**Модуль 6: Реляційні бази даних**

**Заняття 2: Додаткові теми SQL**

**та приклад роботи з Python**

**Стандарт PEP249**

Стандарт [**PEP249**](https://www.python.org/dev/peps/pep-0249/) (https://peps.python.org/pep-0249/) (Python Database API Specification v2.0) був визначений для заохочення подібності між модулями Python, які використовуються для доступу до баз даних. Стандарт прагне досягти узгодженості, що веде до зрозуміліших модулів для підключення до баз даних з Python.

**Методи Connection**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-connection)

Необхідно підтримувати наступні методи підключення до бази даних:

* close() — закриття з'єднання;
* commit() — зафіксувати будь-яку відкладену транзакцію в базу даних;
* rollback() — цей метод є опціональним, оскільки не всі бази даних забезпечують підтримку транзакцій. У разі, якщо база даних надає транзакції, цей метод викликає відкочування бази даних до початку будь-якої очікуваної транзакції. Закриття з'єднання без попередньої фіксації змін спричинить неявне відкочування;
* cursor() — повернути новий об'єкт курсору за допомогою поточного з'єднання;

**Методи Cursor**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-cursor)Ці об'єкти представляють курсор бази даних, який використовується для управління контекстом операції вибірки. Курсори, створені з однієї й тієї самої сполуки, - не ізольовані, тобто будь-які зміни, внесені до бази даних курсором, негайно видно іншим курсорам. Об'єкти курсору повинні реагувати на такі методи.

* close() — закрити курсор та повернути ресурс системі (закриває відкрите з'єднання socket). З цього моменту курсор буде непридатним для використання. Виняток Error (або підклас) буде викликаний, якщо з курсором буде здійснено якусь операцію;
* execute(operation, parameters) — виконує команду SQL з параметрами. Параметри parameters можуть бути надані у вигляді послідовності або зіставлення та будуть прив'язані до змінних в операції. Параметри також можуть бути вказані у вигляді списку кортежів, наприклад, вставити кілька рядків в одну операцію, але таке використання застаріло: натомість варто використовувати .executemany().
* executemany(operation, seq\_of\_parameters) — те саме, що і execute, але seq\_of\_parameters — це список наборів параметрів та SQL operation буде виконаний із ними всіма по черзі. Прискорює виконання операції за рахунок повторного використання вже відкритого з'єднання до бази даних.
* fetchone() — отримати наступний рядок набору результатів запиту, повертаючи одну послідовність, або None, якщо більше немає доступних даних.
* fetchmany(size=cursor.arraysize) — отримати наступний набір рядків результату запиту, повертаючи послідовність послідовностей (наприклад, список кортежів). Порожня послідовність повертається, коли більше немає доступних рядків. Кількість рядків для вибірки за виклик вказується параметром size. Якщо він не вказаний, розмір масиву курсору визначає кількість рядків, що витягуються. Метод повинен спробувати отримати стільки рядків, скільки вказано параметром розміру. Якщо це неможливо через те, що вказана кількість рядків недоступна, може бути повернено меншу кількість рядків.
* fetchall() — отримати всі (решту) рядки результату запиту, повертаючи їх у вигляді послідовності послідовностей (наприклад, списку кортежів). Зверніть увагу, що атрибут масиву курсору може вплинути на продуктивність цієї операції.

**Підключення до бази даних**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%BF%D1%96%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)

Як приклад підключення до бази даних використовуємо SQLite. Це база даних у файлі, але основні принципи стандарту PEP249 до неї застосовані повною мірою.

**connect.py**

import sqlite3

from contextlib import contextmanager

database = './test.db'

@contextmanager

def create\_connection(db\_file):

""" create a database connection to a SQLite database """

conn = sqlite3.connect(db\_file)

yield conn

conn.rollback()

conn.close()

Створимо менеджер контексту у файлі connect.py.

**Створення таблиць**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8C)**create\_table.py**

from sqlite3 import Error

from connect import create\_connection, database

def create\_table(conn, create\_table\_sql):

""" create a table from the create\_table\_sql statement

:param conn: Connection object

:param create\_table\_sql: a CREATE TABLE statement

:return:

"""

try:

c = conn.cursor()

c.execute(create\_table\_sql)

conn.commit()

except Error as e:

print(e)

if \_\_name\_\_ == '\_\_main\_\_':

sql\_create\_projects\_table = """

CREATE TABLE IF NOT EXISTS projects (

id integer PRIMARY KEY,

name text NOT NULL,

begin\_date text,

end\_date text

);

"""

sql\_create\_tasks\_table = """

CREATE TABLE IF NOT EXISTS tasks (

id integer PRIMARY KEY,

name text NOT NULL,

priority integer,

project\_id integer NOT NULL,

status Boolean default False,

begin\_date text NOT NULL,

end\_date text NOT NULL,

FOREIGN KEY (project\_id) REFERENCES projects (id)

);

"""

with create\_connection(database) as conn:

if conn is not None:

# create projects table

create\_table(conn, sql\_create\_projects\_table)

# create tasks table

create\_table(conn, sql\_create\_tasks\_table)

else:

print("Error! cannot create the database connection.")

Створимо допоміжну функцію для створення таблиць create\_table. Потім напишемо SQL запити для створення двох таблиць: таблиця завдань tasks та проектів projects, до яких ці завдання належать:

sql\_create\_projects\_table = """

CREATE TABLE IF NOT EXISTS projects (

id integer PRIMARY KEY,

name text NOT NULL,

begin\_date text,

end\_date text

);

"""

sql\_create\_tasks\_table = """

CREATE TABLE IF NOT EXISTS tasks (

id integer PRIMARY KEY,

name text NOT NULL,

priority integer,

project\_id integer NOT NULL,

status Boolean default False,

begin\_date text NOT NULL,

end\_date text NOT NULL,

FOREIGN KEY (project\_id) REFERENCES projects (id)

);

"""

Для простоти скористаємося вбудованою в Python підтримкою SQLite і створимо базу даних у вигляді файлу.

with create\_connection(database) as conn:

if conn is not None:

# create projects table

create\_table(conn, sql\_create\_projects\_table)

# create tasks table

create\_table(conn, sql\_create\_tasks\_table)

else:

print("Error! cannot create the database connection.")

**Заповнення таблиць**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%B7%D0%B0%D0%BF%D0%BE%D0%B2%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8C)Тепер, коли у нас є дві таблиці з проектами та завданнями, напишемо допоміжні функції для створення проектів та завдань:

**seed.py**

from sqlite3 import Error

from connect import create\_connection, database

def create\_project(conn, project):

"""

Create a new project into the projects table

:param conn:

:param project:

:return: project id

"""

sql = '''

INSERT INTO projects(name,begin\_date,end\_date) VALUES(?,?,?);

'''

cur = conn.cursor()

try:

cur.execute(sql, project)

conn.commit()

except Error as e:

print(e)

finally:

cur.close()

return cur.lastrowid

def create\_task(conn, task):

"""

Create a new task

:param conn:

:param task:

:return:

"""

sql = '''

INSERT INTO tasks(name,priority,status,project\_id,begin\_date,end\_date) VALUES(?,?,?,?,?,?);

'''

cur = conn.cursor()

try:

cur.execute(sql, task)

conn.commit()

except Error as e:

print(e)

finally:

cur.close()

return cur.lastrowid

if \_\_name\_\_ == '\_\_main\_\_':

with create\_connection(database) as conn:

# create a new project

project = ('Cool App with SQLite & Python', '2022-01-01', '2022-01-30')

project\_id = create\_project(conn, project)

print(project\_id)

# tasks

task\_1 = ('Analyze the requirements of the app', 1, True, project\_id, '2022-01-01', '2022-01-02')

task\_2 = ('Confirm with user about the top requirements', 1, False, project\_id, '2022-01-03', '2022-01-05')

# create tasks

print(create\_task(conn, task\_1))

print(create\_task(conn, task\_2))

Для операцій створення/зміни/видалення ми можемо прочитати у курсора id рядка, з якого він востаннє працював в атрибуті lastrowid.

