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

Homework 2: Programming

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 [1]:
```
%load_ext sql
```

You may need to change the password below.

In [2]:
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [3]:
```
%sql SELECT * FROM db_book.student WHERE ID = 12345
```

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

Out[3]:

ID	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32

Python Libraries

```
In [4]: !pip install pandas
        !pip install sqlalchemy
        !pip install requests
```

```
Requirement already satisfied: pandas in /Users/sparshbinjrajka/anaconda3/lib/python3.11/site-pa
ckages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/sparshbinjrajka/anaconda3/lib/py
thon3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/sparshbinjrajka/anaconda3/lib/python3.11/s
ite-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /Users/sparshbinjrajka/anaconda3/lib/python3.1
1/site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in /Users/sparshbinjrajka/anaconda3/lib/python3.11/
site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in /Users/sparshbinjrajka/anaconda3/lib/python3.11/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: sqlalchemy in /Users/sparshbinjrajka/anaconda3/lib/python3.11/sit
e-packages (2.0.27)
Requirement already satisfied: typing-extensions>=4.6.0 in /Users/sparshbinjrajka/anaconda3/lib/
python3.11/site-packages (from sqlalchemy) (4.9.0)
Requirement already satisfied: requests in /Users/sparshbinjrajka/anaconda3/lib/python3.11/site-
packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/sparshbinjrajka/anaconda3/lib/
python3.11/site-packages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/sparshbinjrajka/anaconda3/lib/python3.11/s
ite-packages (from requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/sparshbinjrajka/anaconda3/lib/python
3.11/site-packages (from requests) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/sparshbinjrajka/anaconda3/lib/python
3.11/site-packages (from requests) (2023.7.22)
```

In [5]:
```python
import json

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

You may need to change the password below.

In [6]:
```python
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

In [7]:
```
%%sql

DROP SCHEMA IF EXISTS s24_hw2;
CREATE SCHEMA s24_hw2;
USE s24_hw2;

## Add CREATE TABLEs below

create table student
(
    student_id      int auto_increment primary key ,
    first_name      varchar(50),
    middle_name     varchar(50),
    last_name       varchar(50),
    email           varchar(100) unique not null,
    enrollment_year YEAR CHECK (enrollment_year BETWEEN 2016 AND 2023)
);

create table employee
(
    employee_id     int auto_increment primary key ,
    first_name      varchar(50),
    middle_name     varchar(50),
    last_name       varchar(50),
    email           nvarchar(255) unique not null,
    employee_type   ENUM('Professor', 'Lecturer', 'Staff') not null
);
```

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

Out[7]:
```
[]
```

Used the following sources:

1. https://stackoverflow.com/questions/51149902/sql-table-data-type-for-email-address (https://stackoverflow.com/questions/51149902/sql-table-data-type-for-email-address)
2. https://www.w3schools.com/sql/sql_check.asp (https://www.w3schools.com/sql/sql_check.asp)

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 [8]:
```
df = pd.read_csv("./people_info.csv")
df
```

Out[8]:

	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 [9]:
```
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`
    """
    student =  df.loc[df['employee_type'].isnull()]
    student = student.drop('employee_type', axis=1)
    return student

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_yea
    """
    employee = df.loc[df['enrollment_year'].isnull()]
    employee = employee.drop('enrollment_year', axis=1)
    return employee
```

