Agenda

Database Design

- Entity Relationship Diagrams
  - what they are / why we use them
- Crow's Foot Notation
  - Many to Many relationships
  - Strong and Weak Entities
  - Strong and Weak Relationships
  - ERD editors

- Functional Dependencies
- Data Anomalies

Next Section

Upcoming Section Schedule:
Feb 26, 10am ET
Mar 5, 10am ET

Homework 3 is due this week on Feb 27 by midnight.

Homework 4 will be released this weekend. Homework 4 includes topics from this week, and from next week's lecture.

What is an Entity Relationship Diagram

An entity-relationship (ER) diagram is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems. In other words, it’s a drawing that helps us visualize and plan the layout of our database.

ERDs have three basic components:

- entity types
- relationships
- attributes

Remember from earlier:
An entity is a person, place, thing, object or event about which data will be collected and stored.

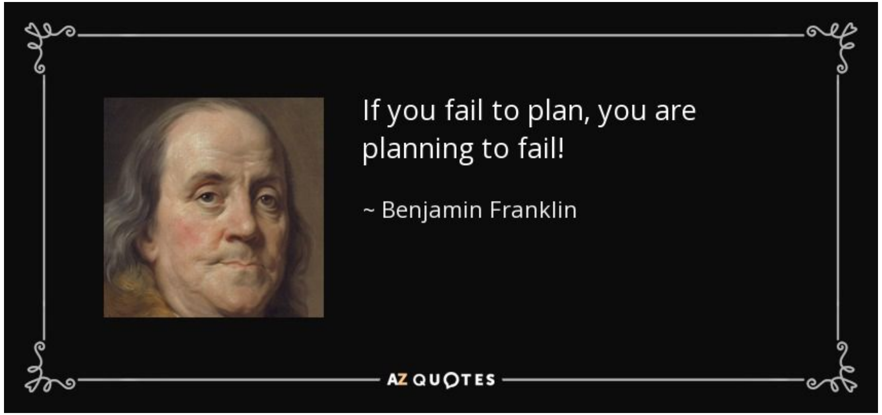
An attribute is a characteristic or trait of an entity type that describes the entity.

A relationship describes an association among entities.

Why is it important?

As much as we might prefer to go straight to the development part of the project, it’s very risky to go into implementation without a clear plan.

What typically happens if you rush into implementation without a good design is that you learn about your system as you are building it, which often results in a lot of refactoring - or in the worst case, it results in an implementation that is difficult or impossible to maintain, and you wish you could go back an refactor it.



If you fail to plan, you are planning to fail!

~ Benjamin Franklin

When is it used?

- **Database design** - Depending on the scale of change, it can be risky to alter a database structure directly in a DBMS. To avoid ruining the data in a production database, it is important to plan out the changes carefully. ERD is a tool that helps. By drawing ER diagrams to visualize database design ideas, you have a chance to identify the mistakes and design flaws, and to make corrections before executing the changes in the database.
- **Database debugging** - To debug database issues can be challenging, especially when the database contains many tables, which require writing complex SQL in getting the information you need. By visualizing a database schema with an ERD, you have a full picture of the entire database schema. You can easily locate entities, view their attributes and identify the relationships they have with others. All these allow you to analyze an existing database and to reveal database problems easier.
- **Database creation and patching** - Visual Paradigm, an ERD tool, supports a database generation tool that can automate the database creation and patching process by means of ER diagrams. So, with this ER Diagram tool, your ER design is no longer just a static diagram but a mirror that reflects truly the physical database structure.
- **Aid in requirements gathering** - Determine the requirements of an information system by drawing a conceptual ERD that depicts the high-level business objects of the system. Such an initial model can also be evolved into a physical database model that aids the creation of a relational database, or aids in the creation of process maps and data flow modes.

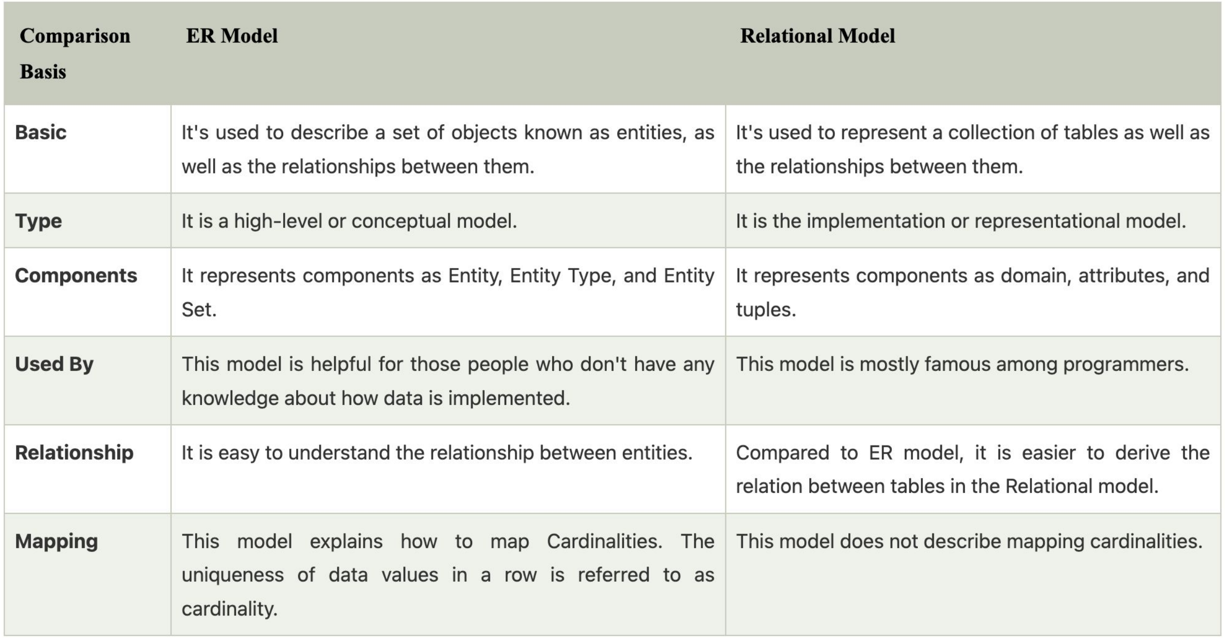
https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

Entity Relationship Diagram vs Relational Model

Technically, an **Entity Relationship Diagram** is an abstract design of our database. It defines the relationships between entities and their attributes. Foreign keys do not need to explicitly identified, they are implied. In an ERD, we are simply visualising the entities, their attributes, and the relation between them. In its early stages it’s a semi-rough sketch, but can evolve into a detailed representation of the database.

In a **Relational Model** we are referring to an implementation of our model. This model is clear about data types used by our attributes, the foreign keys we use, and the linking tables we might need to stitch entities together. A relational model is basically a model of a possible database implementation. A Relational Model can also be referred to as a **Physical ERD**.