Створимо проект та кілька завдань до нього:

with create\_connection(database) as conn:

# create a new project

project = ('Cool App with SQLite & Python', '2022-01-01', '2022-01-30')

project\_id = create\_project(conn, project)

print(project\_id)

# tasks

task\_1 = ('Analyze the requirements of the app', 1, True, project\_id, '2022-01-01', '2022-01-02')

task\_2 = ('Confirm with user about the top requirements', 1, False, project\_id, '2022-01-03', '2022-01-05')

# create tasks

print(create\_task(conn, task\_1))

print(create\_task(conn, task\_2))

Виведення:

1

1

2

**Запити до бази даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%B7%D0%B0%D0%BF%D0%B8%D1%82%D0%B8-%D0%B4%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Тепер у нашій базі зберігається інформація про проект та два завдання. Щоб отримати цю інформацію в застосунку, створимо модуль select.py

**select.py**

from sqlite3 import Error

from connect import create\_connection, database

def select\_projects(conn):

"""

Query all rows in the tasks table

:param conn: the Connection object

:return: rows projects

"""

rows = None

cur = conn.cursor()

try:

cur.execute("SELECT \* FROM projects;")

rows = cur.fetchall()

except Error as e:

print(e)

finally:

cur.close()

return rows

def select\_all\_tasks(conn):

"""

Query all rows in the tasks table

:param conn: the Connection object

:return: rows tasks

"""

rows = None

cur = conn.cursor()

try:

cur.execute("SELECT \* FROM tasks")

rows = cur.fetchall()

except Error as e:

print(e)

finally:

cur.close()

return rows

def select\_task\_by\_status(conn, status):

"""

Query tasks by priority

:param conn: the Connection object

:param status:

:return: rows tasks

"""

rows = None

cur = conn.cursor()

try:

cur.execute("SELECT \* FROM tasks WHERE status=?", (status,))

rows = cur.fetchall()

except Error as e:

print(e)

finally:

cur.close()

return rows

if \_\_name\_\_ == '\_\_main\_\_':

with create\_connection(database) as conn:

print("Projects:")

projects = select\_projects(conn)

print(projects)

print("\nQuery all tasks")

tasks = select\_all\_tasks(conn)

print(tasks)

print("\nQuery task by status:")

task\_by\_priority = select\_task\_by\_status(conn, True)

print(task\_by\_priority)

Де функція для вибору всіх проектів - select\_projects, функція для вибору всіх завдань - select\_all\_tasks і, наприклад, функція вибору всіх завдань за статусом - select\_task\_by\_status.

Результат виконання скрипту select.py в консолі:

Projects:

[(1, 'Cool App with SQLite & Python', '2022-01-01', '2022-01-30')]

Query all tasks

[(1, 'Analyze the requirements of the app', 1, 1, 1, '2022-01-01', '2022-01-02'), (2, 'Confirm with user about the top requirements', 1, 1, 0, '2022-01-03', '2022-01-05')]

Query task by status:

[(1, 'Analyze the requirements of the app', 1, 1, 1, '2022-01-01', '2022-01-02')]

**Зміна даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%B7%D0%BC%D1%96%D0%BD%D0%B0-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Для зміни існуючих завдань використовуємо модуль update.py.

**update.py**

from sqlite3 import Error

from connect import create\_connection, database

def update\_task(conn, parameters):

"""

update priority, begin\_date, and end date of a task

:param conn:

:param parameters:

:return:

"""

sql = '''

UPDATE tasks

SET priority = ?, begin\_date = ?, end\_date = ?

WHERE id = ?

'''

cur = conn.cursor()

try:

cur.execute(sql, parameters)

conn.commit()

except Error as e:

print(e)

finally:

cur.close()

def update\_task\_status(conn, parameters):

"""

update priority, begin\_date, and end date of a task

:param conn:

:param parameters:

:return:

"""

sql = '''

UPDATE tasks

SET status = ?

WHERE id = ?

'''

cur = conn.cursor()

try:

cur.execute(sql, parameters)

conn.commit()

except Error as e:

print(e)

finally:

cur.close()

if \_\_name\_\_ == '\_\_main\_\_':

with create\_connection(database) as conn:

update\_task(conn, (2, '2022-01-04', '2022-01-06', 1))

update\_task\_status(conn, (True, 2))

Ми змінюємо для завдання з id=1 пріоритет та час виконання за допомогою функції update\_task. Також виставимо статус виконано для завдання id=2 за допомогою функції update\_task\_status

**Видалення даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)І остання операція, яку ми розглянемо, - це видалення завдань.

**delete.py**

from sqlite3 import Error

from connect import create\_connection, database

def delete\_task(conn, id):

"""

Delete a task by task id

:param conn: Connection to the SQLite database

:param id: id of the task

:return:

"""

sql = 'DELETE FROM tasks WHERE id=?'

cur = conn.cursor()

try:

cur.execute(sql, (id,))

conn.commit()

except Error as e:

print(e)

finally:

cur.close()

if \_\_name\_\_ == '\_\_main\_\_':

with create\_connection(database) as conn:

delete\_task(conn, 1)

Тут ми видаляємо завдання з id=1. У базі зараз залишається лише завдання з id=2.

**Висновок**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/script_python#%D0%B2%D0%B8%D1%81%D0%BD%D0%BE%D0%B2%D0%BE%D0%BA)

Ми розглянули стандарт PEP249 та основні операції роботи з базою даних у Python. Як бачимо в базі даних зараз залишилося одне завдання для проекту і це легко перевірити запитом отримати всі проекти з їх завданнями.

Тепер напишемо та виконаємо команду для отримання назви за завданням та проектом, якому вона належить:

from sqlite3 import Error

from connect import create\_connection, database

def select\_projects(conn):

"""

Query all rows in the projects table with its tasks

:param conn: the Connection object

:return: rows projects or None

"""

rows = None

cur = conn.cursor()

try:

cur.execute("SELECT \* FROM projects JOIN tasks ON tasks.project\_id = projects.id;")

rows = cur.fetchall()

except Error as e:

print(e)

finally:

cur.close()

return rows

if \_\_name\_\_ == '\_\_main\_\_':

with create\_connection(database) as conn:

print("Projects:")

projects = select\_projects(conn)

print(projects)

Результат:

Projects:

[(1, 'Cool App with SQLite & Python', '2022-01-01', '2022-01-30', 2, 'Confirm with user about the top requirements', 1, 1, 1, '2022-01-03', '2022-01-05')]

**Створюємо проект**

Ми розглянемо процес створення трьох пов'язаних таблиць у нашій базі даних (БД). Це таблиця companies з компаніями, таблиця employees зі співробітниками компаній та, нарешті, таблиця payments з інформацією про заробітну плату співробітників компанії.

У поточному прикладі буде використано стандартну бібліотеку python - sqlite3 та сторонній пакет [Faker](https://faker.readthedocs.io/en/master/) (https://faker.readthedocs.io/en/master/) для генерації випадкових даних.

