint, which was created in 1984 and later acquired by Microsoft. PowerPoint's pushy style seeks to set up a speaker's dominance over the audience. The speaker, after all, is making power points with bullets to followers. Could any metaphor be worse? Voicemail menu systems? Billboards? Television? Stalin?


Следующий Слайд
(NEXT SLIDE, PLEASE)
THERE'S NO BULLET LIST LIKE STALIN'S BULLET LIST!
COMRADE, WHY ARE WE HAVING THIS MEETING? THE RATE OF INFORMATION TRANSFER IS ASYMPTOTICALLY APPROACHING ZERO!
BUT WHY READ ALOUD EVERY SLIDE?

Particularly disturbing is the adoption of the PowerPoint cognitive style in our schools. Rather than learning to write a report using sentences, children are being taught how to formulate client pitches and infomercials. Elementary school PowerPoint exercises (as seen in teacher guides and in student work posted on the Internet) typically consist of 10 to 20 words and a piece of clip art on each slide in a presentation of three to six slides -a total of perhaps 80 words (15 seconds of silent reading) for a week of work. Students would be better off if the schools simply closed down on those days and everyone went to the Exploratorium or wrote an illustrated essay explaining something.

In a business setting, a PowerPoint slide typically shows 40 words, which is about eight seconds' worth of silent reading material. With so little information per slide, many, many slides are needed. Audiences consequently endure a relentless sequentiality, one damn slide after another. When information is stacked in time, it is difficult to understand context and evaluate relationships. Visual reasoning usually works more effectively when relevant information is shown side by side. Often, the more intense the detail, the greater the clarity and understanding. This is especially so for statistical data, where the fundamental analytical act is to make comparisons.



Estimates of relative rates, by cancer site

% survival rates and their standard errors

	5 year		10 year		15 year		20 year	
Prostate	98.8	0.4	95.2	0.9	87.1	1.7	81.1	3.0
Thyroid	96.0	0.8	95.8	1.2	94.0	1.6	95.4	2.1
Testis	94.7	1.1	94.0	1.3	91.1	1.8	88.2	2.3
Melanomas	89.0	0.8	86.7	1.1	83.5	1.5	82.8	1.9
Breast	86.4	0.4	78.3	0.6	71.3	0.7	65.0	1.0
Hodgkin's disease	85.1	1.7	79.8	2.0	73.8	2.4	67.1	2.8
Corpus uteri, uterus	84.3	1.0	83.2	1.3	80.8	1.7	79.2	2.0
Urinary, bladder	82.1	1.0	76.2	1.4	70.3	1.9	67.9	2.4
Cervix, uteri	70.5	1.6	64.1	1.8	62.8	2.1	60.0	2.4

Figure 1GOOD Graphics Press A traditional table: rich, informative, clear.

|

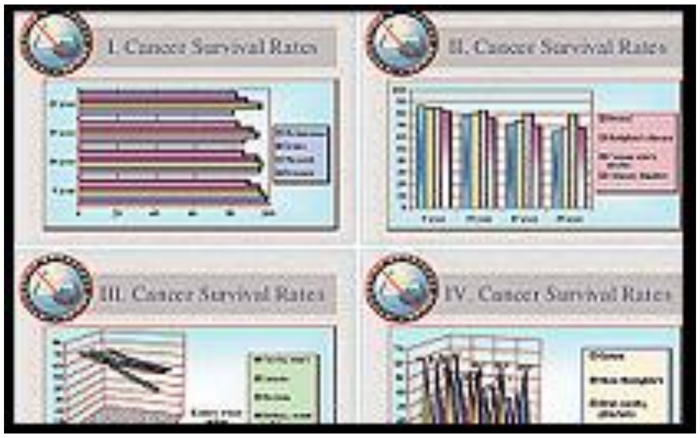


Figure 2 BAD Graphics Press PowerPoint chartjunk: smarmy, chaotic, incoherent.

Cider an important and intriguing table of survival rates for those with cancer relative to those without cancer for the same time period. Some 196 numbers and 57 words describe survival rates and their standard errors for 24 cancers.

Adding the PowerPoint templates to this nice, straightforward table yields an analytical disaster. The data explodes into six separate chaotic slides, consuming 2.9 times the area of the table. Everything is wrong with these smarmy, incoherent graphs: the encoded legends, the meaningless color, the logo-type branding. They are uncomparative, indifferent to content and evidence, and so data-starved as to be almost pointless. Chart-junk is a clear sign of statistical stupidity. Poking a finger into the eye of thought, these data graphics would turn into a nasty travesty if used for a serious purpose, such as helping cancer patients assess their survival chances. To sell a product that messes up data with such systematic intensity, Microsoft abandons any pretense of statistical integrity and reasoning.

Presentations largely stand or fall on the quality, relevance, and integrity of the content. If your numbers are boring, then you've got the wrong numbers. If your words or images are not on point, making them dance in color won't make them relevant. Audience boredom is usually a content failure, not a decoration failure.

At minimum, a presentation format should do no harm. Yet the PowerPoint style routinely disrupts, dominates, and trivializes content. Thus PowerPoint presentations too often resemble a school play -very loud, very slow, and very simple.

The practical conclusions are clear. PowerPoint is a competent slide manager and projector. But rather than supplementing a presentation, it has become a substitute for it. Such misuse ignores the most important rule of speaking: Respect your audience.