Lab 4c: In-class database CrUD lab:

After all of the watching and following along, here is a culminating in-class assignment for this part of the course.

Recall that in Lab 4c, we used the Department table in our company database and performed crud operations on that table. We added validators and new form field types to the forms.
We used inheritance in both the form classes and the templates.
We also ran several different types of queries using flask-sqlalchemy.

For this assignment, which will be (mostly, I hope) completed in class, we'll apply these skills to another part of the company database.

Consider the following three tables in the company database:

employee
project
works-on

There is a m:n relationship between employee:project, and that relationship is captured in works-on.

Using Lab8_CrUD-lab-update-inherit-solution that we use in Lab 4b as a base, please modify it to achieve the following:

Create a web app that allows you to:

- list which employees are assigned to which projects.
  This list should include the Employee Name and the Project Name.
  For debug purposes, you may want to also include the Employee ssn and the Project#.
- assign an employee to a project
- remove an employee from a project

There is no Update functionality required in this lab, unlike our example, which did have an update.

Please include:

- appropriate queries, probably including a 3-table join (as demonstrated in one of the labs)
- using drop down boxes when appropriate:
  Please remember that in our drop-down box, we wanted the ssn to be both the text that was viewed and ALSO the actual value that was selected and returned. Is this what you want?
  Possibly, depending on how you structure your program, you may want to display an employee name, but then select the Employee's SSN (primary key) for that employee.
  Or possibly not (your choice)--but be aware of the difference.
  You will probably want to show two drop-down boxes, one for Employee and one for Project.
  This would be for the Create New Employee-Project Assignment functionality.
  There are notes at the end of this document with hints on how to achieve this, and some limitations.
- appropriate links on the nav bar to allow the user to easily navigate to the different options.

Everything that you need to know for this project was taught in the previous labs, with two minor (but possibly challenging) exceptions. Those two exceptions are how to pass more than one parameter from the form to the route; and how to use two fields in your query.find (since you are dealing with a composite key). However, both of these use syntax that is a simple extension of what we have done previously in labs.

You may work with two other students. I will be walking around and interrupting the class periodically to help.
Online students may also work with two other students. Your team may set up a zoom meeting for help.

Below is a suggestion of a step-by-step process for the delete function (which is the easier function to implement).
It is only one possible way to implement the solution, and you are free to choose other ways.
I have provided you with screen shots that go along with this solution.
For the "assign" (create a new assignment) part of the solution, you're on your own!! (But I provide hints below…)
I chose this implementation, because it closely follows the labs we did in class.
Good luck!!

Began with Lab8_CrUD-lab-update-inherit-solution

__init.py__ no changes
Models.py no changes
Routes.py

Start with the join.
Model this join after the "results2" join from original lab, except change the tables, key, columns
Please note that you will need this query when the user want to view the current employee-project assignments or when the user wants to delete a current employee-project assignment.
You do not need that query when you are only going to insert a new project assignment.

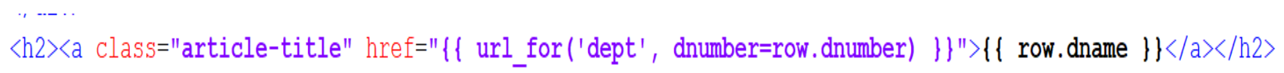
Templates\join.html

- Tested it to see if the join worked: (I will not use join.html in my final version. For testing only.)

If you want to use join.html for testing, be sure to change the field names in the template

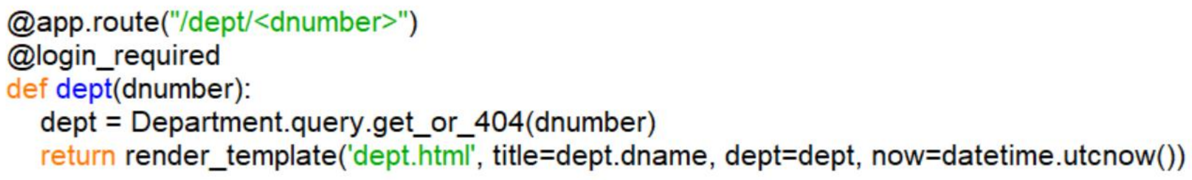
- Created a new template**: assign_home.html,** modeled after **dept_home.html**
  - Changed for *row in outstring* to *for row in joined_m_n %*
  - Changed the *url_for* to *assign* instead of *dept*
  - Added all of the relevant fields to the *url_for*

