Database Design & SDLC

Software Development Life Cycle (SDLC)

SDLC

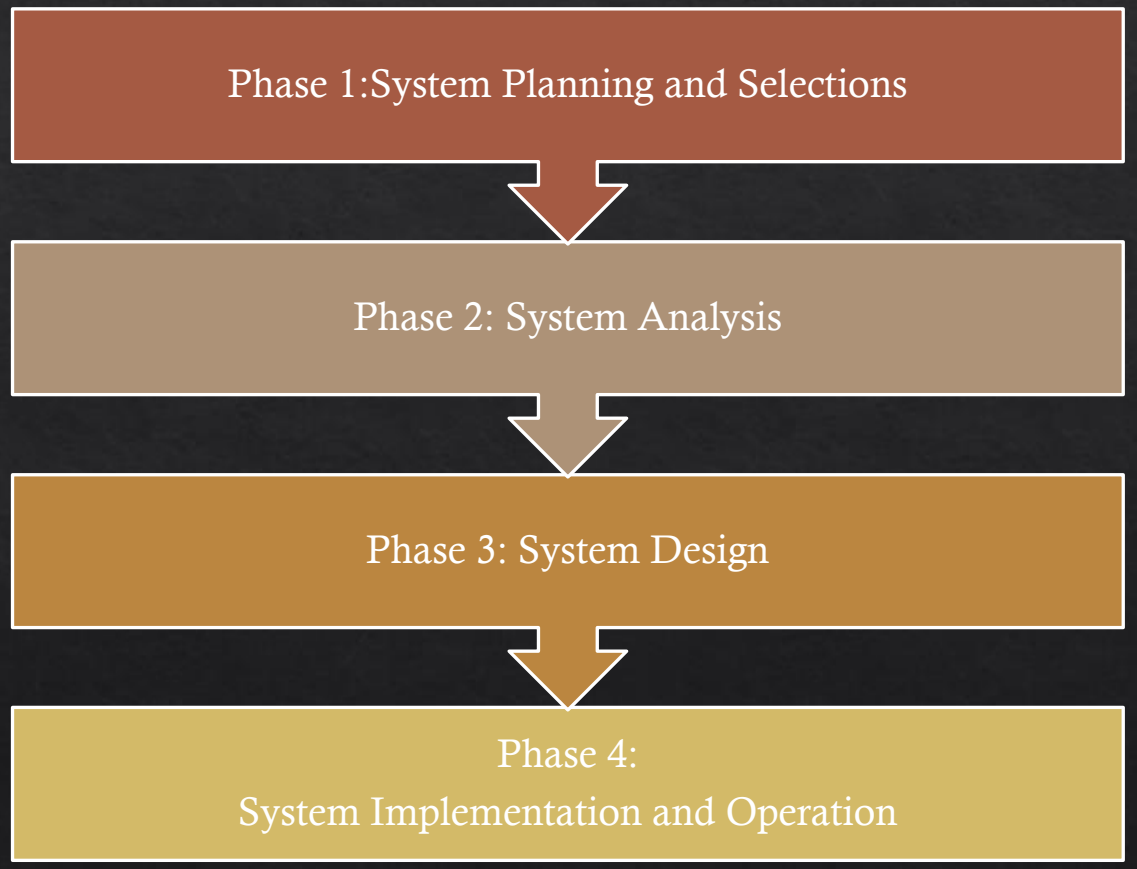
- SDLC has four basic phases
- Phase 1: System Planning and Selections
- Phase 2: System Analysis
- Phase 3: System Design
- Phase 4: System Implementation and Operation

SDLC

- The software development life cycle is the basic structure of creating software
- We are going to talk about a two different types:
  - Waterfall
  - Agile
  - There are more here: https://stackify.com/what-is-sdlc/

