Vincent Ryan & Paul Singleton

7/21/24

Module 10- Milestone 3

For this assignment, a large portion of the work was getting fake data into the system and then learning the ins and outs of writing queries in a " DRY " way. It was a collaborative effort, and despite Python not being new, Python > SQL feels like an entirely new beast. We worked on creating a Table class for more replicative writing to aid in the written queries.

## Table Class

class Table:

*'''*

*Python SQL Table class for functionality*

*'''*

def \_\_init\_\_(*self*, *database*, *table*):

*self*.\_database = *database*

*self*.\_config = CONFIG

*self*.\_table = *table*

*self*.columns = {}

*self*.route\_to\_table()

def \_\_repr\_\_(*self*):

*return* f'{*self*.\_table}'

def route\_to\_table(*self*):

*'''*

*Connects to the database and creates the table.*

*'''*

print(f'\nInitializing {*self*.\_table}')

*self*.query(f'CREATE DATABASE IF NOT EXISTS {*self*.\_database}', True)

*self*.\_config['database'] = *self*.\_database

*self*.query(

f'CREATE TABLE IF NOT EXISTS {*self*.\_table} ('

f'{*self*.\_table}\_id INT NOT NULL AUTO\_INCREMENT,'

f'PRIMARY KEY({*self*.\_table}\_id))', True

)

print(f'Connected to - {*self*.\_table}\n')

*self*.get\_columns()

def connect(*self*):

*'''*

*Returns a connection to the database.*

*'''*

*try*:

*return* mysql.connector.connect(\*\**self*.\_config)

*except* mysql.connector.Error *as* err:

print(err)

@staticmethod

def print\_results(*results*):

*'''*

*Prints query results.*

*'''*

*for* row *in* *results*:

*for* key, value *in* row.items():

print(f'{key}: {value}')

print('\n')

@staticmethod

def fix\_values(*value*):

*'''*

*method to true value*

*'''*

fixed\_values = ''

*for* val *in* *value*:

*if* not val:

fixed\_values += 'NULL, '

*else*:

fixed\_values += f'"{str(val)}", '

fixed\_values = fixed\_values[:-2]

*return* fixed\_values

def fix\_columns(*self*, *value*: tuple):

*if* len(*value*) == 3:

*return* (*value*[0], *value*[1])

*return* (f'{*self*.\_table}\_{*value*[0]}', *value*[1])

def query(*self*, *query*: str = None, *silent*=False):

*'''*

*Runs the query and prints results.*

*'''*

*if* not *query*:

*query* = f'SELECT \* FROM {*self*.\_table}'

*with* *self*.connect() *as* db:

*with* db.cursor(*dictionary*=True) *as* cursor:

*try*:

cursor.execute(*query*)

*self*.print\_results(cursor.fetchall())

*return* True

*except*:

*if* not *silent*:

print(f'{*query*} was not executed')

*return* False

def insert(*self*, *values*: list, *include\_id*=False):

*'''*

*Inserts values into the table.\n*

*Values are a list of tuples allowing multiple line entries at once\n*

*Headers are populated automatically from the table\n*

*The default is to \*not\* include\_id, because this auto-updates and often doesn't matter\n*

*Set include\_id to True to manually input values for id, these cannot be duplicates or they will fail\n*

*'''*

columns = list(*self*.columns.keys())

*if* not *include\_id*:

columns = columns[1:]

fixed\_columns = ', '.join(columns)

*for* val *in* *values*:

*# SQlify values*

fixed\_values = *self*.fix\_values(val)

*# Check for missing data*

*if* len(columns) != len(val):

print('A value is missing see below')

print(f'-- Columns: {fixed\_columns}')

print(f'-- Values: {fixed\_values}')

*else*:

insert = (

f'INSERT INTO {*self*.\_table} '

f'({fixed\_columns}) '

f'VALUES ({fixed\_values})'

)

*if* *self*.query(insert):

print(

f'-- {fixed\_values} inserted into {*self*.\_table} successfully.')