This is one of the "new" things. In dept_home.html, we passed dnumber:



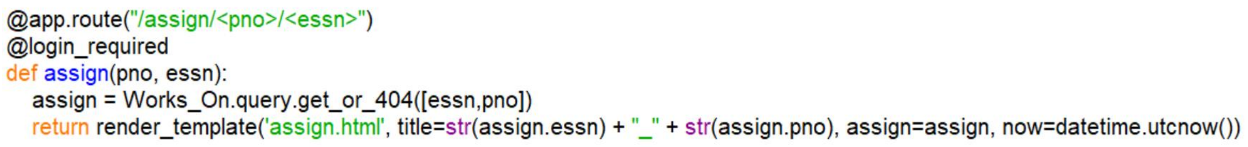
```
<h2><a class="article-title" href="{{ url_for('dept', dnumber=row.dnumber) }}">{{ row.dname }}</a></h2>
```

The url_for (above) is for the function dept, which is expecting one parameter, dnumber.
But you can add more fields to pass back to the route's function. Just separate by commas.
The reason we passed dnumber back to the route is because it's the primary key and we wanted to use that key in our query:



```
@app.route("/dept/<dnumber>")
@login_required
def dept(dnumber):
   dept = Department.query.get_or_404(dnumber)
   return render_template('dept.html', title=dept.dname, dept=dept, now=datetime.utcnow())
```

But we would like to create a page called "assign" or "assignments" or something like that.
Since our primary key is composed of two variables, we want to pass both of them:

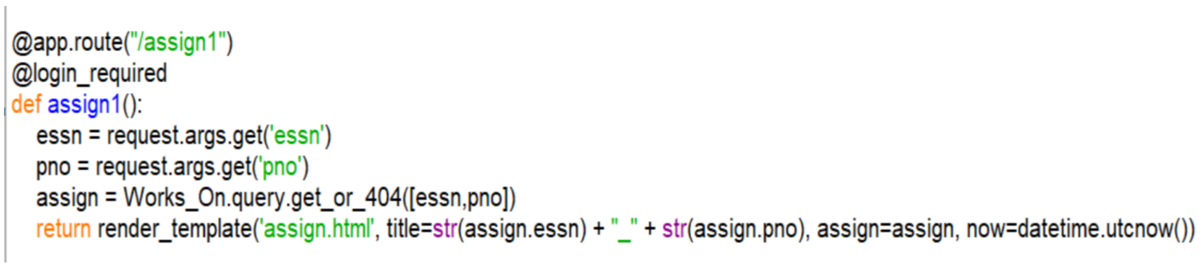


```
@app.route("/assign/<pno>/<essn>")
@login_required
def assign(pno, essn):
   assign = Works_On.query.get_or_404([essn,pno])
   return render_template('assign.html', title=str(assign.essn) + "_" + str(assign.pno), assign=assign, now=datetime.utcnow())
```

If you model assign.html after dept.html, you will have a button for Update and a button for Delete
For this lab, you do not have to implement the Update functionality. Just have that button go to the home route.

That was the new part (2 parameters). Since we haven't seen it before in our labs, I gave it to you.
There is no assign.html page yet, so I just made up a fake one until I was ready to process it.
Right now, we're just looking at Delete, but later in this lab, you will have to create a new assignment as well.
Notice that you have to set up that assign route. Notice also that it is creating a hierarchical url.
Alternative (less kludgy) method of passing parameters using GET variables:

Here is another way to pass those parameters, without creating the contrived directory structure. I created another route which I called assign1, just to demonstrate. This route accepts the get variables by extracting them from the GET superglobal variable array (commonly done in many programming languages). Notice that the parameters are not passed in app.route, nor in the def. I also tested this out with assign.html, and the variables pass correctly. We didn't cover this in previous labs, so you could use the first method.



