ONE TO MANY

*Example:* Dojo >> Ninjas, Ninjas, Ninjas

    @classmethod

    def get\_one\_dojo(cls, data):

        query = """

            SELECT \* FROM dojos

            LEFT JOIN ninjas ON ninjas.dojo\_id = dojos.id

            WHERE dojos.id = %(id)s;

            """ #this is only showing us the things, but we're not storing it yet

        results = connectToMySQL('dojos\_ninjas').query\_db(query, data)

        dojo\_instance = cls(results[0]) #this is passing up to the class Dojo and creating an instance. Without a space for the ninjas, they won't get saved.

        # pprint(results, sort\_dicts=False, width=1)

        for row in results: #refers to the specfic row of the dictionary coming back

            if row['ninjas.id'] == None: #this helps to avoid getting Null data, it would send back the Dojo data, and just ignore the ninja stuff

                return dojo\_instance

            data = { #row must match in both places, and it will update the information passing through, but those things that are named the same in both places like id and created/updated at, you must \_\_. the table you are wanting the information saved from.

                "id" : row['ninjas.id'],

                "first\_name" : row['first\_name'],

                "last\_name" : row['last\_name'],

                "age" : row['age'],

                "dojo\_id" : row['dojo\_id'],

                "created\_at" : row['ninjas.created\_at'],

                "updated\_at" : row['ninjas.updated\_at']

            } #this is for the ninja information

            dojo\_instance.ninjas.append(ninja.Ninja(data)) #this matches the dojo we made on line 29, the ninjas matches up the line inside our constructor on line 10. The stuff inside the parenthesis is

            #code is read inner most ()'s. So we're starting with our data first, going into our Ninja class by using the lowercase ninja file to access it. Then we append to the ninjas in our current model, and the empty list on our dojo object.

        return dojo\_instance

* We run a query, starting with the ONE that will be holding the MANY. Using a LEFT JOIN will guarantee that we can see all Dojo’s, even when there are no ninjas available. If we just do a JOIN and there are no ninjas, it will error/display nothing.

**LEFT JOIN EXAMPLE**

Graphical user interface, text, application, email

Description automatically generated

**JOIN EXAMPLE**

Graphical user interface, text, application, email

Description automatically generated

* Next, we will run our query and make sure to pass the data we are sending it. \*WHENEVER\* you are passing a %()s you need to make sure that data is being passed. Then, you can create the variable results to hold all the data coming through from your query.
* Since we are focusing on the ONE side of things, we will use results[0] to reference just the first row of Dojos, since every line there after will always remain the same Dojo. The only thing that would be differently are the ninjas coming through. We also want to make sure to keep this outside of our for loop because there is no reason to loop through it, since again, it is our ONE.

\*Cls(results[0]) is the process of actually creating the instance of the Dojo (since that is the page we should be in, our Dojo.)\*

* Next, we do an if check to help prevent any potential errors. If there are no ninja’s coming through, then it will send back just the dojo data we received, and we do not need to proceed into our loop if this is the case.
* Now, inside of our loop we’re going to create a dictionary to grab all the data being passed through from the other table, our MANY. In this case, we need to make sure that the ninja data matches the key values of what is in the ninja instance since we will be creating our ninjas this way as well. \*If there is anything that is named the same way in the Dojo table, then we need to make sure we prepend the ninja data with the name of its table: ninjas., so we can be sure that the correct data is getting added/overwritten.

Text

Description automatically generated

MANY TO ONE