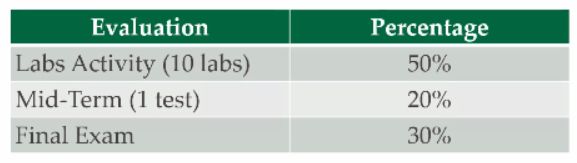
**Database Design & Administration - CST8250**

**Week 1 - January 11th**

Course structure:

*The exams in this course are non-cumulative*



* 2 hours in class and I hour online
* 2 hours of lab
* 4 hours study time

Keys:

* We must have the ability to uniquely identify each row in a table
  + This is achieved by creating keys for each row
  + This ensures that no row in a table can have exactly the same values for all attributes
* What would happen if we didn’t uniquely identify each row?
  + Data would be duplicated
  + We wouldn’t be able to count rows and get any meaningful information
  + The integrity, quality, accuracy, and reliability of our data would be compromised
  + **The main role of keys is to form relationships with other tables, how could we achieve this if the data in our tables were not organized? We wouldn’t be able to form relationships with other tables without keys.**
  + Writing queries would be difficult and complex – WHERE clause

Types of Keys:

* Many types of keys, these are the ones we must understand in this course:
  + **Primary Key:** one or many columns (attributes) that uniquely identify each row in a database
  + **Foreign Key**: a primary key from ANOTHER table that is used to form a relationship with the other table through the key
  + **Composite Key**: a primary key that consists of more than one column
    - It is composite because you are “Glueing shit together so that it works” - Dan Gudreault

*NOTE: these are NOT exclusive terms, a key can fit the criteria of all of these terms and be considered all three i.e. a primary key, a foreign key and a composite key*

Primary Keys:

* Primary Key: one or many columns (attributes) that uniquely identify each row in a database
* a key is typically a sequence of numbers such that no number will be repeated within the table
* it is possible to use other values as long as they will be guaranteed to be unique
* A primary key **CANNOT** be null! A primary key **MUST** be unique! It must have a value
* We use an auto-generated id when none of the attributes in the table are suitable as a primary key
* Ex: SIN (scarier to use), email address (generally pretty safe), auto-generated id (Dan’s favourite), driver’s license number (generally unique, but dangerous as your information can be compromised)
* Use numbers when you can!

Foreign Keys:

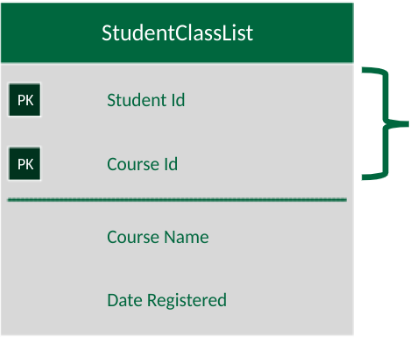
* Foreign Key: a primary key from ANOTHER table that is used to form a relationship with the other table through the key
* Does not have to follow the rules of a primary key: it does not need to be unique, and COULD be null
* A foreign key can become part of the primary key of the new table as it’s likely needed to uniquely identify each row in the table based on the relationship. If the foreign key becomes part of the primary key, it can no longer be null.
* Note that since this follows the definition of a primary key, a foreign key can also be one or more columns in a table
* **Uniquely identifies a row in another table**

Composite Keys:

* Composite Key: a primary key that consists of more than one column
* It is possible that more than one column in the table can be used to uniquely identify each row, in this case the primary key is also called a composite key
* If it makes sense to use existing columns in the table rather than adding an auto-generated id column, this will save some resources. One should consider complexity and clarity when choosing this option.
* There is no large benefit to using composite keys

Keys Example #1:

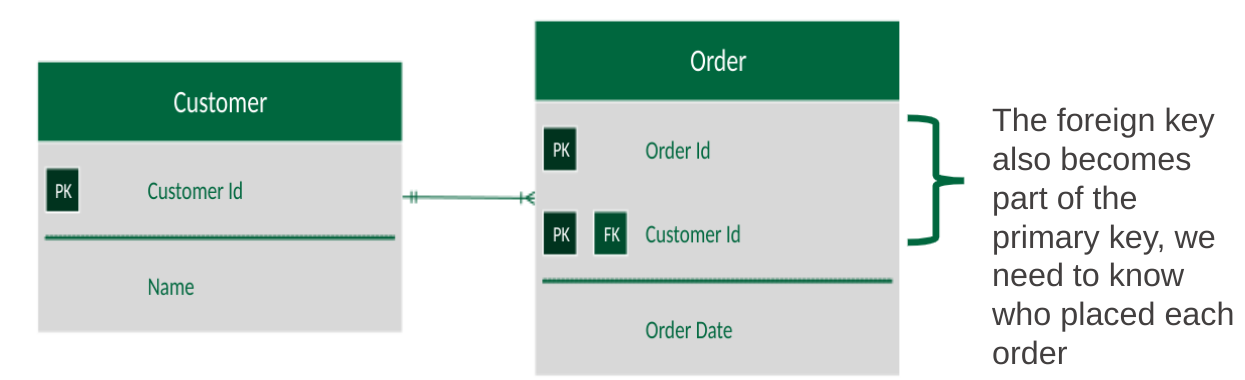
* Capture the historical data of courses a student has registered in.
* Both Student Id and Course Id are part of the primary key, together they form the composite key



* This table bridges two other tables together. This maps both directions of information.

Keys Example #2:

* Link the customer to the order.



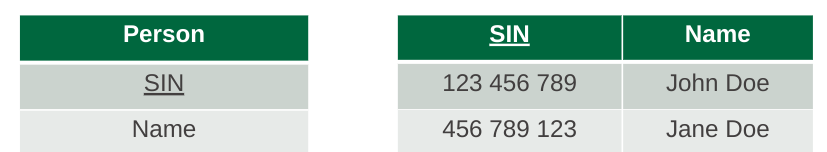
**Data Modelling Intro**

Basic Parts of a Table:

* A table is the physical representation, the “built house”
* All tables normally have:
* Table name: Person
* Columns identified by a name: SIN, Name
* Rows identified by a set of column values

(123 456 789, John Doe)

* A primary key. A primary key is used to uniquely identify a single row of data



**Entities**:

* An entity is the “rough sketch before the built house”
* Entity – a person, a place, an object, an event, or a concept in the user environment about which the organization wishes to maintain data
* Entity type – a collection of entities that share common properties or characteristics
* Entity instance – A single occurrence of an entity type

Entity Example:

* Entity – shortened word we use to refer to entity type
* Entity type – Person
* In the real world, entity and entity type are considered pretty much to be the same thing
* Entity instance – John Doe

**Attributes**:

* An attribute is a property or characteristic of an entity or a relationship
  + A description of an entity
* Often nouns
* All attributes have values
* For example for a prof this could be the course they teach, their office number etc.

Attributes Example:

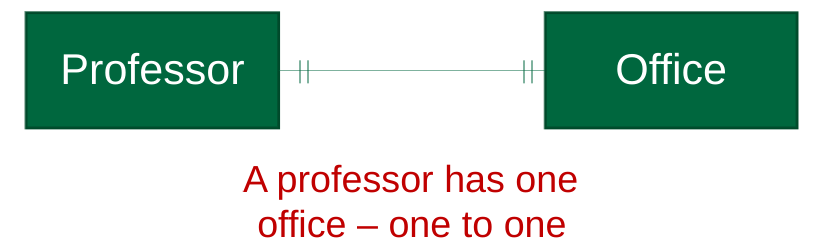
* An attribute is a *property* or characteristic of an entity or a relationship
  + A description of an entity
* A Person can be described by a SIN and their name
* Columns are attributes

