ACIT 1630 - Relational Database Design and SQL

More Relationships with Entity Relationship Diagrams (ERDs) in Draw.io

ACIT 1630 Relational Database Design and SQL
More Relationships with Entity Relationship Diagrams (ERDs)
in Draw.io

Introduction:

The goal of this document is to serve as a walkthrough of the creation of Entity Relationship Diagrams (ERDs) using Microsoft Draw.io.

There are 3 relationship types (sometimes referred to as connectivities and cardinalities):

1. One-to-One (1:1 or 1..1)
2. One-to-Many (1:M or 1..*)
3. Many-to-Many (M:M or M:N or *..*)

We will use Microsoft Draw.io to diagram the Many-To-Many relationship type as well as a unique self-referential unary relationship.

Step 1:
Open Draw.io

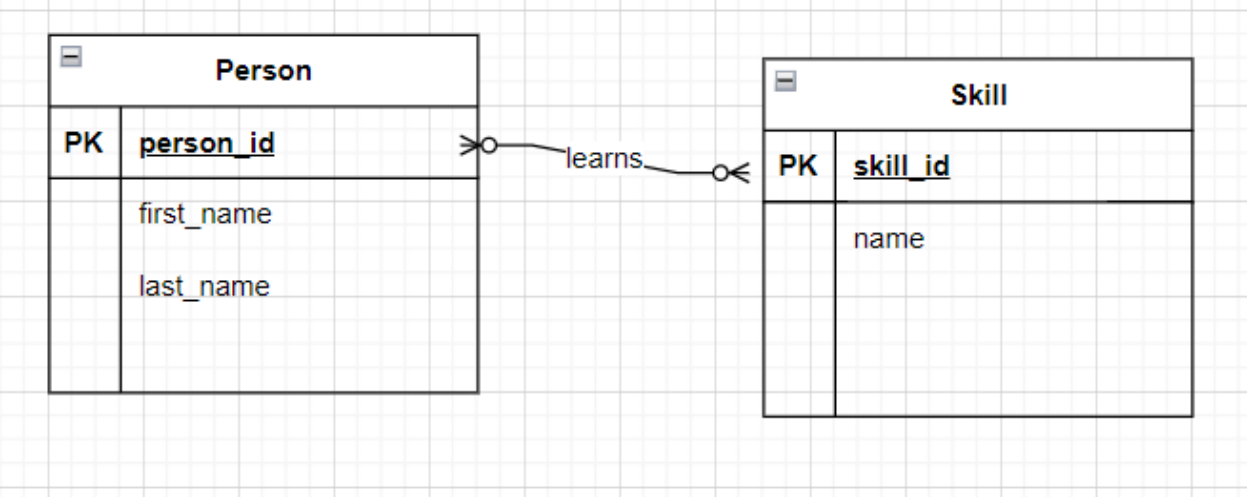
Step 2:
Create a new Entity Relation Diagram.
Expand the Entity Relation object tools and minimize the sandbox and general tools

Step 3:
Create a Many-to-Many Relationship between a Person and Skill.

People can have multiple skills and skills can be mastered by many people. A skill can be anything from Computers to Rollerblading, Sailing to Knitting, or Puzzle-solving to Cooking. Not everyone is going to have all these skills, nor is everyone going to have the same amount of skills.