Створюємо папку з проектом і виконуємо в консолі команду переходу в нього

mkdir EX-08

cd EX-08

Пам'ятаємо, що необхідно створити віртуальне оточення та активувати його (можна використовувати pipenv або poetry на вибір).

python -m venv venv

cd venv/scripts

venv/scripts >>> activate

**Створюємо скрипт із таблицями БД**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/intro#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D1%8E%D1%94%D0%BC%D0%BE-%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82-%D1%96%D0%B7-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8F%D0%BC%D0%B8-%D0%B1%D0%B4)Тепер створимо файл salary.sql зі скриптами SQL. Він необхідний нам для створення таблиць у БД мовою SQL

**salary.sql**

-- Table: companies

DROP TABLE IF EXISTS companies;

CREATE TABLE companies (

id INTEGER PRIMARY KEY AUTOINCREMENT,

company\_name VARCHAR(255) UNIQUE NOT NULL

);

-- Table: employees

DROP TABLE IF EXISTS employees;

CREATE TABLE employees (

id INTEGER PRIMARY KEY AUTOINCREMENT,

employee VARCHAR(255) UNIQUE NOT NULL,

post VARCHAR(120) NOT NULL,

company\_id INTEGER,

FOREIGN KEY (company\_id) REFERENCES companies (id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

-- Table: payments

DROP TABLE IF EXISTS payments;

CREATE TABLE payments (

id INTEGER PRIMARY KEY AUTOINCREMENT,

employee\_id INTEGER,

date\_of DATE NOT NULL,

total INTEGER NOT NULL,

FOREIGN KEY (employee\_id) REFERENCES employees (id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

В принципі вам тут повинно бути все зрозуміло після прочитання теоретичного матеріалу із попереднього розділу. Тут тільки біля поля id з'явилося ключове слово AUTOINCREMENT. Для SQLite це означає, що база сама заповнюватиме це поле автоматично, при вставленні даних, збільшуючи його значення на одиницю (автоінкремент). Вираз DROP TABLE IF EXISTS payments; означає, що потрібно видалити таблицю DROP TABLE, якщо вона існує IF EXISTS. Фактично при кожному запуску скрипту salary.sql ми очищуємо базу даних.

**Створюємо таблиці БД**

Напишемо тепер скрипт create\_db.py, де виконаємо підключення до БД та створимо наші таблиці.

НАГАДУВАННЯ

SQLite - це БД "в одному файлі". І цей файл буде створено автоматично при першому підключенні до неї.

**create\_db.py**

import sqlite3

def create\_db():

# читаємо файл зі скриптом для створення БД

with open('salary.sql', 'r') as f:

sql = f.read()

# створюємо з'єднання з БД (якщо файлу з БД немає, він буде створений)

with sqlite3.connect('salary.db') as con:

cur = con.cursor()

# виконуємо скрипт із файлу, який створить таблиці в БД

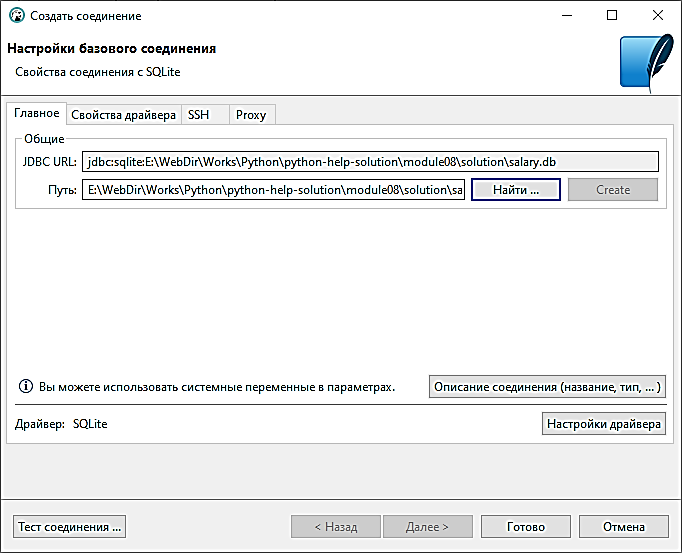
cur.executescript(sql)

if \_\_name\_\_ == "\_\_main\_\_":

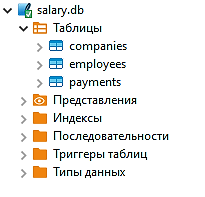
create\_db()

Після виконання скрипту у вас, в корені проекту, повинен з'явитися файл salary.db. Поки що він порожній і ми його ще заповнимо фейковими даними трохи згодом.

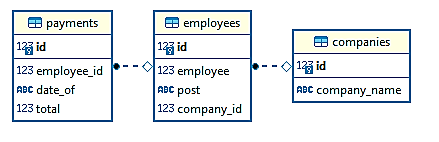
Зараз за допомогою програми [DBeaver](https://dbeaver.io/) (https://dbeaver.io/) виконаємо підключення до БД. Запустіть програму та на панелі інструментів виберіть нове підключення до БД SQLite. Через кнопку 'Знайти...' виберіть створений файл salary.db та натисніть кнопку 'Готово'. Все, підключення створено.



Якщо все було зроблено правильно, зліва на панелі ви повинні побачити наступне.



Двічі клацніть на 'Таблиці' і праворуч виберіть вкладку 'Діаграма'. Там повинна бути зображена ER-діаграма нашої БД



**Наповнюємо БД даними**

Перейдемо до заповнення даними нашої БД. У файлі fill\_data.py ми використовуємо пакет Faker (pip install faker).

У поточному прикладі наша база міститиме дані про 3 компанії NUMBER\_COMPANIES, в яких працює 30 осіб NUMBER\_EMPLOYESS. Персонал випадково розкиданий на 5 посад NUMBER\_POST.

**fill\_data.py**

from datetime import datetime

import faker

from random import randint, choice

import sqlite3

NUMBER\_COMPANIES = 3

NUMBER\_EMPLOYESS = 30

NUMBER\_POST = 5

def generate\_fake\_data(number\_companies, number\_employees, number\_post) -> tuple():

fake\_companies = [] # тут зберігатимемо компанії

fake\_employees = [] # тут зберігатимемо співробітників

fake\_posts = [] # тут зберігатимемо посади

'''Візьмемо три компанії з faker і помістимо їх у потрібну змінну'''

fake\_data = faker.Faker()

# Створимо набір компаній у кількості number\_companies

for \_ in range(number\_companies):

fake\_companies.append(fake\_data.company())

# Згенеруємо тепер number\_employees кількість співробітників'''

for \_ in range(number\_employees):

fake\_employees.append(fake\_data.name())

# Та number\_post набір посад

for \_ in range(number\_post):

fake\_posts.append(fake\_data.job())

return fake\_companies, fake\_employees, fake\_posts

Функція generate\_fake\_data повертає кортеж із даними. Під час її виконання результат приблизно повинен бути таким.

companies, employees, posts = generate\_fake\_data(NUMBER\_COMPANIES, NUMBER\_EMPLOYESS, NUMBER\_POST)

print(companies)

print(employees)

print(posts)

Не забуваємо, що дані випадкові і змінюватимуться при кожному виконанні функції.

['Mullins Group', 'Richardson Inc', 'Burton, Harrington and Long']

['Mary Hunter', 'Susan Scott', 'Jennifer Haley', 'Melanie Schneider', 'Julia Carlson', 'Regina Allen', 'Joshua Cruz', 'Benjamin Herrera', 'Lynn Smith', 'Dale Adkins', 'Melissa Andrews', 'Karen Anderson', 'Sarah Lara', 'Susan Jones', 'Ronald Hernandez', 'Jeffrey Calderon', 'Travis Morrow', 'Angela Reid', 'Shannon Walker', 'Clayton Bray', 'Wendy Harris', 'Albert English', 'Todd Rodriguez', 'Angela Martinez', 'Sarah Patel', 'Luke Martinez', 'Kimberly Walker', 'Andrew Hatfield', 'Jennifer Jackson', 'Anna Young']

['Amenity horticulturist', 'Journalist, magazine', 'Mining engineer', 'Hydrologist', 'Television/film/video producer']

Для виконання скрипту на множинне вставлення даних у таблицю нам знадобиться список кортежів, отже необхідно підготувати такі дані для кожної з таблиць. За це відповідатиме функція prepare\_data

**fill\_data.py**

def prepare\_data(companies, employees, posts) -> tuple():

for\_companies = []

# Готуємо список кортежів назв компаній

for company in companies:

for\_companies.append((company, ))

for\_employees = [] # для таблиці employees

for emp in employees:

'''

Для записів у таблицю співробітників нам потрібно додати посаду та id компанії. Компаній у нас було за замовчуванням

NUMBER\_COMPANIES, при створенні таблиці companies для поля id ми вказували INTEGER AUTOINCREMENT - тому кожен

запис отримуватиме послідовне число збільшене на 1, починаючи з 1. Тому компанію вибираємо випадково

у цьому діапазоні

'''

for\_employees.append((emp, choice(posts), randint(1, NUMBER\_COMPANIES)))

'''

Подібні операції виконаємо й у таблиці payments виплати зарплат. Приймемо, що виплата зарплати у всіх компаніях

виконувалася з 10 по 20 числа кожного місяця. Діапазон зарплат генеруватимемо від 1000 до 10000 у.о.

