МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

спеціальність 121 – Програмна інженерія

на тему: **Система кліматичних показників приміщення**
(назва теми)

Студентка групи КП-02 **Гудзіцька Дарина Сергіївна**
(ПІБ)
_____
(підпис)

Викладач к.т.н **Радченко К. О.**
_____
(підпис)

Захищено з оцінкою _____

Київ – 2021

АНОТАЦІЯ

Дана курсова робота представляє програму, що спроможна керувати базую даних, що використовується для моніторингу кліматичних показників. Якщо розглядати з технічної точки зору, вона складається з двох модулів, що виконують функції генерування даних та безпосередньо контролювання цих даних. Програма написана на мові програмування Python 3.9 у середовищі PyCharm 2020.3, а у якості бази даних – реляційна СУБД PostgreSQL.

ЗМІСТ

ВСТУП

Дана робота призначена для збору інформації про кліматичні показники в приміщенні задля інформування про їх зміни та відстеження як саме ці зміни відбуваються. Метою роботи є створення додатку, що буде забезпечувати збір, аналіз, маніпуляцію, валідацію та фільтрацію великих об’ємів інформації та навчитися коректно зберігати та користуватися цією інформацією.

Робота беззаперечно є актуальною, адже з сучасним кліматичним станом створення моніторингу кліматичних показників та їх подальший аналіз є важливою задачею. Моніториг таких показників у приміщеннях забезпечить комфортне перебування у них. З сучасним епідеміологічним станом важливо підтримувати деякі показники у стабільному стані, наприклад для уражених легень потрібно підтримувати певну вологість повітря. Кліматичні показники у приміщеннях важливі не тільки для людей, що в них перебувають, я й для предметів. Морозильні камери, теплиці, архіви, сховища, склади медичних препаратів, приміщення з серверами – це перелік малої частини приміщень, де необхідний постійний контроль таких показників як вологість, температура, атмосферний тиск.

Щоб забезпечити коректне управління даними, що має у собі база даних було використано великий інструментарій для забезпечення правильної роботи засобів резервації та відновлення бази даних в разі втрати даних, засобів підвищення швидкодії аби програма не перевантажувала систему, працюючи з великим об’ємом даних.

1. Аналіз інструментарію для використання курсової роботи

Для даної курсової роботи була обрана реляційна СУБД PostgreSQL. На доцільність даного даної СУБД вказує велика кількість нормалізованих даних та відношень, невеликий пріоритет швидкості виконання запитів та потреба в аналізі даних, що потребує зручну та наочну маніпуляцією даними.
Мова керування СУБД – Python 3.9. У ній наявна низка зручних бібліотек, що підходять для досягнення мети роботи, а саме psycopg2 (для роботи безпосередньо з базою даних); keyboard, datetime(для спрощення виконання задач); matplotlib.pyplot, numpy, pandas (для візуалізації результатів)

2. Структура бази даних

База даних, що була створена для оперування даними тематики курсової, складається з 5 (п’яти) таблиць:

1. Devices. Таблиця для зберігання інформації щодо пристроїв, які вимірюють показники. Має такі атрибути: device_id, device_name, measureg_object.
2. Premises. Таблиця для зберігання інформації щодо приміщень, в яких вимірюються показники. Має такі атрибути: premise_id, premises_name, rooms_number, device, date. Таблиця має зв’язок один-до-багатьох (one-to-many) між колонкою device та таблицею devices (device_id).
3. Indicators_temperature. Таблиця для зберігання даних показників температури у приміщенні. Має такі атрибути: indicatorT_id, temperature, premises, indicators_states. Таблиця має зв’язок один-до-багатьох (one-to-many) між колонкою premises та таблицею premises (premise _id).
4. Indicators_ humidity. Таблиця для зберігання даних показників температури у приміщенні. Має такі атрибути: indicatorh_id, humidity, premises, indicators_states. Таблиця має зв’язок один-до-багатьох (one-to-many) між колонкою premises та таблицею premises (premise _id).
5. Indicators_ pressure. Таблиця для зберігання даних показників температури у приміщенні. Має такі атрибути: indicatorp_id, pressure, premises, indicators_states. Таблиця має зв’язок один-до-багатьох (one-to-many) між колонкою premises та таблицею premises (premise _id).

ER-модель даної бази даних наведена у додатках.

3. Опис програмного забезпечення

3.1 Загальна структура програмного забезпечення

Програмне забезпечення складається з 2 (двох) програм: генерація даних та контроль бази даних. Це зроблено для того, аби не ускладнювати систему контролю бази даних (аналіз, резервація, перевірка швидкодії).

3.2 Опис модулів програмного забезпечення

1. Генерація даних:
   Програма складається з таких модулів: main.py, config.py, control.py, sql_repo.py.
   Main.py – головний модуль програми, використовується для обробки команд введених з консолі користувачем.
   Config.py – модуль, у якому містить інформація для підключення до бази даних.
   Control.py – модуль, що з’єднує користувача та функції, що виконують SQL запити.
   Sql_repo.py – модуль, що звертається безпосередньо до бази даних та виконує SQL запити.
2. Контроль бази даних
   Програма складається з таких модулів: main.py, config.py, control.py, sql_repo.py, create_hist.py.
   Main.py – головний модуль програми, використовується для обробки команд введених з консолі користувачем.
   Config.py – модуль, у якому містить інформація для підключення до бази даних.
   Control.py – модуль, що з’єднує користувача та функції, що виконують SQL запити.
   Sql_repo.py – модуль, що звертається безпосередньо до бази даних та виконує SQL запити.
   Create_hist.py – модуль, що генерує графіки та діаграми для візуалізації результатів.

4. Аналіз функціонування засобів реплікації

У процесі виконання курсової роботи мала бути використана можливість реплікації даних. Була обрана логічна реплікація в межах однієї локальної мережі. На жаль, з технічних причин, а саме неможливість змінити параметру wal_level з replica на logical, цей функціонал не був реалізований.
Детальний алгоритм виконання реплікації:

1. «ALTER SYSTEM SET wal_level = logical;»
2. Перезавантаження серверу. Можливі варінти виконання: перезавантажити pgAdmin4 та/або перезавантаження комп’ютера; у SQLShell після підключення до бази даних прописати «pg_ctl reload -D C:\Program Files\PostgreSQL\11\data»
3. «SHOW wal_level»
4. «SELECT * FROM pg_create_logical_replication_slot('slotName', 'pgoutput');»
5. Створення публікації в цій базі даних
6. Створення БД для реплікації, створення відповідних таблицб з відповідними колонками (копії основних таблиць)
7. У новій БД створюємо підписку на публікацію, як слот вказуємо 'slotName'.

5. Аналіз функціонування засобів резервування/відновлення бази даних

У даній курсовій роботі, в якості механізму реалізації резервування/відновлення бази даних у Postgresql, було використано сторонній додаток “Effector Saver”. Даний метод доволі легкий у використанні та налаштуваннях. За допомогою додатку можна в один клік резервувати базу даних, або ж увімкнути автоматичне резервування. Backup бази даних створюється на GoogleDrive, тому його можна без проблем завантажити на комп’ютер та відновити базу даних.
Скріншоти результатів виконання наведені у розділі «Додатки».

6. Аналіз результатів підвищення швидкодії використання запитів

У даній курсовій роботі було використано два способи підвищення швидкодії використання запитів: індекси та запит “IF EXIST()”.
Для індексації було використано hash-індекси. Вони підходять для будь-яких типів даних та практично не викликають колізій. Найкращий випадок використання hash-індексів для текстових стовбців – пошук за допомогою оператора «=».
У другому способі було порівняно швидкість виконування запитів SELECT COUNT(1) та EXIST(). Останній показав кращий результат, адже при виконуванні першого, здійснюється прохід всієї бази, а EXIST() зупиняється після знаходження 1 співпадіння.
Графіки порівняння результатів для обох способів наведені у розділі «Додатки».

7. Опис результатів аналізу предметної галузі

У аналізі предметної галузі зображено статистику використання пристроїв для вимірювання показників. Результати наведено у вигляді діаграми та наведено у додатках.

ВИСНОВКИ

Виконуючи дану курсову роботу, було створено базу даних та програмне забезпечення для її використання да контролю. При створенні бази було проаналізовано, які показники мають значення для системи кліматичних показників. Для симуляцію великої кількості таких даних було створено відповідні SQL-запити, які генерують рандомізовані дані. Також було додано можливість вручну додавати коректні дані.

Задля зручності керування СУБД було використано сторонній додаток, він спрощує резервацію даних, та забезпечує безпечне та автоматичне створення Backup. На мій погляд це доволі зручно, адже якщо трапиться непередбачувана ситуація через яку втратиться доступ до даних, можна завантажити останній зроблений Backup та не хвилюватися наскільки актуальними буду дані.

Щоб система не перевантажувалась, при аналізі великої кількості рядків було використано підвищення швидкодії у вигляді hash-індексів та запитів “IF EXIST()”. За результатами можна сказати, що це скоротило час виконання запитів майже у два рази, що доволі непоганий результат.

Отже, по завершенню виконання роботи, можна сказати, що початкова мета проєкту, аналіз кліматичних показників у приміщенні, була досягнута з використанням СУБД PostgreSQL та мови програмування Python 3.9 у середовищі PyCharm 2020.3, що дозволяє роботі використовуватися за призначенням у своїй предметній галузі.

ДОДАТКИ

Додатки А.

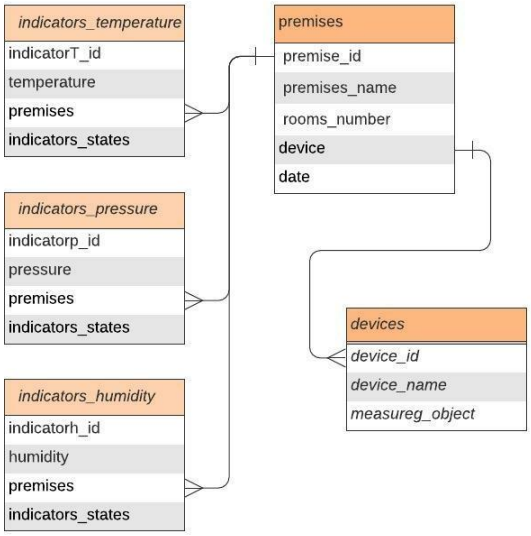

Рис 1. ER-діаграма бази даних

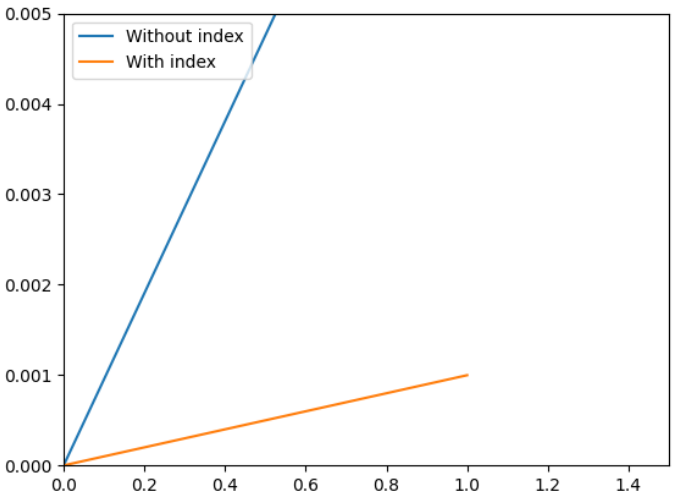

Рис 2. Графік порівння часу виконання запиту з індексами

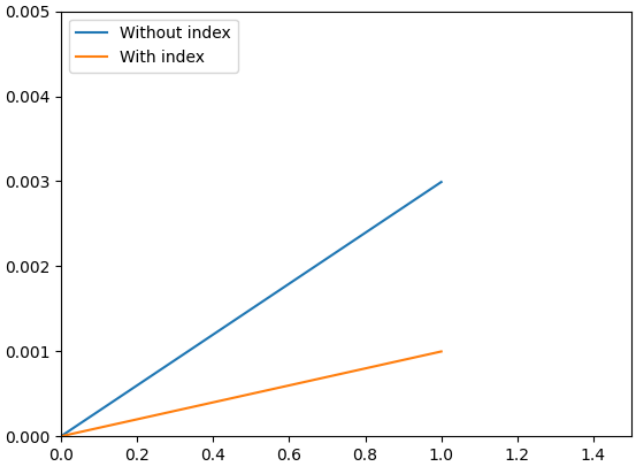

Рис 3. Графік порівння часу виконання запиту з індексами

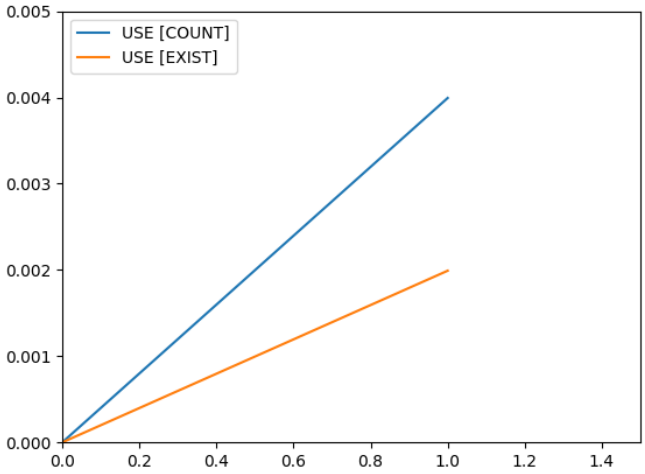

Рис 5. Графік порівняння часу виконная запиту COUNT()

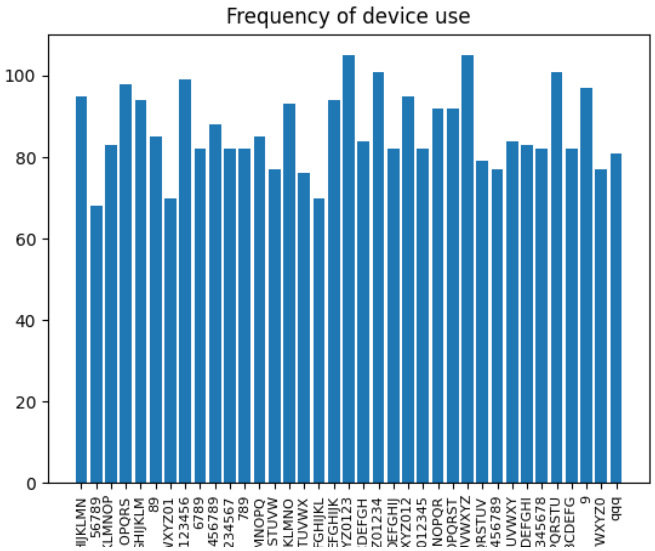

Рис 6. Частота використання пристроїв

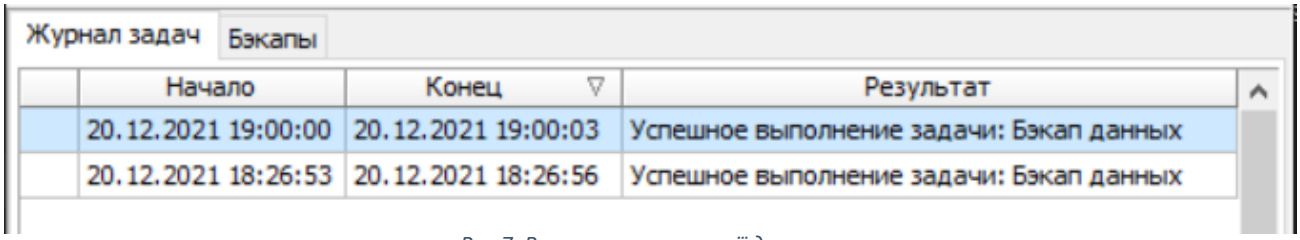

Рис 7. Виконання резервації даних

Додатки Б.

Приклад генерування даних SQL-запитом

```
def sql_generate_device(num, connection):
    n = 0
    with connection.cursor() as cursor:
        while n <= num:
            cursor.execute(
                """INSERT INTO devices (device_name, measureg_object) values(
                    substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',((random()*(36-
1)+1)::integer),7),
                    substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ',((random()*(26-1)+1)::integer),4)
                    )"""
            )
            n += 1


def sql_generate_indicator_temp(num, connection):
    n = 0
    min = __getfirstid(connection, "premise")
    max = __getcount(connection, "premises")
    with connection.cursor() as cursor:
        while n <= num:
            cursor.execute(
                """INSERT INTO indicators_temperature (temperature, premises,
indicators_states)
                    VALUES(
                    %s, %s,
                    substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ',((random()*(26-1)+1)::integer),4)
                    )""" %
                (random.randint(-10, 30), random.randint(min, max + min - 1))
            )
            n += 1
```

Приклад генерування гістограми

```
def unic_name(list_name):
    a = []
    for i in list_name:
        if i.lower() not in [j.lower() for j in a]:
            a.append(i)
    return a


def generate_device_hist(connection):
    list_name = sql_repo.__getdevicesname(connection)
    tick_list = unic_name(list_name)

    num_elements = len(tick_list)
    index = np.arange(num_elements)
```

```
    values1 = []
    for i in np.arange(len(tick_list)):
        values1.append(list_name.count(list_name[i]))
        if list_name.count(list_name[i]) == 2:
            list_name.remove(list_name[i])
            i -= 1

    tick_values = range(0, len(tick_list))

    plt.title('Frequency of device use')
    plt.bar(index, values1)
    plt.xticks(ticks=tick_values, labels=tick_list, rotation=90, fontsize=8)
    plt.show()
```

Приклад створення індекса

```
def create_index(connection):
    with connection.cursor() as cursor:
        cursor.execute(
            """CREATE INDEX index_device ON devices USING btree (device_name)"""
        )
```

Приклад дослідження швидкодії індексації

```
noind = control.check_speed_device(command[1], connection)
sql_repo.create_index(connection)
ind = control.check_speed_device(command[1], connection)
sql_repo.drop_index(connection, 'index_device')
val = [noind, ind]
create_hist.generate_speed_index(val)


def data_sampling_device(connection, value):
    with connection.cursor() as cursor:
        cursor.execute(
            """SELECT (device_name) FROM devices
            WHERE device_name = '%s'
            ORDER BY device_name ASC""" % str(value)
        )
        c = cursor.fetchall()
    return c
```