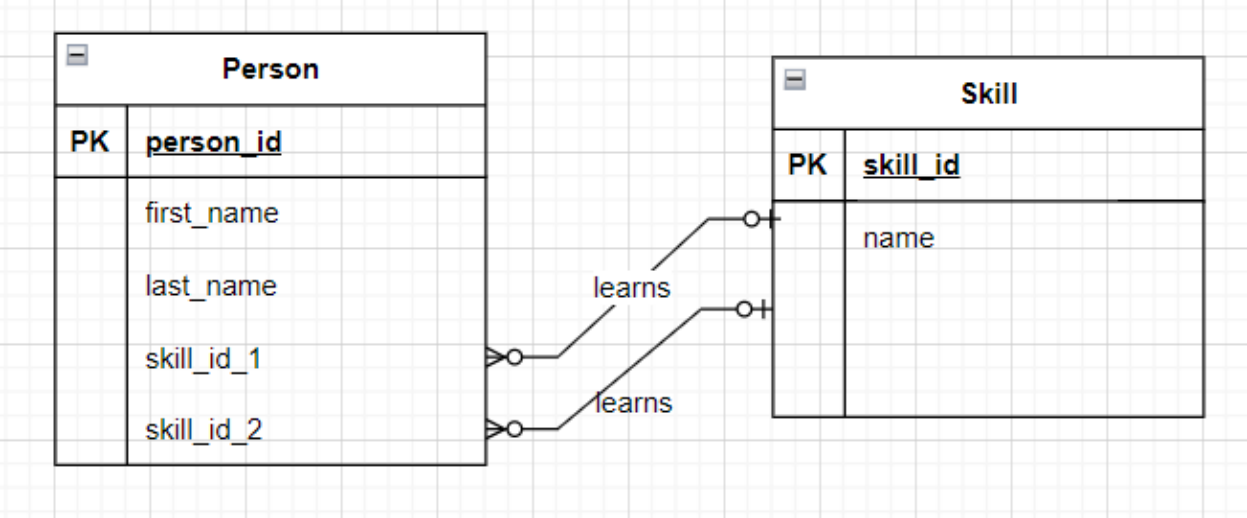
As we've seen in Draw.io (and this is the same in other ERD software as well) when we have a relationship between 2 entities, all we need to do to create a relationship between them, is add a line to represent it.

It is natural to represent our Person - Skill relationship like this:



But a problem comes up when we try to add a foreign key to represent this relationship. Which side do we add the foreign key to? To the Person Entity? If we add single Foreign Key to the Person Entity, then we can only have a *single* Skill for each Person; this results in a One-to-Many relationship, **not** a Many-to-Many. Add the Foreign Key to the Skill Entity? The same thing happens here, except now, Skills can *only* belong to a single Person.

We could allow for multiple foreign keys on the Person table like this:

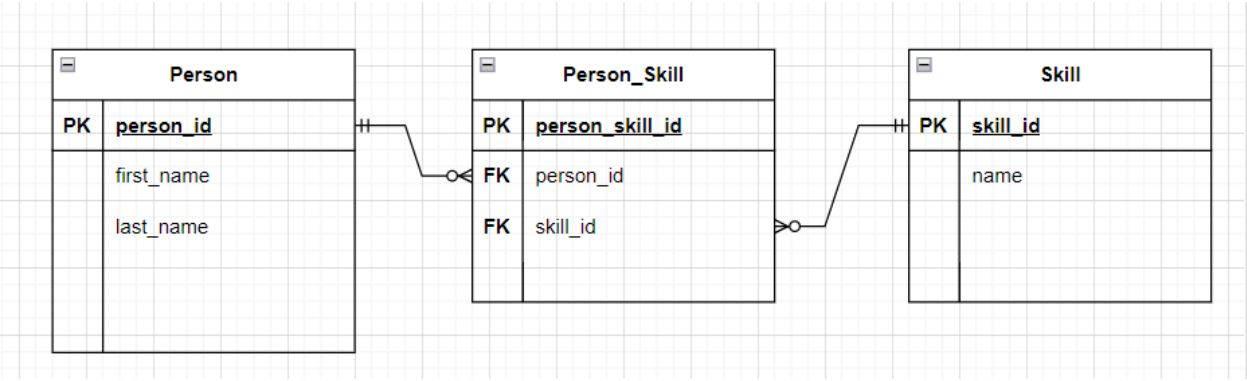


However, the downside of this is, it would limit the number of skills a person could have. Also, it adds a number of redundant columns, which if a person had fewer skills than someone else, all these other skills would be left blank (NULL) and would take up space in our database for nothing.