Software Development Life Cycle Waterfall

SDLC – Waterfall

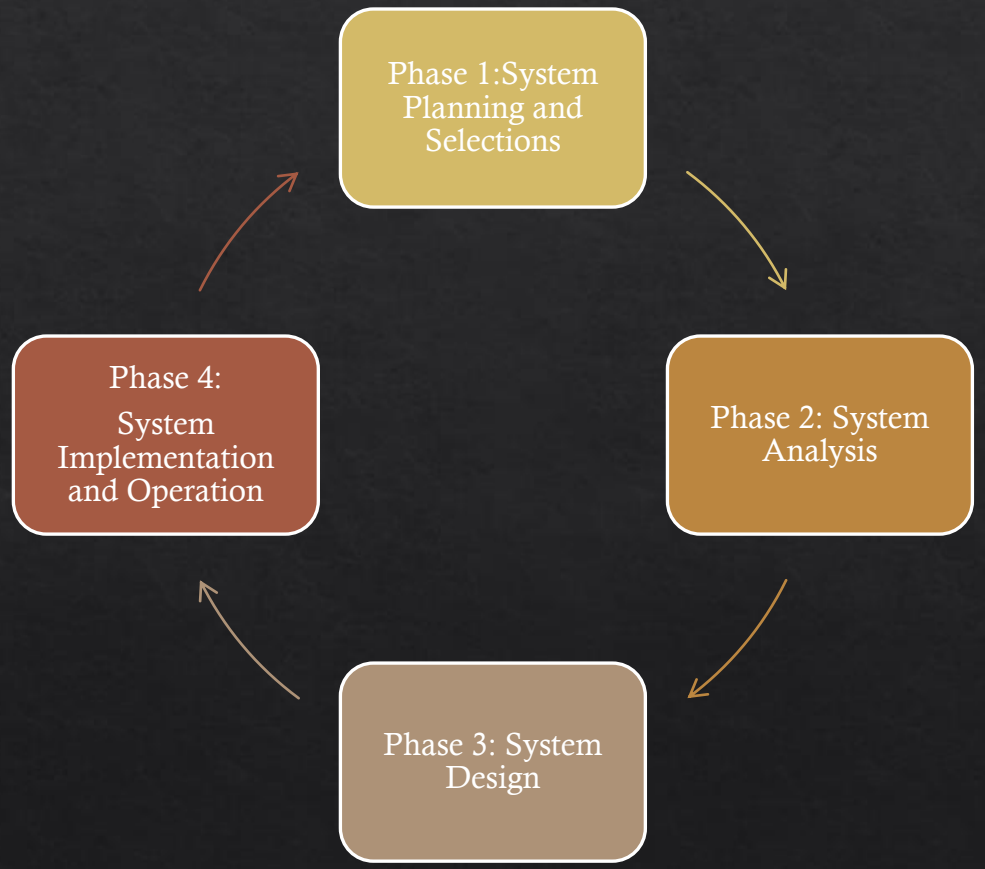


Waterfall

- Waterfall Software development is one of the oldest way to make software
- Waterfall phases flow like a water fall top to the bottom.
- Waterfall is a ridged approach that starts with phase one and ends at phase four
- Waterfall is the opposite of Agile
- Some industries need to be done this way
- Example, Health Care:
- Lets say you are making a pacemaker, and you use agile, you deploy the software and the person uses it then their pacemaker stops working, you as a company will not be in a good state, that's why using the waterfall method will allow you to be very strict on your flow and how things are deployed

