</style>

COMS W4111-002, V02 (Spring 2022) Introduction to Databases

</span>

Homework 2: Programming

Due Sunday, February 27, 2022 at 11:59 PM

Introduction

Overview

This homework has 1 section:

1. A section for programming track.

Submission

You will **submit 2 files** for this assignment.

1. Submit a zip file titled `<your_uni>_hw2_programming.zip` to **HW2 Programming - Zip** on Gradescope.
   - Replace `<your_uni>` with your uni. My submission would be `dff9_hw2_programming.zip` .
   - The zipped directory should include:
     - `classicmodels.sql`
     - `src`
       - `application.py`
       - `resources`
         - `__init__.py`
         - `base_resource.py`
         - `imdb_artists.py`
       - `rest_utils.py`
     - `<your_uni>_hw2_programming.ipynb` (substitute with your uni as above)
     - Any image files you embed in your notebook.

1. Submit a PDF title `<your_uni>_hw2_programming.pdf` to **HW2 Programming - PDF** on Gradescope.
   - This should be a PDF of your completed HW2 Programming Python notebook.
     - **Tag pages for each problem**. Per course policy, any untagged submission will receive an automatic 0.
     - Double check your submission on Gradescope to ensure that the PDF conversion worked and that your pages are appropriately tagged.

Collaboration and Information

- Answering some of the questions may require independent research to find information. We encourage you to try troubleshooting problems independently before reaching out for help.
- You may use any information you get in TA or Prof. Ferguson's office hours, from lectures or from recitations. This includes slides related to the recommended textbook.
- You may use information that you find on the web.
- You are NOT allowed to collaborate with other students outside of office hours.

Programming

Setup

- Modify the cells below to setup your environment.
- The change should just be setting the DB user ID and password, replacing my user ID and password with yours for MySQL.

In [225…

```
database_user_id = "root"
database_pwd = "Xcz990208!"
```

In [226…

```
database_url = "mysql+pymysql://" + \
    database_user_id + ":" + database_pwd + "@localhost"
database_url
```

Out[226…

```
'mysql+pymysql://root:Xcz990208!@localhost'
```

In [227…

```
%reload_ext sql
```

In [228…

```
%sql $database_url
```

Out[228…

```
'Connected: root@None'
```

In [229…

```
from sqlalchemy import create_engine
```

In [230…

```
sqla_engine = create_engine(database_url)
```

In [231…

```
#
# We are going to create a schema and some tables for the HW.
#
%sql drop schema if exists S22_W4111_HW2_B
%sql create schema if not exists S22_W4111_HW2_B
%sql select 1;
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[231…

1
1

Install Dataset

Classic Models

- We will use the Classic Models Tutorial database for HW 2 Programming, other homework assignments, and exams.

- Lecture 5 briefly explained why this data model is interesting for educational purposes. The problems on homework assignments and exams will further explore why it's interesting.

- The zip file for HW 2 Programming contains an SQL script for creating a database `classicmodels` and loading the data. The script is `classicmodels.sql` .

- Use DataGrip to run the script. You performed this task for HW 0 with different SQL scripts. The basic approach is:
  - Right click on `@localhost`
  - Choose `Run SQL Script` .
  - Navigate to and select `classicmodels.sql` .

- The following cells test for correct installation.

- These cells are also examples of DDL statements and querying the "catalog."

In [81]:

```
%sql show tables from classicmodels
```

```
 * mysql+pymysql://root:***@localhost
8 rows affected.
```

Out[81]:

Tables_in_classicmodels
customers
employees
offices
orderdetails
orders
payments
productlines
products

In [82]:

```
%%sql
```

```
select
    table_schema, table_name, column_name, IS_NULLABLE, DATA_TYPE from information
where
    table_schema='classicmodels'
order by
    table_schema, table_name, ORDINAL_POSITION;
```

```
 * mysql+pymysql://root:***@localhost
59 rows affected.
```

Out[82]:

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	IS_NULLABLE	DATA_TYPE
classicmodels	customers	customerNumber	NO	int
classicmodels	customers	customerName	NO	varchar
classicmodels	customers	contactLastName	NO	varchar
classicmodels	customers	contactFirstName	NO	varchar
classicmodels	customers	phone	NO	varchar
classicmodels	customers	addressLine1	NO	varchar
classicmodels	customers	addressLine2	YES	varchar
classicmodels	customers	city	NO	varchar
classicmodels	customers	state	YES	varchar
classicmodels	customers	postalCode	YES	varchar
classicmodels	customers	country	NO	varchar
classicmodels	customers	salesRepEmployeeNumber	YES	int
classicmodels	customers	creditLimit	YES	decimal
classicmodels	employees	employeeNumber	NO	int
classicmodels	employees	lastName	NO	varchar
classicmodels	employees	firstName	NO	varchar
classicmodels	employees	extension	NO	varchar
classicmodels	employees	email	NO	varchar
classicmodels	employees	officeCode	NO	varchar
classicmodels	employees	reportsTo	YES	int
classicmodels	employees	jobTitle	NO	varchar
classicmodels	offices	officeCode	NO	varchar
classicmodels	offices	city	NO	varchar
classicmodels	offices	phone	NO	varchar
classicmodels	offices	addressLine1	NO	varchar
classicmodels	offices	addressLine2	YES	varchar
classicmodels	offices	state	YES	varchar
classicmodels	offices	country	NO	varchar
classicmodels	offices	postalCode	NO	varchar
classicmodels	offices	territory	NO	varchar
classicmodels	orderdetails	orderNumber	NO	int

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	IS_NULLABLE	DATA_TYPE
classicmodels	orderdetails	productCode	NO	varchar
classicmodels	orderdetails	quantityOrdered	NO	int
classicmodels	orderdetails	priceEach	NO	decimal
classicmodels	orderdetails	orderLineNumber	NO	smallint
classicmodels	orders	orderNumber	NO	int
classicmodels	orders	orderDate	NO	date
classicmodels	orders	requiredDate	NO	date
classicmodels	orders	shippedDate	YES	date
classicmodels	orders	status	NO	varchar
classicmodels	orders	comments	YES	text
classicmodels	orders	customerNumber	NO	int
classicmodels	payments	customerNumber	NO	int
classicmodels	payments	checkNumber	NO	varchar
classicmodels	payments	paymentDate	NO	date
classicmodels	payments	amount	NO	decimal
classicmodels	productlines	productLine	NO	varchar
classicmodels	productlines	textDescription	YES	varchar
classicmodels	productlines	htmlDescription	YES	mediumtext
classicmodels	productlines	image	YES	mediumblob
classicmodels	products	productCode	NO	varchar
classicmodels	products	productName	NO	varchar
classicmodels	products	productLine	NO	varchar
classicmodels	products	productScale	NO	varchar
classicmodels	products	productVendor	NO	varchar
classicmodels	products	productDescription	NO	text
classicmodels	products	quantityInStock	NO	smallint
classicmodels	products	buyPrice	NO	decimal
classicmodels	products	MSRP	NO	decimal

In [232…