This is where we create a composite entity. A composite entity has many names:

- Composite entity
- Associative entity
- Bridging entity
- Linking entity
- Intermediate entity

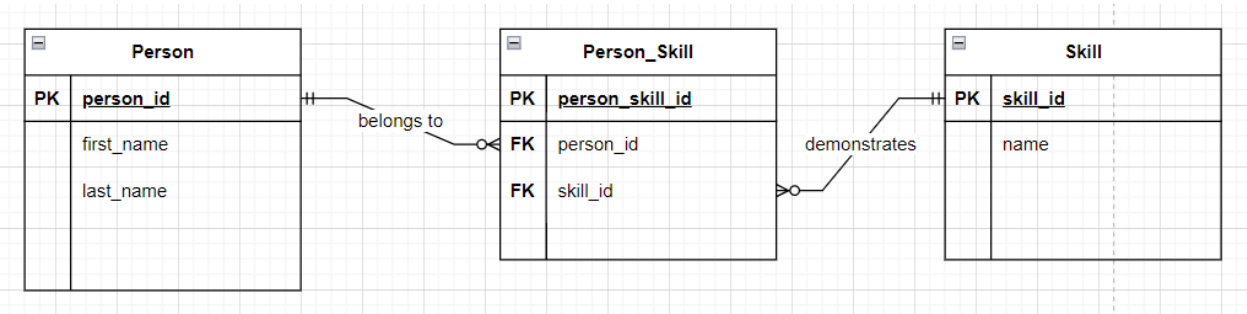
As the name(s) suggest, this composite entity exists to serve as an intermediate bridging entity to associate 2 entities in this Many-to-Many relationship to link them together. The result is the Many-to-Many relationship changes from a single Many-to-Many relationship to two Many-to-One relationships. Our composite entity goes in between the original 2 entities like so:



In our example our composite entity is the Person_Skill entity. This would represent a physical table in our database with foreign keys to both the Person and Skill tables. This now allows for multiple people to have as many skills as the want, but without the need to add multiple (and possibly unnecessary

columns) in the person table. One note, we often add a requirement on the combination of our foreign keys to be unique. This eliminates the possibility of a single person having the same skill multiple times (which doesn't make sense).

Use Draw.io to create the Person - Skill Many-to-Many Relationship like this:

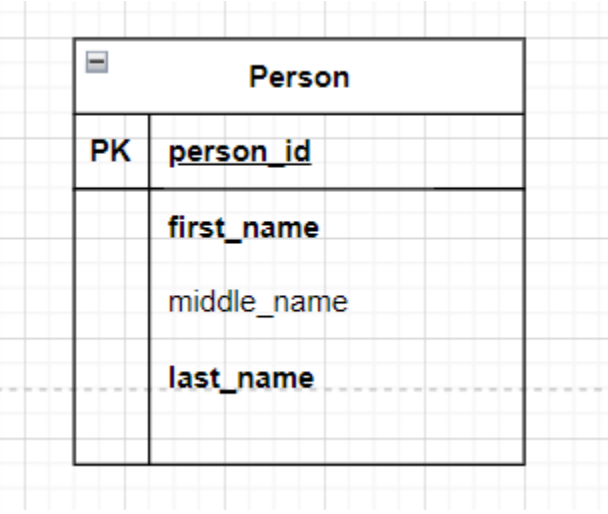


Step 4:
Set all required attributes to 'required'.

In Draw.io we can indicate that our entity attributes are required. They will show up in **bold text**. Optional attributes are then indicated by normal text (non-bold). Unless we mark attributes as required, they will default to optional.

To set an attribute as required in Draw.io, select the attribute by double-clicking on the name, while the text is highlight hit the CTRL and B keys

For the Person Entity below all attributes are required, except middle_name. The middle_name attribute is optional.