Software Development Life Cycle Agile

Agile – Iterative



Agile

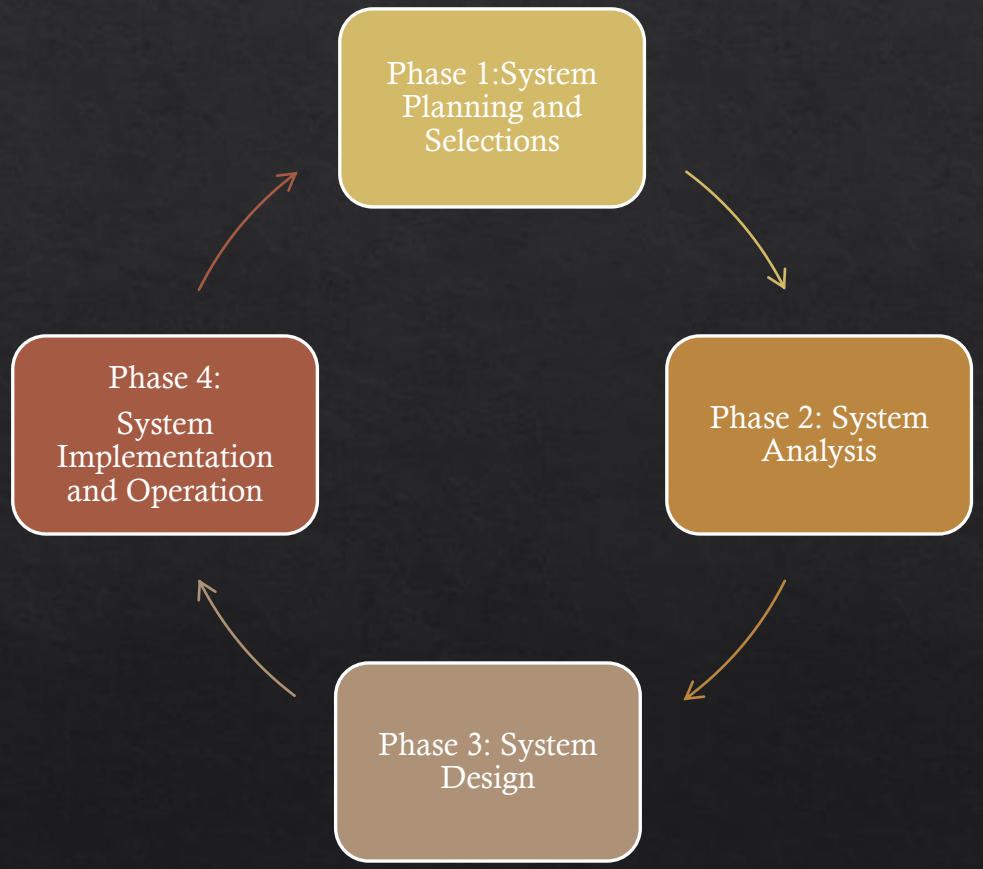
- Agile means that the process has a lot of flexibility.
- Agile moves through the phases when it needs to and can each phase out of order if need be
- It is common to use the Agile SDLC with an iterative method.
- With agile the idea is to ship frequently and test, test, test.
- Good is better than great.
- Example would be games, when people make games sometimes the first release is so buggy that most people hate the game, then in about 6 months, they get so much feed back that the are able to fix the bugs and the game become the greatest of all time.

Flow of Phases

Phases Flow

- Though Agile is flexible it will generally use an iterative flow through the phases.
- Different Models Cycle through each phase at different Paces
- Agile vs Waterfall are diametrically opposed in their pacing

SDLC - Software Development Life Cycle



Flow of Phases

- We constantly iterate over the Phases

1. Phase 1:System Planning and Selections
    - This is where we procure a generalized idea of what we want and why we want it.
2. Then based on what we want we do some analysis on what we need:
    - Databases, Teams, Mobile application
3. Design The System:
    - Creating architecture diagrams, Look at how we can build Phase 1
4. System Implementation and Operation
    - This is where you build it
5. Repeat, you are never done, what you create is perishable, if it stops growing, it dies

End Goal

- The goal is to create robust systems that live for a long time.
- We want a system that can be evolved with low coupling and high cohesion
  - This means we want it to be interdependent, or independent together
  - Low coupling is where parts are not coupled together
  - High cohesion means that they work in an integrated fashion

Phase 1: System Planning and Selections

- What you are doing is Requirements Analysis
- This is getting a broad scope of what needs to be done
- What are we trying to build?

Phase 2: System Analysis

- Now we have the requirements we ask if it can be done on time and on budget
- What will be needed

Phase 3: System Design

- We design the Architecture for all related components
  - Database
  - Apps
  - Websites
- We will putting our puzzle pieces together

Phase 4: System Implementation

- BUILD!!

More Phases??