```
%%sql
use classicmodels;
with
    customer_orders_details as
        (
            select customerNumber, orderNumber, status, orderDate, shippedDate,
                   productCode, quantityOrdered, priceEach
            from orders natural join orderdetails
        ),
     customer_orders_totals as
         (
             select customerNumber, orderNumber,
                   concat(
                          '$',
                          format(sum(priceEach * quantityOrdered), 2)
                      ) as order_value
               from customer_orders_details
               group by customerNumber, orderNumber
        )
select * from customer_orders_totals limit 10;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
10 rows affected.
```

Out[232…

customerNumber	orderNumber	order_value
103	10123	$14,571.44
103	10298	$6,066.78
103	10345	$1,676.14
112	10124	$32,641.98
112	10278	$33,347.88
112	10346	$14,191.12
114	10120	$45,864.03
114	10125	$7,565.08
114	10223	$44,894.74
114	10342	$40,265.60

Tasks

- There is a sub-folder `src` of this directory that contains:
  - `application.py` which is a Flask application.
  - `rest_utils.py` is some helpful code for dealing with Flask and other objects.
  - `resources` is a package that contains:
    - `base_resource.py` defines the abstract class that all REST resources must implement.
    - `imdb_artists.py` contains a partially completed REST resource implementation.
- You must complete the implementation of `application.py` and implement a file `orders.py` that implements a class `Orders` . The class must implement the abstract methods defined in `base_resource` .
- In `application.py` you must implement support for the paths:
  - `/resource_collection`
    - GET on URLs of the forms `/orders?customerNumber=101&status=shipped&fields=customerNumber,orderNumber`
    - POST that has a JSON body defining the data for the new row.
  - `/resource_collection/id`

- GET on URLs of the `/orders/10100`
- DELETE
- UPDATE, which takes a JSON body and updates the fields.

- You must test your paths below. The following is an example that tests GET.

In [85]:
```
import requests
```

In [91]:
```
#
# Test get
#
url = "http://localhost:5003/api/imdb_artists/nm0000980"
res = requests.get(url)
res = res.json()

res
```

Out[91]:
```
{'nconst': 'nm0000980',
 'primaryName': 'Jim Broadbent',
 'birthYear': '1949',
 'deathYear': '',
 'primaryProfession': 'actor,writer,soundtrack',
 'knownForTitles': 'tt0203009,tt1007029,tt0151568,tt1431181'}
```

- Include at least one test for each remaining supported path below. You **must** display the output of each test.

In [ ]:
```
#
# Test GET on URLs of the forms /orders?customerNumber=101&status=shipped&fields=custo
#
```

In [184…
```
import requests
url = "http://localhost:5003/api/orders?customerNumber=101&status=Shipped&fields=custo
res = requests.get(url)
res = res.json()

res
```

Out[184…
```
{'data': [{'customerNumber': 101,
   'orderNumber': 9999,
   'status': 'Shipped',
   'orderDate': '2020-03-02'}],
 'links': [{'rel': 'self',
   'href': 'http://localhost:5003/api/orders?customerNumber=101&status=Shipped&fields=
customerNumber,%20orderNumber,%20status,%20orderDate'}]}
```

In [193…
```
# Just a try, please ignore
import json
payload = {'orderNumber':9999,
             'orderDate': '2020-03-02',
             'requiredDate':'2020-03-02',
             'status':'Shipped',
             'customerNumber':101}
json.dumps(payload)
```

Out[193…
```
'{"orderNumber": 9999, "orderDate": "2020-03-02", "requiredDate": "2020-03-02", "statu
s”: ”Shipped”, ”customerNumber”: 101}’
```

In [ ]:

```
#
# Test POST that has a JSON body defining the data for the new row
#
```

In [196…

```
import requests
d = {’orderNumber’:9999,
     ’orderDate’: ’2020-03-02’,
     ’requiredDate’:’2020-03-02’,
     ’status’:’Shipped’,
     ’customerNumber’:101}

res = requests. post(”http://localhost:5003/api/orders”, json = d)
print(res)
print(res. content)
print(res. headers)
```