*# for val in values:*

*# insert = (*

*# f'INSERT INTO {self.\_table} '*

*# f'({headers}) '*

*# f'VALUES ({val})'*

*# )*

*# if self.query(insert):*

*# print(f'-- {val} inserted into {self.\_table} successfully.')*

def delete(*self*, *criteria*: str = None):

*'''*

*Deletes rows matching the criteria.\n*

*Leave empty to clear all data*

*'''*

delete = (

f'DELETE FROM {*self*.\_table} '

)

*if* *criteria*:

delete += f'WHERE {*criteria*}'

*if* *self*.query(delete):

print(f'-- Rows in {*self*.\_table} deleted successfully')

*if* not *criteria*:

*self*.query(f'ALTER TABLE {*self*.\_table} AUTO\_INCREMENT = 1')

def update(*self*, *update*: str, *criteria*: str):

*'''*

*Updates rows matching the criteria.*

*'''*

*update* = (

f'UPDATE {*self*.\_table} '

f'SET {*update*} '

f'WHERE {*criteria*}'

)

*if* *self*.query(*update*):

print(f'-- {*self*.\_table} updated successfully')

def select(*self*, *values*: str = None, *joins*: tuple = None, *where*: str = None, *group\_by*: str = None, *order\_by*: str = None, *limit*: int = None):

*'''*

*Selects and prints rows\n*

*Default is that all special modifiers are set to none.\n*

*Any can be set independantly of every other\n*

*Joins are a tuple of strings, or Tables which will be processed with the Table.join function\n*

*'''*

*# Values*

*if* not *values*:

*values* = '\*'

selection = f'SELECT {*values*} FROM {*self*.\_table} '

*# Join*

*if* *joins*:

*for* val *in* *joins*:

*if* type(val) == Table:

selection += *self*.join(val)

*else*:

selection += val

*# Where*

*if* *where*:

selection += f'WHERE {*where*} '

*# Group*

*if* *group\_by*:

selection += f'GROUP BY {*group\_by*} '

*# Order*

*if* *order\_by*:

selection += f'ORDER BY {*order\_by*} '

*if* *limit*:

selection += f'LIMIT {*limit*} '

*self*.query(selection)

def show(*self*):

*'''*

*Shortcut to show the table content.*

*'''*

*self*.select()

def add\_columns(*self*, \**columns*: tuple):

*'''*

*Adds columns to the table as tuples\n*

*(Column Name, Column Type)\n\n*

*\*\* DO NOT include table name in Column Name.\*\*\n*

*\*\* Table name is added automatically for consistency \*\*\n*

*\*\* You can add foreign keys directly with link\_to() \*\*\n\n*

*Example: (Name, VARCHAR(250) NOT NULL)\n*

*'''*

*for* col *in* *columns*:

column\_name, column\_type = *self*.fix\_columns(col)

*if* column\_name not in *self*.columns.keys():

*if* *self*.query(f'ALTER TABLE {*self*.\_table} ADD {column\_name.lower()} {column\_type.upper()}'):

print(f'Added {column\_name} to {*self*.\_table} successfully')

*self*.get\_columns()

*else*:

print(f'{column\_name} already exists in {*self*.\_table}\n')

def rename(*self*, *change\_to*: str, *change\_from*: str = None):

*'''*

*leave "change\_from" blank to change table name\n*

*Otherwise, put old column names in change\_from, and new ones in change\_to.*

*'''*

query = f'ALTER TABLE {*self*.\_table} '

*if* *change\_from*: *# updating column*

*if* *change\_from* in *self*.columns.keys():

query += f'RENAME COLUMN {*change\_from*} TO {*change\_to*}'

*else*:

print(f'{*change\_from*} is not in {*self*.\_table}\n')

print('Please check the spelling')

*self*.show\_columns()

*else*: *# Updating table*

query += f'RENAME TO {*change\_to*}'

*self*.rename(f'{*change\_to*}\_id', f'{*self*.\_table}\_id')

*self*.\_table = *change\_to*

*if* *self*.query(query):

print(f'Renamed to {*change\_to*} successfully')

*if* *change\_from*:

*self*.get\_columns()

def drop\_columns(*self*, \**columns*: str):

*'''*

*Drops columns from the table.*

*'''*

*for* col *in* *columns*:

*if* *self*.query(f'ALTER TABLE {*self*.\_table} DROP COLUMN {col}'):

print(f'{col} dropped from {*self*.\_table} successfully')

*del* *self*.columns[col]

def link\_to(*self*, \**tables*):

*'''*

*Adds and links a foreign key to the provided Tables*

*'''*

*for* table *in* *tables*:

link\_id = f'{table}\_id'

*self*.add\_columns((link\_id, 'INT NOT NULL', True))

*if* *self*.query(f'ALTER TABLE {*self*.\_table} ADD FOREIGN KEY({link\_id}) '

f'REFERENCES {table}({link\_id})'):

print(f'{*self*.\_table} linked to {table} successfully')

def get\_columns(*self*):

*with* *self*.connect() *as* db:

*with* db.cursor(*dictionary*=True) *as* cursor:

*try*:

cursor.execute(f'SELECT \* FROM {*self*.\_table}')

cursor.fetchall()

*self*.columns = {

col[0]: FieldType.get\_info(col[1]) *for* col *in* cursor.description}

*except*:

print(f'Could not retrieve headers from {*self*.\_table}')

def show\_columns(*self*):

print(f'-- {*self*.\_table} table columns --')

*for* k, v *in* *self*.columns.items():

print(f'\n Column: {k}'

f'\n Type: {v}')

def join(*self*, \**tables*):

*'''*

*Supply tables as tables\n*

*An inner join is created where ID's match across tables assuming they reference the same thing.*

*'''*

joint = ''

me = set(*self*.columns.keys())

*for* table *in* *tables*:

them = set(table.columns.keys())

shared\_key = list(me.intersection(them))[0]

*if* shared\_key:

joint += (f'INNER JOIN {table} ON '

f'{*self*.\_table}.{shared\_key} = {table}.{shared\_key} ')

*return* joint

## Reports:

orders = Table('bacchus', 'orders')

order\_details = Table('bacchus', 'order\_details')

distributors = Table('bacchus', 'distributors')

products = Table('bacchus', 'products')

employees = Table('bacchus', 'employees')

hours = Table('bacchus', 'hours')

suppliers = Table('bacchus', 'suppliers')

supplies = Table('bacchus', 'supplies')

supply\_orders = Table('bacchus', 'supply\_orders')

print('---------- Wine Sales ----------')

order\_details.select(

*values*='products\_name as Name, SUM(order\_details\_qty) as "Total Sold"', *group\_by*='products\_name', *joins*=(products,))

print('---------- Which distributor carries which wine? ----------')

orders.select(*values*='distributors\_name as Distributor, products\_name as Product, SUM(order\_details\_qty) as "Total Ordered"',

*joins*=(order\_details, distributors,

order\_details.join(products)),

*group\_by*='distributors\_name, products\_name',

*order\_by*='distributors\_name')

print('---------- During the last four quarters, how many hours did each employee work? ----------')

hours.select(*values*='QUARTER(hours\_week\_end) as Quarter, employees\_name AS Employee, SUM(hours\_qty) AS "Hours Worked"',

*joins*=(employees,),

*group\_by*='QUARTER(hours\_week\_end), employees\_name',

*order\_by*='QUARTER(hours\_week\_end)')

print('---------- Orders received later than expected ----------')

supplies.select(*values*=('suppliers\_name as Supplier, supplies\_description as Supply, '

'supply\_orders\_expected\_date as Expected, supply\_orders\_received\_date as Received'),

*joins*=(supply\_orders, suppliers),

*where*='supply\_orders\_received\_date > supply\_orders\_expected\_date ')

print('---------- Supply orders not received yet ----------')

supplies.select(*values*=('suppliers\_name as Supplier, supplies\_description as Supply, '

'supply\_orders\_expected\_date as Expected'),

*joins*=(supply\_orders, suppliers), *where*='supply\_orders\_received\_date IS NULL')

print('---------- A month by month report of supply orders ----------')

supply\_orders.select(

*values*=('YEAR(supply\_orders\_expected\_date) as Year, Month(supply\_orders\_expected\_date) as Month, '

'supplies\_description as Supply'),

*joins*=(supplies,),

*order\_by*='YEAR(supply\_orders\_expected\_date)')