- Yes, there are more phases that we can add such as:
  - Testing, test early, test often
  - Maintenance

Our Focus

- We have been working on phase 4, building
- Now what we are going to take a look at is phase 3 for our databases
- Phase 1 and 2 are more connected to the business side of things
- While 3 and 4 are more connected the development of the software itself

System Design

System Design – Deep Dive

- When we start with creating a database we could just start with CREATE DATABASE and CREATE TABLE
- But when it comes to actually creating databases, just jumping in head first can cause a lot of issues
- The goal of System design is to go from chaos to structure that can be understood and communicated
- As fun and creative as software is, we want to make sure we are having a business and logical approach to the design. This way we are ensuring we can communicate ease and understandable software to the end users.
- Have a *you* attitude, this means think about how much better can I make this experience for the user.
- What we are doing is distilling what the business people want to accomplish and turning it into a solid plan that can be executed with ease

System Design

- Data is the most important part of any system
- It's the driving force behind the day to day operations of any business
- Our job as database designers it to make sure this data keeps its integrity because the good data can make or break a business
- I know Carpet Cleaners, Marketers, Shop owners and more who all rely on good solid data

System Design

- There two main different techniques to design databases:
  - Top-Down
  - Bottom-up

Top Down vs Bottom-Up

- Depending on what phase your are in you can either do top-down or bottom up database design
- Top-down:
  - From general to a specific
- Bottom-up
  - From specific to general

Top-Down

Top-Down

- Top-down:
  - You are starting from a birds-eye view
  - Generally there is no existing system
  - And there is no existing data
  - Someone comes to you with phase 1 and 2 completed but now they need you to implement a database system
  - This is the one we are going to be working with

Bottom-Up

- Bottom-up
  - There already is an existing system
  - Or there is specific data is place
  - Example, you have a company with Google sheets filled with customer. The data is disorganized and they need to upgrade to a new system that will improve their data integrity

Starting to build Top-Down

- We are going to make a Driving school data base for RainyCity Driving School here in Vancouver
- Every school has instructors on payroll and an inventory of cars, trucks and motorcycles for teaching
- After Gathering all of our requitements we were able to gather an understanding of the company and it positioning and growth
- Their goal is to become known as the number 1 driving school in Canada
- Currently RainyCity has an outdated website and they get their customers by word of mouth
- The goal is to get more customers from their online presence
- Our goal is to implement what RainyCity wants

Building Top-Down: Requirements

1. There is a vehicle inventory for students to rent
2. There are employees at every branch
3. There is Maintenance for the vehicles
4. There is an optional exam at the end of your lessons
5. You can only take the exam twice, fail twice then you will have to take more lessons

Database Modeling

How do we Model a Database?

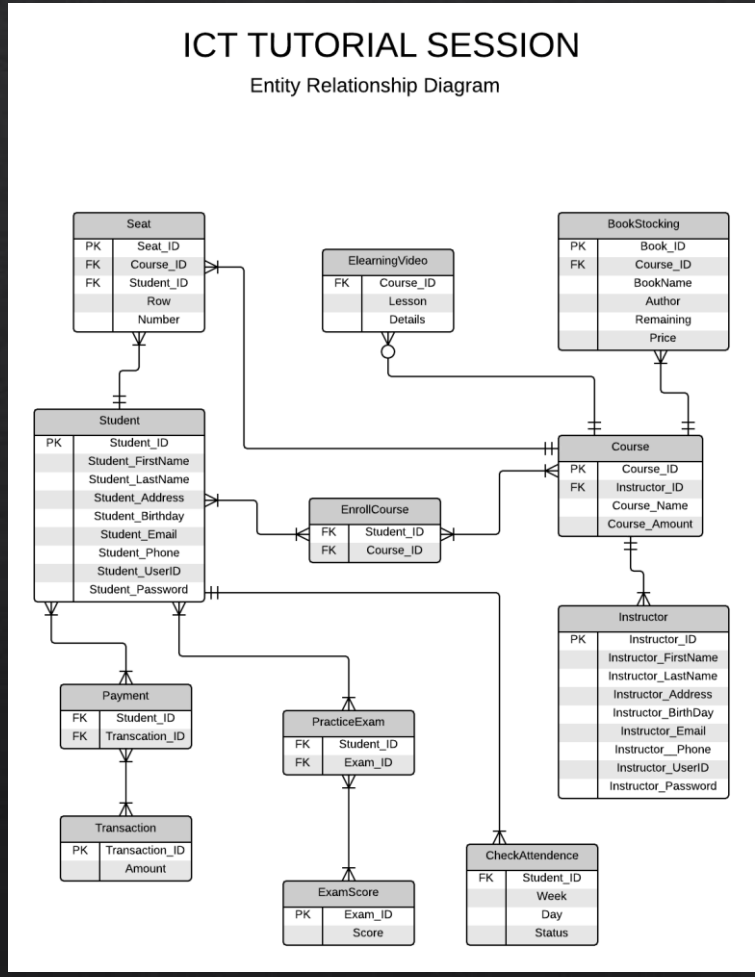
- We know with these requirements we will be doing a top-down design
- First let's look at what is necessary

