COMS W4111: Introduction to Databases
Spring 2024, Sections 002/V02

Homework 2: Programming

In [ ]:

Introduction

This notebook contains HW2 Programming. **Only students on the programming track should complete this part.** To ensure everything runs as expected, work on this notebook in Jupyter.

Submission instructions:

- You will submit **PDF and ZIP files** for this assignment. Gradescope will have two separate assignments for these.
- For the PDF:
  - The most reliable way to save as PDF is to go to your browser's menu bar and click `File -> Print`. **Switch the orientation to landscape mode**, and hit save.
  - **MAKE SURE ALL YOUR WORK (CODE AND SCREENSHOTS) IS VISIBLE ON THE PDF. YOU WILL NOT GET CREDIT IF ANYTHING IS CUT OFF.** Reach out for troubleshooting.
- For the ZIP:
  - Zip the folder that contains this notebook, any screenshots, and the code you write.
  - To avoid freezing Gradescope with too many files, when you finish this assignment, delete any unnecessary directories. Such directories include `venv`, `.idea`, and `.git`.

Setup

SQL Magic

The `sql` extension was installed in HW0. Double check that if this cell doesn't work.

In [ ]:
```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

You may need to change the password below.

In [ ]:
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [ ]:
```
%sql SELECT * FROM db_book.student WHERE ID = 12345
```

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

Out[ ]:

ID	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32

Python Libraries

In [ ]:
```
!pip install pandas
!pip install sqlalchemy
!pip install requests
```

```
Requirement already satisfied: pandas in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/ava/opt/anaconda3/lib/python3.9/site-packag
es (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (from p
andas) (2023.3.post1)
Requirement already satisfied: numpy>=1.20.3 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (from
pandas) (1.26.1)
Requirement already satisfied: six>=1.5 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (from pytho
n-dateutil>=2.8.1->pandas) (1.16.0)

[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
Requirement already satisfied: sqlalchemy in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (2.0.21)
Requirement already satisfied: typing-extensions>=4.2.0 in /Users/ava/opt/anaconda3/lib/python3.9/site-pack
ages (from sqlalchemy) (4.7.1)
Requirement already satisfied: greenlet!=0.4.17 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (fr
om sqlalchemy) (2.0.1)

[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
Requirement already satisfied: requests in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/ava/opt/anaconda3/lib/python3.9/site-pack
ages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages (from r
equests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages
(from requests) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /Users/ava/opt/anaconda3/lib/python3.9/site-packages
(from requests) (2023.7.22)

[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
```

In [ ]:

```python
import json

import pandas as pd
from sqlalchemy import create_engine
import requests
```

You may need to change the password below.

In [ ]:
```
engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

Data Definition and Insertion

Create Tables

- The directory contains a file `people_info.csv`. The columns are
  - `first_name`
  - `middle_name`
  - `last_name`
  - `email`
  - `employee_type`, which can be one of `Professor`, `Lecturer`, `Staff`. The value is empty if the person is a student.
  - `enrollment_year` which must be in the range `2016–2023`. The value is empty if the person is an employee.
- In the cell below, create two tables, `student` and `employee`
  - You should choose appropriate data types for the attributes
  - You should add an attribute `student_id` to `student` and `employee_id` to `employee`. **These attributes should be auto-incrementing numbers.** They are the PKs of their tables.
  - `email` should be unique and non-null in their tables. You don't need to worry about checking whether `email` is unique across both tables.
  - `student` should have all the columns listed above except `employee_type`. You should have some way to ensure that `enrollment_year` is always in range.
  - `employee` should have all the columns listed above except `enrollment_year`. You should have some way to ensure that `employee_type` is one of the valid values.