Entity Relationship Diagram vs Relational Model



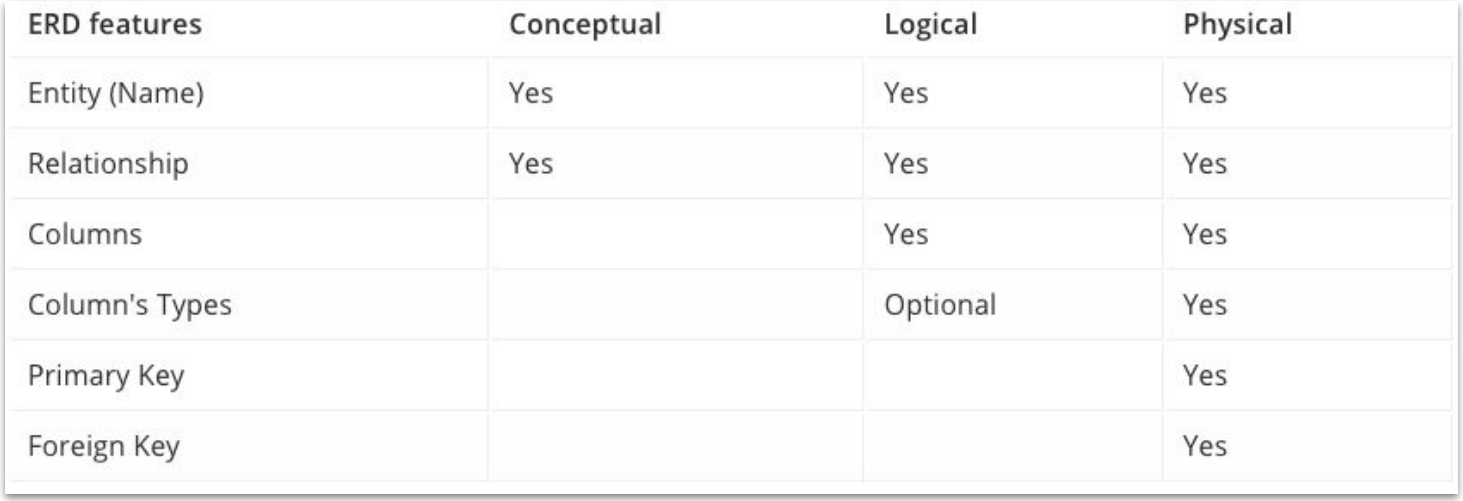
Comparison Basis	ER Model	Relational Model
Basic	It's used to describe a set of objects known as entities, as well as the relationships between them.	It's used to represent a collection of tables as well as the relationships between them.
Type	It is a high-level or conceptual model.	It is the implementation or representational model.
Components	It represents components as Entity, Entity Type, and Entity Set.	It represents components as domain, attributes, and tuples.
Used By	This model is helpful for those people who don't have any knowledge about how data is implemented.	This model is mostly famous among programmers.
Relationship	It is easy to understand the relationship between entities.	Compared to ER model, it is easier to derive the relation between tables in the Relational model.
Mapping	This model explains how to map Cardinalities. The uniqueness of data values in a row is referred to as cardinality.	This model does not describe mapping cardinalities.

https://www.javatpoint.com/er-model-vs-relational-model

@GregMisicko

Conceptual / Logical / Physical

These terms apply to schemas as well as ERDs. The characteristics of each maps out as follows:



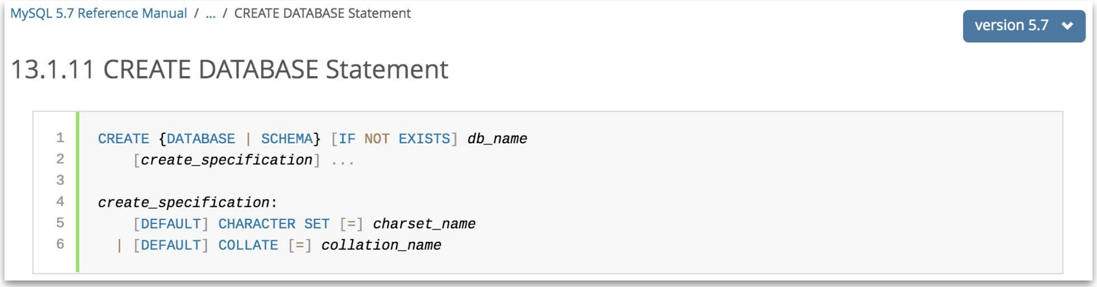
ERD features	Conceptual	Logical	Physical
Entity (Name)	Yes	Yes	Yes
Relationship	Yes	Yes	Yes
Columns		Yes	Yes
Column's Types		Optional	Yes
Primary Key			Yes
Foreign Key			Yes

https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

What Is A Schema?

You may have heard of a database schema before, and may have seen examples of them. What is a schema? How does it relate to the diagrams we are discussing?

In MySQL it refers to a database, and in fact “schema” can be used interchangeably with “database” when forming SQL statements:



MySQL 5.7 Reference Manual / ... / CREATE DATABASE Statement

13.1.11 CREATE DATABASE Statement

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

What Is A Schema?

As usual, there are various definitions for what a schema is and there is also variance from one database vendor to the next. Since we are working with MySQL we'll go by their definition and consider our schema to be the overall database.

You may also hear of different flavors of schema:

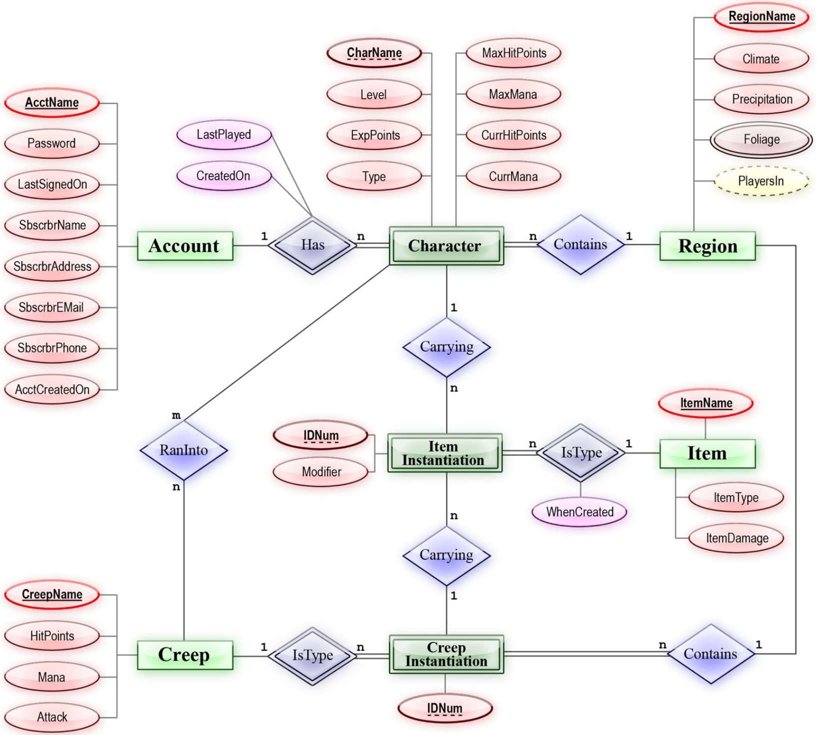
- **physical or internal schema** refers to how the RDBMS itself stores and organizes data
- **logical or conceptual schema** refers to how developers and database administrators view the database
- **external or view level schema** refers to how end users view the database

If it helps to think of it this way: a schema is a mapping of your database that you can actually insert data into. An ERD is a diagram of your database.

Entity Relationship Diagram Notation

- We will use Crow’s Foot notation, however you will find many different types of notations used and you will even see variations of the Crow’s Foot notation.
- While Crow's Foot notation is often recognized as the most intuitive style you’ll also find Chen Models, Object Modeling Technique (OMT), Integration Definition (IDEF), Bachman, or UML notation used to describe a database.
- There is no “standard” ERD notation (I wish that there was), and no market vendor has enough influence to drive a standard.
- The closest thing to a standard is Universal Modeling Language (UML), but is more commonly used for software modeling
- It can be frustrating and confusing to see different notations used within a single design, or example, but this too can be pretty common.