Modeling Database

- Set our Goal
  - We need to create the database model based on their requirements
- How do we get Requirements?
  - First we have high-level requirements
    - These come from communication with the client assess their specific needs, and their implicit needs
    - Business owners they may not know how to properly communicate what they really need
    - Its up to us as the architects to distill their thoughts into concrete workable requirements
  - Second User Feedback
    - Talk to users and find out if there are any things we can add based on what the users think they need
  - Third Data Collection:
    - We can look at the current data and see what they already have
  - Fourth get a deep understanding of what they need
    - The moment you are most valuable to a client is the moment you understand their business and how it functions

ER Modeling

- Entity Relationship Modeling
- This is the relationship in Relational Databases



https://sites.google.com/site/runggaga12/entity-relationship-diagram

ER Model

- An ER Model is a diagram that functions as a way to structure high-level requirements
- This allows us to make the shape of our database
- The Squares are Entities and the lines are relationships

Step 1: Entities

- What is an entity?
  - A person place or thing
  - Has a singular name
  - Has an identifier – key
  - Should contain more than one instance of data

- Determine the entities in the system
- Give them meaningful names
- Don’t create relationships. Just give a first impression about the relationships, reasoning why we may want them

UML

- UML stands for Unified Modeling Language
- It is a standardized way of communicating software designs through charts
- https://www.lucidchart.com

Find the Entities in the Requirements

1. There is a vehicle for students to rent
2. There are employees at every branch
3. There is Maintenance for the vehicles
4. There is an optional exam at the end of your lessons
5. You can only take the exam twice, fail twice then you will have to take more lessons

RainyCity

- Core requirements broken down:

1. There is a vehicle for students to rent – Vehicle, Students
2. There are employees at every branch – Employees/Instructors, branch/schools
3. There is Maintenance for the vehicles – Maintenance
4. There is an optional exam at the end of your lessons – Exams, Lessons
5. You can only take the exam twice, fail twice then you will have to take more lessons

Making Entities

- Now we are going to make our UML
- When we are making our UML, we can name things based on the company we are working with, Calling Instructor vs Employee, School vs Branch
- When it comes to something like maintenance, this could either be an attribute(column) or it could be a stand alone entity
  - Using it as a stand alone entity would be best
  - This is because it might have more than one instance of data
  - If we break down what maintenance entails
  - We find it may have a date, what kind of maintenance, who did it, where did it happen

Making Entities

- Nothing is set in stone
- You may create your entities, then realize you are missing some, that’s ok
- Take your time and think it through, but also remember that you can iterate through this again and find more and better ways, Great is the enemy of the good

Making Entities

- When reasoning your relationships, say:
  - What does this have?
  - Is the relationship direct or indirect, the school may have vehicles but really the school only has vehicles because of lessons. This means that the vehicles may be more connected to the lessons vs the school. Or could the vehicle be connected to the student, which means that the student can have the same vehicle every time they take the lesson.
- If we connect school to instructor then instructor to student, that could cause issues if the student has a different instructor
- We could connect the School to lessons, if different schools have different lesson plans, like some places offer winter driving

Step 2: Attributes

- This is where we create the columns for our entities
- Lets think about this at a higher level than implementation
- Attributes of Attributes:
  - Must Be a property of the entity
    - You can't have a student with an instructors birthday
  - Must be atomic – one singular value:
    - Lets say we have an address, this is going to be too broad, we would need to break this down to street name, number, postal code, city, country etc.…
  - Single vs Multivalued
    - You may have an array of phone numbers, but they are still an atomic value of phone numbers
  - Keys

Relational Model

- We have talked a bit about primary keys and foreign keys
- In practice these are the keys which will be implemented
- In Theory, we may use other key concepts that will allow us to design our databases more effectively