**Relationships**:

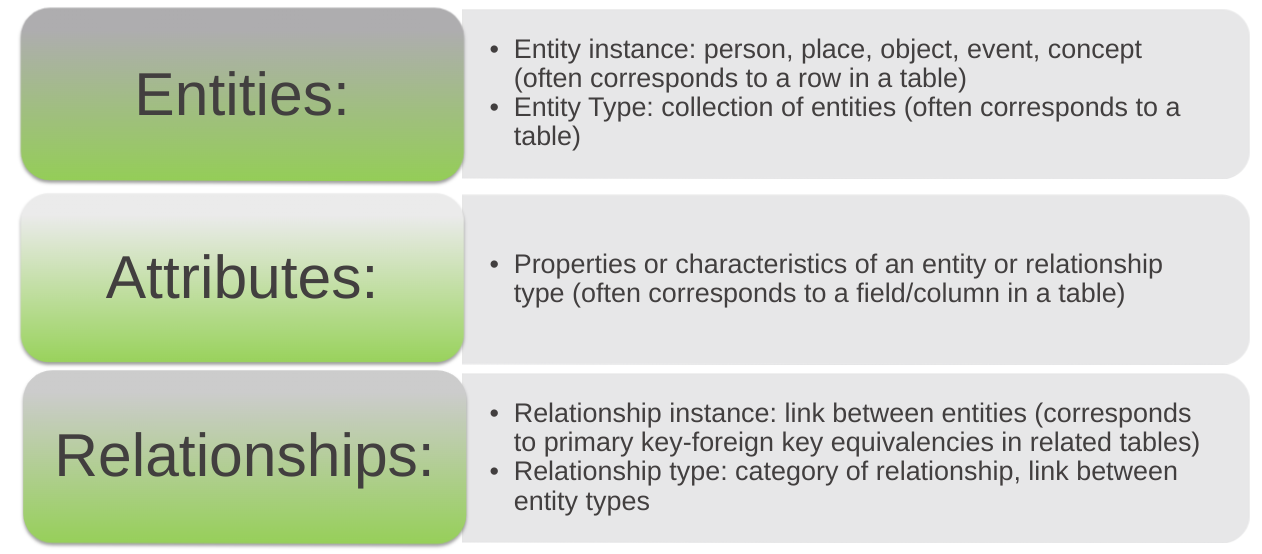
* Describes how entities associate with one another
* Often verbs (ex: has, teaches, attends etc.)
* Three types:
  + One to one
  + One to many
  + Many to many

Relationships Example:

* Describes how entities associate with one another
* We use Crow’s Foot notation in this course
* Crow’s foot is a visual representation of the rules of a relationship



**Entity-Relationship (ER) Model Constructs**



An Entity:

* SHOULD BE:
  + An object that will have many instances in the database
  + An object that will be composed of multiple attributes
  + An object that we are trying to model
* SHOULD NOT BE:
  + A user of the database system
  + An output of the database system (e.g. a report)

Types of Entities:

* Strong Entity
  + Has a primary key
  + Independent of other entities
* Weak Entity
  + Does not have a primary key
  + Dependent on another entity
* A loan is a strong entity as it can exist on its own and can be uniquely identified by a primary key such as loan\_number
* A payment is a weak entity as we cannot make a payment unless it has a Loan, it is dependent on Loan

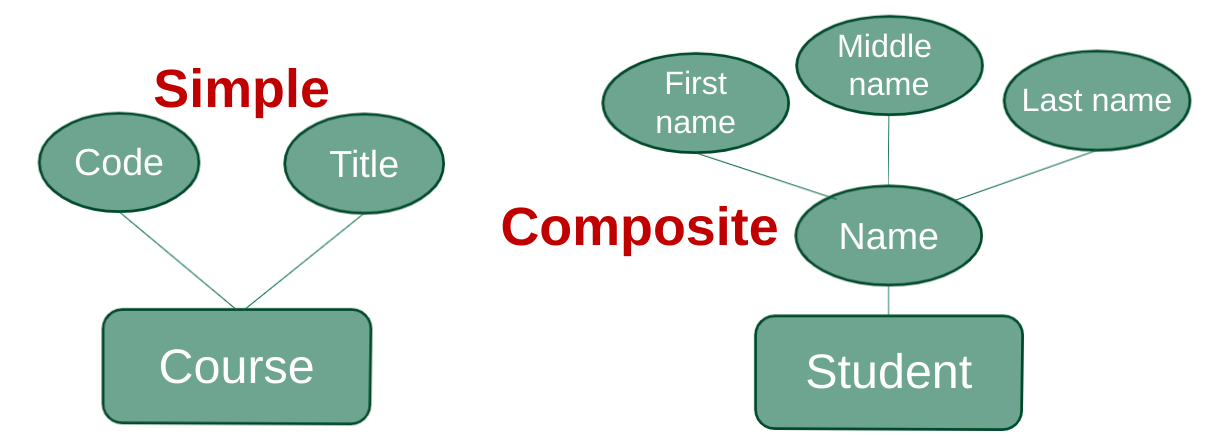
Types of **Attributes**:

* Attribute: property or characteristic of an entity or relationship type
* Classifications of attributes:
  + Required versus Optional Attributes (email address)
  + Simple versus Composite Attribute (Date of Birth vs Address)
  + Single-Valued versus Multivalued Attribute (Schools Attended)
  + Stored versus Derived Attributes (DOB vs age)
  + Identifier Attributes

Required vs. Optional Attributes

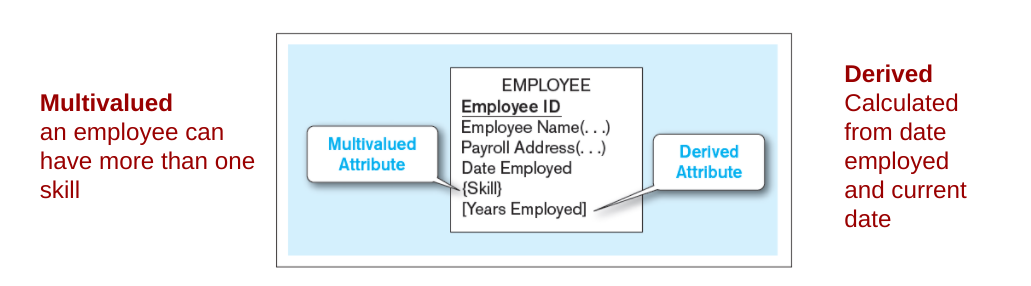
* Required – must have a value for every entity (or relationship) instance with which it is associated
* Optional – may not have a value for every entity (or relationship) instance with which it is associated

Simple vs. Composite Attributes

* Simple attribute – An single attribute that can hold one value
* Composite attribute – An attribute that has meaningful component parts (attributes)

Multivalued and Derived Attributes

* Multivalued – may take on more than one value for a given entity (or relationship) instance
* Derived – values can be calculated from related attribute values (not physically stored in the database)
* Entity with multivalued attribute (Skill) and derived attribute (Years Employed)

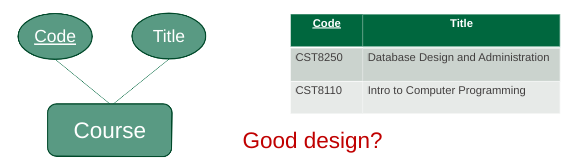


Defining Attributes:

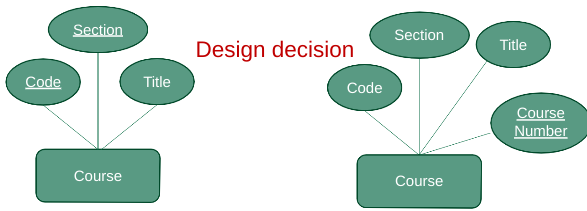
* State what the attribute is and possibly why it is important
* Make it clear what is and is not included in the attribute’s value
* Include aliases in documentation
* No one wants to inherit a database with undefined attribute names
* State source of values
* State whether attribute value can change once set
* Specify required vs. optional
* State min and max number of occurrences allowed
* Indicate relationships with other attributes

Identifiers (Keys)

* An Identifier becomes a key later
* Identifier (Key): an attribute (or combination of attributes) that uniquely identifies individual instances of an entity type
* Simple versus Composite Identifier
* Candidate Identifier: an attribute that could be an identifier, satisfies the requirements for being an identifier



* This is kind of a good design but does not take into account that there are multiple courses, multiple sections, terms etc. So what happens in that case:



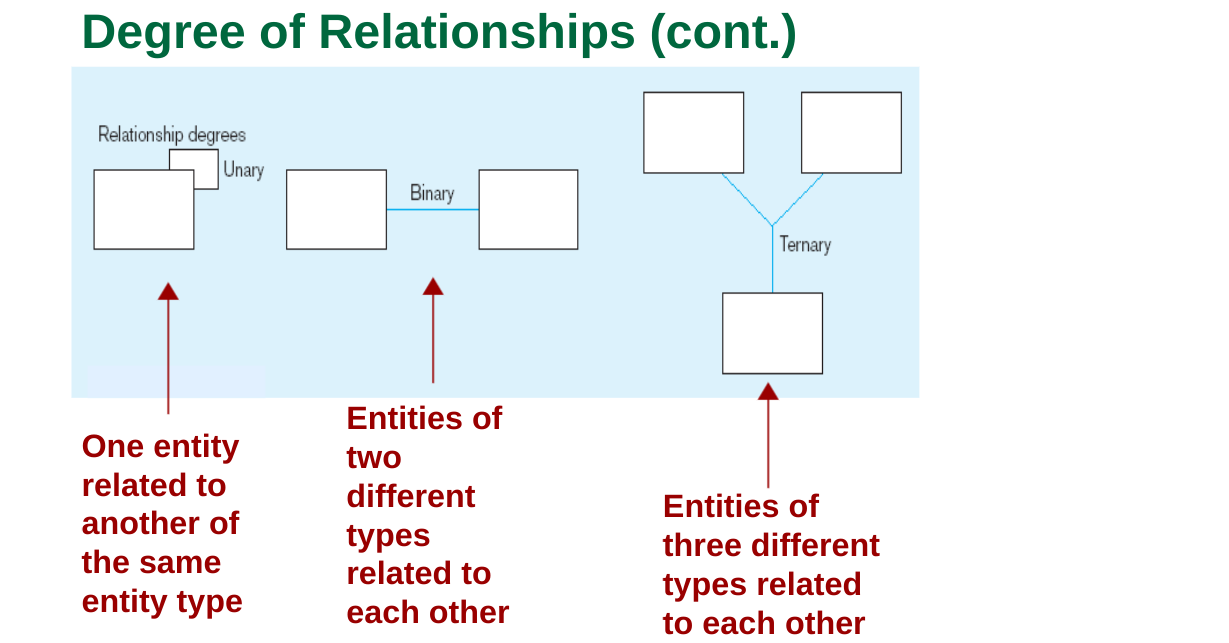
Criteria for Identifiers

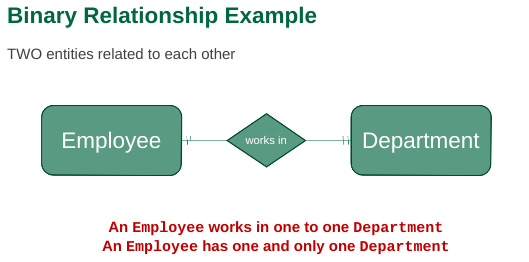
* Choose Identifiers that:
  + Will not change in value
  + Will not be null
* Avoid intelligent identifiers (e.g., containing locations or people that might change ex: never use a person's name because they can change that)
* Substitute new, simple keys for long, composite keys

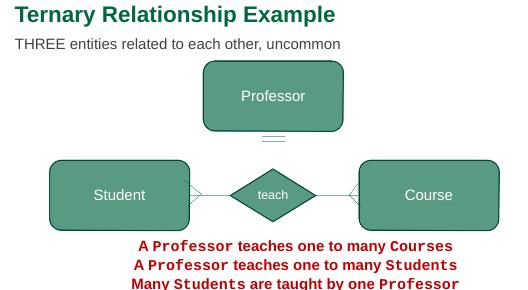
Types of Relationships

* One-to-One
  + Each entity in the relationship will have exactly one related entity
* One-to-Many
  + An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
* Many-to-Many
  + Entities on both sides of the relationship can have many related entities on the other side
  + “The alabama relationship” - Dan Gaudreault

Degree of Relationships

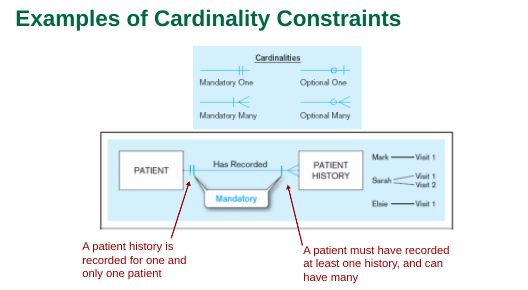
* Degree of a relationship is the number of entity types that participate in it
  + Unary Relationship
  + Binary Relationship (parent + child)
  + Ternary Relationship (party relationship because everything is connected)