```
<Response [201]>
b’CREATED’
{’Location’: ’http://localhost:5003/users/[9999]’, ’Content-Type’: ’text/plain’, ’Cont
ent-Length’: ’7’, ’Access-Control-Allow-Origin’: ’*’, ’Server’: ’Werkzeug/2.0.2 Pytho
n/3.6.13’, ’Date’: ’Thu, 03 Mar 2022 02:40:15 GMT’}
```

In [ ]:

```
#
# Test GET on URLs of the /orders/10100
#
```

In [96]:

```
url = ”http://localhost:5003/api/orders/10100”
res = requests. get(url)
res = res. json()

res
```

Out[96]:

```
{’orderNumber’: 10100,
 ’orderDate’: ’2003-01-06’,
 ’requiredDate’: ’2003-01-13’,
 ’shippedDate’: ’2003-01-10’,
 ’status’: ’Shipped’,
 ’comments’: None,
 ’customerNumber’: 363}
```

In [ ]:

```
#
# Test Delete by ID
#
```

In [199…

```
url = ”http://localhost:5003/api/orders/8888”
res = requests. delete(url)
res = res. json()

res
```

Out[199…

```
1
```

In [ ]:

```
#
# Test UPDATE, which takes a JSON body and updates the fields
#
```

In [200…

```
# Firsr inset a new row to be updated
import requests
d = {'orderNumber':8888,
     'orderDate': '2020-03-02',
     'requiredDate':'2020-03-02',
     'status':'unShipped',
     'customerNumber':505}

res = requests.post("http://localhost:5003/api/orders", json = d)
print(res)
print(res.content)
print(res.headers)
```

```
<Response [201]>
b'CREATED'
{'Location': 'http://localhost:5003/users/[8888]', 'Content-Type': 'text/plain', 'Cont
ent-Length': '7', 'Access-Control-Allow-Origin': '*', 'Server': 'Werkzeug/2.0.2 Pytho
n/3.6.13', 'Date': 'Thu, 03 Mar 2022 02:55:04 GMT'}
```

In [213…

```
# Update the body
d = {'status':'Shipped',
     'customerNumber':101}

res = requests.put("http://localhost:5003/api/orders/8888", json = d)
res = res.json()
res
```

Out[213…

```
1
```

In [208…

```
# Check the update
url = "http://localhost:5003/api/orders/8888"
res = requests.get(url)
res = res.json()

res
```

Out[208…

```
{'orderNumber': 8888,
 'orderDate': '2020-03-02',
 'requiredDate': '2020-03-02',
 'shippedDate': None,
 'status': 'Shipped',
 'comments': None,
 'customerNumber': 101}
```

- Include screenshots of all the code you wrote in `application.py` , `orders.py` , and any other Python files below.

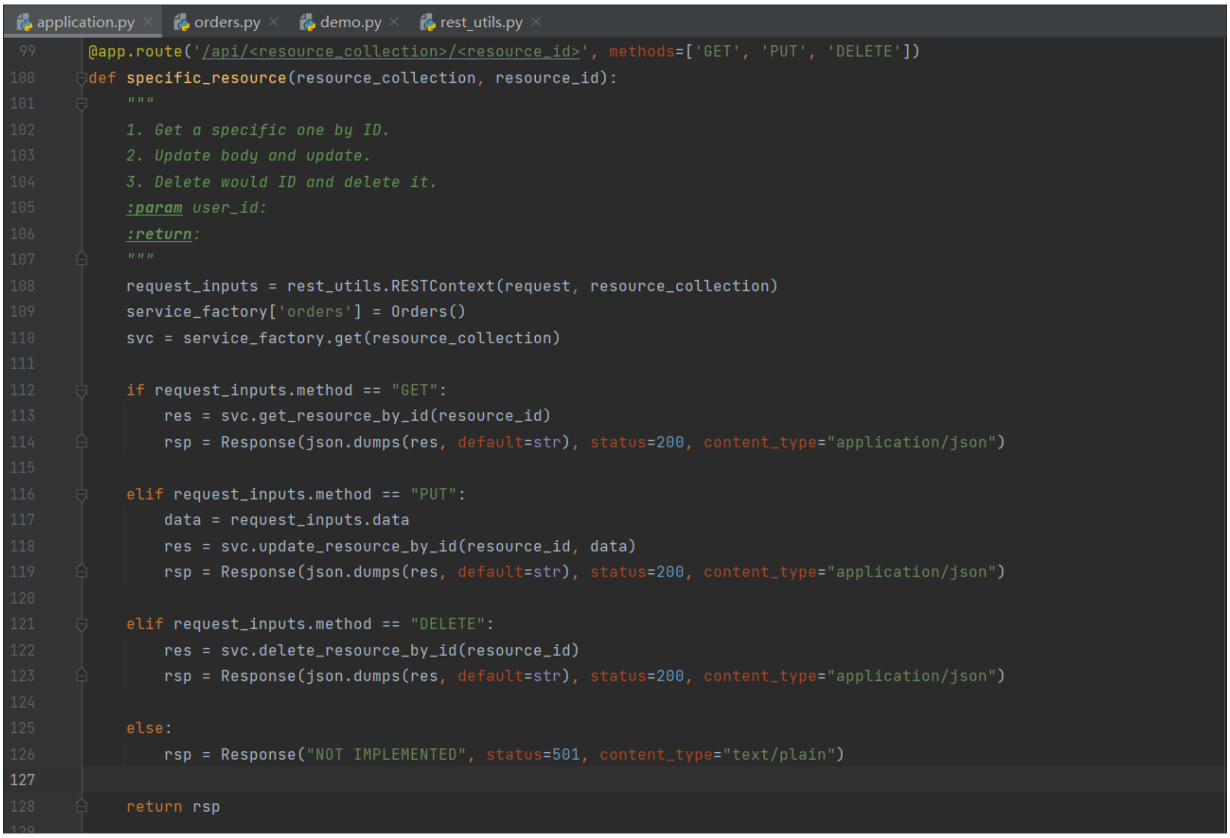
In [ ]:

```
# ScreenShots in application.py
```

In [216…

```
# GET PUT DELETE
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[216…

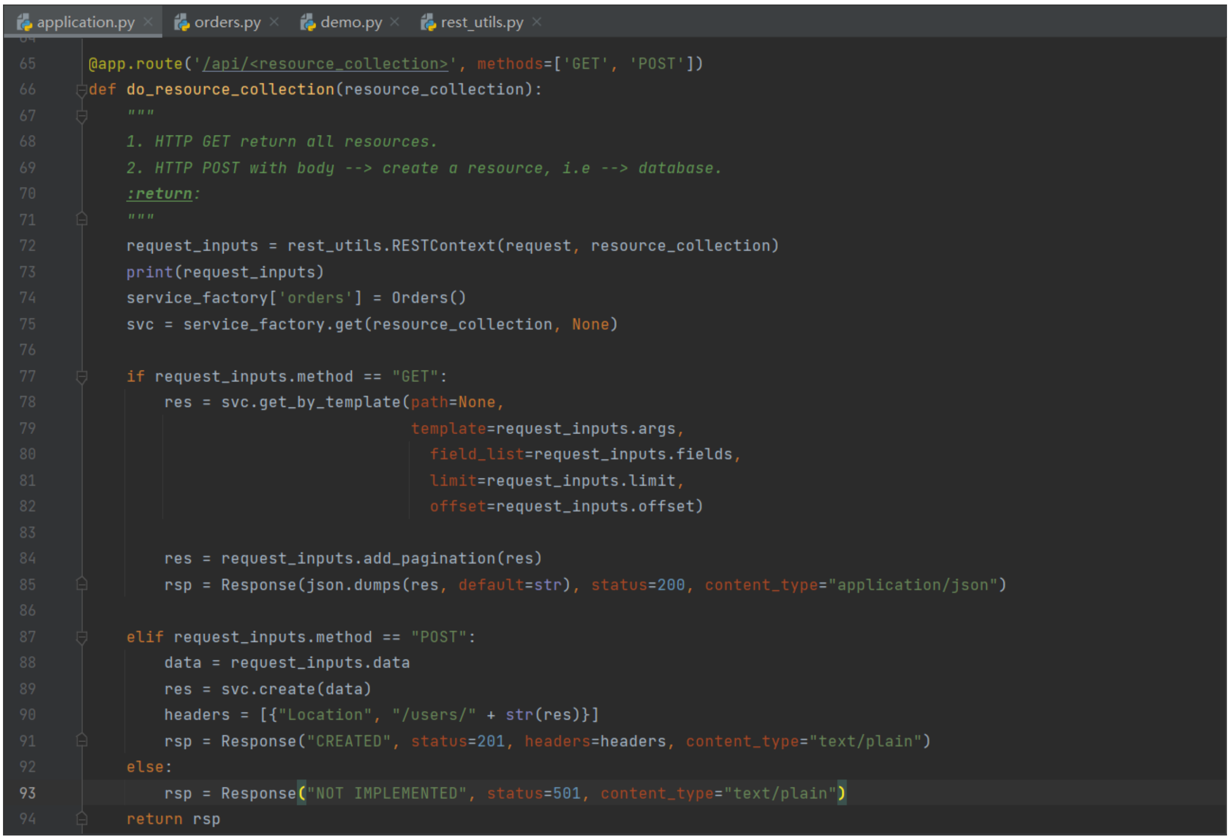


```
@app.route('/api/<resource_collection>/<resource_id>', methods=['GET', 'PUT', 'DELETE'])
def specific_resource(resource_collection, resource_id):
    """
    1. Get a specific one by ID.
    2. Update body and update.
    3. Delete would ID and delete it.
    :param user_id:
    :return:
    """
    request_inputs = rest_utils.RESTContext(request, resource_collection)
    service_factory['orders'] = Orders()
    svc = service_factory.get(resource_collection)

    if request_inputs.method == "GET":
        res = svc.get_resource_by_id(resource_id)
        rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    elif request_inputs.method == "PUT":
        data = request_inputs.data
        res = svc.update_resource_by_id(resource_id, data)
        rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    elif request_inputs.method == "DELETE":
        res = svc.delete_resource_by_id(resource_id)
        rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    else:
        rsp = Response("NOT IMPLEMENTED", status=501, content_type="text/plain")

    return rsp
```

In [217…

```
# GET POST
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[217…