## Results:

---------- Wine Sales ----------

Name: Merlot

Total Sold: 582

Name: Chablis

Total Sold: 516

Name: Cabernet

Total Sold: 325

Name: Chardonnay

Total Sold: 322

---------- Which distributor carries which wine? ----------

Distributor: Exquisite Taste

Product: Chablis

Total Ordered: 50

Distributor: Exquisite Taste

Product: Merlot

Total Ordered: 32

Distributor: New Wines

Product: Cabernet

Total Ordered: 200

Distributor: New Wines

Product: Chablis

Total Ordered: 150

Distributor: New Wines

Product: Chardonnay

Total Ordered: 135

Distributor: New Wines

Product: Merlot

Total Ordered: 250

Distributor: Vineyard Ventures

Product: Cabernet

Total Ordered: 125

Distributor: Vineyard Ventures

Product: Merlot

Total Ordered: 300

Distributor: Wine Distributors Inc.

Product: Chablis

Total Ordered: 316

Distributor: Wine Distributors Inc.

Product: Chardonnay

Total Ordered: 187

---------- During the last four quarters, how many hours did each employee work? ----------

Quarter: 1

Employee: Bob Ulrich

Hours Worked: 489.54

Quarter: 1

Employee: Henry Doyle

Hours Worked: 510.76

Quarter: 1

Employee: Janet Collins

Hours Worked: 456.04

Quarter: 1

Employee: Maria Costanza

Hours Worked: 446.70

Quarter: 1

Employee: Roz Murphy

Hours Worked: 460.87

Quarter: 2

Employee: Bob Ulrich

Hours Worked: 528.92

Quarter: 2

Employee: Henry Doyle

Hours Worked: 527.28

Quarter: 2

Employee: Janet Collins

Hours Worked: 519.71

Quarter: 2

Employee: Maria Costanza

Hours Worked: 509.21

Quarter: 2

Employee: Roz Murphy

Hours Worked: 520.78

Quarter: 3

Employee: Bob Ulrich

Hours Worked: 569.09

Quarter: 3

Employee: Henry Doyle

Hours Worked: 548.12

Quarter: 3

Employee: Janet Collins

Hours Worked: 603.21

Quarter: 3

Employee: Maria Costanza

Hours Worked: 543.36

Quarter: 3

Employee: Roz Murphy

Hours Worked: 550.69

Quarter: 4

Employee: Bob Ulrich

Hours Worked: 541.43

Quarter: 4

Employee: Henry Doyle

Hours Worked: 545.46

Quarter: 4

Employee: Janet Collins

Hours Worked: 523.05

Quarter: 4

Employee: Maria Costanza

Hours Worked: 554.73

Quarter: 4

Employee: Roz Murphy

Hours Worked: 519.70

---------- Orders received later than expected ----------

Supplier: Pack It

Supply: Labels

Expected: 2021-03-08 00:00:00

Received: 2021-03-21 00:00:00

Supplier: Pack It

Supply: Labels

Expected: 2023-01-14 00:00:00

Received: 2023-02-01 00:00:00

---------- Supply orders not received yet ----------

Supplier: Bottle It Up

Supply: Bottles

Expected: 2024-02-03 00:00:00

Supplier: Bottle It Up

Supply: Corks

Expected: 2024-02-03 00:00:00

---------- A month by month report of supply orders ----------

Year: 2021

Month: 2

Supply: Bottles

Year: 2021

Month: 2

Supply: Corks

Year: 2021

Month: 3

Supply: Labels

Year: 2021

Month: 2

Supply: Boxes

Year: 2021

Month: 4

Supply: Vats

Year: 2021

Month: 4

Supply: Tubing

Year: 2022

Month: 2

Supply: Bottles

Year: 2022

Month: 2

Supply: Corks

Year: 2022

Month: 3

Supply: Boxes

Year: 2023

Month: 2

Supply: Bottles

Year: 2023

Month: 2

Supply: Corks

Year: 2023

Month: 1

Supply: Labels

Year: 2024

Month: 2

Supply: Bottles

Year: 2024

Month: 2

Supply: Corks

Year: 2024

Month: 1

Supply: Tubing