In the database we can implement this with the "Allows Null" property of the table column. Allowing NULL means the field is *optional*. Not allowing NULL means the field is *required*. When entering data, marking a field in a row as NULL means that it is was not recorded. Kind of like leaving the email field blank on a form on a website when you don't subscribe to the weekly newsletter - it's optional.

In our diagram mark all fields as required.
Person:

attribute name	required
person_id	Yes
first_name	Yes
last_name	Yes

Person_Skill:

attribute name	required
person_skill_id	Yes
person_id	Yes
skill_id	Yes

Skill:

attribute name	required
skill_id	Yes
name	Yes

Step 5:
Add attribute data types to our ERD.

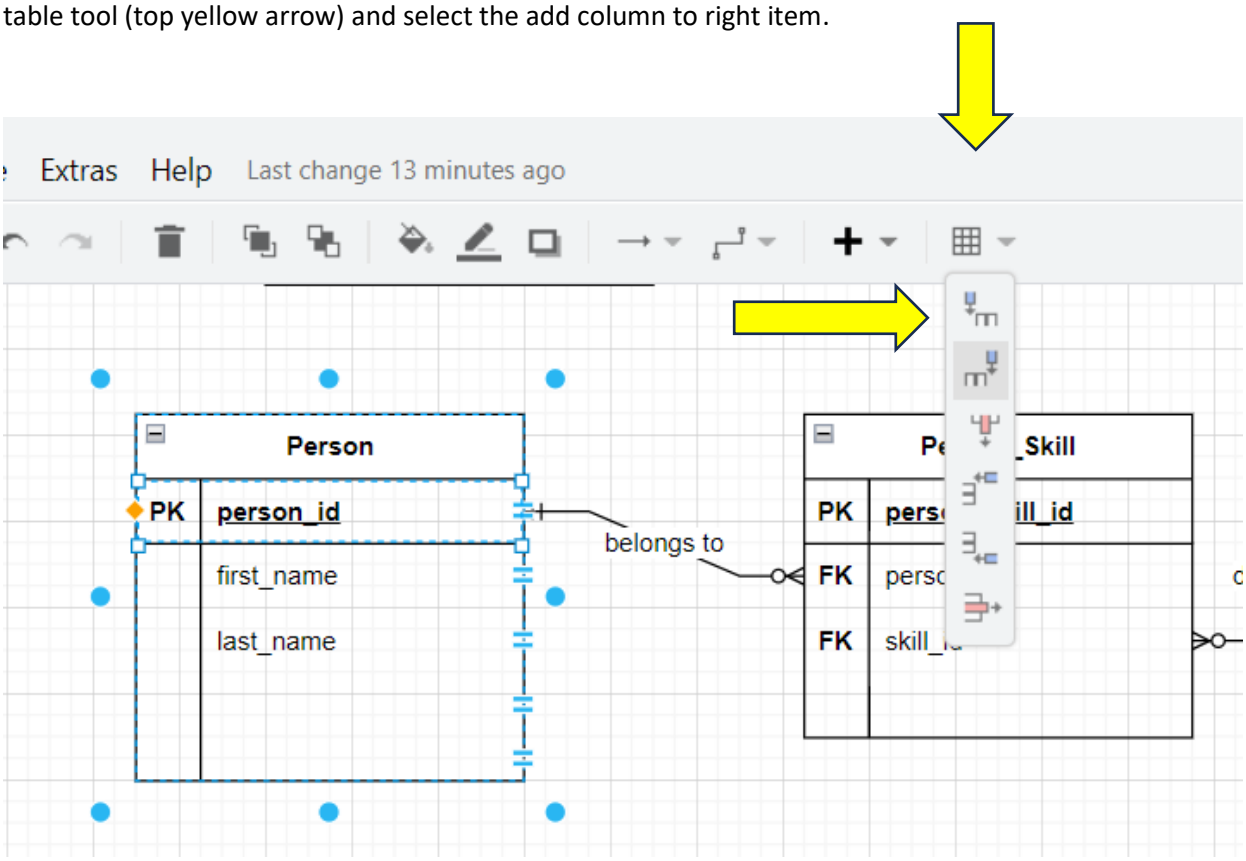
Our ERD can also include a suggestion for which data type would be appropriate for our database.