```
@app.route('/api/<resource_collection>', methods=['GET', 'POST'])
def do_resource_collection(resource_collection):
    """
    1. HTTP GET return all resources.
    2. HTTP POST with body --> create a resource, i.e --> database.
    :return:
    """
    request_inputs = rest_utils.RESTContext(request, resource_collection)
    print(request_inputs)
    service_factory['orders'] = Orders()
    svc = service_factory.get(resource_collection, None)

    if request_inputs.method == "GET":
        res = svc.get_by_template(path=None,
                                  template=request_inputs.args,
                                    field_list=request_inputs.fields,
                                    limit=request_inputs.limit,
                                    offset=request_inputs.offset)

        res = request_inputs.add_pagination(res)
        rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    elif request_inputs.method == "POST":
        data = request_inputs.data
        res = svc.create(data)
        headers = [{"Location", "/users/" + str(res)}]
        rsp = Response("CREATED", status=201, headers=headers, content_type="text/plain")
    else:
        rsp = Response("NOT IMPLEMENTED", status=501, content_type="text/plain")
    return rsp
```

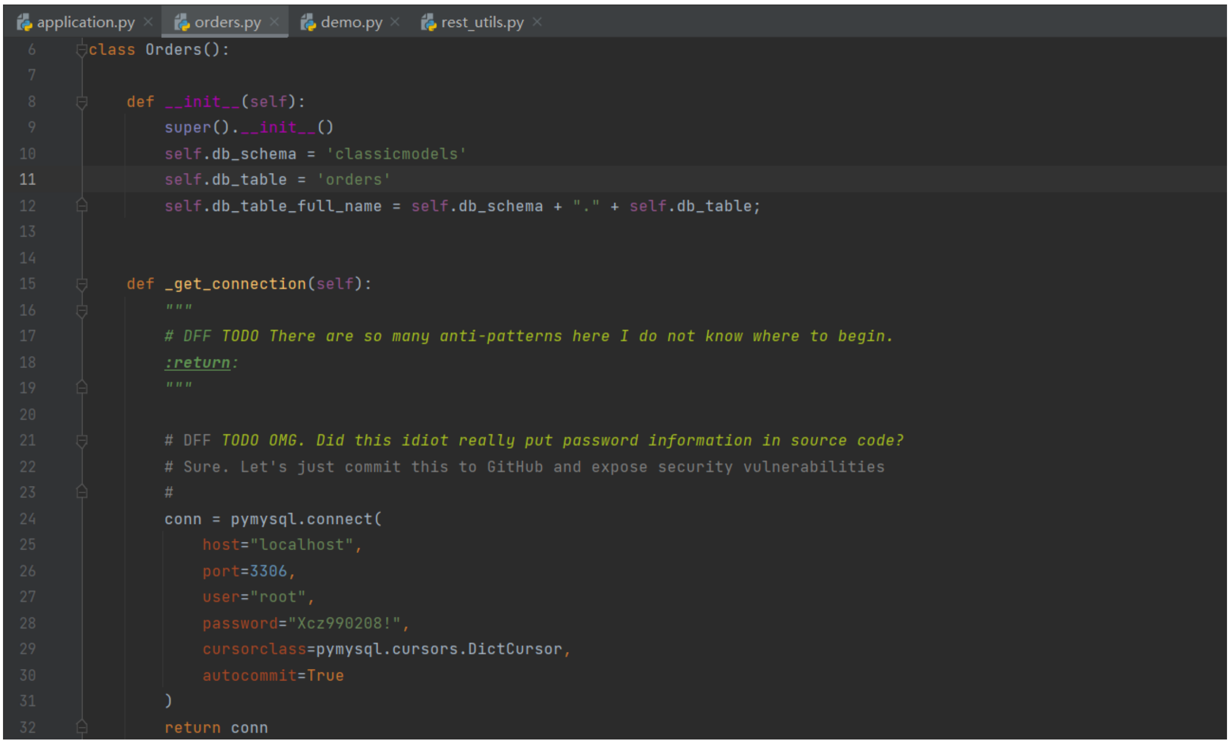
In [ ]:

```
# ScreenShots in orders.py
```

In [218…

```
# Initialization
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[218…

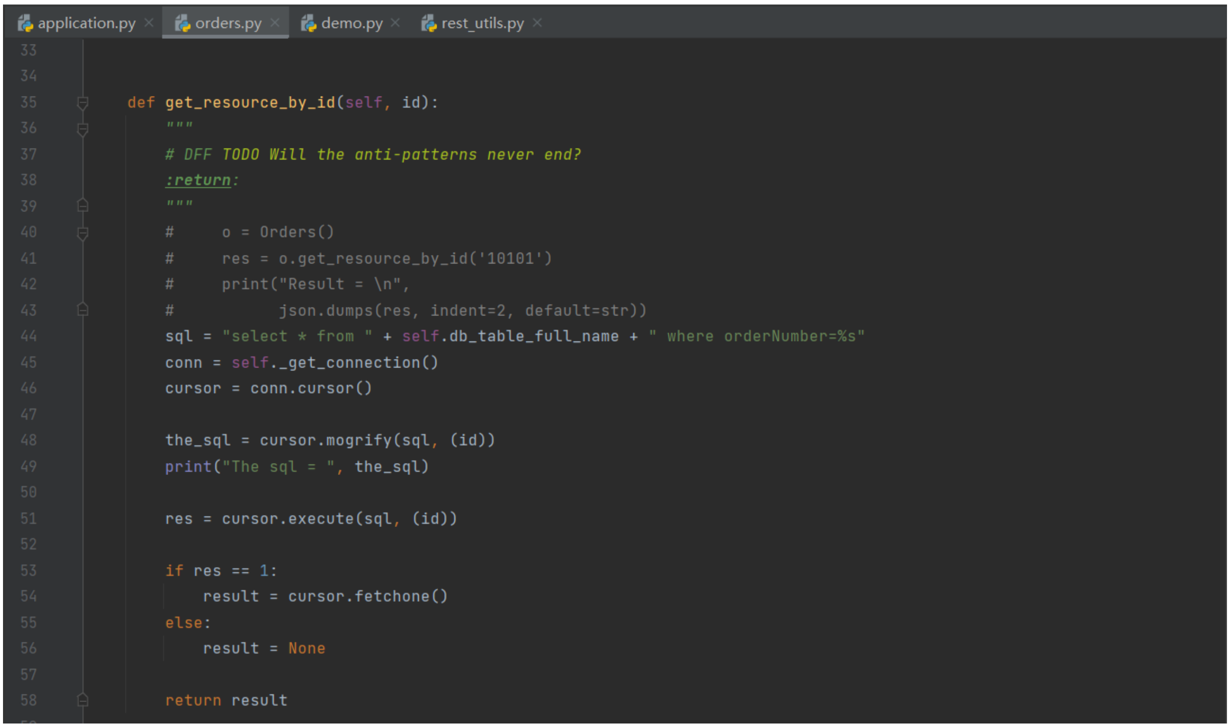


```
class Orders():

    def __init__(self):
        super().__init__()
        self.db_schema = 'classicmodels'
        self.db_table = 'orders'
        self.db_table_full_name = self.db_schema + "." + self.db_table;


    def _get_connection(self):
        """
        # DFF TODO There are so many anti-patterns here I do not know where to begin.
        :return:
        """

        # DFF TODO OMG. Did this idiot really put password information in source code?
        # Sure. Let's just commit this to GitHub and expose security vulnerabilities
        #
        conn = pymysql.connect(
            host="localhost",
            port=3306,
            user="root",
            password="Xcz990208!",
            cursorclass=pymysql.cursors.DictCursor,
            autocommit=True
        )
        return conn
```