In [ ]:
```
%%sql

DROP SCHEMA IF EXISTS s24_hw2;
```

```
CREATE SCHEMA s24_hw2;
USE s24_hw2;

## Add CREATE TABLEs below
CREATE TABLE student (
    student_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    email varchar(40) NOT NULL UNIQUE,
    enrollment_year int NOT NULL CHECK ( enrollment_year >= 2016 AND enrollment_year <= 2023 ),
    first_name varchar(50) NOT NULL,
    middle_name varchar(50),
    last_name varchar(50) NOT NULL

);
 CREATE TABLE employee (
    employee_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    email varchar(40) NOT NULL UNIQUE,
    employee_type enum('Professor', 'Lecturer', 'Staff'),
    first_name varchar(50) NOT NULL,
    middle_name varchar(50),
    last_name varchar(50) NOT NULL

 );
```

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

Out[ ]: []

Inserting Data

- Below we read `people_info.csv` into a Pandas Dataframe
- You should implement `get_students` and `get_employees`, which extract the student/employee rows from a dataframe of people
- If you implement the functions correctly, the next cell should run with no errors and insert data into the tables you created above

In [ ]:
```
df = pd.read_csv("./people_info.csv")
df
```

Out[ ]:

	first_name	middle_name	last_name	email	employee_type	enrollment_year
0	Sanders	Arline	Breckell	abreckell1x@fotki.com	Professor	NaN
1	Zared	NaN	Fenelon	afenelona@themeforest.net	NaN	2021.0
2	Ethelin	NaN	Fidele	afidele12@google.ru	Lecturer	NaN
3	Bibbye	Annabal	Guesford	aguesfordb@tumblr.com	NaN	2018.0
4	Xenia	Ardella	Kief	akieft@free.fr	Staff	NaN
...	...	...	...	...	...	...
95	Norry	NaN	Rubinchik	trubinchik16@howstuffworks.com	NaN	2016.0
96	Doug	NaN	Medforth	vmedforth1o@homestead.com	Staff	NaN
97	Gerty	NaN	O'Donegan	vodoneganf@clickbank.net	NaN	2020.0
98	Anabelle	Wallas	Quimby	wquimby1c@nba.com	NaN	2022.0
99	Sasha	Win	Ruffli	wruffli2q@wordpress.com	Lecturer	NaN

100 rows × 6 columns

In [ ]:
```python
def get_students(df):
    """Given a dataframe of people df, returns a new dataframe that only contains students.
    The returned dataframe should have all the attributes of the people df except `employee_type`.
    """
    students = df[df["enrollment_year"].notnull()].drop(labels="employee_type", axis=1)
    # print(students)
    return students

def get_employees(df):
    """Given a dataframe of people df, returns a new dataframe that only contains employees.
    The returned dataframe should have all the attributes of the people df except `enrollment_year`.
    """
    employees = df[df["employee_type"].notnull()].drop(labels="enrollment_year", axis=1)
    # print(employees)
    return employees
```

In [ ]:
```python
student_df = get_students(df)
employee_df = get_employees(df)

student_df.to_sql("student", schema="s24_hw2", index=False, if_exists="append", con=engine)
employee_df.to_sql("employee", schema="s24_hw2", index=False, if_exists="append", con=engine)
```

Out[ ]: 50

In [ ]:
```
%%sql

 Students
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.ProgrammingError) (1064, "You have an error in your SQL syntax; check the manual that correspo
nds to your MySQL server version for the right syntax to use near 'Students' at line 1")
[SQL: Students]
(Background on this error at: https://sqlalche.me/e/20/f405)
```

API Implementation

- You will create an API that allows users to read, create, update, and delete students and employees

- The `src/` directory has the following structure:

```
src
|
|- db.py
|
|- db_test.py
|
|- main.py
```

Python Environment

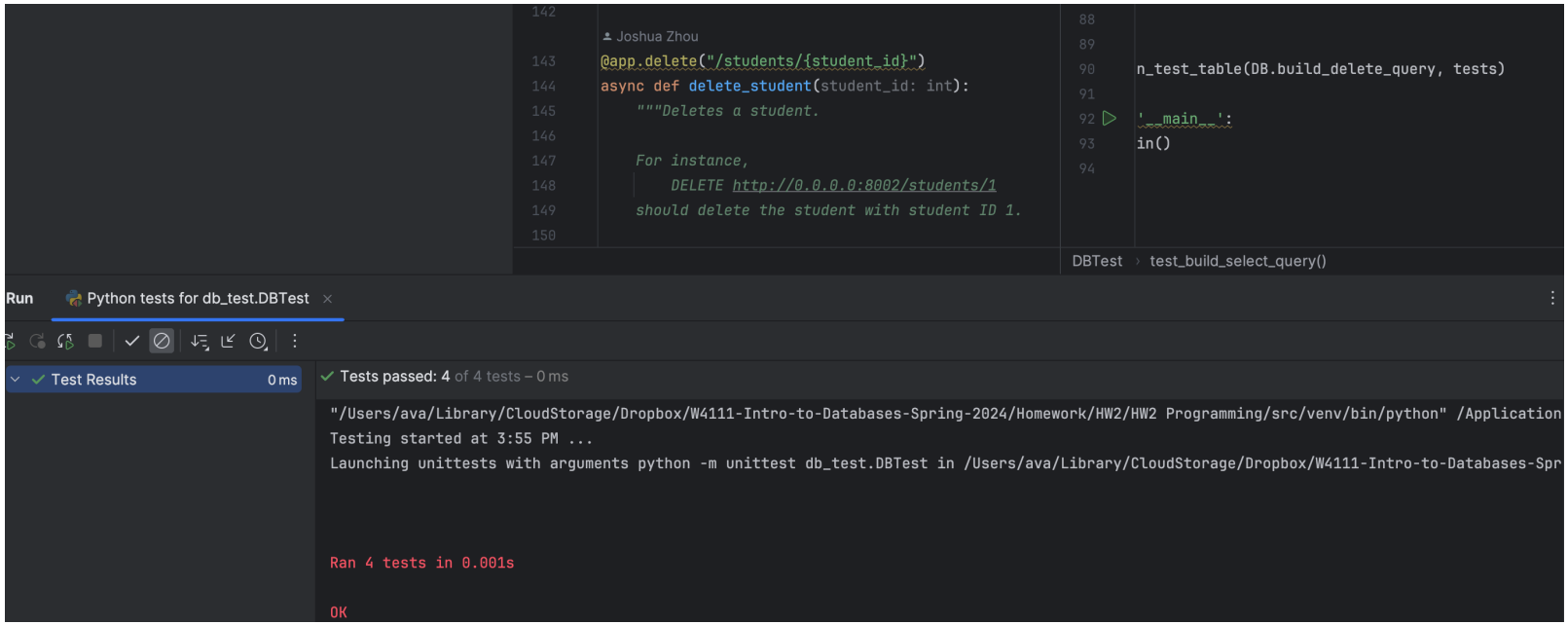
1. Open the `src/` folder in PyCharm
2. Follow these instructions to set up a virtual environment. This'll give us an blank, isolated environment for packages that we install. It's fine to use the `Virtualenv Environment` tab.
3. Open the Terminal in PyCharm. Make sure your virtual environment is active (you'll probably see `(venv)` somewhere).
    A. If you don't, the docs may be helpful
4. Run `pip install -r requirements.txt`
    A. `requirements.txt` contains all the packages that the project needs, such as `pymysql`

db.py

- Implement the eight methods in `db.py`: `build_select_query`, `select`, `build_insert_query`, `insert`, `build_update_query`, `update`, `build_delete_query`, and `delete`
    - To see examples of the inputs and expected outputs for the `build_*` functions, see `db_test.py`

db_test.py

- To test your `build_*` methods, run the `db_test.py` file. This file defines some unit tests.
- **Post a screenshot of your successful tests below**



Successful Unit Tests

main.py

- `main.py` declares our API and defines paths for it
  - The `@app` decorator above each method describes the HTTP method and the path associated with that method
- Implement the ten endpoints in `main.py`: `get_students`, `get_student`, `post_student`, `put_student`, `delete_student`, `get_employees`, `get_employee`, `post_employee`, `put_employee`, and `delete_employee`

Testing Your API

Student Testing

- With your API running, execute the following cells
  - Successful cells may have no output. However, failing cells will generate an error.

In [ ]:
```
BASE_URL = "http://localhost:8002/"

def print_json(j):
    print(json.dumps(j, indent=2))
```

In [ ]:
```
# Healthcheck

r = requests.get(BASE_URL)
print(r.text)
```

```
<h1>Heartbeat</h1>
```

In [ ]:
```
# Get all students

r = requests.get(BASE_URL + "students")
print(r)
j = r.json()
# print(j)

assert len(j) == 50, "There should be 50 students after inserting data"
```

```
<Response [200]>
```

In [ ]:
```
# Get specific attributes

r = requests.get(BASE_URL + "students?enrollment_year=2018&fields=first_name,last_name")
j = r.json()

print_json(j)
assert len(j) == 7, "There should be 7 students that graudated in 2018"
assert all(map(lambda o: len(o) == 2 and "first_name" in o and "last_name" in o, j)), \
"All student JSONs should have two attributes, first_name and last_name"
```

```
[
  {
    "first_name": "Bibbye",
    "last_name": "Guesford"
  },
  {
    "first_name": "Barry",
    "last_name": "Elias"
  },
  {
    "first_name": "Avie",
    "last_name": "Blissitt"
  },
  {
    "first_name": "Shea",
    "last_name": "Bates"
  },
  {
    "first_name": "Mal",
    "last_name": "Issett"
  },
  {
    "first_name": "Rozelle",
    "last_name": "Vigar"
  },
  {
    "first_name": "Drona",
    "last_name": "McKinie"
  }
]
```


In [ ]:
```python
# Test bad gets

# Invalid ID
r = requests.get(BASE_URL + "students/100")
assert r.status_code == 404, f"Invalid ID: Expected 404 Not Found but got {r.status_code}"
```

In [ ]:
```python
# Create a new student

or_student = {
    "first_name": "Michael",
    "last_name": "Jan",
```

```
    "email": "ap@columbia.edu",
    "enrollment_year": 2019,
}

r = requests.post(BASE_URL + "students", json=or_student)
assert r.status_code == 201, f"Expected 201 Created but got {r.status_code}"
```

In [ ]:
```
# Get that student

r = requests.get(BASE_URL + "students/51")
j = r.json()

print_json(j)
assert j == {
    'student_id': 51,
    'first_name': 'Michael',
    'middle_name': None,
    'last_name': 'Jan',
    'email': 'ap@columbia.edu',
    'enrollment_year': 2019,
}, "Newly inserted student does not match what we specified"
```


```
{
  "student_id": 51,
  "email": "ap@columbia.edu",
  "enrollment_year": 2019,
  "first_name": "Michael",
  "middle_name": null,
  "last_name": "Jan"
}
```


In [ ]:
```
# Test bad posts

# Duplicate email
bad_student = {
    "first_name": "Foo",
    "last_name": "Bar",
    "email": "ap@columbia.edu",
    "enrollment_year": 2018,
}
r = requests.post(BASE_URL + "students", json=bad_student)
assert r.status_code == 400, f"Duplicate email: Expected 400 Bad Request but got {r.status_code}"
```

```
# Email not specified
bad_student = {
    "first_name": "Foo",
    "last_name": "Bar",
    "enrollment_year": 2018,
}
r = requests.post(BASE_URL + "students", json=bad_student)
assert r.status_code == 400, f"Email not specified: Expected 400 Bad Request but got {r.status_code}"

# Invalid year
bad_student = {
    "first_name": "Foo",
    "last_name": "Bar",
    "email": "fb@columbia.edu",
    "enrollment_year": 2011,
}
r = requests.post(BASE_URL + "students", json=bad_student)
assert r.status_code == 400, f"Invalid year: Expected 400 Bad Request but got {r.status_code}"
```

In [ ]:
```
# Update the student

r = requests.put(BASE_URL + "students/51", json={"enrollment_year": "2020"})
assert r.status_code == 200, f"Expected 200 OK but got {r.status_code}"
```

In [ ]:
```
# Test bad puts

# Duplicate email
bad_student = {
    "email": "csimeons2@microsoft.com",
}
r = requests.put(BASE_URL + "students/51", json=bad_student)
assert r.status_code == 400, f"Duplicate email: Expected 400 Bad Request but got {r.status_code}"

# Email set to null
bad_student = {
    "email": None
}
r = requests.put(BASE_URL + "students/51", json=bad_student)
assert r.status_code == 400, f"Null email: Expected 400 Bad Request but got {r.status_code}"
```

```python
# Invalid year
bad_student = {
    "enrollment_year": 2011
}
r = requests.put(BASE_URL + "students/51", json=bad_student)
assert r.status_code == 400, f"Invalid year: Expected 400 Bad Request but got {r.status_code}"

# Invalid ID
bad_student = {
    "enrollment_year": 2018
}
r = requests.put(BASE_URL + "students/100", json=bad_student)
assert r.status_code == 404, f"Invalid ID: Expected 404 Not Found but got {r.status_code}"
```

In [ ]:
```python
# Delete the student

r = requests.delete(BASE_URL + "students/51")
assert r.status_code == 200, f"Expected 200 OK but got {r.status_code}"
```

In [ ]:
```python
# Try to get deleted student

r = requests.get(BASE_URL + "students/51")
assert r.status_code == 404, f"Expected 404 Not Found but got {r.status_code}"
```

In [ ]:
```python
# Test bad deletes

r = requests.delete(BASE_URL + "students/100")
assert r.status_code == 404, f"Invalid ID: Expected 404 Not Found but got {r.status_code}"
```

Employee Testing

- Write similar tests below to test your employee endpoints

In [ ]:
```python
# Get all employees

r = requests.get(BASE_URL + "employees")
print(r)
j = r.json()
print(j)
```

```
assert len(j) == 50, "There should be 50 employees after inserting data"
```

<Response [200]>
[{'employee_id': 1, 'email': 'abreckell1x@fotki.com', 'employee_type': 'Professor', 'first_name': 'Sanders'
, 'middle_name': 'Arline', 'last_name': 'Breckell'}, {'employee_id': 2, 'email': 'afidele12@google.ru', 'em
ployee_type': 'Lecturer', 'first_name': 'Ethelin', 'middle_name': None, 'last_name': 'Fidele'}, {'employee_
id': 3, 'email': 'akieft@free.fr', 'employee_type': 'Staff', 'first_name': 'Xenia', 'middle_name': 'Ardella
', 'last_name': 'Kief'}, {'employee_id': 4, 'email': 'aleask1n@devhub.com', 'employee_type': 'Lecturer', 'f
irst_name': 'Cari', 'middle_name': 'Andriana', 'last_name': 'Leask'}, {'employee_id': 5, 'email': 'bbradnoc
kek@nifty.com', 'employee_type': 'Lecturer', 'first_name': 'Lemmy', 'middle_name': 'Burr', 'last_name': 'Br
adnocke'}, {'employee_id': 6, 'email': 'blalley2d@rediff.com', 'employee_type': 'Lecturer', 'first_name': '
Sibylle', 'middle_name': 'Bearnard', 'last_name': 'Lalley'}, {'employee_id': 7, 'email': 'cflaxman1b@cdbaby
.com', 'employee_type': 'Lecturer', 'first_name': 'Lu', 'middle_name': 'Cinnamon', 'last_name': 'Flaxman'},
{'employee_id': 8, 'email': 'dcroalx@purevolume.com', 'employee_type': 'Professor', 'first_name': 'Hobart',
'middle_name': 'Dominic', 'last_name': 'Croal'}, {'employee_id': 9, 'email': 'dfavey2p@mozilla.com', 'emplo
yee_type': 'Staff', 'first_name': 'Marylin', 'middle_name': 'Darcy', 'last_name': 'Favey'}, {'employee_id':
10, 'email': 'dwarmishame@soundcloud.com', 'employee_type': 'Staff', 'first_name': 'Ailbert', 'middle_name'
: 'Danie', 'last_name': 'Warmisham'}, {'employee_id': 11, 'email': 'ebree1z@creativecommons.org', 'employee
_type': 'Professor', 'first_name': 'Karon', 'middle_name': None, 'last_name': 'Bree'}, {'employee_id': 12,
'email': 'ecella26@mail.ru', 'employee_type': 'Staff', 'first_name': 'Maybelle', 'middle_name': 'Esteban',
'last_name': 'Cella'}, {'employee_id': 13, 'email': 'efulk1d@discuz.net', 'employee_type': 'Staff', 'first_
name': 'Lelah', 'middle_name': 'Ellette', 'last_name': 'Fulk'}, {'employee_id': 14, 'email': 'gblagden1q@bu
zzfeed.com', 'employee_type': 'Professor', 'first_name': 'Gisela', 'middle_name': None, 'last_name': 'Blagd
en'}, {'employee_id': 15, 'email': 'gform18@blogger.com', 'employee_type': 'Staff', 'first_name': 'Niki', '
middle_name': 'Gardiner', 'last_name': 'Form'}, {'employee_id': 16, 'email': 'ghellyar2a@cornell.edu', 'emp
loyee_type': 'Staff', 'first_name': 'Suki', 'middle_name': None, 'last_name': 'Hellyar'}, {'employee_id': 1
7, 'email': 'gtolmanr@slideshare.net', 'employee_type': 'Staff', 'first_name': 'Carmine', 'middle_name': No
ne, 'last_name': 'Tolman'}, {'employee_id': 18, 'email': 'gyousef2r@spotify.com', 'employee_type': 'Profess
or', 'first_name': 'Wells', 'middle_name': None, 'last_name': 'Yousef'}, {'employee_id': 19, 'email': 'hsie
gertsz21@instagram.com', 'employee_type': 'Professor', 'first_name': 'Christie', 'middle_name': None, 'last
_name': 'Siegertsz'}, {'employee_id': 20, 'email': 'jaslin24@redcross.org', 'employee_type': 'Lecturer', 'f
irst_name': 'Doyle', 'middle_name': None, 'last_name': 'Aslin'}, {'employee_id': 21, 'email': 'jcharte1y@me
rriam-webster.com', 'employee_type': 'Staff', 'first_name': 'Robinett', 'middle_name': 'Jami', 'last_name':
'Charte'}, {'employee_id': 22, 'email': 'jhedley1m@disqus.com', 'employee_type': 'Staff', 'first_name': 'Ol
wen', 'middle_name': None, 'last_name': 'Hedley'}, {'employee_id': 23, 'email': 'jplessing0@samsung.com', '
employee_type': 'Staff', 'first_name': 'Renae', 'middle_name': 'Jaquith', 'last_name': 'Plessing'}, {'emplo
yee_id': 24, 'email': 'jsnodin20@princeton.edu', 'employee_type': 'Lecturer', 'first_name': 'Lilllie', 'mid
dle_name': None, 'last_name': 'Snodin'}, {'employee_id': 25, 'email': 'kmcknishs@reddit.com', 'employee_typ
e': 'Lecturer', 'first_name': 'Sayers', 'middle_name': 'Karon', 'last_name': 'McKnish'}, {'employee_id': 26
, 'email': 'kmorfell2g@istockphoto.com', 'employee_type': 'Professor', 'first_name': 'Electra', 'middle_nam
e': 'Krystle', 'last_name': 'Morfell'}, {'employee_id': 27, 'email': 'kslipper2f@nytimes.com', 'employee_ty
pe': 'Staff', 'first_name': 'Matthieu', 'middle_name': 'Kalle', 'last_name': 'Slipper'}, {'employee_id': 28
, 'email': 'ktrehearn19@tinyurl.com', 'employee_type': 'Lecturer', 'first_name': 'Nadia', 'middle_name': No
ne, 'last_name': 'Trehearn'}, {'employee_id': 29, 'email': 'lguislin2o@chicagotribune.com', 'employee_type'