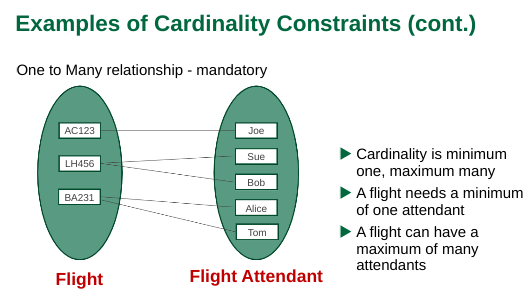


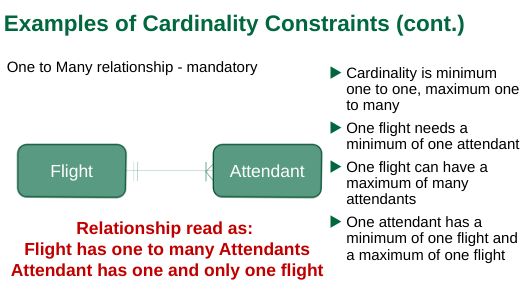


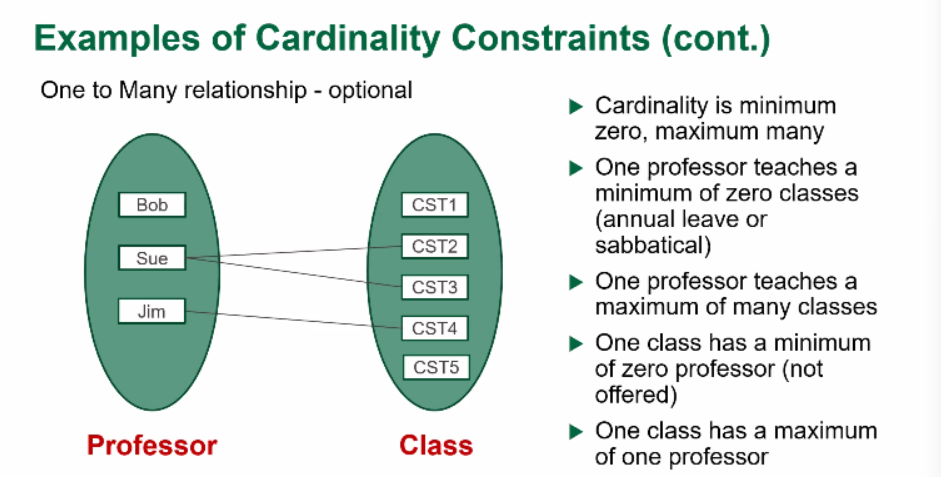
Cardinality Constraints

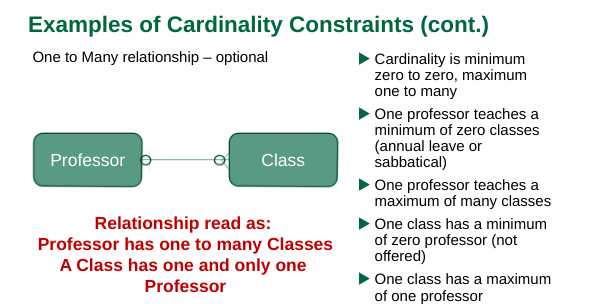
* Cardinality Constraints: the number of instances of one entity that can or must be associated with each instance of another entity
* We add minimum and maximum constraints to the relationship
* Minimum Cardinality
  + If zero, then the relationship is optional
  + If one or many, then mandatory
* Maximum Cardinality
  + The maximum number











Naming conventions:

* Used to be loose and free
* Because of original space constraints, naming used to be pretty cryptic
* Each company had its own standards
* Often each developer had his/her own version
* Could cause all kinds of grief
* Thanks to modern development frameworks, a de facto standard (not written down) is starting to emerge
* The concepts are still the same, but the approach is changing

Naming conventions used in this course:

* Everything is lower case. No exceptions.
* No spaces. Use an underscore.
* Tables are plural whenever possible. Exceptions would be names that imply plurality such as “log”
* Primary keys are called “id”. NOT “person id” or “user id”
* Foreign keys are named using the rule of” singular parent table name + underscore + primary key name”. i.e. “Users” table with PK “id” would have a foreign key name of “user\_id”.

*You only need to attend labs if you need help*