ERDs can either be very general and leave all the implementation details out, or can be very specific and and include many very specific implementation details. This is a choice the designer makes when they create the ERD. Sometimes when designers create the ERD, the specific database software (DBMS) has not been determined yet, so it doesn't make sense to include details specific to Microsoft SQL Server data types for example if the DBMS eventually changes to use MySQL, Oracle or PostgreSQL.

For our attribute data types, we use generic data types that could apply to any DBMS:

- Use **number** for whole numbers such as 0, -120 and 37.
- Use **text** for names and descriptions that are text.
- Use **date** for values relating to dates or times.
- Use **money** for values relating to financial data or where rounding errors could be problematic.
- Use **float** for fractional numbers that are very large or very small (positive or negative).

To add attribute data types to our ERD in Draw.io we will click on our entities and click on the menu bar table tool (top yellow arrow) and select the add column to right item.



Often the attribute data type column is too small to show the text and it wraps to 2 lines. You can resize the column by hovering over the border and dragging it to an appropriate size. Remember that you can always use the undo icon or CTRL Z to undo mistakes.

Resize the entity and the attribute type section to the appropriate sizes.

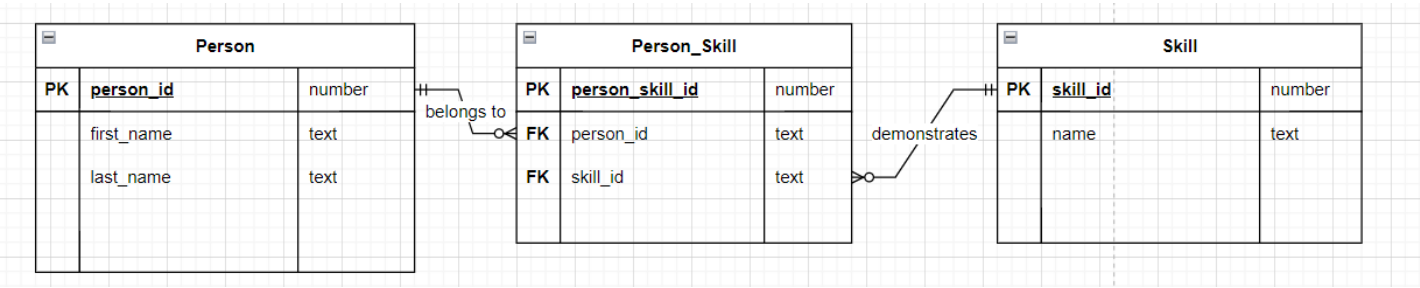
Add the following attribute data types to your entities:

Person	
attribute name	**data type**
person_id	number
first_name	text
last_name	text

Person_Skill	
attribute name	**data type**
person_skill_id	number
person_id	number
skill_id	number

Skill	
attribute name	**data type**
skill_id	number
name	text

Your finished Many-to-Many relationship (that we converted to two Many-to-One relationships using a composite entity) should look like this:

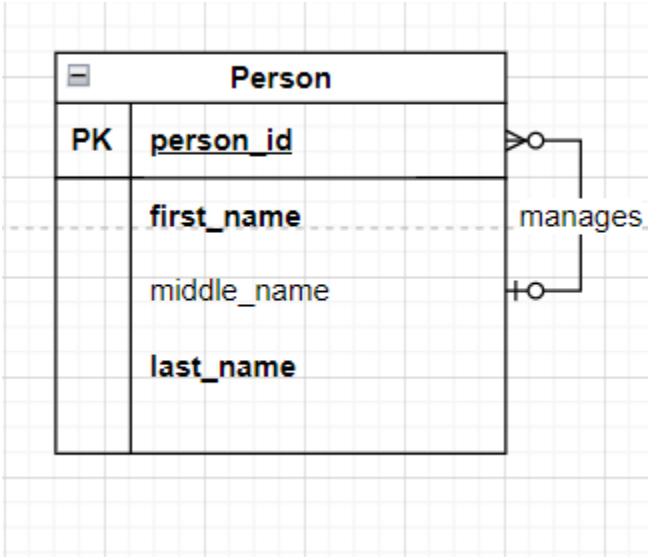


Step 6:
Create an ERD model of a self-referential (unary) relationship.

A self-referential or unary relationship can occur when an entity instance (or row in a table) relates to another entity instance within the same entity (or table). A few examples include:

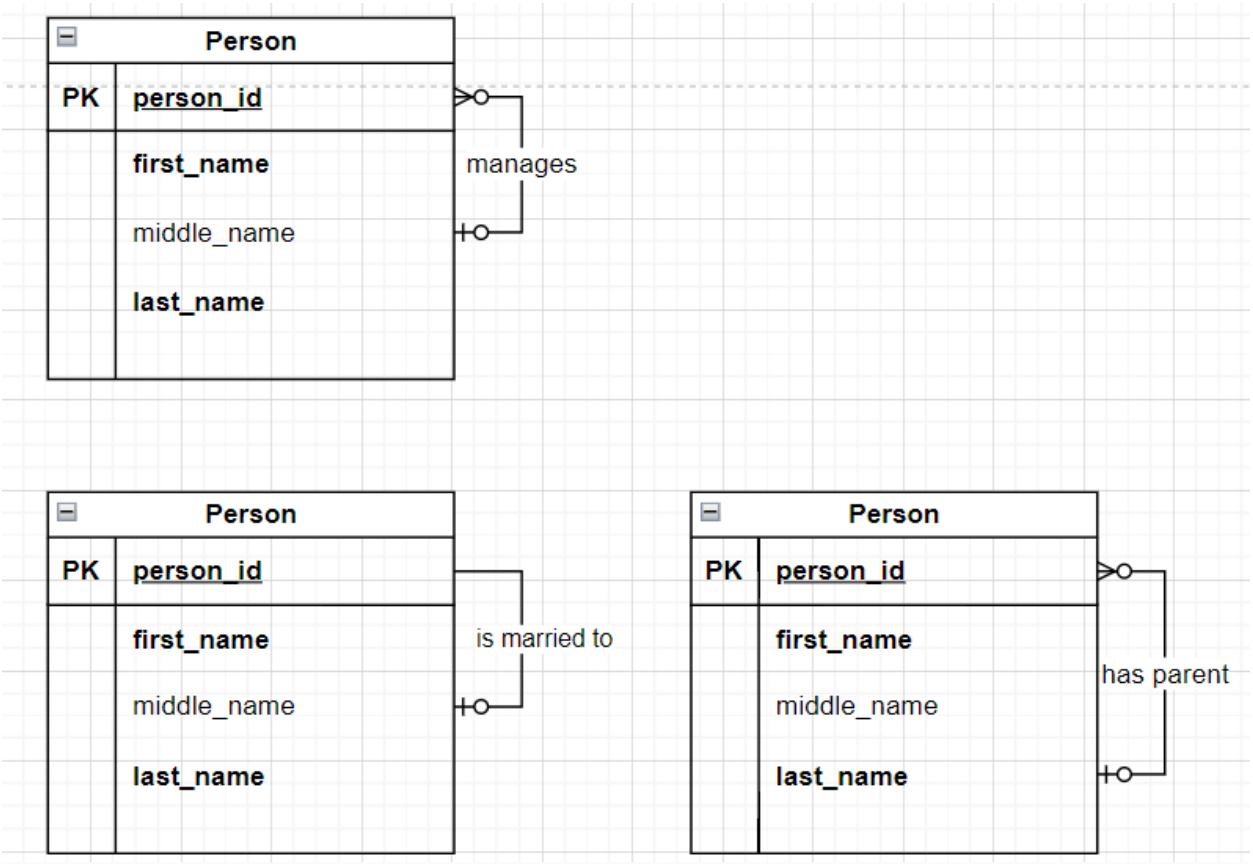
- Courses that have other courses as pre-requisites.
- Employees that have other employees as their bosses and managers.
- People that are married to other people.
- People have parents (other people).
- Parts that are made up of other smaller parts.
- Ingredients in a recipe that can be made up other ingredients.

