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

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 [1]:
```
database_user_id = "root"
database_pwd = "dvuserdvuser"
```

In [2]:
```
database_url = "mysql+pymysql://" + \
    database_user_id + ":" + database_pwd + "@localhost"
database_url
```

Out[2]:
```
'mysql+pymysql://root:dvuserdvuser@localhost'
```

In [3]:
```
%reload_ext sql
```

In [4]:
```
%sql $database_url
```

Out[4]:
```
'Connected: root@None'
```

In [5]:
```
from sqlalchemy import create_engine
```

In [6]:
```
sqla_engine = create_engine(database_url)
```

In [7]:
```
#
```

```
# We are going to create a schema and some tables for the HW.
#
%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.
```

Out[7]:

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 [26]:

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

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (2006, "MySQL server has gone away (BrokenPipeErr
or(32, 'Broken pipe'))")
[SQL: show tables from classicmodels]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

In [27]:

```
%%sql

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

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

Out[27]:

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
classicmodels	orderdetails	productCode	NO	varchar
classicmodels	orderdetails	quantityOrdered	NO	int

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	IS_NULLABLE	DATA_TYPE
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 [28]:
```
%sql use classicmodels
```

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

Out[28]:
```
[]
```

In [29]:
```
%%sql
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;
```

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

Out[29]:

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
114	10347	$41,995.62
119	10275	$47,924.19
119	10315	$19,501.82
119	10375	$49,523.67
119	10425	$41,623.44
121	10103	$50,218.95
121	10158	$1,491.38
121	10309	$17,876.32
121	10325	$34,638.14
124	10113	$11,044.30
124	10135	$55,601.84
124	10142	$56,052.56
124	10182	$45,084.38
124	10229	$43,369.30
124	10271	$37,430.89

customerNumber	orderNumber	order_value
124	10282	$47,979.98
124	10312	$55,639.66
124	10335	$6,466.44
124	10357	$40,676.26
124	10368	$13,874.75
124	10371	$35,137.54
124	10382	$47,765.59
124	10385	$4,466.71
124	10390	$55,902.50
124	10396	$27,695.54
124	10421	$7,639.10
128	10101	$10,549.01
128	10230	$33,820.62
128	10300	$24,101.81
128	10323	$7,466.32
129	10111	$16,537.85
129	10201	$23,923.93
129	10333	$26,248.78
131	10107	$22,292.62
131	10248	$41,445.21
131	10292	$35,321.97
131	10329	$50,025.35
141	10104	$40,206.20
141	10128	$13,884.99
141	10133	$22,366.04
141	10153	$44,939.85
141	10156	$4,599.52
141	10190	$10,721.86
141	10203	$40,062.53
141	10205	$13,059.16
141	10212	$59,830.55
141	10244	$26,155.91
141	10246	$35,420.74
141	10262	$47,065.36
141	10279	$20,009.53

customerNumber	orderNumber	order_value
141	10311	$36,140.38
141	10350	$46,493.16
141	10355	$25,529.78
141	10358	$44,185.46
141	10378	$32,289.12
141	10379	$16,621.27
141	10380	$34,404.21
141	10383	$36,851.98
141	10386	$46,968.52
141	10394	$18,102.74
141	10412	$46,895.48
141	10417	$28,574.90
141	10424	$29,310.30
144	10112	$7,674.94
144	10320	$16,799.03
144	10326	$19,206.68
144	10334	$23,014.17
145	10105	$53,959.21
145	10238	$28,211.70
145	10256	$4,710.73
145	10327	$20,564.86
145	10406	$21,638.62
146	10194	$39,712.10
146	10208	$49,614.72
146	10227	$40,978.53
148	10117	$44,380.15
148	10150	$38,350.15
148	10165	$67,392.85
148	10277	$2,611.84
148	10387	$3,516.04
151	10127	$58,841.35
151	10204	$58,793.53
151	10267	$20,314.44
151	10349	$39,964.63
157	10272	$23,715.70

customerNumber	orderNumber	order_value
157	10281	$39,641.43
157	10318	$35,152.12
157	10422	$5,849.44
161	10140	$38,675.13
161	10168	$50,743.65
161	10317	$2,434.25
161	10362	$12,692.19
166	10217	$22,474.17
166	10259	$44,160.92
166	10288	$38,785.48
166	10409	$2,326.18
167	10181	$55,069.55
167	10188	$29,954.91
167	10289	$12,538.01
171	10180	$42,783.81
171	10224	$18,997.89
172	10114	$33,383.14
172	10286	$1,960.80
172	10336	$51,209.58
173	10228	$20,355.24
173	10249	$11,843.45
175	10172	$24,879.08
175	10263	$42,044.77
175	10413	$28,500.78
177	10210	$47,177.59
177	10240	$15,183.63
181	10102	$5,494.78
181	10237	$22,602.36
181	10324	$44,400.50
186	10155	$37,602.48
186	10299	$34,341.08
186	10377	$23,602.90
187	10110	$48,425.69
187	10306	$52,825.29
187	10332	$47,159.11

customerNumber	orderNumber	order_value
189	10220	$32,538.74
189	10297	$17,359.53
198	10130	$6,036.96