: 'Professor', 'first_name': 'Clim', 'middle_name': None, 'last_name': 'Guislin'}, {'employee_id': 30, 'email': 'lhenstridgeh@sogou.com', 'employee_type': 'Lecturer', 'first_name': 'Holli', 'middle_name': None, 'last_name': 'Henstridge'}, {'employee_id': 31, 'email': 'lkleinschmidtl@squarespace.com', 'employee_type': 'Lecturer', 'first_name': 'Meghan', 'middle_name': None, 'last_name': 'Kleinschmidt'}, {'employee_id': 32, 'email': 'lpurbrick25@canalblog.com', 'employee_type': 'Professor', 'first_name': 'Genni', 'middle_name': None, 'last_name': 'Purbrick'}, {'employee_id': 33, 'email': 'lscheffel7@taobao.com', 'employee_type': 'Professor', 'first_name': 'Bonny', 'middle_name': None, 'last_name': 'Scheffel'}, {'employee_id': 34, 'email': 'ltennick3@aboutads.info', 'employee_type': 'Staff', 'first_name': 'Mendie', 'middle_name': None, 'last_name': 'Tennick'}, {'employee_id': 35, 'email': 'minkinc@google.de', 'employee_type': 'Staff', 'first_name': 'Karole', 'middle_name': None, 'last_name': 'Inkin'}, {'employee_id': 36, 'email': 'mpenzer14@dailymail.co.uk', 'employee_type': 'Professor', 'first_name': 'Kahaleel', 'middle_name': 'Meg', 'last_name': 'Penzer'}, {'employee_id': 37, 'email': 'mwynrahamem@admin.ch', 'employee_type': 'Professor', 'first_name': 'Darrin', 'middle_name': 'Mario', 'last_name': 'Wynrahame'}, {'employee_id': 38, 'email': 'mwyrallj@scientificamerican.com', 'employee_type': 'Staff', 'first_name': 'Carey', 'middle_name': 'Maudie', 'last_name': 'Wyrall'}, {'employee_id': 39, 'email': 'nbolles23@ucoz.ru', 'employee_type': 'Staff', 'first_name': 'Jacky', 'middle_name': 'Nydia', 'last_name': 'Bolles'}, {'employee_id': 40, 'email': 'nmacneely22@cpanel.net', 'employee_type': 'Lecturer', 'first_name': 'Francene', 'middle_name': None, 'last_name': 'MacNeely'}, {'employee_id': 41, 'email': 'rmabbs4@xing.com', 'employee_type': 'Lecturer', 'first_name': 'Bamby', 'middle_name': 'Rubetta', 'last_name': 'Mabbs'}, {'employee_id': 42, 'email': 'rsellek6@oakley.com', 'employee_type': 'Lecturer', 'first_name': 'Ari', 'middle_name': 'Rheta', 'last_name': 'Sellek'}, {'employee_id': 43, 'email': 'shiggonet2b@163.com', 'employee_type': 'Lecturer', 'first_name': 'Bettina', 'middle_name': 'Sonya', 'last_name': 'Higgonet'}, {'employee_id': 44, 'email': 'sjohlg@soundcloud.com', 'employee_type': 'Professor', 'first_name': 'Jany', 'middle_name': 'Sherry', 'last_name': 'Johl'}, {'employee_id': 45, 'email': 'slyles1j@amazon.de', 'employee_type': 'Staff', 'first_name': 'Pattie', 'middle_name': None, 'last_name': 'Lyles'}, {'employee_id': 46, 'email': 'smalacrida1w@economist.com', 'employee_type': 'Staff', 'first_name': 'Kristin', 'middle_name': None, 'last_name': 'Malacrida'}, {'employee_id': 47, 'email': 'smccolleyq@amazon.com', 'employee_type': 'Staff', 'first_name': 'Yehudi', 'middle_name': 'Sile', 'last_name': 'McColley'}, {'employee_id': 48, 'email': 'ssillars2l@unicef.org', 'employee_type': 'Professor', 'first_name': 'Duncan', 'middle_name': 'Shellie', 'last_name': 'Sillars'}, {'employee_id': 49, 'email': 'vmedforth1o@homestead.com', 'employee_type': 'Staff', 'first_name': 'Doug', 'middle_name': None, 'last_name': 'Medforth'}, {'employee_id': 50, 'email': 'wruffli2q@wordpress.com', 'employee_type': 'Lecturer', 'first_name': 'Sasha', 'middle_name': 'Win', 'last_name': 'Ruffli'}]