A Unary relationship is modeled by drawing a line from the entity back to itself like this:



In a Unary relationship, it is particularly important to label the relationship by giving it a name.

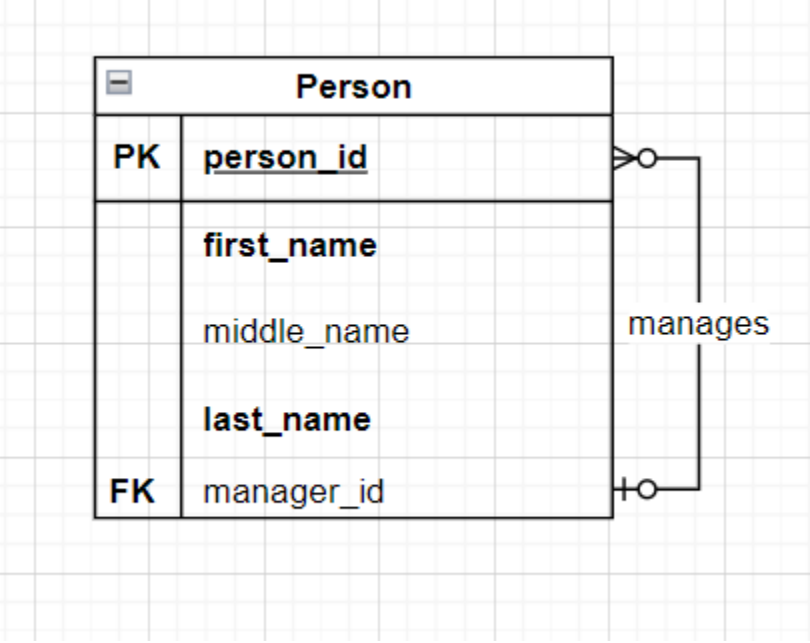
different relationship types



Because Draw.io does not have a one optional to one optional the married to relationship is drawn with the most accurate type. This can still be read as “if the optional relationship of is-married-to exists, then it must be to one other person”.

All of these are Unary relationships on the Person Entity, but represent ***very*** different relationships. The only way to tell them apart is by reading the relationship label. Keep in mind too, that there can multiple simultaneous unary relationships on the same table at the same time. You may want to know everyone's boss, spouse and children and have them all stored in the same person table!

Create a Person Entity that has a manager that is also a Person.



Create a new Person Entity.
Add to your Person Entity a foreign key called manager_id (which will reference person_id in the Person Entity). As indicated, draw a unary relationship with the name "manages".

Since the manager relationship indicates that not everyone will have a manager, we will make the manager_id foreign key optional.

ACIT 1630 Relational Database Design and SQL
More Relationships with Entity Relationship Diagrams (ERDs)
in Draw.io

Save your diagram (as below) and submit it to the learning hub at learn.bcit.ca.
Your drawing should include **both** your Person - Skill relationship and your unary Person relationship.

Filename: **01_ERD_with_Many_to_Many_Relationship.xml** or .jpg or .gif

Submit