Chen Notation



https://en.wikipedia.org/wiki/Entity–relationship_model

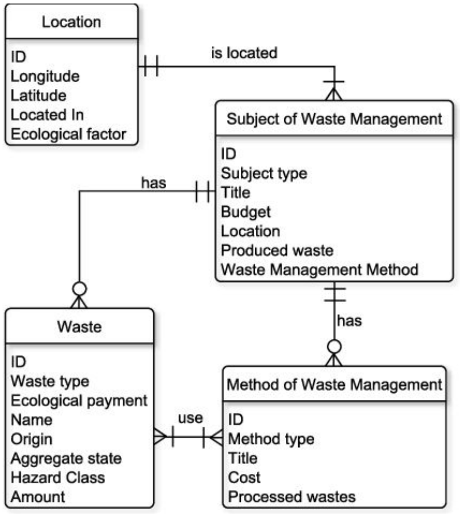
Entity–relationship modeling was developed for database and design by Peter Chen and published in a 1976 paper.

Chen notation favors conceptual modeling. It is very detailed, but this detail takes more time to learn and is not going to be easy to white-board.

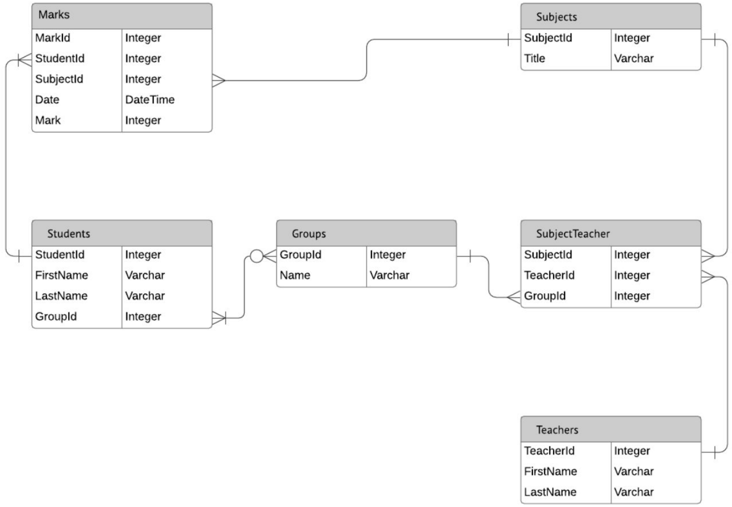
@GregMisicko

Crow’s Foot Notation

Crow’s Foot notation favors a more implementation-oriented approach. Easy to sketch by hand (good for white-boarding) and fairly intuitive to understand with a small amount of practice.



https://www.researchgate.net/figure/ER-diagram-of-the-subject-domain-Crows-Foot-notation_fig6_311756970

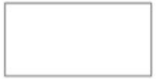


https://lucidchart.zendesk.com/hc/en-us/articles/360029676532-Template-Simple-ERD-Crow-s-Foot-

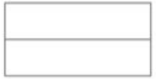
@GregMisicko

Crow’s Foot Notation

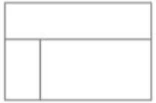
Crow’s foot notation is going to be inconsistent from one user to the next, unfortunately. If you search around for Crow’s Foot Notation references, you’ll likely find various ways of presenting a simple entity box, as shown with the examples laid around this slide.



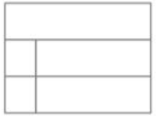
Entity
(with no attributes)



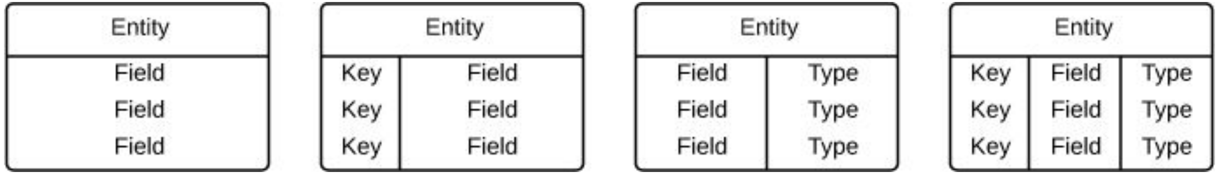
Entity
(with attributes field)



Entity
(attributes field with columns)



Entity
(attributes field with columns and variable number of rows)



Entity
Field
Field
Field

Entity	
Key	Field
Key	Field
Key	Field

Entity	
Field	Type
Field	Type
Field	Type

Entity		
Key	Field	Type
Key	Field	Type
Key	Field	Type

@GregMisicko

Cardinality and Ordinality/Modality

Cardinality refers to the relationship between a row of one table and a row of another table. The only two options for cardinality are one or many. Cardinality refers to the maximum number of times an instance in one entity can relate to instances of another entity. Ordinality (or modality), on the other hand, is the minimum number of times an instance in one entity can be associated with an instance in the related entity.

Cardinality can be 1 or Many and the symbol is placed on the outside ends of the relationship line, closest to the entity, Modality can be 1 or 0 and the symbol is placed on the inside, next to the cardinality symbol.

For a cardinality of 1 a straight, perpendicular line is drawn.
For a cardinality of Many a foot with three toes is drawn - a crow's foot.
For a modality of 1 a straight line is drawn.
For a modality of 0 a circle is drawn.

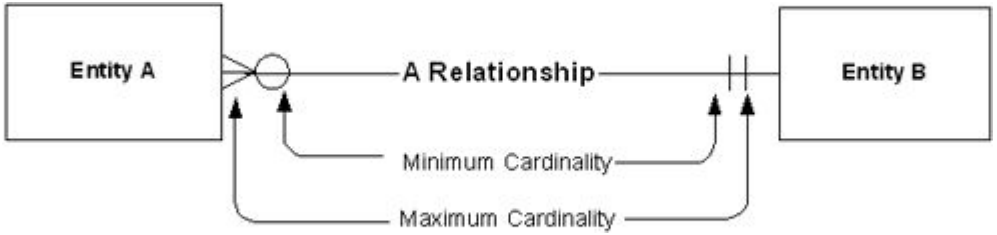
Cardinality: the max number
Ordinality or Modality: the min number

http://www2.cs.uregina.ca/~bernatja/crowsfoot.html

Crow’s Foot Notation

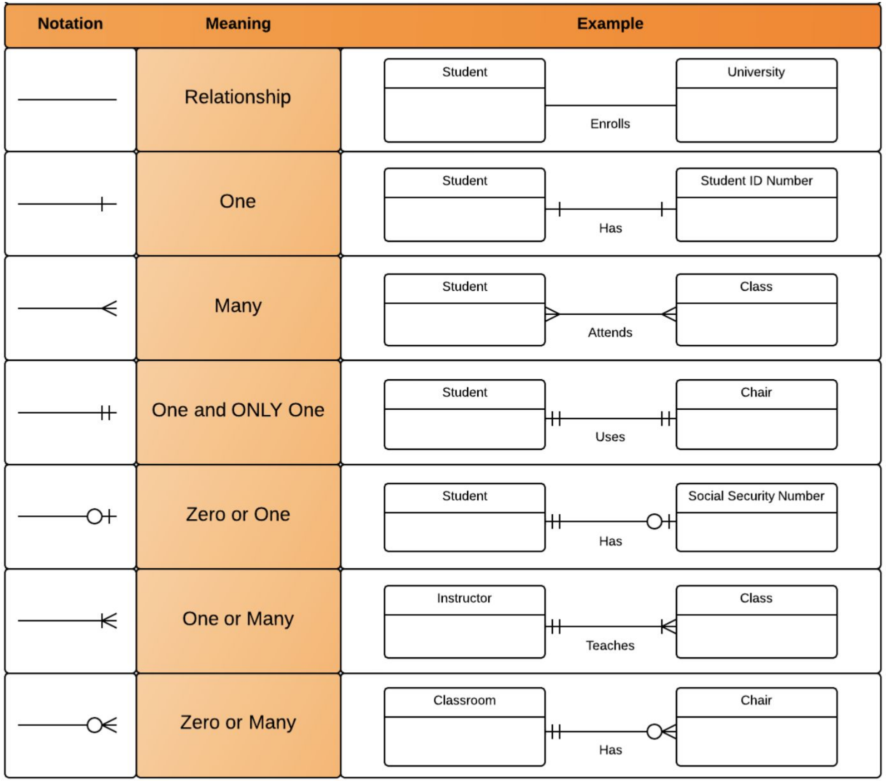
Read “minimum cardinality” as ordinality. Or modality. Yes, there always seems to be several ways of doing or saying or doing the exact same thing in the database world...