для кожного місяця, та кожного співробітника.

'''

for\_payments = []

for month in range(1, 12 + 1):

# Виконуємо цикл за місяцями'''

payment\_date = datetime(2021, month, randint(10, 20)).date()

for emp in range(1, NUMBER\_EMPLOYESS + 1):

# Виконуємо цикл за кількістю співробітників

for\_payments.append((emp, payment\_date, randint(1000, 10000)))

return for\_companies, for\_employees, for\_payments

Тепер дані набудуть наступного вигляду:

companies, employees, posts =

prepare\_data(\*generate\_fake\_data(NUMBER\_COMPANIES, NUMBER\_EMPLOYESS, NUMBER\_POST))

print(companies)

print(employees)

print(posts)

[('Small, Lee and Hoover',), ('Hernandez-Mercado',), ('Clark-Garcia',)]

[('Nicole Jones', 'Designer, ceramics/pottery', 2), ('Yolanda Mitchell', 'Theatre stage manager', 1), ('Robert Oliver', 'Theatre stage manager', 1), ('Mary Fernandez', 'Theatre stage manager', 3), ('Justin Richard', 'Occupational therapist', 3), ('Kevin Glover', 'Occupational therapist', 1), ('Jonathan Gonzalez Jr.', 'Occupational therapist', 1), ('Nathan Franklin', 'Personal assistant', 1), ('William Walsh', 'Theatre stage manager', 2), ('Scott Fowler', 'Designer, ceramics/pottery', 1), ('Cynthia Robinson', 'Administrator, local government', 3), ('Michael Cole', 'Personal assistant', 1), ('Linda Davis', 'Designer, ceramics/pottery', 1), ('Michael Gates', 'Administrator, local government', 3), ('Karen Williams DVM', 'Personal assistant', 3), ('Victor Watts', 'Administrator, local government', 2), ('Jacob Tucker', 'Theatre stage manager', 3), ('Miranda Lambert', 'Personal assistant', 1), ('Joyce Mason', 'Personal assistant', 2), ('Rachel Lopez', 'Personal assistant', 3), ('Jennifer Dennis', 'Theatre stage manager', 3), ('Patricia Guerrero', 'Administrator, local government', 2), ('Ashley Collins', 'Theatre stage manager', 3), ('Samantha Norris', 'Designer, ceramics/pottery', 2), ('Aaron Gomez', 'Personal assistant', 2), ('Dr. Matthew Schultz', 'Occupational therapist', 1), ('David Arellano', 'Personal assistant', 1), ('Matthew Austin', 'Theatre stage manager', 1), ('Carrie Pitts', 'Occupational therapist', 3), ('Melissa Mckenzie', 'Administrator, local government', 3)]

[(1, datetime.date(2021, 1, 18), 4847), (2, datetime.date(2021, 1, 18), 5553), (3, datetime.date(2021, 1, 18), 9511), (4, datetime.date(2021, 1, 18), 1293), (5, datetime.date(2021, 1, 18), 6010), (6, datetime.date(2021, 1, 18), 9246), (7, datetime.date(2021, 1, 18), 4590), (8, datetime.date(2021, 1, 18), 2096), (9, datetime.date(2021, 1, 18), 5446), (10, datetime.date(2021, 1, 18), 1113), (11, datetime.date(2021, 1, 18), 7590), (12, datetime.date(2021, 1, 18), 7173), (13, datetime.date(2021, 1, 18), 8944), (14, datetime.date(2021, 1, 18), 8818), (15, datetime.date(2021, 1, 18), 9963), (16, datetime.date(2021, 1, 18), 2338), (17, datetime.date(2021, 1, 18), 8662), (18, datetime.date(2021, 1, 18), 6352), (19, datetime.date(2021, 1, 18), 2474), (20, datetime.date(2021, 1, 18), 3430), (21, datetime.date(2021, 1, 18), 2396), (22, datetime.date(2021, 1, 18), 2716), (23, datetime.date(2021, 1, 18), 2564), (24, datetime.date(2021, 1, 18), 3670), (25, datetime.date(2021, 1, 18), 7756), (26, datetime.date(2021, 1, 18), 2189), (27, datetime.date(2021, 1, 18), 7212), (28, datetime.date(2021, 1, 18), 1674), (29, datetime.date(2021, 1, 18), 6326), (30, datetime.date(2021, 1, 18), 4246), (1, datetime.date(2021, 2, 15), 1328), (2, datetime.date(2021, 2, 15), 1452), (3, datetime.date(2021, 2, 15), 2414), (4, datetime.date(2021, 2, 15), 2907), (5, datetime.date(2021, 2, 15), 4431), (6, datetime.date(2021, 2, 15), 5998), (7, datetime.date(2021, 2, 15), 6472), (8, datetime.date(2021, 2, 15), 8282), (9, datetime.date(2021, 2, 15), 3166), (10, datetime.date(2021, 2, 15), 5262), (11, datetime.date(2021, 2, 15), 3604), (12, datetime.date(2021, 2, 15), 2384), (13, datetime.date(2021, 2, 15), 8905), (14, datetime.date(2021, 2, 15), 8571), (15, datetime.date(2021, 2, 15), 9269), (16, datetime.date(2021, 2, 15), 8778), (17, datetime.date(2021, 2, 15), 3838), (18, datetime.date(2021, 2, 15), 4166), (19, datetime.date(2021, 2, 15), 1858), (20, datetime.date(2021, 2, 15), 1843), (21, datetime.date(2021, 2, 15), 5721), (22, datetime.date(2021, 2, 15), 6127), (23, datetime.date(2021, 2, 15), 2648), (24, datetime.date(2021, 2, 15), 4277), (25, datetime.date(2021, 2, 15), 8196), (26, datetime.date(2021, 2, 15), 9433), (27, datetime.date(2021, 2, 15), 4742), (28, datetime.date(2021, 2, 15), 3069), (29, datetime.date(2021, 2, 15), 7536), (30, datetime.date(2021, 2, 15), 5901), (1, datetime.date(2021, 3, 11), 8701), (2, datetime.date(2021, 3, 11), 4498), (3, datetime.date(2021, 3, 11), 5334), (4, datetime.date(2021, 3, 11), 6523), (5, datetime.date(2021, 3, 11), 6288), (6, datetime.date(2021, 3, 11), 3788), (7, datetime.date(2021, 3, 11), 3377), (8, datetime.date(2021, 3, 11), 6119), (9, datetime.date(2021, 3, 11), 1509), (10, datetime.date(2021, 3, 11), 5911), (11, datetime.date(2021, 3, 11), 9588), (12, datetime.date(2021, 3, 11), 7998), (13, datetime.date(2021, 3, 11), 3283), (14, datetime.date(2021, 3, 11), 5399), (15, datetime.date(2021, 3, 11), 6814), (16, datetime.date(2021, 3, 11), 3818), (17, datetime.date(2021, 3, 11), 6385), (18, datetime.date(2021, 3, 11), 2259), (19, datetime.date(2021, 3, 11), 3918), (20, datetime.date(2021, 3, 11), 1427), (21, datetime.date(2021, 3, 11), 4702), (22, datetime.date(2021, 3, 11), 3157), (23, datetime.date(2021, 3, 11), 9119), (24, datetime.date(2021, 3, 11), 7652), (25, datetime.date(2021, 3, 11), 1571), (26, datetime.date(2021, 3, 11), 2616), (27, datetime.date(2021, 3, 11), 4997), (28, datetime.date(2021, 3, 11), 5817), (29, datetime.date(2021, 3, 11), 4027), (30, datetime.date(2021, 3, 11), 4949), (1, datetime.date(2021, 4, 12), 7570), (2, datetime.date(2021, 4, 12), 1645), (3, datetime.date(2021, 4, 12), 9217), (4, datetime.date(2021, 4, 12), 7131), (5, datetime.date(2021, 4, 12), 5602), (6, datetime.date(2021, 4, 12), 8320), (7, datetime.date(2021, 4, 12), 5536), (8, datetime.date(2021, 4, 12), 1987), (9, datetime.date(2021, 4, 12), 6338), (10, datetime.date(2021, 4, 12), 2285), (11, datetime.date(2021, 4, 12), 3765), (12, datetime.date(2021, 4, 12), 3488), (13, datetime.date(2021, 4, 12), 3437), (14, datetime.date(2021, 4, 12), 1995), (15, datetime.date(2021, 4, 12), 5405), (16, datetime.date(2021, 4, 12), 8761), (17, datetime.date(2021, 4, 12), 3549), (18, datetime.date(2021, 4, 12), 1485), (19, datetime.date(2021, 4, 12), 2398), (20, datetime.date(2021, 4, 12), 8883), (21, datetime.date(2021, 4, 12), 2142), (22, datetime.date(2021, 4, 12), 8252), (23, datetime.date(2021, 4, 12), 3472), (24, datetime.date(2021, 4, 12), 6060), (25, datetime.date(2021, 4, 12), 7245), (26, datetime.date(2021, 4, 12), 1420), (27, datetime.date(2021, 4, 12), 1415), (28, datetime.date(2021, 4, 12), 9867), (29, datetime.date(2021, 4, 12), 1820), (30, datetime.date(2021, 4, 12), 6299), (1, datetime.date(2021, 5, 18), 5677), (2, datetime.date(2021, 5, 18), 2975), (3, datetime.date(2021, 5, 18), 7255), (4, datetime.date(2021, 5, 18), 7951), (5, datetime.date(2021, 5, 18), 2428), (6, datetime.date(2021, 5, 18), 5849), (7, datetime.date(2021, 5, 18), 9140), (8, datetime.date(2021, 5, 18), 2132), (9, datetime.date(2021, 5, 18), 3388), (10, datetime.date(2021, 5, 18), 1685), (11, datetime.date(2021, 5, 18), 6233), (12, datetime.date(2021, 5, 18), 6822), (13, datetime.date(2021, 5, 18), 6109), (14, datetime.date(2021, 5, 18), 8513), (15, datetime.date(2021, 5, 18), 2104), (16, datetime.date(2021, 5, 18), 8388), (17, datetime.date(2021, 5, 18), 4337), (18, datetime.date(2021, 5, 18), 9399), (19, datetime.date(2021, 5, 18), 4882), (20, datetime.date(2021, 5, 18), 2952), (21, datetime.date(2021, 5, 18), 5566), (22, datetime.date(2021, 5, 18), 5850), (23, datetime.date(2021, 5, 18), 3768), (24, datetime.date(2021, 5, 18), 8597), (25, datetime.date(2021, 5, 18), 6742), (26, datetime.date(2021, 5, 18), 4081), (27, datetime.date(2021, 5, 18), 9009), (28, datetime.date(2021, 5, 18), 8104), (29, datetime.date(2021, 5, 18), 7010), (30, datetime.date(2021, 5, 18), 2756), (1, datetime.date(2021, 6, 15), 7022), (2, datetime.date(2021, 6, 15), 6699), (3, datetime.date(2021, 6, 15), 6598), (4, datetime.date(2021, 6, 15), 3087), (5, datetime.date(2021, 6, 15), 8623), (6, datetime.date(2021, 6, 15), 8534), (7, datetime.date(2021, 6, 15), 7940), (8, datetime.date(2021, 6, 15), 4377), (9, datetime.date(2021, 6, 15), 9974), (10, datetime.date(2021, 6, 15), 2970), (11, datetime.date(2021, 6, 15), 4049), (12, datetime.date(2021, 6, 15), 7975), (13, datetime.date(2021, 6, 15), 5320), (14, datetime.date(2021, 6, 15), 6537), (15, datetime.date(2021, 6, 15), 5165), (16, datetime.date(2021, 6, 15), 1547), (17, datetime.date(2021, 6, 15), 5729), (18, datetime.date(2021, 6, 15), 8115), (19, datetime.date(2021, 6, 15), 4044), (20, datetime.date(2021, 6, 15), 7772), (21, datetime.date(2021, 6, 15), 1362), (22, datetime.date(2021, 6, 15), 7118), (23, datetime.date(2021, 6, 15), 6335), (24, datetime.date(2021, 6, 15), 3464), (25, datetime.date(2021, 6, 15), 3493), (26, datetime.date(2021, 6, 15), 4787), (27, datetime.date(2021, 6, 15), 3607), (28, datetime.date(2021, 6, 15), 5241), (29, datetime.date(2021, 6, 15), 1409), (30, datetime.date(2021, 6, 15), 4045), (1, datetime.date(2021, 7, 12), 9920), (2, datetime.date(2021, 7, 12), 8880), (3, datetime.date(2021, 7, 12), 4894), (4, datetime.date(2021, 7, 12), 7714), (5, datetime.date(2021, 7, 12), 7427), (6, datetime.date(2021, 7, 12), 4949), (7, datetime.date(2021, 7, 12), 2291), (8, datetime.date(2021, 7, 12), 9301), (9, datetime.date(2021, 7, 12), 1882), (10, datetime.date(2021, 7, 12), 8078), (11, datetime.date(2021, 7, 12), 7943), (12, datetime.date(2021, 7, 12), 7660), (13, datetime.date(2021, 7, 12), 4200), (14, datetime.date(2021, 7, 12), 1862), (15, datetime.date(2021, 7, 12), 8715), (16, datetime.date(2021, 7, 12), 7289), (17, datetime.date(2021, 7, 12), 8543), (18, datetime.date(2021, 7, 12), 4203), (19, datetime.date(2021, 7, 12), 2241), (20, datetime.date(2021, 7, 12), 5422), (21, datetime.date(2021, 7, 12), 3596), (22, datetime.date(2021, 7, 12), 4958), (23, datetime.date(2021, 7, 12), 8880), (24, datetime.date(2021, 7, 12), 3178), (25, datetime.date(2021, 7, 12), 9245), (26, datetime.date(2021, 7, 12), 8433), (27, datetime.date(2021, 7, 12), 5607), (28, datetime.date(2021, 7, 12), 8990), (29, datetime.date(2021, 7, 12), 4530), (30, datetime.date(2021, 7, 12), 2960), (1, datetime.date(2021, 8, 13), 4140), (2, datetime.date(2021, 8, 13), 1178), (3, datetime.date(2021, 8, 13), 4474), (4, datetime.date(2021, 8, 13), 5737), (5, datetime.date(2021, 8, 13), 9189), (6, datetime.date(2021, 8, 13), 1771), (7, datetime.date(2021, 8, 13), 9659), (8, datetime.date(2021, 8, 13), 6407), (9, datetime.date(2021, 8, 13), 4404), (10, datetime.date(2021, 8, 13), 4993), (11, datetime.date(2021, 8, 13), 3056), (12, datetime.date(2021, 8, 13), 5586), (13, datetime.date(2021, 8, 13), 3027), (14, datetime.date(2021, 8, 13), 8063), (15, datetime.date(2021, 8, 13), 7699), (16, datetime.date(2021, 8, 13), 3203), (17, datetime.date(2021, 8, 13), 7982), (18, datetime.date(2021, 8, 13), 3623), (19, datetime.date(2021, 8, 13), 2067), (20, datetime.date(2021, 8, 13), 2626), (21, datetime.date(2021, 8, 13), 9751), (22, datetime.date(2021, 8, 13), 3174), (23, datetime.date(2021, 8, 13), 9290), (24, datetime.date(2021, 8, 13), 9344), (25, datetime.date(2021, 8, 13), 6074), (26, datetime.date(2021, 8, 13), 3429), (27, datetime.date(2021, 8, 13), 2329), (28, datetime.date(2021, 8, 13), 5297), (29, datetime.date(2021, 8, 13), 9907), (30, datetime.date(2021, 8, 13), 2148), (1, datetime.date(2021, 9, 20), 7861), (2, datetime.date(2021, 9, 20), 3997), (3, datetime.date(2021, 9, 20), 5483), (4, datetime.date(2021, 9, 20), 7368), (5, datetime.date(2021, 9, 20), 1742), (6, datetime.date(2021, 9, 20), 5137), (7, datetime.date(2021, 9, 20), 1595), (8, datetime.date(2021, 9, 20), 5992), (9, datetime.date(2021, 9, 20), 8394), (10, datetime.date(2021, 9, 20), 1396), (11, datetime.date(2021, 9, 20), 7890), (12, datetime.date(2021, 9, 20), 2380), (13, datetime.date(2021, 9, 20), 2938), (14, datetime.date(2021, 9, 20), 1983), (15, datetime.date(2021, 9, 20), 7980), (16, datetime.date(2021, 9, 20), 8930), (17, datetime.date(2021, 9, 20), 2473), (18, datetime.date(2021, 9, 20), 2794), (19, datetime.date(2021, 9, 20), 8587), (20, datetime.date(2021, 9, 20), 4162), (21, datetime.date(2021, 9, 20), 5347), (22, datetime.date(2021, 9, 20), 6838), (23, datetime.date(2021, 9, 20), 2450), (24, datetime.date(2021, 9, 20), 2513), (25, datetime.date(2021, 9, 20), 1454), (26, datetime.date(2021, 9, 20), 8169), (27, datetime.date(2021, 9, 20), 2507), (28, datetime.date(2021, 9, 20), 2062), (29, datetime.date(2021, 9, 20), 5666), (30, datetime.date(2021, 9, 20), 1864), (1, datetime.date(2021, 10, 18), 1222), (2, datetime.date(2021, 10, 18), 2049), (3, datetime.date(2021, 10, 18), 2379), (4, datetime.date(2021, 10, 18), 1966), (5, datetime.date(2021, 10, 18), 1266), (6, datetime.date(2021, 10, 18), 1235), (7, datetime.date(2021, 10, 18), 7626), (8, datetime.date(2021, 10, 18), 9416), (9, datetime.date(2021, 10, 18), 5914), (10, datetime.date(2021, 10, 18), 1711), (11, datetime.date(2021, 10, 18), 5882), (12, datetime.date(2021, 10, 18), 5299), (13, datetime.date(2021, 10, 18), 4694), (14, datetime.date(2021, 10, 18), 2910), (15, datetime.date(2021, 10, 18), 4655), (16, datetime.date(2021, 10, 18), 5180), (17, datetime.date(2021, 10, 18), 5483), (18, datetime.date(2021, 10, 18), 3157), (19, datetime.date(2021, 10, 18), 6307), (20, datetime.date(2021, 10, 18), 3379), (21, datetime.date(2021, 10, 18), 2554), (22, datetime.date(2021, 10, 18), 7205), (23, datetime.date(2021, 10, 18), 7405), (24, datetime.date(2021, 10, 18), 4271), (25, datetime.date(2021, 10, 18), 8462), (26, datetime.date(2021, 10, 18), 3468), (27, datetime.date(2021, 10, 18), 3870), (28, datetime.date(2021, 10, 18), 6753), (29, datetime.date(2021, 10, 18), 7498), (30, datetime.date(2021, 10, 18), 1094), (1, datetime.date(2021, 11, 11), 2049), (2, datetime.date(2021, 11, 11), 1754), (3, datetime.date(2021, 11, 11), 7254), (4, datetime.date(2021, 11, 11), 5488), (5, datetime.date(2021, 11, 11), 8489), (6, datetime.date(2021, 11, 11), 6038), (7, datetime.date(2021, 11, 11), 8062), (8, datetime.date(2021, 11, 11), 7224), (9, datetime.date(2021, 11, 11), 3066), (10, datetime.date(2021, 11, 11), 3809), (11, datetime.date(2021, 11, 11), 5287), (12, datetime.date(2021, 11, 11), 6497), (13, datetime.date(2021, 11, 11), 7906), (14, datetime.date(2021, 11, 11), 4365), (15, datetime.date(2021, 11, 11), 9700), (16, datetime.date(2021, 11, 11), 2312), (17, datetime.date(2021, 11, 11), 6022), (18, datetime.date(2021, 11, 11), 3361), (19, datetime.date(2021, 11, 11), 9895), (20, datetime.date(2021, 11, 11), 2889), (21, datetime.date(2021, 11, 11), 3511), (22, datetime.date(2021, 11, 11), 8639), (23, datetime.date(2021, 11, 11), 5784), (24, datetime.date(2021, 11, 11), 8019), (25, datetime.date(2021, 11, 11), 4917), (26, datetime.date(2021, 11, 11), 1187), (27, datetime.date(2021, 11, 11), 6522), (28, datetime.date(2021, 11, 11), 9382), (29, datetime.date(2021, 11, 11), 7802), (30, datetime.date(2021, 11, 11), 8606), (1, datetime.date(2021, 12, 20), 3104), (2, datetime.date(2021, 12, 20), 2027), (3, datetime.date(2021, 12, 20), 8289), (4, datetime.date(2021, 12, 20), 6695), (5, datetime.date(2021, 12, 20), 1254), (6, datetime.date(2021, 12, 20), 1950), (7, datetime.date(2021, 12, 20), 2053), (8, datetime.date(2021, 12, 20), 4030), (9, datetime.date(2021, 12, 20), 4864), (10, datetime.date(2021, 12, 20), 4234), (11, datetime.date(2021, 12, 20), 1650), (12, datetime.date(2021, 12, 20), 5951), (13, datetime.date(2021, 12, 20), 6566), (14, datetime.date(2021, 12, 20), 4413), (15, datetime.date(2021, 12, 20), 3879), (16, datetime.date(2021, 12, 20), 6328), (17, datetime.date(2021, 12, 20), 4568), (18, datetime.date(2021, 12, 20), 5581), (19, datetime.date(2021, 12, 20), 4098), (20, datetime.date(2021, 12, 20), 2770), (21, datetime.date(2021, 12, 20), 6993), (22, datetime.date(2021, 12, 20), 6008), (23, datetime.date(2021, 12, 20), 7955), (24, datetime.date(2021, 12, 20), 1788), (25, datetime.date(2021, 12, 20), 5279), (26, datetime.date(2021, 12, 20), 6357), (27, datetime.date(2021, 12, 20), 9456), (28, datetime.date(2021, 12, 20), 7160), (29, datetime.date(2021, 12, 20), 8705), (30, datetime.date(2021, 12, 20), 9899)]

Ці дані вже готові для вставлення в базу даних функцією prepare\_data. За саме вставлення даних відповідатиме остання третя функція insert\_data\_to\_db

Повний код скрипту fill\_data.py:

**fill\_data.py**

from datetime import datetime

import faker

from random import randint, choice

import sqlite3

NUMBER\_COMPANIES = 3

NUMBER\_EMPLOYESS = 30

NUMBER\_POST = 5

def generate\_fake\_data(number\_companies, number\_employees, number\_post) -> tuple():

fake\_companies = [] # тут зберігатимемо компанії

fake\_employees = [] # тут зберігатимемо співробітників

fake\_posts = [] # тут зберігатимемо посади

'''Візьмемо три компанії з faker і помістимо їх у потрібну змінну'''

fake\_data = faker.Faker()

# Створимо набір компаній у кількості number\_companies

for \_ in range(number\_companies):

fake\_companies.append(fake\_data.company())

# Згенеруємо тепер number\_employees кількість співробітників'''

for \_ in range(number\_employees):

fake\_employees.append(fake\_data.name())

# Та number\_post набір посад

for \_ in range(number\_post):

fake\_posts.append(fake\_data.job())

return fake\_companies, fake\_employees, fake\_posts

def prepare\_data(companies, employees, posts) -> tuple():

for\_companies = []

# готуємо список кортежів назв компаній

for company in companies:

for\_companies.append((company, ))

for\_employees = [] # для таблиці employees

for emp in employees:

'''

Для записів у таблицю співробітників нам потрібно додати посаду та id компанії. Компаній у нас було за замовчуванням

NUMBER\_COMPANIES, при створенні таблиці companies для поля id ми вказували INTEGER AUTOINCREMENT - тому кожен

запис отримуватиме послідовне число збільшене на 1, починаючи з 1. Тому компанію вибираємо випадково

у цьому діапазоні

'''

for\_employees.append((emp, choice(posts), randint(1, NUMBER\_COMPANIES)))

'''

Подібні операції виконаємо й у таблиці payments виплати зарплат. Приймемо, що виплата зарплати у всіх компаніях

виконувалася з 10 по 20 числа кожного місяця. Діапазон зарплат генеруватимемо від 1000 до 10000 у.о.

для кожного місяця, та кожного співробітника.

'''

for\_payments = []

for month in range(1, 12 + 1):

# Виконуємо цикл за місяцями'''

payment\_date = datetime(2021, month, randint(10, 20)).date()

for emp in range(1, NUMBER\_EMPLOYESS + 1):

# Виконуємо цикл за кількістю співробітників

for\_payments.append((emp, payment\_date, randint(1000, 10000)))

return for\_companies, for\_employees, for\_payments

def insert\_data\_to\_db(companies, employees, payments) -> None:

# Створимо з'єднання з нашою БД та отримаємо об'єкт курсору для маніпуляцій з даними

with sqlite3.connect('salary.db') as con:

cur = con.cursor()

'''Заповнюємо таблицю компаній. І створюємо скрипт для вставлення, де змінні, які вставлятимемо, відзначимо

знаком заповнювача (?) '''

sql\_to\_companies = """INSERT INTO companies(company\_name)

VALUES (?)"""

'''Для вставлення відразу всіх даних скористаємося методом executemany курсора. Першим параметром буде текст

скрипта, а другим - дані (список кортежів).'''

cur.executemany(sql\_to\_companies, companies)

# Далі вставляємо дані про співробітників. Напишемо для нього скрипт і вкажемо змінні

sql\_to\_employees = """INSERT INTO employees(employee, post, company\_id)

VALUES (?, ?, ?)"""

# Дані були підготовлені заздалегідь, тому просто передаємо їх у функцію

cur.executemany(sql\_to\_employees, employees)

# Останньою заповнюємо таблицю із зарплатами

sql\_to\_payments = """INSERT INTO payments(employee\_id, date\_of, total)

VALUES (?, ?, ?)"""

# Вставляємо дані про зарплати

cur.executemany(sql\_to\_payments, payments)

# Фіксуємо наші зміни в БД

con.commit()

if \_\_name\_\_ == "\_\_main\_\_":

companies, employees, posts = prepare\_data(\*generate\_fake\_data(NUMBER\_COMPANIES, NUMBER\_EMPLOYESS, NUMBER\_POST))

insert\_data\_to\_db(companies, employees, posts)

Після виконання скрипту наша БД буде заповнена фейковими даними, з якими ми можемо вже працювати та створювати запити.

**Виконуємо запити до БД**

Після заповнення БД напишемо кілька запитів до нашої бази даних.

**Знайдемо середню зарплату за посадами**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/select#%D0%B7%D0%BD%D0%B0%D0%B9%D0%B4%D0%B5%D0%BC%D0%BE-%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BD%D1%8E-%D0%B7%D0%B0%D1%80%D0%BF%D0%BB%D0%B0%D1%82%D1%83-%D0%B7%D0%B0-%D0%BF%D0%BE%D1%81%D0%B0%D0%B4%D0%B0%D0%BC%D0%B8)Ми робимо запит до таблиці payments, приєднуємо таблицю employees, щоб при цьому збігалися поля employee\_id з таблиці payments та id з таблиці employees.

Середнє значення знаходимо за полем total таблиці payments та округляємо його до двох знаків після коми. Головне - це виконати групування за полем post, щоб запит точно розумів процес обчислення середнього значення до кожної посади.

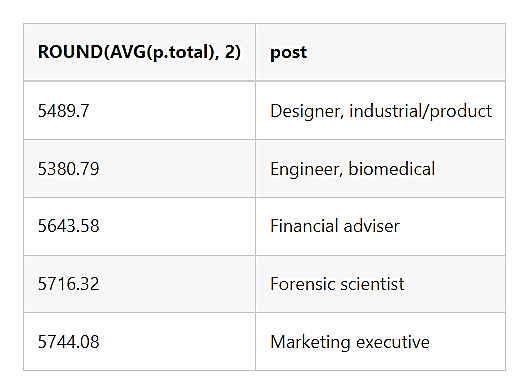
SELECT ROUND(AVG(p.total), 2), e.post

FROM payments as p

LEFT JOIN employees as e ON p.employee\_id = e.id

GROUP BY e.post;

Результат:



У вигляді python коду

**select\_first.py**

import sqlite3

def execute\_query(sql: str) -> list:

with sqlite3.connect('salary.db') as con:

cur = con.cursor()

cur.execute(sql)

return cur.fetchall()

sql = """

SELECT ROUND(AVG(p.total), 2), e.post

FROM payments as p

LEFT JOIN employees as e ON p.employee\_id = e.id

GROUP BY e.post;

"""

print(execute\_query(sql))

Результат буде список кортежів:

[(5489.7, 'Designer, industrial/product'), (5380.79, 'Engineer, biomedical'), (5643.58, 'Financial adviser'), (5716.32, 'Forensic scientist'), (5744.08, 'Marketing executive')]

Вибірка: кількість співробітників за компаніями[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/select#%D0%B2%D0%B8%D0%B1%D1%96%D1%80%D0%BA%D0%B0-%D0%BA%D1%96%D0%BB%D1%8C%D0%BA%D1%96%D1%81%D1%82%D1%8C-%D1%81%D0%BF%D1%96%D0%B2%D1%80%D0%BE%D0%B1%D1%96%D1%82%D0%BD%D0%B8%D0%BA%D1%96%D0%B2-%D0%B7%D0%B0-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D1%96%D1%8F%D0%BC%D0%B)

Тут використовуємо функцію COUNT для підрахунку кількості рядків, головне - це з'єднати таблиці companies та employees за ключами id та company\_id, а потім виконати групування за полем id або company\_name таблиці companies. У нашому випадку ми виконали групування за полем id.