198	10290	$5,858.56
198	10352	$9,658.74
201	10253	$45,443.54
201	10302	$23,908.24
201	10403	$37,258.94
202	10206	$36,527.61
202	10313	$33,594.58
204	10276	$51,152.86
204	10294	$4,424.40
205	10145	$50,342.74
205	10189	$3,879.96
205	10367	$39,580.60
209	10241	$36,069.26
209	10255	$4,632.31
209	10405	$35,157.75
211	10187	$28,287.73
211	10200	$17,193.06
216	10118	$3,101.40
216	10197	$40,473.86
216	10340	$24,945.21
219	10154	$4,465.85
219	10376	$3,452.75
227	10161	$36,164.46
227	10314	$53,745.34
233	10171	$16,909.84
233	10261	$22,997.45
233	10411	$29,070.38
239	10222	$56,822.65
239	10226	$23,552.59
240	10232	$24,995.61
240	10316	$46,788.14
242	10136	$14,232.70

customerNumber	orderNumber	order_value
242	10178	$33,818.34
242	10397	$12,432.32
249	10280	$48,298.99
249	10293	$33,924.24
250	10134	$23,419.47
250	10356	$26,311.63
250	10395	$17,928.09
256	10216	$5,759.42
256	10304	$53,116.99
259	10191	$27,988.47
259	10310	$61,234.67
260	10235	$29,284.42
260	10283	$37,527.58
276	10148	$41,554.73
276	10169	$38,547.19
276	10370	$27,083.78
276	10391	$29,848.52
278	10106	$52,151.81
278	10173	$37,723.79
278	10328	$37,654.09
282	10139	$24,013.52
282	10270	$35,806.73
282	10361	$31,835.36
282	10420	$42,251.51
286	10285	$43,134.04
286	10305	$47,411.33
298	10225	$47,375.92
298	10287	$61,402.00
299	10284	$32,260.16
299	10301	$36,798.88
311	10151	$32,723.04
311	10239	$16,212.59
311	10373	$46,770.52
314	10221	$16,901.38
314	10273	$45,352.47

customerNumber	orderNumber	order_value
314	10423	$8,597.73
319	10195	$36,092.40
319	10308	$42,339.76
320	10143	$41,016.75
320	10185	$52,548.49
320	10365	$8,307.28
321	10159	$54,682.68
321	10162	$30,876.44
321	10381	$32,626.09
321	10384	$14,155.57
323	10132	$2,880.00
323	10254	$37,281.36
323	10354	$39,440.59
323	10393	$33,593.32
323	10404	$41,426.81
324	10129	$29,429.14
324	10175	$37,455.77
324	10351	$13,671.82
328	10233	$7,178.66
328	10251	$31,102.85
328	10401	$43,525.04
333	10152	$9,821.32
333	10174	$23,936.53
333	10374	$21,432.31
334	10141	$29,716.86
334	10247	$28,394.54
334	10363	$45,785.34
339	10183	$34,606.28
339	10307	$23,333.06
344	10177	$31,428.21
344	10231	$15,322.93
347	10160	$20,452.50
347	10209	$21,053.69
350	10122	$50,824.66
350	10344	$18,888.31

customerNumber	orderNumber	order_value
350	10364	$1,834.56
353	10121	$16,700.47
353	10137	$13,920.26
353	10343	$17,104.91
353	10359	$32,600.61
353	10398	$46,656.94
357	10202	$20,220.04
357	10260	$37,769.38
357	10410	$36,442.34
362	10264	$18,473.71
362	10295	$15,059.76
362	10414	$50,806.85
363	10100	$10,223.83
363	10192	$55,425.77
363	10322	$50,799.69
379	10147	$32,680.31
379	10274	$12,530.51
379	10369	$28,322.83
381	10116	$1,627.56
381	10144	$1,128.20
381	10338	$12,081.52
381	10366	$14,379.90
382	10119	$35,826.33
382	10269	$6,419.84
382	10341	$42,813.83
382	10419	$52,420.07
385	10108	$51,001.22
385	10198	$20,644.24
385	10330	$15,822.84
386	10176	$38,524.29
386	10266	$51,619.02
386	10416	$35,362.26
398	10258	$22,037.91
398	10339	$48,927.64
398	10372	$33,967.73

customerNumber	orderNumber	order_value
398	10408	$615.45
406	10211	$49,165.16
406	10252	$25,080.96
406	10402	$12,190.85
412	10234	$31,670.37
412	10268	$35,034.57
412	10418	$23,627.44
415	10296	$31,310.09
424	10115	$21,665.98
424	10163	$22,042.37
424	10337	$25,505.98
447	10131	$17,032.29
447	10146	$6,631.36
447	10353	$26,304.13
448	10167	$44,167.09
448	10291	$48,809.90
448	10389	$27,966.54
450	10250	$42,798.08
450	10257	$16,753.30
450	10400	$31,755.34
450	10407	$52,229.55
452	10164	$27,121.90
452	10170	$15,130.97
452	10392	$8,807.12
455	10196	$38,139.18
455	10245	$32,239.47
456	10242	$1,679.92
456	10319	$27,550.51
458	10126	$57,131.92
458	10214	$22,162.61
458	10348	$33,145.56
462	10166	$9,977.85
462	10321	$48,355.87
462	10388	$30,293.77
471	10193	$35,505.63

customerNumber	orderNumber	order_value
471	10265	$9,415.13
471	10415	$10,945.26
473	10157	$17,746.26
473	10218	$7,612.06
475	10199	$7,678.25
475	10215	$36,070.47
484	10184	$47,513.19
484	10303	$3,474.66
486	10109	$25,833.14
486	10236	$5,899.38
486	10331	$45,994.07
487	10149	$29,997.09
487	10219	$12,573.28
489	10186	$22,275.73
489	10213	$7,310.42
495	10207	$59,265.14
495	10243	$6,276.60
496	10138	$32,077.44
496	10179	$22,963.60
496	10360	$52,166.00
496	10399	$30,253.75

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/101000`
  - 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 [11]:
```
import requests
```

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

In [19]:
```
url = "http://160.39.213.248:5003/api/orders_resources/10101"
res = requests.get(url)
res = res.json()