Read the symbols from left to right (from Entity A to Entity B) and you will determine that Entity A can have 1 relationship to Entity B (a minimum of 1, and a maximum of 1). Read from right to left (from Entity B toward Entity A) and you will determine that Entity B can have a relationship with zero or many instances of Entity A.



@GregMisicko

Crow’s Foot Notation



Notation	Meaning	Example
	Relationship	Student Enrolls University
	One	Student Has Student ID Number
	Many	Student Attends Class
	One and ONLY One	Student Uses Chair
	Zero or One	Student Has Social Security Number
	One or Many	Instructor Teaches Class
	Zero or Many	Classroom Has Chair

Let’s use this chart as our basic reference for our Entity Relationship Diagram notation as far as relationships are concerned.

As you can see, relationships are identified by a line connecting one entity to another. To make the relationship more human readable, text is included to describe the association.

Symbols exist at the end of each line to identify the type of relationship. Let’s look more closely at what these symbols mean.

Note: the top three examples here are ambiguous relationships.

https://www.vivekmchawla.com/erd-crows-foot-relationship-symbols-cheat-sheet/

@GregMisicko

Crow’s Foot Notation Example

How would we draw up the relationship between students in a class, and the seats available within the classroom?

How would we draw up the relationship between instructors, and courses they teach?

How would we draw up the relationship between students, and the courses they enroll in?

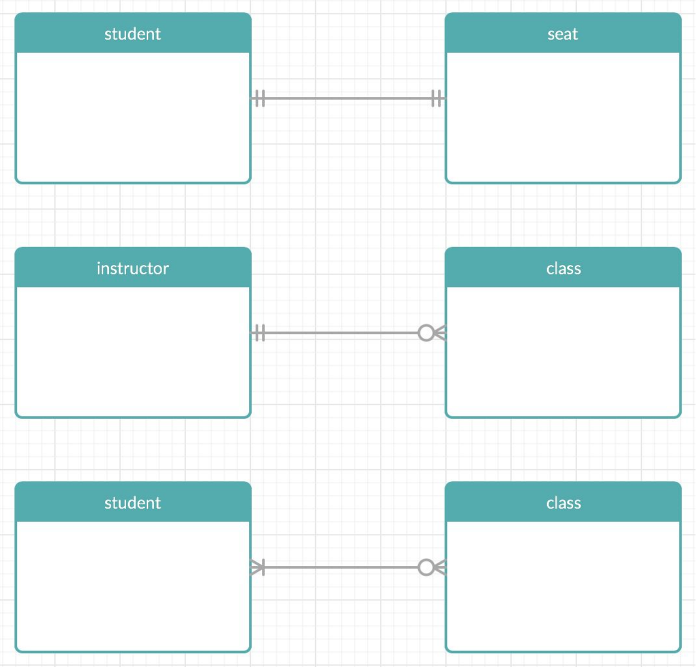
ERD Editors

How can I draw clean diagrams on the computer? There are a few great options which require no installation as they are both browser-based.

Sign up is free for either one. You will have the ability to create a limited number of projects, and you will not have access to “premium” features.

Creately: https://creately.com/
Lucidchart: https://www.lucidchart.com/pages/

Crow’s Foot Notation Example



A student can have one and only one seat, and a seat only holds one student.

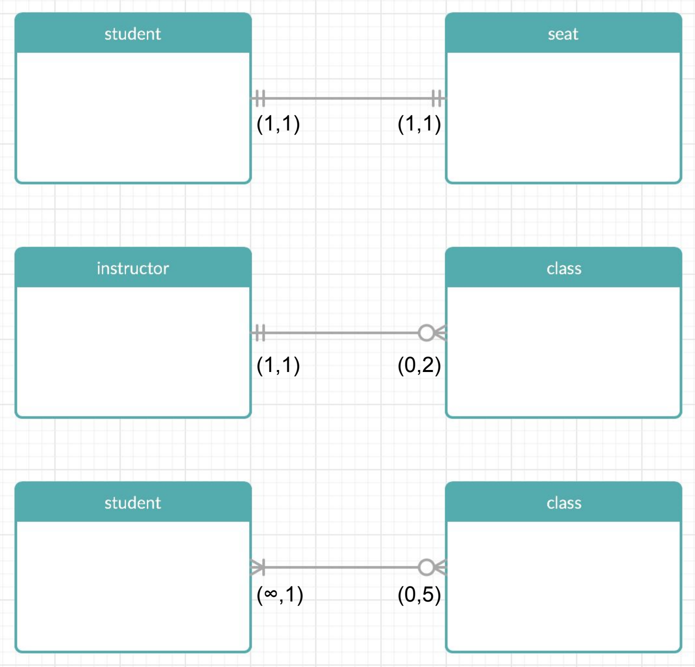
An instructor can teach no classes in a semester, or an instructor can teach one or more.

A student can take no classes in a semester, or they can take one or more classes. If a class had zero students, I think that class would need to be canceled. A class needs to have at least one student, or many.

Note that a “1” indicates a mandatory relationship, and a “0” indicates an optional relationship.

@GregMisicko

Crow’s Foot Notation Example



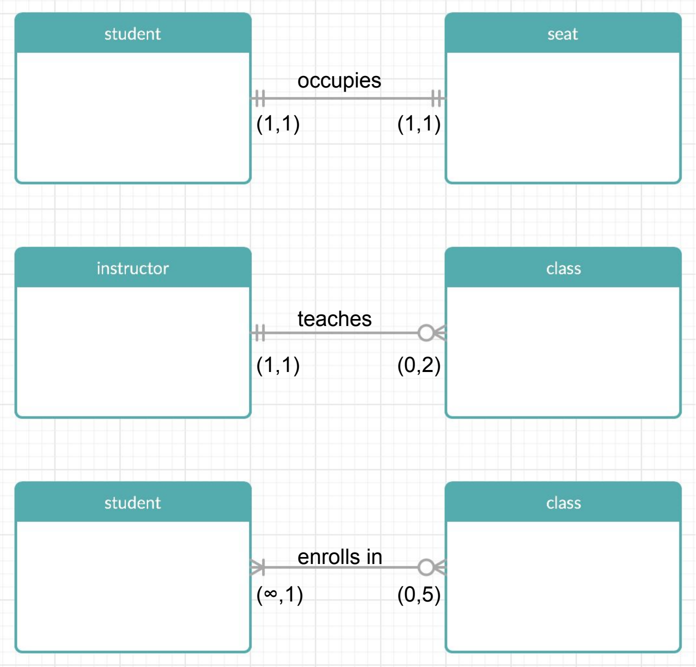
What if “many” is not specific enough?

It is possible that you’ll find situations where you know of a strict upper limit. For example, let’s say that Harvard has a policy where an instructor can not teach more than one class per semester. Let’s also say that a student is prohibited from ever taking more than 5 classes per semester.

These kinds of constraints can be represented using the notation shown here.

@GregMisicko

Crow’s Foot Notation Example



In the early stages of design, when we are simply mapping out entities and relationships, we identify the relationship without describing the implementation details.

How do we know which seat a student occupies? We don’t yet. We might even decide later that this information doesn’t serve a justifiable purpose, and we might remove it. For now, we think we might want to know that a student occupies a seat.

An instructor teaches a class. How do we implement that relationship? At this point, we don’t know. We just know that we will keep track of which class(es) an instructor teaches.

@GregMisicko

Crow’s Foot Notation

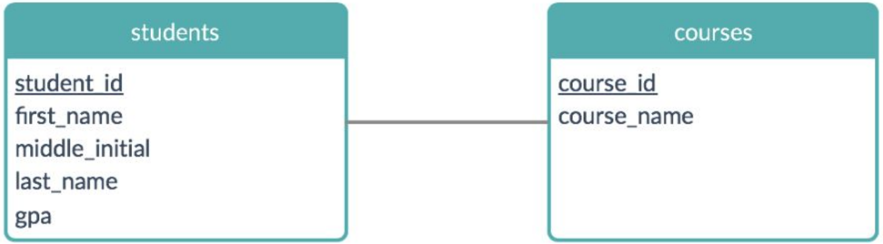
How do we identify KEYS in our diagram? For our examples here, we will identify keys with an underline as well as an indicator for Primary Key or Foreign Key as so:

- (PK)
- (FK)

How do we specify data types?
Using a multi-column box for our entities, we can list data types to the side of our attributes.

Conceptual / Logical / Physical

Last week we worked on examples using students enrolled in courses. This is what we were working from:



students

student_id
first_name
middle_initial
last_name
gpa

courses

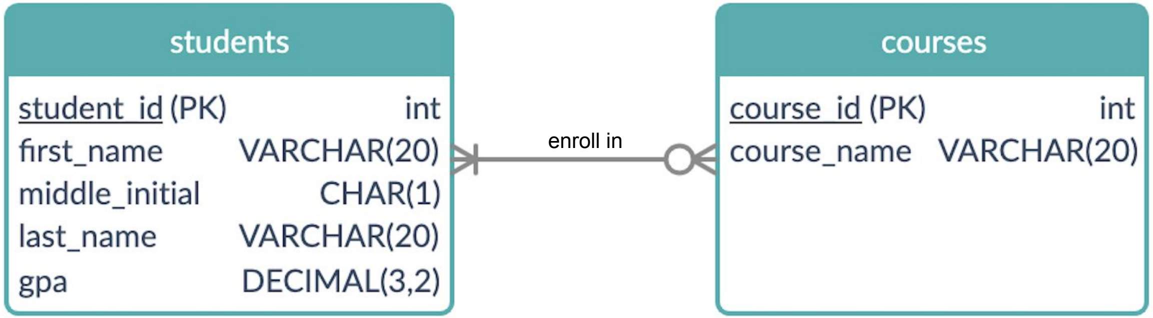
course_id
course_name

Would the above diagram be considered a conceptual, logical, or physical diagram?

Crow’s Foot Notation Example

Early in the design process our plans would have looked more like the slide preceding this one where we have identified entities and relationships, but did not yet include attributes or additional details such as primary keys. That diagram would be considered conceptual.

This diagram below is logical, as it provides more details such as attributes and their types. Overall this diagram should be easily readable. If we look at it, it is not difficult to determine that students have relationships with courses. Students enroll in courses. What isn’t clear is how we would implement that relationship - how does that “enroll in” relationship map to something usable in our RDBMS?



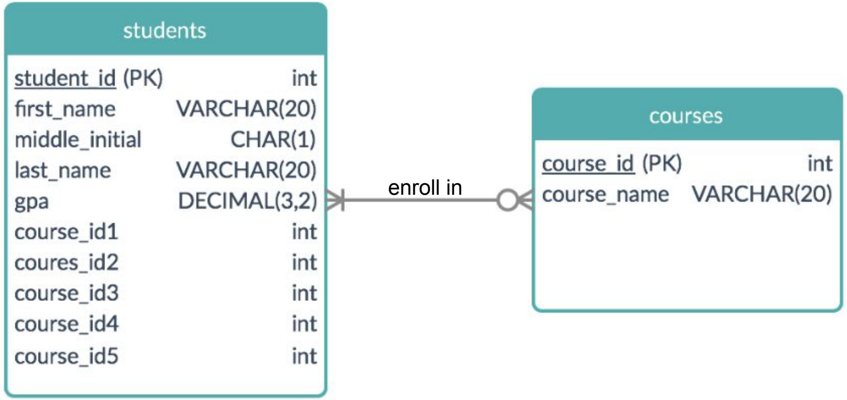
students	
student_id (PK)	int
first_name	VARCHAR(20)
middle_initial	CHAR(1)
last_name	VARCHAR(20)
gpa	DECIMAL(3,2)

courses	
course_id (PK)	int
course_name	VARCHAR(20)

@GregMisicko

Crow’s Foot Notation Example

You might think that you could get away with adding a column in students for “course_id”, but this would only be reasonable in a 1-1 relationship. It would not work well with our one-to-many relationship. Even if we know a student can only enroll in up to five classes, it would be inefficient to add columns such as: course_id1, course_id2, etc.



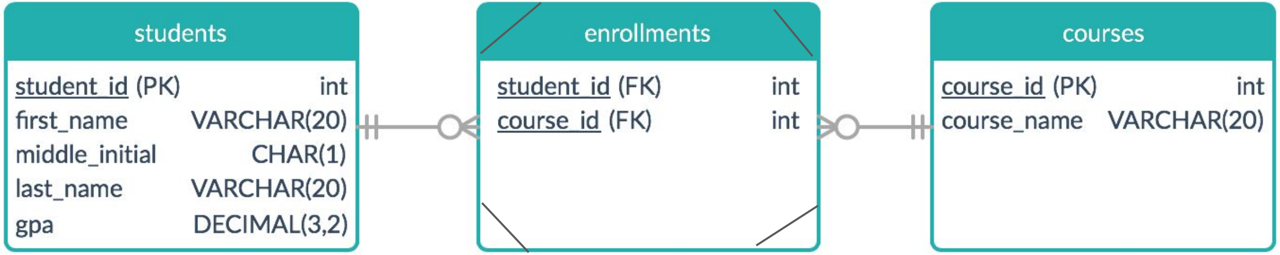
students	
student_id (PK)	int
first_name	VARCHAR(20)
middle_initial	CHAR(1)
last_name	VARCHAR(20)
gpa	DECIMAL(3,2)
course_id1	int
coures_id2	int
course_id3	int
course_id4	int
course_id5	int

courses	
course_id (PK)	int
course_name	VARCHAR(20)

@GregMisicko

Many To Many Relationships

We implement many to many relationships using what’s called a **join table**. Also known as a **bridge**. Or a **junction**. And the entity itself is referred to as an **associative entity** type.



students	
student_id (PK)	int
first_name	VARCHAR(20)
middle_initial	CHAR(1)
last_name	VARCHAR(20)
gpa	DECIMAL(3,2)

enrollments	
student_id (FK)	int
course_id (FK)	int

courses	
course_id (PK)	int
course_name	VARCHAR(20)

@GregMisicko

Existence Dependence

An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with another related entity occurrence. This type of entity is known as a **weak entity**.

A **weak entity** is one that meets two conditions:

1. The entity is existence-dependent; it cannot exist without the entity with which it has a relationship
2. The entity has a primary key that is partially or totally derived from the parent entity in the relationship

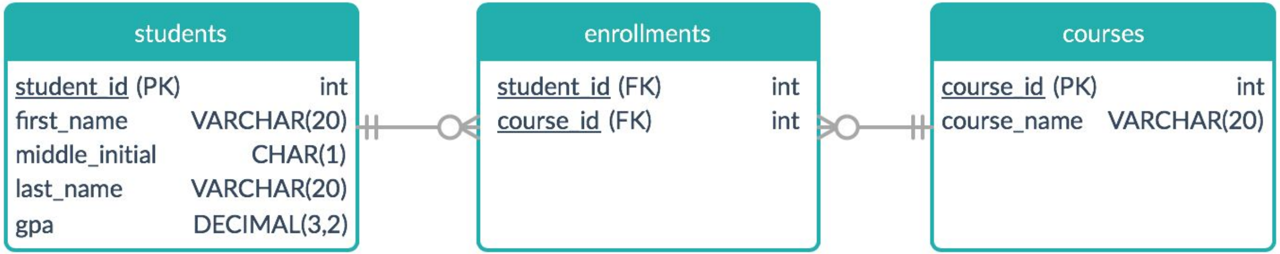
If an entity can exist apart from all of its related entities, then it is **existence-independent** and it is referred to as a **strong entity** or a regular entity.

(1) Coronel, Morris. *Database Systems Design, Implementation, & Management.* Cengage, 2019

Strong and Weak Entity Types

A simpler way to think of it is to say that a weak entity is one that can only exist when owned by another one. For example, an order would not exist without any products (if there are no products to sell, there is no order to create). A bank account could not exist without a bank. An enrollment would not exist without students, and an enrollment could not exist without a course… so yes, our `enrollments` table would be considered a weak entity in our current example.

Is a join table (also known as a bridge or junction table) always going to be a weak entity type?



students	
student_id (PK)	int
first_name	VARCHAR(20)
middle_initial	CHAR(1)
last_name	VARCHAR(20)
gpa	DECIMAL(3,2)

enrollments	
student_id (FK)	int
course_id (FK)	int

courses	
course_id (PK)	int
course_name	VARCHAR(20)

Strong and Weak Relationship Types

We refer to relationships as being strong, or weak. Formally, these relationships are defined as such:

1. Weak (**Non-Identifying**) Relationship
   - Entity is existence-independent of other entities
   - PK of Child doesn’t contain PK component of Parent Entity
2. Strong (**Identifying**) Relationship
   - Child entity is existence-dependent on parent
   - PK of Child Entity contains PK component of Parent Entity
   - Usually occurs utilizing a composite key for primary key, which means one of this composite key components must be the primary key of the parent entity.

A dashed line means that the relationship is strong, whereas a solid line means that the relationship is weak.

Why Identify Weak Entities?

We know the definitions of strong and weak entities… but why does it matter to us?

Consider when we were creating enrollments as a join table. We knew this table was a weak entity with a strong relationship dependency on it's associated attributes. Thus, we knew that if either of its dependencies were deleted, we would want that deletion to cascade. What if we had not implemented the table with deletions set to cascade?

👍

```
CREATE TABLE enrollments
(
  course_id  INT unsigned NOT NULL,
  student_id INT unsigned NOT NULL,
  FOREIGN KEY (course_id)
        REFERENCES courses(course_id)
        ON DELETE CASCADE,
  FOREIGN KEY (student_id)
        REFERENCES students(student_id)
        ON DELETE CASCADE,
  PRIMARY KEY (course_id, student_id)
);
```

👎

```
CREATE TABLE enrollments
(
  course_id  INT unsigned NOT NULL,
  student_id INT unsigned NOT NULL,
  FOREIGN KEY (course_id)
        REFERENCES courses(course_id),
  FOREIGN KEY (student_id)
        REFERENCES students(student_id),
  PRIMARY KEY (course_id, student_id)
);
```

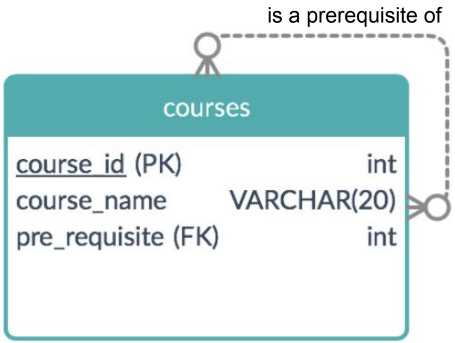
Self Referencing (Reflexive) Relationship

It is possible for an entity to have a relationship with itself. We call this a self referencing or reflexive relationship. For example, in our school database example we could say that some courses may have prerequisites.

A prerequisite is simply another course, and it is understood that the prerequisite course should be taken as preparation for the course in question.

Self Referencing (Reflexive) Relationship

In our ER Diagram, we would represent a reflexive relationship as shown below.



courses

course_id (PK)	int
course_name	VARCHAR(20)
pre_requisite (FK)	int

@GregMisicko

What is a database engine?

This question comes up from time to time, as you might see references to the “engine” in various places.

In general, referring to an "engine" for a technology implies that that specific module contains the core code for that technology’s operations. In database design, a database engine is composed of the component of the system that actually stores and retrieves data using the create, read, update and delete.

MySQL offers a selection of several engines which can be listed using the command:

```
SHOW ENGINES;
```

This is an advanced topic that is outside of the scope of this class, but for more information you can start here:
https://dev.mysql.com/doc/refman/8.0/en/storage-engines.html

Retrieve Table Details

Also very useful is the ability to sort-of “reverse engineer” a table. This command will give us the SQL statement used to create this table:

```
SHOW CREATE TABLE students;
```

You may notice when running this command that the SQL statement returned is different from the SQL command we entered earlier.

Designing from scratch

There are a variety of steps involved in designing a database from scratch. A typical process would be:

- define your business rules and data constraints
- identify functional dependencies, entities, attributes
- apply normalization concepts to refine the model

We’re going to get into some concepts now which may at first appear to be unimportant for database implementation, but are in fact essential for building a reliable database.

What Are Functional Dependencies

A Functional Dependency is when one attribute determines another attribute in a DBMS system. A functional dependency occurs when the value of one or more column(s) in each record of a relation uniquely determines the value of another column in that same record of the relation.

Functional Dependencies (FD) are important constraints for removing unwanted redundancy in a table. An FD is a constraint about the uniqueness of a column if used in a table alone with other columns. It is represented by the → arrow sign.

In mathematics, a **function** is a mathematical object that produces an output, when given an input - it could be a number, a vector, or anything that can exist inside a set of things. So a function is like a machine, that takes values of x and returns an output y. (https://simple.wikipedia.org/wiki/Function_(mathematics))

How to Write Functional Dependencies

We write out FD’s as:
X functionally determines Y, or **X→Y**
X: left-hand side (LHS), or determinant
Y: right-hand side

For each X value there is at most one Y value

You can also write out FD’s with multiple LHS or RHS values, such as:
X → Y, Z
(A, B) → C
(A, B) → (C, D, E)

The Value of Functional Dependencies

What is the value in identifying FD’s? What do they do for me?

Essentially, Functional Dependencies identify columns which should be placed together in the same table.

Or, to put it slightly differently: we take potentially disorganized data and identify functional dependencies in order to identify how to best organize our data into individual tables. Knowing how to identify Functional Dependencies leads us toward better database design.