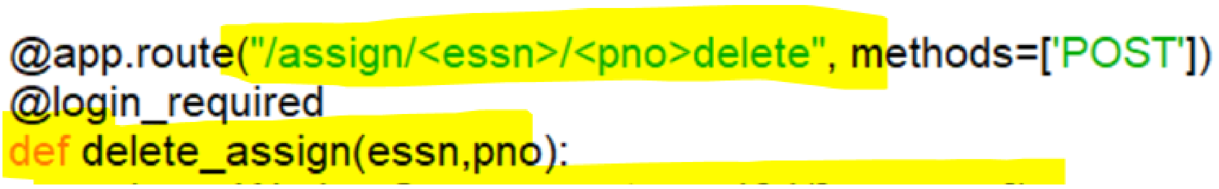
```
@app.route("/assign1")
@login_required
def assign1():
    essn = request.args.get('essn')
    pno = request.args.get('pno')
    assign = Works_On.query.get_or_404([essn,pno])
    return render_template('assign.html', title=str(assign.essn) + "_" + str(assign.pno), assign=assign, now=datetime.utcnow())
```

Following along with the same logic flow, let's create the real **assign.html**, model it after **dept.html:**
Assign.html is a landing page for a specific employee-project assignment, after the user clicked on it from "home"

- delete or rename the fake assign.html if you created one.
- copy dept.html and renamed it assign.html
- in assign.html, change all references of dept to assignment.
  Many of these were "canned" from bootstrap; (I am showing line #s.)
  in previous labs, we walked through what they do.

○ Change "dead" link and text to be displayed
○ Change url for update_dept to update_assign (which doesn't exist yet) or just to home
○ Change the delete button and links from "dept" to "assign":
  In your "action=" be sure to pass both parts of the primary key!
  Notice that we don't have the urls/routes for update_assign and delete_assign yet:
    (We changed update_dept and delete_dept, but haven't actually created those urls/routes yet.)

- So back to **routes.py** to add delete_assign and update_assign. (We'll work on delete_assign first.)
  (delete_assign is my shorthand for delete assignment)

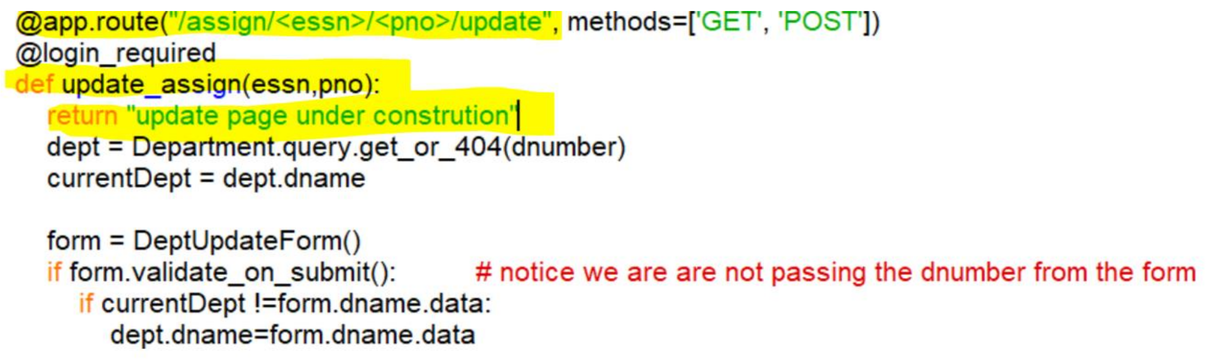


```
@app.route("/assign/<essn>/<pno>delete", methods=['POST'])
@login_required
def delete_assign(essn,pno):
```

…you can complete the rest, modeled after def delete_dept, just make sure to use the entire primary key

- Note that we can also use "request", similarly to the alternative way that I showed you before, how we did before, so we don't have to use a deeper directory structure. Again, we didn't cover this in our labs, and it works fine the first way, as shown in the screen shot.

- Since the reference to update_assign will trigger an error, I copied that too and put in a dummy return, just until we get around to working on it. Other than the signatures, it’s still copied from update_dept Or, as I said earlier, since you don't have to implement the update functionality for this lab, just have the template send the user "home" if they click on Update.



```
@app.route("/assign/<essn>/<pno>/update", methods=['GET', 'POST'])
@login_required
def update_assign(essn,pno):
    return "update page under constrution"
    dept = Department.query.get_or_404(dnumber)
    currentDept = dept.dname

    form = DeptUpdateForm()
    if form.validate_on_submit():        # notice we are are not passing the dnumber from the form
        if currentDept !=form.dname.data:
            dept.dname=form.dname.data
```

- Now it’s time to test our delete! Refresh localhost:5000 (re-run if necessary), click on one of the assignments, and click delete. It works, and the update is reflected in the database.