res
```

Out[19]:
```
{'orderNumber': 10101,
 'orderDate': '2003-01-09',
 'requiredDate': '2003-01-18',
 'shippedDate': '2003-01-11',
 'status': 'Shipped',
 'comments': 'Check on availability.',
 'customerNumber': 128}
```

In [35]:
```
url = "http://160.39.213.248:5003/api/orders_resources/orders?customerNumber=103
res = requests.get(url)
res = res.json()

res
```

Out[35]:
```
{'data': [{'customerNumber': 103, 'orderNumber': 10123},
  {'customerNumber': 103, 'orderNumber': 10298},
  {'customerNumber': 103, 'orderNumber': 10345}],
 'links': [{'rel': 'self',
   'href': 'http://160.39.213.248:5003/api/orders_resources/orders?customerNumbe
r=103&status=shipped&fields=customerNumber,%20orderNumber'}]}
```

In [112…
```
#create function
#returns 201 when an object is created

url = 'http://160.39.213.248:5003/api/orders_resources'
payload = {'orderNumber': 11001, 'orderDate': '2003-01-10', 'requiredDate': '200
res = requests.post(url, json=payload)
```

```
#res = res.json()

res
```

Out[112…
```
<Response [201]>
```

In [109…
```
#update function
#return 1 when the item is successfully updated, 0 otherwise

url = 'http://160.39.213.248:5003/api/orders_resources/10100'
payload = {'customerNumber': 112, 'status': 'Shipped'}
res = requests.put(url, json=payload)
res = res.json()

res
```

Out[109…
```
1
```

In [111…
```
#delete function
#return 200 when successfully deleted
url = "http://160.39.213.248:5003/api/orders_resources/11001"
res = requests.delete(url)

res
```

Out[111…
```
<Response [200]>
```

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

In [122…
```
pic1 = 'application1.png'
pic2 = 'application2.png'
pic3 = 'getbytemplate.png'
pic4 = 'create.png'
pic5 = 'update.png'
pic6 = 'delete.png'
pic7 = 'getbyid.png'