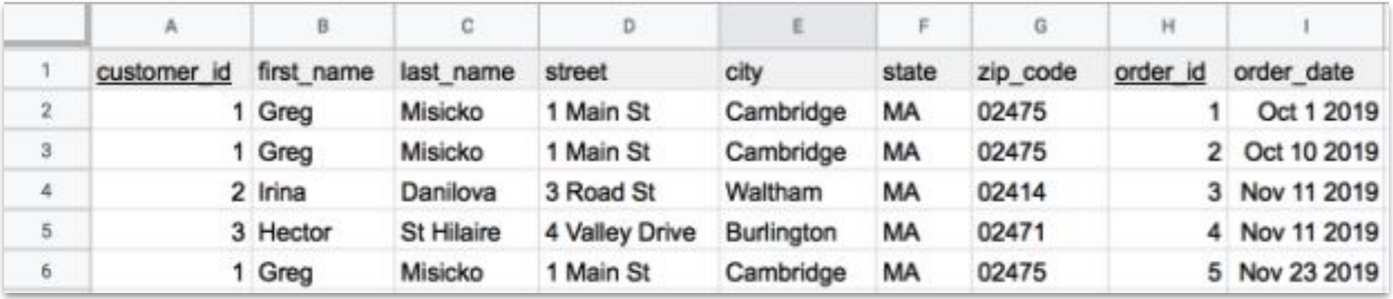
This might still be confusing, but some examples should (hopefully) make it easier to understand.

Falsification

When you prove that an FD is not true, this is called a falsification.
An FD cannot be proven by examining the rows of a table.
However, **a falsification can be proven by examining the rows of a table**: when two rows have the same X value, but a different Y value.

Again this may be confusing, but we will look at an example in later slides which should (hopefully) make it clear.

Falsification: Example

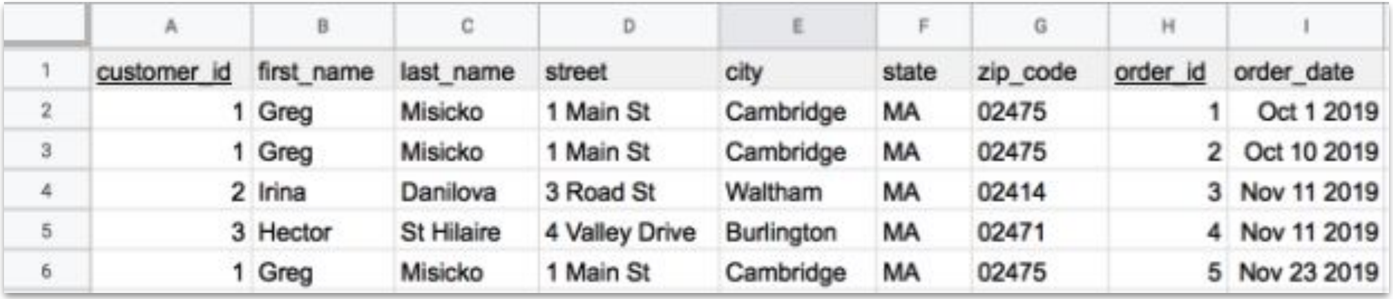


	A	B	C	D	E	F	G	H	I
1	customer_id	first_name	last_name	street	city	state	zip_code	order_id	order_date
2	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	1	Oct 1 2019
3	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	2	Oct 10 2019
4	2	Irina	Danilova	3 Road St	Waltham	MA	02414	3	Nov 11 2019
5	3	Hector	St Hilaire	4 Valley Drive	Burlington	MA	02471	4	Nov 11 2019
6	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	5	Nov 23 2019

We cannot prove a Functional Dependency using samples from a data table (as mentioned in the previous slide). Why? Because your sample data can't guarantee coverage of all potential scenarios which may arise.

You can, however, falsify an FD using sample data. Take our order data for example: what if we suspected that order_date could uniquely identify our order_id? (not a very clever assumption to begin with, but let's pretend…)
Based on the data in rows four and five above we can see that this assumption is wrong because 'Nov 11 2019' leads to two separate order_id's.
The statement X functionally determines Y, or **X→Y** does not hold.

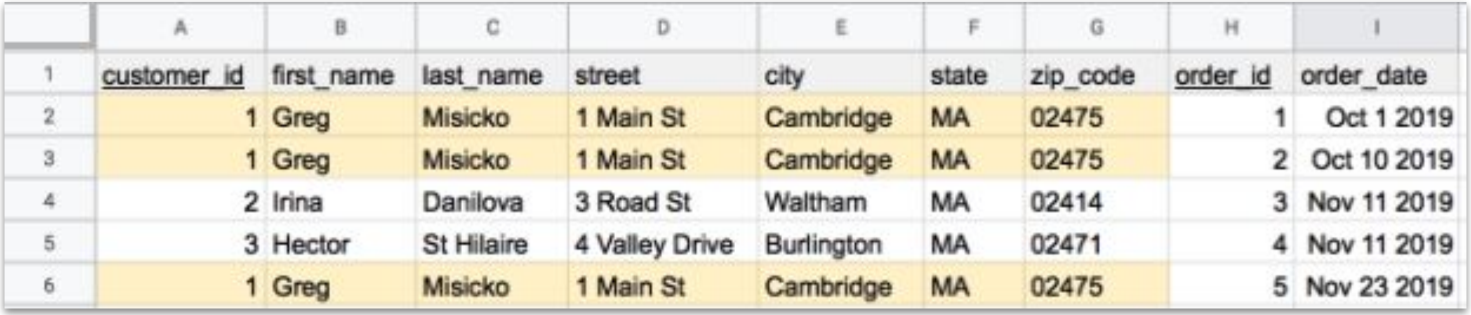
Data Redundancy



	A	B	C	D	E	F	G	H	I
1	customer_id	first_name	last_name	street	city	state	zip_code	order_id	order_date
2	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	1	Oct 1 2019
3	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	2	Oct 10 2019
4	2	Irina	Danilova	3 Road St	Waltham	MA	02414	3	Nov 11 2019
5	3	Hector	St Hilaire	4 Valley Drive	Burlington	MA	02471	4	Nov 11 2019
6	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	5	Nov 23 2019

Let’s say we have the above data, and we want to move it into a database from a spreadsheet. All of the data is in a single table, and it’s not too difficult to determine what it means. Maintaining this information in a single table is simpler than the multiple tables we’ve been using in our examples so far, right?

Data Redundancy



	A	B	C	D	E	F	G	H	I
1	customer_id	first_name	last_name	street	city	state	zip_code	order_id	order_date
2	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	1	Oct 1 2019
3	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	2	Oct 10 2019
4	2	Irina	Danilova	3 Road St	Waltham	MA	02414	3	Nov 11 2019
5	3	Hector	St Hilaire	4 Valley Drive	Burlington	MA	02471	4	Nov 11 2019
6	1	Greg	Misicko	1 Main St	Cambridge	MA	02475	5	Nov 23 2019

There are three main problems caused by data redundancy:

1. **Storing multiple values is a waste of storage capacity.** With a five row example it doesn't seem like much, but if this example were applied at an enterprise level it would be significant. Additionally, you'll be wasting time and bandwidth every time you perform a backup of this data.
2. **When a redundant value changes, it needs to be updated in multiple rows.** If customer_id='1' were to change his address, you would need to update the values for street, city, state, and zip_code multiple times.
3. If an update to any of those multiple rows were missed, we would have a **problem with inconsistent data**. If two addresses were found for customer_id='1', how would we know which was the correct address?

Data Anomalies

Data redundancy is a bad thing for the reasons just mentioned. The problems mentioned can lead to data anomalies. There are three types of anomalies we aim to avoid:

update (or modification) anomaly: An update anomaly is a data inconsistency that results from data redundancy and a partial update.

deletion anomaly: A deletion anomaly is the unintended loss of data due to deletion of other data.

insertion anomaly: An insertion anomaly is the inability to add data to the database due to absence of other data.

Example

This example data is taken from the book:

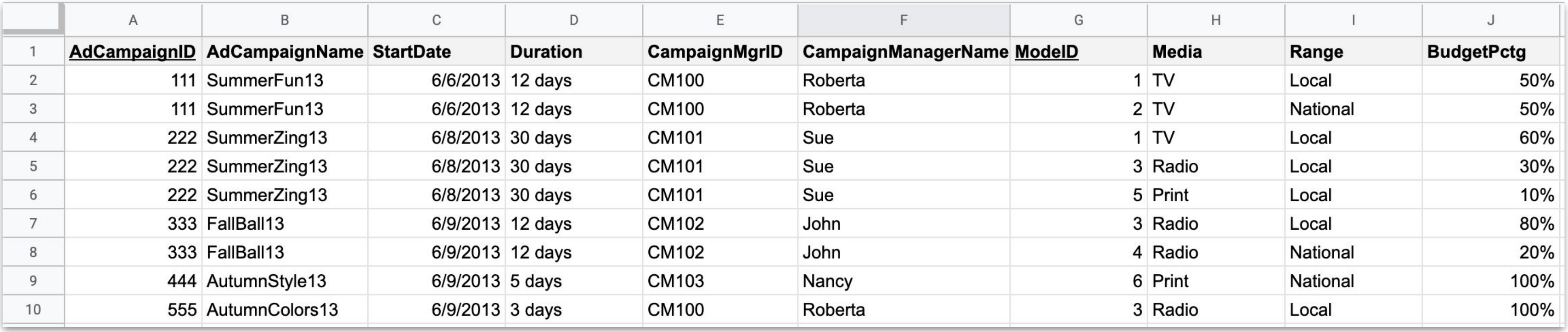
Jukic, Vrbsky, Nestorov. *Database Systems Introduction to Databases and Data Warehouses.* Prospect Press, 2017

Example Scenario:

An ad agency manages ad campaigns through a variety of campaign modes. Each campaign mode has a mode identifier "ModeID" and uses one particular media (TV, radio, or print) to cover one particular range (Local, National). For example, the campaign mode using TV media for Local range has the ModeID of 1, the campaign mode using TV media for National range has ModeID of 2, and so on.

Each ad campaign has a unique identifier, unique name, start date, duration, and campaign manager who has a name and a unique identifier. Each ad campaign can use a number of different campaign modes. When an ad campaign uses multiple campaign modes, a percentage of the overall budget is allocated to each mode. A campaign with a single mode uses 100% of the budget.

Example



	A	B	C	D	E	F	G	H	I	J
1	**AdCampaignID**	**AdCampaignName**	**StartDate**	**Duration**	**CampaignMgrID**	**CampaignManagerName**	**ModelID**	**Media**	**Range**	**BudgetPctg**
2	111	SummerFun13	6/6/2013	12 days	CM100	Roberta	1	TV	Local	50%
3	111	SummerFun13	6/6/2013	12 days	CM100	Roberta	2	TV	National	50%
4	222	SummerZing13	6/8/2013	30 days	CM101	Sue	1	TV	Local	60%
5	222	SummerZing13	6/8/2013	30 days	CM101	Sue	3	Radio	Local	30%
6	222	SummerZing13	6/8/2013	30 days	CM101	Sue	5	Print	Local	10%
7	333	FallBall13	6/9/2013	12 days	CM102	John	3	Radio	Local	80%
8	333	FallBall13	6/9/2013	12 days	CM102	John	4	Radio	National	20%
9	444	AutumnStyle13	6/9/2013	5 days	CM103	Nancy	6	Print	National	100%
10	555	AutumnColors13	6/9/2013	3 days	CM100	Roberta	3	Radio	Local	100%

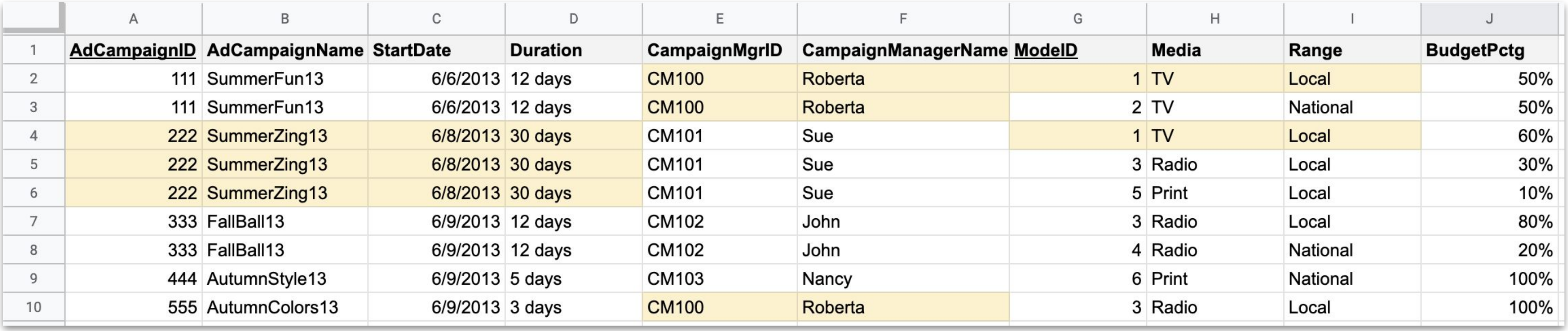
This is a sample of the data that could be stored for the ad campaign.

Where do you find the possibilities for the three types of anomalies we’ve identified?

What are the functional dependencies?

Jukic, Vrbsky, Nestorov. *Database Systems Introduction to Databases and Data Warehouses.* Prospect Press, 2017

Example



	A	B	C	D	E	F	G	H	I	J
1	AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID	CampaignManagerName	ModelID	Media	Range	BudgetPctg
2	111	SummerFun13	6/6/2013	12 days	CM100	Roberta	1	TV	Local	50%
3	111	SummerFun13	6/6/2013	12 days	CM100	Roberta	2	TV	National	50%
4	222	SummerZing13	6/8/2013	30 days	CM101	Sue	1	TV	Local	60%
5	222	SummerZing13	6/8/2013	30 days	CM101	Sue	3	Radio	Local	30%
6	222	SummerZing13	6/8/2013	30 days	CM101	Sue	5	Print	Local	10%
7	333	FallBall13	6/9/2013	12 days	CM102	John	3	Radio	Local	80%
8	333	FallBall13	6/9/2013	12 days	CM102	John	4	Radio	National	20%
9	444	AutumnStyle13	6/9/2013	5 days	CM103	Nancy	6	Print	National	100%
10	555	AutumnColors13	6/9/2013	3 days	CM100	Roberta	3	Radio	Local	100%

Here is the same table with some redundancies highlighted.

Jukic, Vrbsky, Nestorov. *Database Systems Introduction to Databases and Data Warehouses.* Prospect Press, 2017

Trivial Functional Dependencies

All of the functional dependencies from the preceding table are as follows:

(Set 1) CampaignManagerID → CampaignMgrName
(Set 2) ModeID → Media, Range
(Set 3) AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName
(Set 4) AdCampaignName → AdCampaignID, StartDate, Duration, CampaignMgrID, CampaignMgrName
(Set 5) AdCampaignID, ModeID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName, Media, Range, BudgetPctg
(Set 6) AdCampaignName, ModeID → AdCampaignID, StartDate, Duration, CampaignMgrID, CampaignMgrName, Media, Range, BudgetPctg

Note that we would not identify what are referred to as **trivial functional dependencies**. These are defined as FD's which contain an attribute functionally determining itself. For example:
CampaignMgrID, CampaignMgrName → CampaignMgrName

Jukic, Vrbsky, Nestorov. *Database Systems Introduction to Databases and Data Warehouses.* Prospect Press, 2017