In [ ]:
```
r = requests.get(BASE_URL + "employees/25")
j = r.json()

print(j)
```

{'employee_id': 25, 'email': 'kmcknishs@reddit.com', 'employee_type': 'Lecturer', 'first_name': 'Sayers', 'middle_name': 'Karon', 'last_name': 'McKnish'}

In [ ]:
```
# post employees
```

```
good_employee = {"first_name": "Ava",
                 "last_name": "Hajratwala",
                 "employee_type": "Staff",
                 "email": "avahaj@hotmail.com",
                 "middle_name": "Sarah"
                 }

r = requests.post(BASE_URL + "employees", json=good_employee)

assert r.status_code == 201, f"status code is {r.status_code}"

r = requests.post(BASE_URL + "employees", json=good_employee)
assert r.status_code == 400, f"duplicate email: {r.status_code}"

bad_employee = {
    "first_name": "Ava",
    "last_name": "Hajratwala",
    "employee_type": "slacker",
    "email": "avahaj@gmail.com",
    "middle_name": "Sarah",
}
r = requests.post(BASE_URL + "employees", json=bad_employee)
assert r.status_code == 400, f"invalid employee type of {bad_employee['employee_type']}: {r.status_code}"
```

In [ ]:

```
# change employees
new_employee = {
    "first_name": "Donald"
}

r = requests.put(BASE_URL + "employees/1", json=new_employee)
assert r.status_code == 200, f"change unsuccessful: {r.status_code}"


r = requests.get(BASE_URL + "employees/1")
j = r.json()

assert j['first_name'] == 'Donald', f"change not showing in db; id 1 is {j['first_name']}"

# should be a duplicate email
new_employee = {"email": "avahajr@gmail.com"}

r = requests.put(BASE_URL + "employees/1", json=new_employee)
```

```
assert r.status_code == 400, f"got wrong code: {r.status_code}"

new_employee = {"email": None}
r = requests.put(BASE_URL + "employees/1", json=new_employee)

assert r.status_code == 400, f"got wrong code: {r.status_code}"
```

In [ ]:

```
# test delete

r = requests.delete(BASE_URL+"employees/1")
assert r.status_code == 200, f"failed delete with status code {r.status_code}"

# verify that id 1 is gone
r = requests.get(BASE_URL + "employees/1")
assert r.status_code == 404, f"despite just being deleted, the record for id 1 is still there: status code

r = requests.delete(BASE_URL + "employees/38902840238403094829")
assert r.status_code == 404, f"Nonsensical query has status code {r.status_code}"
```