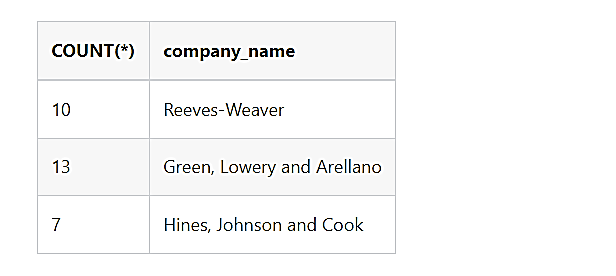
SELECT COUNT(\*), c.company\_name

FROM employees e

LEFT JOIN companies c ON e.company\_id = c.id

GROUP BY c.id;

Результат:



У вигляді python коду

**select\_second.py**

import sqlite3

def execute\_query(sql: str) -> list:

with sqlite3.connect('salary.db') as con:

cur = con.cursor()

cur.execute(sql)

return cur.fetchall()

sql = """

SELECT COUNT(\*), c.company\_name

FROM employees e

LEFT JOIN companies c ON e.company\_id = c.id

GROUP BY c.id;

"""

print(execute\_query(sql))

[(10, 'Reeves-Weaver'), (13, 'Green, Lowery and Arellano'), (7, 'Hines, Johnson and Cook')]

Вибрати співробітників компаній, у яких у 7 місяці була зарплата > 5000[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-02/select#%D0%B2%D0%B8%D0%B1%D1%80%D0%B0%D1%82%D0%B8-%D1%81%D0%BF%D1%96%D0%B2%D1%80%D0%BE%D0%B1%D1%96%D1%82%D0%BD%D0%B8%D0%BA%D1%96%D0%B2-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D1%96%D0%B9-%D1%83-%D1%8F%D0%BA%D0%B8%D1%85-%D1%83-7-%D0%BC%D1%96%D1%81%D1%8F%D1%86%D1%96-)

Тут запит дещо складніший, необхідно послідовно з'єднати таблиці між собою.

Знайти всі записи, де виплати були більшими 5000 та час виплат повинен бути між датами '2021-07-10' та '2021-07-20'.

SELECT c.company\_name, e.employee, e.post, p.total

FROM companies c

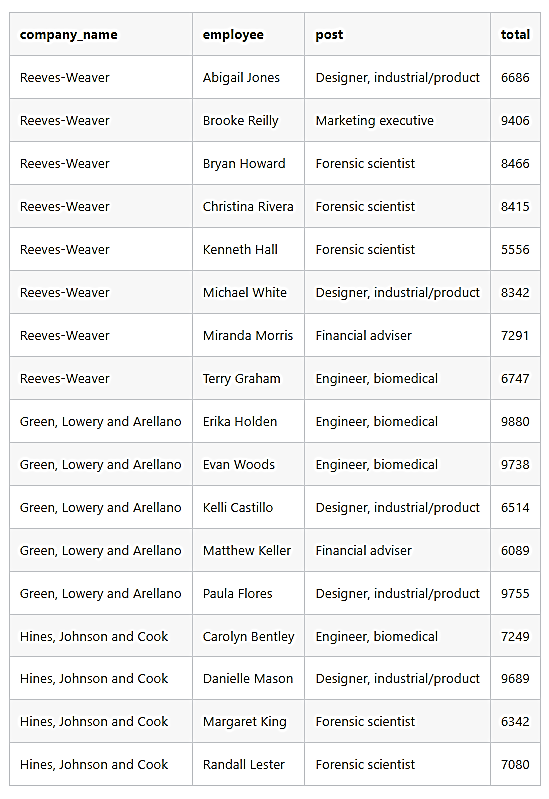
LEFT JOIN employees e ON e.company\_id = c.id

LEFT JOIN payments p ON p.employee\_id = e.id

WHERE p.total > 5000

AND p.date\_of BETWEEN '2021-07-10' AND '2021-07-20'

Результат



У вигляді python коду

**select\_third.py**

import sqlite3

def execute\_query(sql: str) -> list:

with sqlite3.connect('salary.db') as con:

cur = con.cursor()

cur.execute(sql)

return cur.fetchall()

sql = """

SELECT c.company\_name, e.employee, e.post, p.total

FROM companies c

LEFT JOIN employees e ON e.company\_id = c.id

LEFT JOIN payments p ON p.employee\_id = e.id

WHERE p.total > 5000

AND p.date\_of BETWEEN '2021-07-10' AND '2021-07-20'

"""

print(execute\_query(sql))

Результат буде список кортежів:

[('Reeves-Weaver', 'Abigail Jones', 'Designer, industrial/product', 6686), ('Reeves-Weaver', 'Brooke Reilly', 'Marketing executive', 9406), ('Reeves-Weaver', 'Bryan Howard', 'Forensic scientist', 8466), ('Reeves-Weaver', 'Christina Rivera', 'Forensic scientist', 8415), ('Reeves-Weaver', 'Kenneth Hall', 'Forensic scientist', 5556), ('Reeves-Weaver', 'Michael White', 'Designer, industrial/product', 8342), ('Reeves-Weaver', 'Miranda Morris', 'Financial adviser', 7291), ('Reeves-Weaver', 'Terry Graham', 'Engineer, biomedical', 6747), ('Green, Lowery and Arellano', 'Erika Holden', 'Engineer, biomedical', 9880), ('Green, Lowery and Arellano', 'Evan Woods', 'Engineer, biomedical', 9738), ('Green, Lowery and Arellano', 'Kelli Castillo', 'Designer, industrial/product', 6514), ('Green, Lowery and Arellano', 'Matthew Keller', 'Financial adviser', 6089), ('Green, Lowery and Arellano', 'Paula Flores', 'Designer, industrial/product', 9755), ('Hines, Johnson and Cook', 'Carolyn Bentley', 'Engineer, biomedical', 7249), ('Hines, Johnson and Cook', 'Danielle Mason', 'Designer, industrial/product', 9689), ('Hines, Johnson and Cook', 'Margaret King', 'Forensic scientist', 6342), ('Hines, Johnson and Cook', 'Randall Lester', 'Forensic scientist', 7080)]

Була створена БД із трьома пов'язаними таблицями: companies з назвами компаній, employees зі співробітниками компаній та, нарешті, payments з інформацією про заробітну плату співробітників компанії. Ми наповнили БД фейковими даними, а також виконали приклади запитів до отриманої БД.

Розберіть код у живому прикладі, який ми тільки що розібрали.

**Розгортання контейтера PostgreSQL за допомогою Docker в певному каталозі**

Щоб розгорнути контейнер PostgreSQL за допомогою Docker в певному каталозі, ви можете використати Docker Compose, який дозволить вам налаштувати та запускати контейнери з описом у файлі конфігурації. Ось кроки, які вам потрібно буде виконати:

1. **Створіть файл docker-compose.yml**: Створіть новий файл з іменем **docker-compose.yml** у каталозі, де ви хочете розгорнути контейнер PostgreSQL. В цьому файлі ви зможете визначити всі налаштування для вашого контейнера.
2. **Визначте сервіс PostgreSQL у файлі docker-compose.yml**: У файлі **docker-compose.yml** визначте сервіс для PostgreSQL, вказавши образ, порт, мережу та інші налаштування. Ось приклад простого файлу **docker-compose.yml**:

version: '3.7'

services:

postgres:

image: postgres

restart: always

ports:

- "5432:5432"

environment:

POSTGRES\_PASSWORD: your\_password

volumes:

- ./data:/var/lib/postgresql/data

В цьому прикладі ми використовуємо офіційний образ PostgreSQL, вказуємо порт 5432 для зовнішнього доступу, задаємо пароль для користувача **postgres** та монтуємо каталог **./data** локально на контейнер **/var/lib/postgresql/data** для зберігання даних.

1. **Запустіть контейнер за допомогою Docker Compose**: Відкрийте термінал або командний рядок у вашому каталозі з файлом **docker-compose.yml** і виконайте команду:

docker-compose up –d

Ця команда запустить Docker Compose та розгорне ваш контейнер PostgreSQL у фоновому режимі (**-d**).

Тепер контейнер PostgreSQL буде розгорнутий у вашому каталозі, як ви вказали у файлі **docker-compose.yml**.