CSE 183 Spring 2019
Homework Assignment 3
Due: Thursday May 9

The Assignment

In this assignment, we will learn how to create multiple database tables, insert data into them via web forms, and query them.

I created this starter code for you.

The Database

The database will have the following tables:

Product table. One of our tables will be the product table, familiar to you from homework 2, containing at least the following fields:

- Name
- Quantity in stock
- Sales Price (to customer)

User profile table. This table is separate from the auth.user table; you can call it “user_profile”. It needs to hold:

- Email
- Name
- Street
- City
- Zip Code

Order table. Orders are a many-to-many relationship between users and products, so the orders table should contain:

- Email (used to refer to a user), or reference to the profile, as you like.
- product_id (Reference to the product)
- Quantity
- Date
- Amount paid (this is useful, since the price of the product can change, so we cannot reconstruct the amount paid via the current price).

The Website

The Store page.

Make a “store” page, at /default/store, that shows the list of all products, with a button to buy it. When a user clicks on the button, you redirect the user to the create_orders page, passing the id of the product you wish to purchase as an argument, as in:

```
URL(‘default’, ‘create_order’, args=[product_id])
```

There needs to be a way to create products to test the code, so use create=True in the grid, which will enable logged in users to create products.

The Create_Order page.

The /default/create_order page has a @auth.requires_login() decorator. In the create order page, you first check whether the user (who needs to be logged in) has a profile (an entry in the profile table).

If the user does not have a profile page, you redirect them to create a profile via the Profile page, to:

```
URL(‘default’, ‘profile’, vars=dict(
      next=URL(‘default’, ‘create_order’, args=[product_id]),
      edit=’y’
      ))
```

Where:

- default/profile is the profile page
- `next` is a variable containing a URL when to send the user after the profile is edited, so instead of doing redirect(URL(‘default’, ‘index’)), the controller for default/profile will do:

  ```
  redirect(request.vars.next)
  ```

- `edit` is a variable that, when set to ‘y’, indicates that the profile needs editing.

If the user does have a profile, the create order page displays the product name, and a form to create an order – the user just needs to fill in the quantity – with a “submit” button. Check that the quantity is positive, and no greater than the amount of products in stock. Suggestion: for the

latter, you can either use custom form verification, or you can do something like (to give you the idea; you might have to adapt this code):

```
product = db( ... ).select().first()
db.product_orders.order_quantity.requires = IS_INT_IN_RANGE(1,
product.quantity)

# You may also write:
db.product_orders.order_date.readable = False
# etc etc.
```

In the form, you need to tie the order to the user, and to the product. You can do this by doing:

```
db.product_orders.product_id.default = product_id
db.product_orders.user.default = auth.user.email
db.product_orders.order_date.default = datetime.datetime.utcnow()
```

In the controller, before you define the form!

The profile page.

The profile page enables one to view his/her own profile, or, if request.vars.edit == 'y', to create/edit it.

The page accepts these variables:

- `request.vars.edit` : if 'y', then it asks the user to create/edit the profile; otherwise, it displays the profile.
- `request.vars.email` : if specified, it displays the profile of this specific user; otherwise, for the currently logged in user.
- `request.vars.next` : where to redirect the user after a successful creation/edit.

Let us remind you how to create forms; see here for the reference. A form to create a profile is created with

```
SQLFORM(db.user_profile)
```

A form to edit a profile p (which you have previously read from the database) is created with:

```
SQLFORM(db.user_profile, p)
```

A form to display a profile p (which you have previously read from the database) is created with:

```
SQLFORM(db.user_profile, p, readonly=True)
```

Note: forms for creation or edit need .process() to have an effect; forms for viewing, not.

The profile page needs to include one of these three types of forms, according to `request.vars.edit == 'y'` (in which case the form has to be a create form, if the user does not have a profile, or an edit form, if the user does have a profile), or otherwise (in which case the form is a view form).

If you display a view form, you can put on the page a button to go to /default/store. If you display an edit or create form, you redirect as specified by `request.vars.next`, via:

```
redirect(request.vars.next)
```

The order list page.

Finally, the order list page, at /default/order_list, should display a list of all orders, linking to the profile of the user who ordered, and to the product that was ordered. To do so, you can use a SQLFORM.grid. Now the trick is that you want that user emails are displayed as links to the profile, and products are represented via their product names and links to the product page. You can obtain the effect via:

```
# Define this function in tables.py

def get_product_name(p):
      return None if p is None else p.product_name

# Put these statements in the controller for the order list, before you do
# grid = SQLFORM.grid( ... )

db.product_order.user_email.represent = lambda v, r : A(v,
_href=URL('default', 'user_profile', vars=dict(email=v)))

db.product_order.product_id.represent = lambda v, r :
A(get_product_name(db.product(v)), _href=URL('default', 'view_product',
args=[v]))
```

Where /default/view_product (or however you have called it) is the page that is used to view an order. Generate it via a SQLFORM(db.product, p, readonly=True).

One word about security

This assignment is very naive, in that it allows every logged in user to see the list of all orders. In a real shopping site, we would need to be able to distinguish between customers and store employees, and let only the latter see the list of all orders. Worse, we allow everybody to see every user profile. We will see how to fix this in a later homework assignment.

Implementation Steps

1. Get the starter code (not required but highly recommended).
2. Define the profile and orders tables.
3. Notice you can create products already. Now build the store page with the Buy button for each item.
4. Build the profile page, completing the function profile in default.py. You can test it from the menu bar of the site, for your convenience.
5. Build the create_order page, completing the function create_order in default.py.
6. Build the order_list page, as above.
7. Test, test, test.
8. Now write create_order.html, profile.html, order_list.html, etc. Keep them simple.
9. Done :-) Test, test, and submit.