In [10]:
```
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[10]:
```
50
```

API Implementation

- You will create an API that allows users to [read, create, update, and delete (https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) students and employees
- The `src/` directory has the following structure:

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

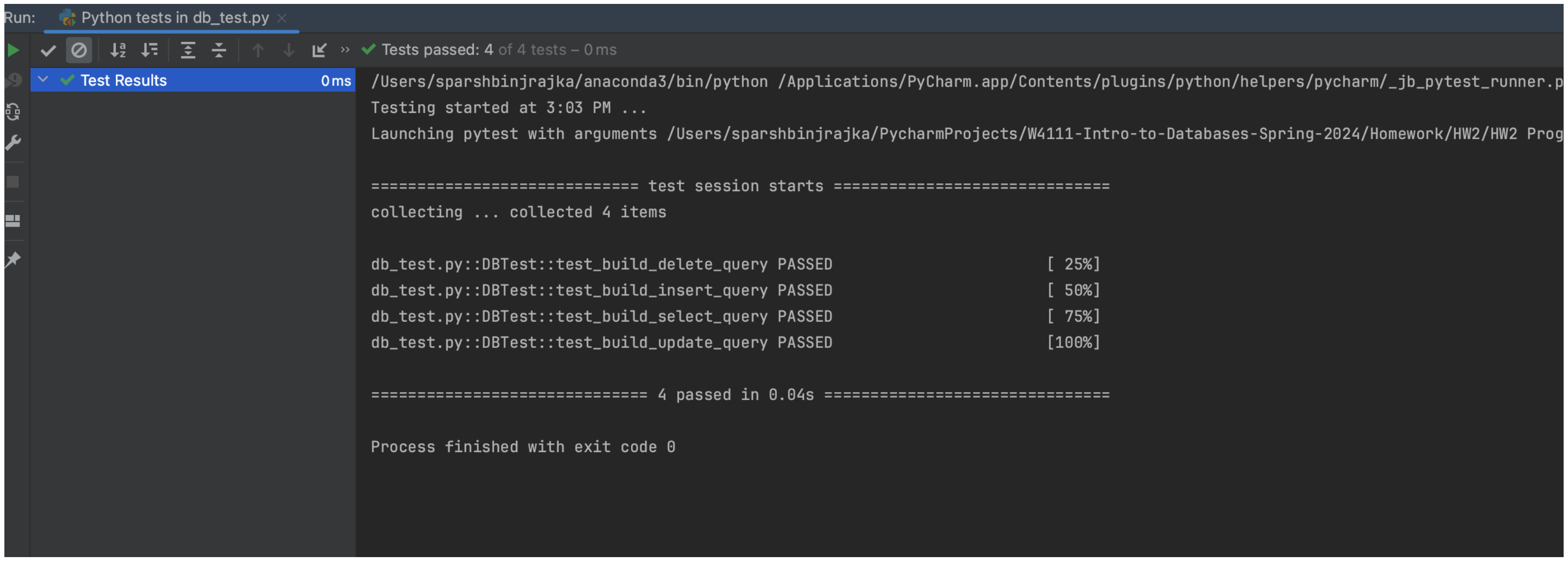
Python Environment

1. Open the `src/` folder in PyCharm
2. Follow [these instructions (https://www.jetbrains.com/help/pycharm/creating-virtual-environment.html#python_create_virtual_env)](https://www.jetbrains.com/help/pycharm/creating-virtual-environment.html#python_create_virtual_env) 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 (https://docs.python.org/3/library/venv.html#how-venvs-work)](https://docs.python.org/3/library/venv.html#how-venvs-work) 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



```
/Users/sparshbinjrajka/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pycharm/_jb_pytest_runner.p
Testing started at 3:03 PM ...
Launching pytest with arguments /Users/sparshbinjrajka/PycharmProjects/W4111-Intro-to-Databases-Spring-2024/Homework/HW2/HW2 Prog

============================ test session starts ==============================
collecting ... collected 4 items

db_test.py::DBTest::test_build_delete_query PASSED                       [ 25%]
db_test.py::DBTest::test_build_insert_query PASSED                       [ 50%]
db_test.py::DBTest::test_build_select_query PASSED                       [ 75%]
db_test.py::DBTest::test_build_update_query PASSED                       [100%]

============================= 4 passed in 0.04s ==============================

Process finished with exit code 0
```

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 [11]:
```
BASE_URL = "http://localhost:8002/"

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

In [12]:
```
# Healthcheck

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

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

In [13]:
```
# Get all students

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

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

```
50
```

```
In [14]: # 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 [15]:
```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 [16]:
```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)
print(r)
assert r.status_code == 201, f"Expected 201 Created but got {r.status_code}"
```

```
<Response [201]>
```

In [17]:
```python
# 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,
  "first_name": "Michael",
  "middle_name": null,
  "last_name": "Jan",
  "email": "ap@columbia.edu",
  "enrollment_year": 2019
}
```

In [18]:
```
# 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_co

# 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 [19]:
```
# 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 [20]:
```
# 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}"

# 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 [21]:
```
# 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 [22]:
```
# 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 [23]:
```
# 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 [24]:
```
# Get all employeed

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

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

```
50
```

In [25]:
```python
# Get specific attributes

r = requests.get(BASE_URL + "employees?employee_type=Professor&fields=first_name,email")
j = r.json()

print_json(j)
assert len(j) == 14, "There should be 14 employees that are Professors"
assert all(map(lambda o: len(o) == 2 and "first_name" in o and "email" in o, j)), \
"All employee JSONs should have two attributes, first_name and email"
```

```
[
  {
    "first_name": "Sanders",
    "email": "abreckell1x@fotki.com"
  },
  {
    "first_name": "Hobart",
    "email": "dcroalx@purevolume.com"
  },
  {
    "first_name": "Karon",
    "email": "ebree1z@creativecommons.org"
  },
  {
    "first_name": "Gisela",
    "email": "gblagden1q@buzzfeed.com"
  },
  {
    "first_name": "Wells",
    "email": "gyousef2r@spotify.com"
  },
  {
    "first_name": "Christie",
    "email": "hsiegertsz21@instagram.com"
  },
  {
    "first_name": "Electra",
    "email": "kmorfell2g@istockphoto.com"
  },
  {
    "first_name": "Clim",
    "email": "lguislin2o@chicagotribune.com"
  },
  {
    "first_name": "Genni",
    "email": "lpurbrick25@canalblog.com"
  },
  {
    "first_name": "Bonny",
    "email": "lscheffel7@taobao.com"
  },
  {
    "first_name": "Kahaleel",
    "email": "mpenzer14@dailymail.co.uk"
  },
  {
    "first_name": "Darrin",
    "email": "mwynrahamem@admin.ch"
  },
  {
    "first_name": "Jany",
    "email": "sjohlg@soundcloud.com"
  },
  {
    "first_name": "Duncan",
    "email": "ssillars2l@unicef.org"
  }
]
```

In [26]:
```
# Test bad gets

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

In [27]:
```
# Create a new employee

or_emp = {
    "first_name": "Don",
    "last_name": "Ferguson",
    "email": "dff@columbia.edu",
    "employee_type": "Professor",
}

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

```
<Response [201]>
```

In [28]:
```python
# Get that employee

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

print_json(j)
assert j == {
    'employee_id': 51,
    'first_name': 'Don',
    'middle_name': None,
    'last_name': 'Ferguson',
    'email': 'dff@columbia.edu',
    'employee_type': 'Professor',
}, "Newly inserted employee does not match what we specified"
```

```
{
  "employee_id": 51,
  "first_name": "Don",
  "middle_name": null,
  "last_name": "Ferguson",
  "email": "dff@columbia.edu",
  "employee_type": "Professor"
}
```

In [29]:
```python
# Test bad posts

# Duplicate email
bad_emp = {
    "first_name": "Foo",
    "last_name": "Bar",
    "email": "dff@columbia.edu",
    "employee_type": "Lecturer",
}
r = requests.post(BASE_URL + "employees", json=bad_emp)
assert r.status_code == 400, f"Duplicate email: Expected 400 Bad Request but got {r.status_code}"

# Email not specified
bad_emp = {
    "first_name": "Foo",
    "last_name": "Bar",
    "employee_type": "Lecturer",
}
r = requests.post(BASE_URL + "employees", json=bad_emp)
assert r.status_code == 400, f"Email not specified: Expected 400 Bad Request but got {r.status_co

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

In [30]:
```python
# Update the employee

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

In [31]:
```
# Test bad puts

# Duplicate email
bad_emp = {
    "email": "rsellek6@oakley.com",
}
r = requests.put(BASE_URL + "employees/51", json=bad_emp)
assert r.status_code == 400, f"Duplicate email: Expected 400 Bad Request but got {r.status_code}"

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

# Invalid job_type
bad_emp = {
    "employee_type": "Teacher"
}
r = requests.put(BASE_URL + "employees/51", json=bad_emp)
assert r.status_code == 400, f"Invalid year: Expected 400 Bad Request but got {r.status_code}"

# Invalid ID
bad_emp = {
    "employee_type": "Lecturer"
}
r = requests.put(BASE_URL + "employees/100", json=bad_emp)
assert r.status_code == 404, f"Invalid ID: Expected 404 Not Found but got {r.status_code}"
```

In [32]:
```
# Delete the employee

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

In [33]:
```python
# Try to get deleted employee

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

In [34]:
```python
# Test bad deletes

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