In [219…

```
# get_resource_by_id
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[219…

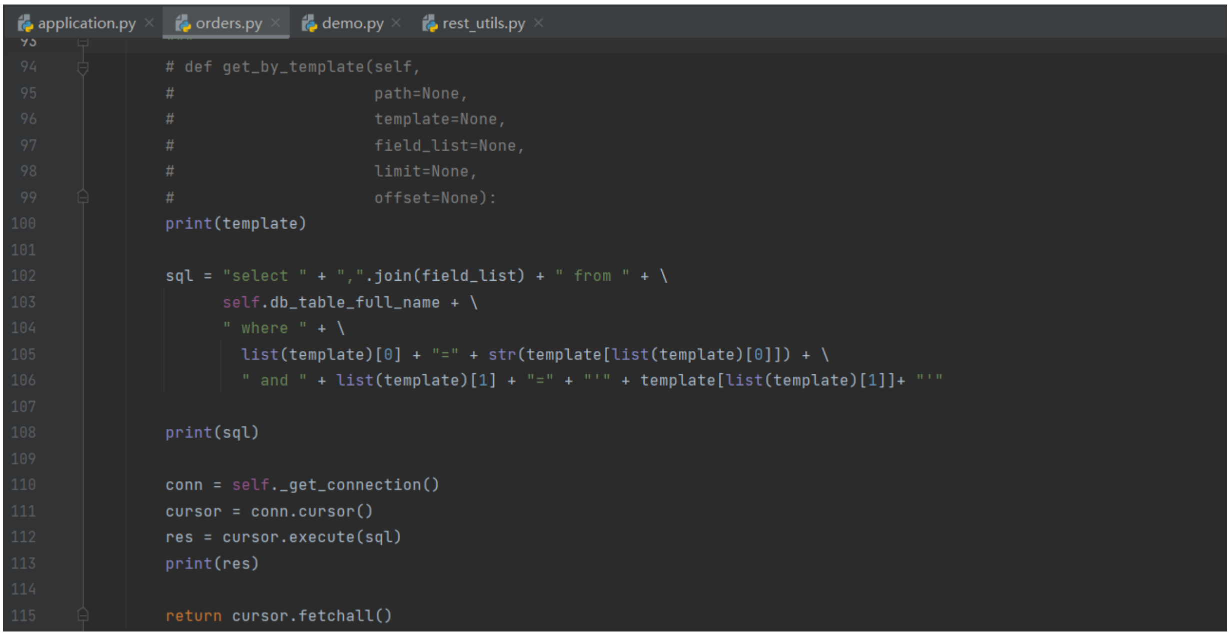


```python
    def get_resource_by_id(self, id):
        """
        # DFF TODO Will the anti-patterns never end?
        :return:
        """
        #     o = Orders()
        #     res = o.get_resource_by_id('10101')
        #     print("Result = \n",
        #           json.dumps(res, indent=2, default=str))
        sql = "select * from " + self.db_table_full_name + " where orderNumber=%s"
        conn = self._get_connection()
        cursor = conn.cursor()

        the_sql = cursor.mogrify(sql, (id))
        print("The sql = ", the_sql)

        res = cursor.execute(sql, (id))

        if res == 1:
            result = cursor.fetchone()
        else:
            result = None

        return result
```

In [220…

```python
# get_by_template
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[220…

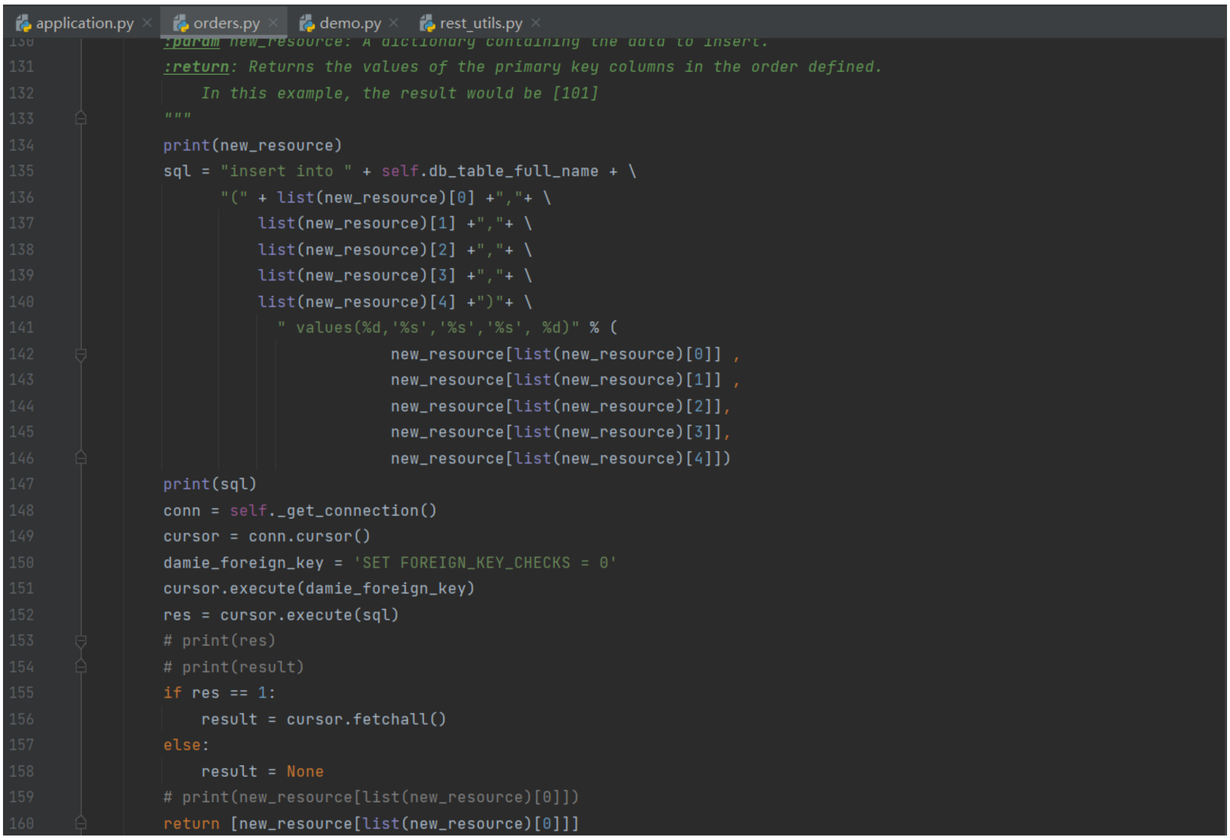


```python
        # def get_by_template(self,
        #                      path=None,
        #                      template=None,
        #                      field_list=None,
        #                      limit=None,
        #                      offset=None):
        print(template)

        sql = "select " + ",".join(field_list) + " from " + \
              self.db_table_full_name + \
              " where " + \
                list(template)[0] + "=" + str(template[list(template)[0]]) + \
                " and " + list(template)[1] + "=" + "'" + template[list(template)[1]]+ "'"

        print(sql)

        conn = self._get_connection()
        cursor = conn.cursor()
        res = cursor.execute(sql)
        print(res)

        return cursor.fetchall()
```

In [221…

```python
# create
er_model_file_name_1 = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[221…

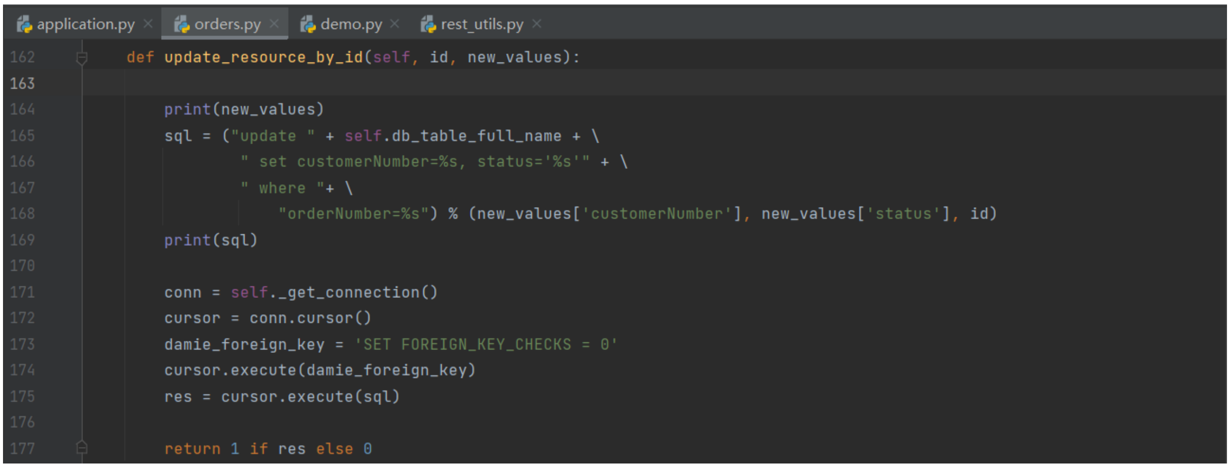


```
        :param new_resource: A dictionary containing the data to insert.
        :return: Returns the values of the primary key columns in the order defined.
            In this example, the result would be [101]
        """
        print(new_resource)
        sql = "insert into " + self.db_table_full_name + \
              "(" + list(new_resource)[0] +","+ \
                  list(new_resource)[1] +","+ \
                  list(new_resource)[2] +","+ \
                  list(new_resource)[3] +","+ \
                  list(new_resource)[4] +")"+ \
                    " values(%d,'%s','%s','%s', %d)" % (
                                new_resource[list(new_resource)[0]] ,
                                new_resource[list(new_resource)[1]] ,
                                new_resource[list(new_resource)[2]],
                                new_resource[list(new_resource)[3]],
                                new_resource[list(new_resource)[4]])
        print(sql)
        conn = self._get_connection()
        cursor = conn.cursor()
        damie_foreign_key = 'SET FOREIGN_KEY_CHECKS = 0'
        cursor.execute(damie_foreign_key)
        res = cursor.execute(sql)
        # print(res)
        # print(result)
        if res == 1:
            result = cursor.fetchall()
        else:
            result = None
        # print(new_resource[list(new_resource)[0]])
        return [new_resource[list(new_resource)[0]]]
```

In [222…

```
# update
er_model_file_name_1 = ’C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print(”\n”)
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[222…

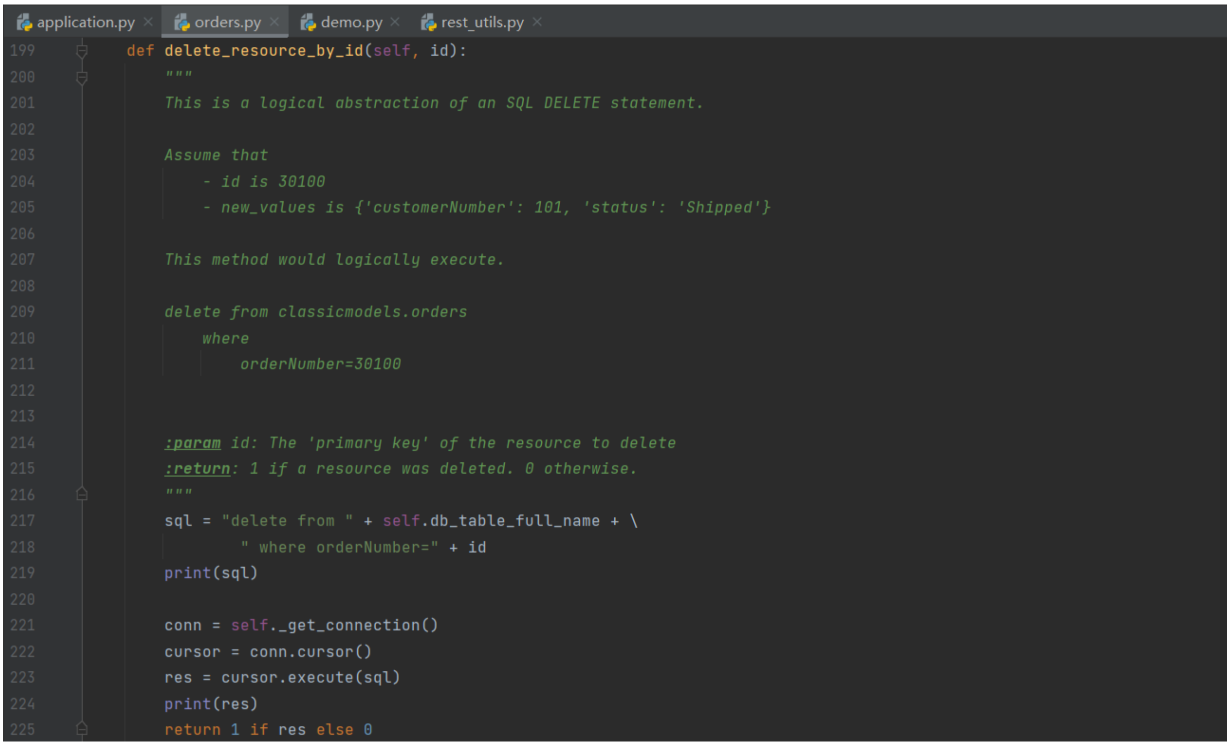


```
    def update_resource_by_id(self, id, new_values):

        print(new_values)
        sql = ("update " + self.db_table_full_name + \
                " set customerNumber=%s, status='%s'" + \
                " where "+ \
                    "orderNumber=%s") % (new_values['customerNumber'], new_values['status'], id)
        print(sql)

        conn = self._get_connection()
        cursor = conn.cursor()
        damie_foreign_key = 'SET FOREIGN_KEY_CHECKS = 0'
        cursor.execute(damie_foreign_key)
        res = cursor.execute(sql)

        return 1 if res else 0
```

In [223…

```
# delete
er_model_file_name_1 = ’C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\HW2_progra

print(”\n”)
from IPython.display import Image
Image(filename=er_model_file_name_1)
```

Out[223…



```
    def delete_resource_by_id(self, id):
        """
        This is a logical abstraction of an SQL DELETE statement.

        Assume that
            - id is 30100
            - new_values is {'customerNumber': 101, 'status': 'Shipped'}

        This method would logically execute.

        delete from classicmodels.orders
            where
                orderNumber=30100


        :param id: The 'primary key' of the resource to delete
        :return: 1 if a resource was deleted. 0 otherwise.
        """
        sql = "delete from " + self.db_table_full_name + \
                " where orderNumber=" + id
        print(sql)

        conn = self._get_connection()
        cursor = conn.cursor()
        res = cursor.execute(sql)
        print(res)
        return 1 if res else 0
```