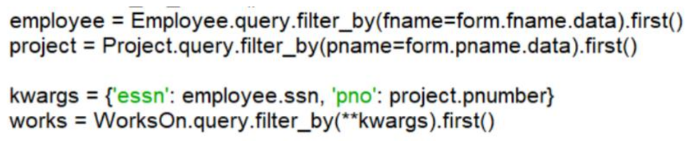
→ ***Notice that at this point, we did not even user drop-down boxes or forms at all. The forms that used were the update and delete button, built directly into assign.html . For the "assign" (or you might call it "create new") functionality, you will have to use drop-down boxes and forms.***

→ ***Notice also that this is a straightforward modification of Lab-4-b. The only difference is using two parameters instead of one.***

○ ***For the routes, using the same syntax as @app.route("/dept/<dnumber>/delete", simply creating a deeper path: @app.route("/assign/<essn>/<pno>delete"***

○ ***For the query: assign = Works_On.query.get_or_404([essn,pno]) Using the brackets is something new, but that is a straightforward google.***

- Some teams googled other methods for the query:



```
employee = Employee.query.filter_by(fname=form.fname.data).first()
project = Project.query.filter_by(pname=form.pname.data).first()

kwargs = {'essn': employee.ssn, 'pno': project.pnumber}
works = WorksOn.query.filter_by(**kwargs).first()
```

This method uses a python dictionary or arguments, which is passed as the query argument. Again, this is not something you are responsible for knowing, but it’s another option if you understand it.

After you get the delete to run, your real lab challenge is to add an assign (or create) functionality, that is, allow the user to assgin an employee to a project. I did this with two drop-down boxes on the same page. Some teams presented one drop-down for Employee, and then took the user's input and presented a separate page with the project drop-down box. (Without javascript or similar, you can't have the two boxes dependent on each other in one page, so I can't require that in this course.)

You can model this after the existing forms.py, using two separate queries to create the "choices" for the employee and the project drop-down boxes and similar logic to the existing application.

Hints for validating your form fields:

Some comments and hints about validation:

1. You can show both drop down boxes, populating one of them with all employees and one of them with all projects. Then, after the user selects an employee and a project, you do in fact have to validate that that combination doesn't already exist.

2. You can show just the employee (or just the project) drop-down box, and let the user select an employee, and then, using that employee-SSN, query the works_on table to extract those project that are already assigned to that employee, and populate a new, separate drop-down box with the projects that are NOT already assigned to that employee. In this case, you do NOT have to validate the assignment in the form, because you only populated the drop-down form wtih valid project numbers.

One method is more complicated for validation; the otehr may be more complicated for programming and querying.

If you do choose to validate, you don't want to validate ssn separately and then the project number separately, because then all you are validating is that there exists such an employee, and that there exists such a project, but you are not validating that the combination exists. You would have to do something similar to what we did to how we tested dnumber and dname in the example given in lab-4-b.

Hints for creating a new employee-project assignment

How many queries: As I mentioned earlier, the easiest way to do the drop-down lists is to have two independent drop-down lists, one for employee and one for project.

What are the queries: For these lists, you would create two separate queries, one on Employee and one on Project. Why not one on Works_on? Because you want to have a list of all employees and all projects. What if there's a new project, and no one has been assigned to it yet? Then that project won't be in Works_on yet. What if there's a new Employee, who has not yet been assigned to project? Then that employee won't be in Works_on. So run the two separate queries at the top of your forms.py, and model them similarly to how you did for ssn for the department example. You don't need distinct. Since you are retrieving the primary key of each, it will already be distinct.

What should the queries return: When we ran the dept.ssn query, we only retrieved the ssn. IN our case, however, you'd like more than that. You want the employee essn, lname, fname, and the Project's pnumber and pname. Set up two separate choices (I named mine empChoices and projChoices), and use them in your Select Field almost identically to how we did it in the example.

The pnumber is an integer: If you are using a SelectField, then you can't also make the pnumber an IntegerField.
So you can do this instead: pnumber = SelectField("Project", choices = projChoices, coerce = int)

Don't forget about hours: Although I have not required you to enter anything for hours, which is a field in Works_on, the database does require something. If you look at the DDL, or at the structure of the works_on table in phpMyAdmin, then you will see that it is a "not null" field in the database. YOu can address this in two ways:

Add a field to your form for the hours. Then use that to create the new works_on record just as you use the empname and pno.

OR:

Example in your routes: `assignment = Works_On(essn=form.ssn.data, pno=form.pnumber.data,hours=form.hours.data)`

Don't add a field to your form, but still give works_on some kind of value for hours.

```
assignment = Works_On(essn=form.ssn.data, pno=form.pnumber.data,hours=0)
```