Relation Schema

- This is more the abstract portion of the relational model the more we talk about the relational model we are talking about concepts
- The relation schema refers to the table schema which can be defined as
  - The data that is going into the table
- From a high level the relation schema looks like this:
  - We have a table called users
  - In the users table we have columns called id, FirstName, LastName, Gender, DOB

Id	firstName	lastName	Gender	DOB

Relation Instance

- The relation instance is the set of data that is inside the relation schema
- This is refereeing to all the data inside the table

Id	firstName	lastName	Gender	DOB
1	Ann	Bauer	F	1969-05-04
2	AL	Smith	M	1969-05-04
3	Sara	Ericson	A	1969-05-04
4	Man	Alpha	M	1969-05-04

Relation Key

- Relation Keys are the glue that hold our relationships together
- They are vital for us to understand how we can structure our relationships
- The relation key has two jobs:
  1. To uniquely identify each row
  2. To uniquely identify the relationships
- FirstNames and LastNames don’t matter because they are not unique, but the keys do, they are the bridge that creates the relationships

Super and Candidate Keys

The keys that allow us to distill a the primary key

Super Keys

- Super keys are a combination of one or more attributes which can uniquely identify a row
- This could be something like first name and last name, but this comes with issues because there could be more people with that first name and last name
- Even first name, last name and phone number, though this could cause issues too because people may have had the phone number before, and someone could have had it with the same first and last name.
- A better example is first name and email, emails are unique as no one can have the same email, so you could even use just an email.

Candidate Key

- A candidate key is a subset of the super key
- This is a more specific version of a super key
- This is a minimal amount of keys that need to be used to identify the row
- For example a super key technically could use all of the columns to identify the row, meanwhile a candidate key would break that down so we have the minimal amount of columns as our keys
- First name and email would be better than First name, last name, DOB, and phone number.

Primary Key

- If super keys are any combination of attributes to identify a row
- And a candidate key is the least amount of attributes to identify the row
- Then a Primary key is the materialization of the candidate key
- This is the key that allows us to uniquely identify our relationships
- This will often be the ID column as quite often you may not have any attribute that does the job
- You still could have multiple columns as a primary key, but this adds an extra layer of complexity to your design

Foreign Key

- Now that we have our primary key we then have to create relationships
- We do this with our forging keys
- We add a column of primary keys from another table

EmployeeId	FirstName	LastName	DOB	Email	ManagerID
1	Stella	Smith	1986	io@h.com	M1
2	Harry	Apple	1985	ps@o.com	M2
3	Steve	Fork	1978	r@s.com	M1

ManagerID	FirstName	LastName	DOB
M1	Jeff	Stew	1987
M2	Get	Back	1888

Foreign Key

- What we do is define the attribute as a Foreign key and link it to the table we want, now we can only fill out the rows with a primary key from the manager table
- The attribute(column) has an automatic constraint
- And only one employee can report to one manager, this is called a one to one relationship

More Keys!

- Super keys are groupings of attributes to identify the row
- Candidate keys are the minimum amount of columns that will be needed to identify the row and is the candidate for the primary key
- Primary Keys are keys that are chosen to identify the row
- Foreign keys are primary keys from a different table that are used to connect the tables relationships
- Compound keys are keys that use a foreign key and at least one other column as its identifier.
- Composite keys are keys that use more than one attribute to identify a row where all the attributes are from the same table
- Surrogate keys are primary keys that have nothing to do with our data, like an ID a non surrogate primary key would be an email

RainCity Attributes

- Tips:
  - When naming things make sure you use words the client uses, ask them if you aren't sure
  - When creating your attributes don't sweat doing it wrong, the back button was made for a reason
  - Don't worry about missing attributes, you can always add more later
  - We can include our relationships as a general rule

Step 3: Create Relationships

- We are now determining the relationships between our entities
- In step one and two we thought a bit about relationships, but now we are going to make them a little more concrete
- But what really is a relationship?
- Simply put, a relationship links entities by primary keys and foreign keys
- A relationship links two entities together in:
    1. 1 to 1
    2. 1 to many
    3. Many to many

1 to 1

- One to one is the idea that one record in one table has a relationship to one record in another table
- One to one examples:
- One Teacher Teaches at one school;
- One customerID has one contactID
- One Country has One capitol city

1 to Many

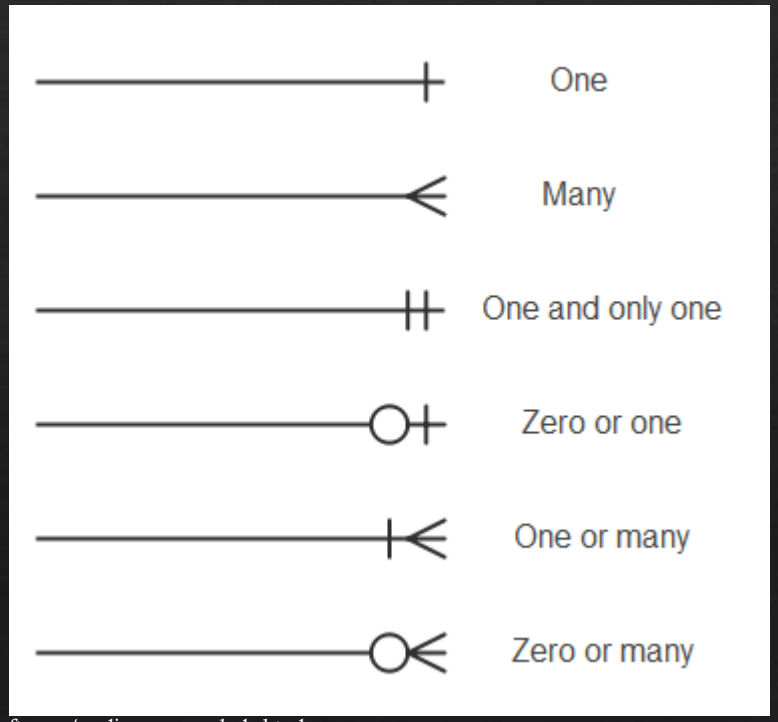
- This would be where one record has one relationship with many records in another table
- Example:
  - One Customer has Many orders
  - One teacher to Many courses, generally speaking a course will have one teacher though a teacher will have more then one courses
  - One country many provinces

Many To Many

- This is where multiple rows in one table have the id of another table and vice versa
- Example:
  - Students have multiple classes and classes may have multiple students
  - Movie theater has many movie showings and the movie may have multiple movie theaters its showing at

Relationships and Crows Feet

- In step three we are connecting entities with lines that tell us what the relationships is



https://www.edrawsoft.com/er-diagram-symbols.html

Creating the Relationships

- Back to Lucid Charts:

First Three Steps Summed Up

1: Entity

- Person place or thing
- Has a singular name
- Has an identifier
- Should contain more than one instance of data

2: Attribute

- Is a property of an entity
- Must be atomic – one singular value
- Be useful to the entity

3: Relationship

- Shows the association between two entities
- Has Cardinality 1 to 1, 1 to M, M to M
- Described as a verb or phrase. A student takes lessons, a school has students
- Is null or not null (Modality)
- Is it dependant or independent

Step 4 & 5

Step 4: Many To Many

- Now when it comes down to many to many it is best practice to avoid them
- You want to avoid it because there is:
  - Insert Overhead
  - Update overhead
  - Delete overhead
  - Potential redundancies
- It’s not always the case but in about 95% of the times there is someway you can find a solution that isn’t many to many

Many to many

- Solving many to many
  - Books and authors
  - we may have a book written by multiple people and those multiple people may have multiple books

Author_id_1	name	Book_id_1	Book_id_2
1	Abook	1	null

Book_id	name	Publish_date	Author_id_1	Author_id_2	Author_id_3
1	Abook	1/1/20	1	2	3

Many To Many Resolution

The resolution for many to many relationships to use intermediate tables

Author_id_1	name
1	Human
2	Person

Author_ID	book_id
1	1
2	1

book_id	name	Publish_date
1	Abook	1/1/20

Step 5: Subject Areas

- Divide entities into logical groups that are related
- Rules:
  - All entities must belong to a subject area
  - An entity can only belong to one subject area
  - You can nest subject area

Thank You!