print("\n")
from IPython.display import Image
Image(filename=pic1)
```

Out[122…

```
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)
    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)
        if res is None:
            rsp = Response("ERROR", status=404, content_type="text/plain")
        else:
            rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    elif request_inputs.method == "POST":
        data = request_inputs.data
        print(data)
        res = svc.create(data)
        if res is None:
            rsp = Response("ERROR", status=404, content_type="text/plain")
        else:
            rsp = Response("CREATED", status=201, content_type="text/plain")
    else:
        rsp = Response("NOT IMPLEMENTED", status=501, content_type="text/plain")

    return rsp
```

In [116…

```
Image(filename=pic2)
```

Out[116…

```
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)
    svc = service_factory.get(resource_collection)

    if request_inputs.method == "GET":
        res = svc.get_resource_by_id(resource_id)
        if res is None:
            rsp = Response("ERROR", status=404, content_type="text/plain")
        else:
            rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")

    elif request_inputs.method == "PUT":
        res = svc.update_resource_by_id(resource_id, request_inputs.data)
        if res == 1:
            rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")
        else:
            rsp = Response(json.dumps(res, default=str), status=404, content_type="application/json")

    elif request_inputs.method == "DELETE":
        res = svc.delete_resource_by_id(resource_id)
        if res == 1:
            rsp = Response(json.dumps(res, default=str), status=200, content_type="application/json")
        else:
            rsp = Response(json.dumps(res, default=str), status=404, content_type="application/json")



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

In [117…

```
Image(filename=pic3)
```

Out[117…

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

    """This is a logical abstraction of an SQL SELECT statement...."""
    sql = "select " \
          + ",".join(field_list) \
          + " from " + self.db_table_full_name + " where " \
          + " and ".join(["%s = '%s'" % (key, val) if not type(val) == int
                          else "%s = %s" % (key, val)
                          for (key, val) in template.items()])

    conn = self._get_connection()
    cursor = conn.cursor()
    print(sql)
    res = cursor.execute(sql)
    if res == 0:
        result = None
    else:
        result = cursor.fetchall()
    return result
```

In [118…

```
Image(filename=pic4)
```

Out[118…

```
def create(self, new_resource):
    """..."""

    requiredCol = {"orderNumber", "orderDate", "requiredDate", "status", "customerNumber"}
    for name in requiredCol:
        if name not in new_resource:
            return None

    conn = self._get_connection()
    cursor = conn.cursor()
    check_statement = "select * from classicmodels.orders where orderNumber=" + str(new_resource['orderNumber'])
    row = cursor.execute(check_statement)
    if row > 0:
        return None

    sql = "insert into " + self.db_table_full_name \
          + '(' + ','.join(new_resource) \
          + ") values(" \
          + ','.join(["'%s'" % val if not type(val) == int
                      else "%s" % val
                      for val in new_resource.values()]) + ')'

    res = cursor.execute(sql)
    result = new_resource['orderNumber']
    return result
```

In [119…
```
Image(filename=pic5)
```

Out[119…
```
def update_resource_by_id(self, id, new_values):
    """This is a logical abstraction of an SQL UPDATE statement...."""

    conn = self._get_connection()
    cursor = conn.cursor()
    check_statement = "select * from classicmodels.customers where customerNumber=%s"
    row = cursor.execute(check_statement, (new_values['customerNumber']))
    if row == 0:
        print("555sad")
        return 0

    sql = "update " + self.db_table_full_name \
          + " set " \
          + ", ".join(["%s = '%s'" % (key, val) if not type(val) == int
                       else "%s = %s" % (key, val)
                       for (key, val) in new_values.items()]) \
          + " where orderNumber =%s"
    res = cursor.execute(sql, id)
    return 1
```

In [120…
```
Image(filename=pic6)
```

Out[120…
```
def delete_resource_by_id(self, id):
    """This is a logical abstraction of an SQL DELETE statement...."""

    conn = self._get_connection()
    cursor = conn.cursor()

    check_statement = "select * from classicmodels.orders where orderNumber=%s"
    row = cursor.execute(check_statement, (id))
    if row == 0:
        return 0

    sql = "delete from " + self.db_table_full_name + " where orderNumber = %s"
    res = cursor.execute(sql, (id))
    return 1
```

In [121…
```
Image(filename=pic7)
```

Out[121…

```
def get_resource_by_id(self, id):
    sql = "select * from " + self.db_table_full_name + " where orderNumber=%s"
    conn = self._get_connection()
    cursor = conn.cursor()
    # the_sql = cursor.mogrify(sql,(id))
    res = cursor.execute(sql, id)
    if res == 1:
        print(res)
        result = cursor.fetchone()
    else:
        result = None
    return result
```

In [ ]: