МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота №2
з дисципліни “Бази даних”
тема ««Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»»

Виконав
студент ІІІ курсу
групи КП-81
Янковський Дмитро
Олексійович

Посилання на репозиторій:

https://github.com/Dizzzmas/DB_course_labs/tree/lab2

Мета роботи

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Постановка завдання

Загальне завдання роботи полягає у наступному:
1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Приклади коду

db_labs/cli/__init__.py (View)

```
from pprint import pprint
import click
from db_labs.domain.cli import (
   handle_creating_developer,
   handle_updating_developer,
   handle_searching_for_developers,
   handle_getting_developers,
)

APP_DEV_URL = "http://localhost:5000/api"


@click.command()
@click.option(
   "--option",
   prompt="Your name",
   help="Options:
create_developer\nupdate_develiper\nsearch_developers",
)
def main(option):
   if option == "create_developer":
       email = click.prompt("Please enter an email",
type=str)
       first_name = click.prompt("Please enter a first
```

```
name", type=str)

        response = handle_creating_developer(email,
first_name)

        print("New developer created.")
        return pprint(response.json())

    if option == "update_developer":
        developer_id = click.prompt("Please enter a
developer id", type=int)
        email = click.prompt("Please enter an email",
type=str)
        first_name = click.prompt("Please enter a first
name", type=str)

        response = handle_updating_developer(developer_id,
email, first_name)

        print(f"Developer with id: {developer_id} was
updated.")
        return pprint(response.json())

    if option == "search_developers":
        query_string = click.prompt(
            "Please enter a search keyword(first/last name
or skill name)", type=str
        )

        response =
handle_searching_for_developers(query_string)

        print(
            f"{len(response.json())} developers found for
the keyword: {query_string}"
```

```
        )
        return pprint(response.json())

    if option == "get_developers":
        response = handle_getting_developers()

        print(f"{len(response.json())} developers
fetched")
        return pprint(response.json())


if __name__ == "__main__":
    main()
```

db_labs/domain/cli/__init__.py (Controller)

```
from os import abort

import requests

APP_DEV_URL = "http://localhost:5000/api"


def handle_creating_developer(email: str, first_name:
str):
    try:
        import requests

        response = requests.post(
            f"{APP_DEV_URL}/developer",
json=dict(email=email, first_name=first_name)
```

```
        )
    except Exception:
        print("An error occurred while trying to reach the
API.")
        return abort()

    if response.status_code != 200:
        print("An error occurred during the API request.")
        return abort()

    return response


def handle_updating_developer(id: int, email: str,
first_name: str):
    try:
        response = requests.patch(
            f"{APP_DEV_URL}/developer/{id}",
            json=dict(email=email, first_name=first_name),
        )
    except Exception:
        print("An error occurred while trying to reach the
API.")
        return abort()

    if response.status_code != 200:
        print("An error occurred during the API request.")
        return abort()

    return response


def handle_searching_for_developers(query_string: str):
    try:
        response =
```

```
requests.get(f"{APP_DEV_URL}/developer?query={query_strin
g}")
    except Exception:
        print("An error occurred while trying to reach the
API.")
        return abort()

    if response.status_code != 200:
        print("An error occurred during the API request.")
        return abort()

    if not response.json():
        print(f"No results found for the keyword:
{query_string}")
        return abort()

    return response


def handle_getting_developers():
    try:
        response =
requests.get(f"{APP_DEV_URL}/developer")
    except Exception:
        print("An error occurred while trying to reach the
API.")
        return abort()

    if response.status_code != 200:
        print("An error occurred during the API request.")
        return abort()

    if not response.json():
        print(f"No results found")
        return abort()
```

```
    return response
```

db_labs/model/developer.py (Model)

```
from jetkit.db.model import TSTZ
from sqlalchemy import Integer, ForeignKey, Text, Index
from db_labs.db import db
from db_labs.model.trgm_extension import TrgmExtension


class Developer(db.Model, TrgmExtension):
    first_name = db.Column(Text)
    last_name = db.Column(Text)
    email = db.Column(Text)
    birthdate = db.Column(TSTZ)

    vacancy_id = db.Column(Integer,
ForeignKey("vacancy.id", ondelete="SET NULL"))
    vacancy = db.relationship("Vacancy",
back_populates="developers")

    skills = db.relationship("Skill",
secondary="developer_skill")

    developer_first_name_trgm_idx =
Index('developer_first_name_trgm_idx',
          first_name, postgresql_using='gin',
          postgresql_ops={
              'first_name': 'gin_trgm_ops',
          })
```

```
   developer_last_name_trgm_idx =
Index('developer_last_name_trgm_idx',
         last_name, postgresql_using='gin',
         postgresql_ops={
             'last_name': 'gin_trgm_ops',
         })


Developer.add_create_trgm_extension_trigger()
```

db_labs/domain/developer/__init__.py (Controller)

```
from typing import Dict, Union
from flask_smorest import abort
from sqlalchemy.orm import joinedload

from db_labs.db import db
from db_labs.model import Developer, DeveloperSkill,
Skill


def
handle_getting_and_searching_for_developers(query_string:
str):
   """Get all developers(limit is 50 per query) or search
for specific developers by first_name,last_name or
skill_name."""
   if query_string:
       query_string = f"%{query_string}%"  # Enclosed in
'%' as per ILIKE syntax

       # Query for search
```

```
        # UNION needed here to speed up ILIKE across 2
tables. SELECT * also fetches vacancy and skills that
were JOINed. We don't process and output them however.
        query = """SELECT *
FROM developer LEFT OUTER JOIN developer_skill ON
developer.id = developer_skill.developer_id LEFT OUTER
JOIN skill ON skill.id = developer_skill.skill_id LEFT
OUTER JOIN vacancy AS vacancy_1 ON vacancy_1.id =
developer.vacancy_id LEFT OUTER JOIN (developer_skill AS
developer_skill_1 JOIN skill AS skill_1 ON skill_1.id =
developer_skill_1.skill_id) ON developer.id =
developer_skill_1.developer_id
WHERE CAST(developer.first_name AS VARCHAR) ILIKE
:query_string ESCAPE '~' OR CAST(developer.last_name AS
VARCHAR) ILIKE :query_string ESCAPE '~' UNION SELECT  *
FROM developer LEFT OUTER JOIN developer_skill ON
developer.id = developer_skill.developer_id LEFT OUTER
JOIN skill ON skill.id = developer_skill.skill_id LEFT
OUTER JOIN vacancy AS vacancy_1 ON vacancy_1.id =
developer.vacancy_id LEFT OUTER JOIN (developer_skill AS
developer_skill_1 JOIN skill AS skill_1 ON skill_1.id =
developer_skill_1.skill_id) ON developer.id =
developer_skill_1.developer_id
WHERE CAST(skill.name AS VARCHAR) ILIKE :query_string
ESCAPE '~' LIMIT 50;"""

        developers = db.session.execute(query,
dict(query_string=query_string))
    else:
        query = """SELECT * FROM developer LIMIT 50;"""
        developers = db.session.execute(query)

    return developers
```

```
def handle_creating_developer(args: Dict[str, str]):
    # developer = Developer(**args) For ORM
    # db.session.add(developer)
    # db.session.commit()

    create_developer_query = """INSERT INTO developer
(first_name, email) VALUES (:first_name, :email)
RETURNING developer.id, developer.email,
developer.first_name"""

    result = db.session.execute(create_developer_query,
args)

    db.session.commit()

    developer = {}
    for entry in result:
        developer = entry

    return developer


def handle_updating_developer(args: Dict[str, Union[str,
int]], developer_id: int):
    # developer = Developer.query.get(developer_id)

    # if not developer:
    #     abort(404, message=f"No developer with id:
${developer_id} found.")

    # remove None values so they do not override existing
data
    values = {key: value for key, value in args.items() if
value is not None}
    # Developer.query.update(values) for ORM
```

```
    update_developer_query = """UPDATE developer SET
updated_at=NOW(), first_name=:first_name, email=:email
WHERE id=:id RETURNING developer.id, developer.email,
developer.first_name"""
    values["id"] = developer_id
    result = db.session.execute(update_developer_query,
values)

    db.session.commit()

    developer = {}
    for entry in result:
        developer = entry

    if not developer:
        abort(404, message=f"No developer with id:
${developer_id} found.")

    return developer
```

db_labs/api/developer/__init__.py (View)

```
from typing import Dict, Union

from flask import request
from jetkit.api import combined_search_by
from flask_smorest import Blueprint, abort
from sqlalchemy.orm import joinedload

from db_labs.api.developer.decorators import
searchable_by_skills
```

```
from db_labs.api.developer.schema import DeveloperSchema
from db_labs.db import db
from db_labs.domain.developer import (
    handle_getting_and_searching_for_developers,
    handle_creating_developer, handle_updating_developer,
)
from db_labs.model import Developer, DeveloperSkill,
Skill

blp = Blueprint("Developer", __name__,
url_prefix=f"/api/developer")


@blp.route("", methods=["GET"])
@blp.response(DeveloperSchema(many=True))
# @combined_search_by(  # For use with ORM
#     Developer.first_name,
#     Developer.last_name,
#     Skill.name,
#     search_parameter_name="query",
# )
def get_developers():
    """Get all developers(limit is 50 per query) or search
for specific developers by first_name,last_name or
skill_name."""
    query_string = request.args.get("query")

    developers =
handle_getting_and_searching_for_developers(query_string)

    return developers


@blp.route("", methods=["POST"])
@blp.response(DeveloperSchema)
```

```
@blp.arguments(DeveloperSchema)
def create_developer(args: Dict[str, str]):
    """Create a developer entry."""
    developer = handle_creating_developer(args)

    return developer


@blp.route("/<string:developer_id>", methods=["PATCH"])
@blp.response(DeveloperSchema)
@blp.arguments(DeveloperSchema)
def update_developer(args: Dict[str, Union[str, int]],
developer_id: int):
    """Check if developer with given id exists, then
update the entry."""
    developer = handle_updating_developer(args,
developer_id)

    return developer
```

Приклади роботи програми

```
~/Projects/University/DB_course_labs on master
❯ python3 db_labs/cli/__init__.py --option get_developers
50 developers fetched
[{'birthdate': '2020-10-18T00:00:00-07:00',
  'email': 'zhanson@hotmail.com',
  'first_name': 'Paula',
  'id': 1,
  'last_name': 'Bray'},
 {'birthdate': '2020-10-11T00:00:00-07:00',
  'email': 'woodsdenise@yahoo.com',
  'first_name': 'Gabriel',
  'id': 2,
```

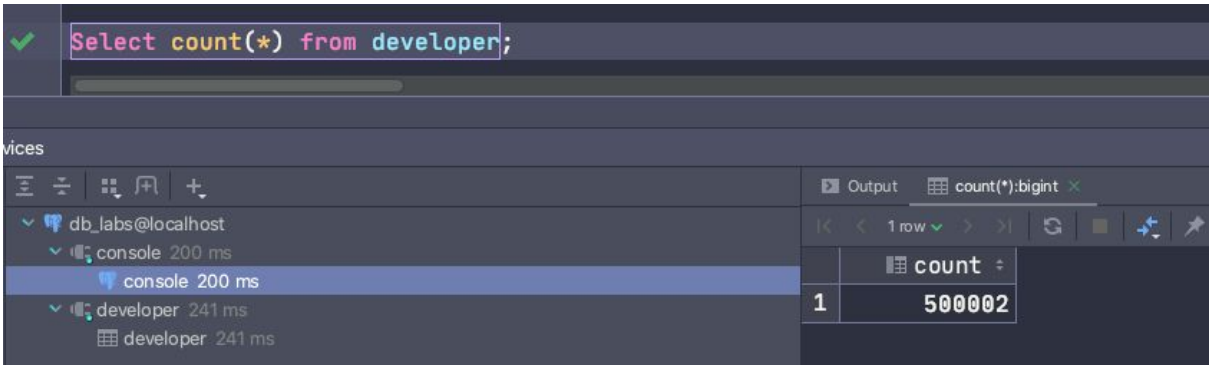
Вивід правильних даних

```
~/Projects/University/DB_course_labs on master
❯ python3 db_labs/cli/__init__.py --option get_developers
An error occurred while trying to reach the API.
[1]    20026 abort      python3 db_labs/cli/__init__.py --option get_developers
```

Обробка помилок

	id	created_at	updated_at	first_name	last_name	email	birthdate	vacancy_id
1	1	2020-10-25 15:59:35.470290	<null>	Paula	Bray	zhanson@hotmail.com	2020-10-18 07:00:00.000000	1
2	2	2020-10-25 15:59:35.470290	<null>	Gabriel	Collins	woodsdenise@yahoo.com	2020-10-11 07:00:00.000000	1
3	3	2020-10-25 15:59:35.470290	<null>	Holly	Thompson	johnsonpeggy@gmail.com	2020-10-22 07:00:00.000000	1
4	4	2020-10-25 15:59:35.470290	<null>	Glen	Herrera	wendy91@palmer.com	2020-09-26 07:00:00.000000	1
5	5	2020-10-25 15:59:35.470290	<null>	Carla	Gutierrez	ericdean@white.info	2020-10-05 07:00:00.000000	1
6	6	2020-10-25 15:59:35.470290	<null>	Scott	Palmer	juliaharrison@young.com	2020-09-28 07:00:00.000000	2
7	7	2020-10-25 15:59:35.470290	<null>	Theresa	Brown	heatherspence@hotmail.com	2020-10-12 07:00:00.000000	2
8	8	2020-10-25 15:59:35.470290	<null>	Aaron	Miller	judithvargas@hood.com	2020-10-10 07:00:00.000000	2
9	9	2020-10-25 15:59:35.470290	<null>	Brenda	Johnson	nbell@jones.com	2020-10-04 07:00:00.000000	2
10	10	2020-10-25 15:59:35.470290	<null>	Lorraine	Pruitt	mullentonya@leach.com	2020-10-04 07:00:00.000000	2
11	11	2020-10-25 15:59:35.470290	<null>	Samantha	Miller	awilcox@yahoo.com	2020-10-14 07:00:00.000000	3
12	12	2020-10-25 15:59:35.470290	<null>	Tanner	Mullen	duffysamuel@harvey.com	2020-09-30 07:00:00.000000	3
13	13	2020-10-25 15:59:35.470290	<null>	Alexander	Golden	pchristian@gmail.com	2020-10-23 07:00:00.000000	3
14	14	2020-10-25 15:59:35.470290	<null>	Crystal	Diaz	panderson@yahoo.com	2020-10-16 07:00:00.000000	3
15	15	2020-10-25 15:59:35.470290	<null>	Lisa	Erickson	gutierrezkevin@yahoo.com	2020-09-28 07:00:00.000000	3
16	16	2020-10-25 15:59:35.470290	<null>	Brooke	Long	albert74@gonzalez-cooper.net	2020-10-12 07:00:00.000000	4
17	17	2020-10-25 15:59:35.470290	<null>	Diane	Garza	astanley@yahoo.com	2020-10-20 07:00:00.000000	4
18	18	2020-10-25 15:59:35.470290	<null>	Christine	Jones	hoffmanmartha@yahoo.com	2020-09-29 07:00:00.000000	4
19	19	2020-10-25 15:59:35.470290	<null>	John	Harvey	cmcclure@gmail.com	2020-10-02 07:00:00.000000	4
20	20	2020-10-25 15:59:35.470290	<null>	Heidi	Hamilton	anna53@wilson.com	2020-10-04 07:00:00.000000	4

Приклад даних у БД



Приклад кількості даних у БД

```
~/Projects/University/DB_course_labs on master
python3 db_labs/cli/__init__.py --option search_developers
Please enter a search keyword(first/last name or skill name): Dizzzmas
1 developers found for the keyword: Dizzzmas
[{'birthdate': None,
  'email': 'dmytro@jetbridge.com',
  'first_name': 'Dizzzmas',
  'id': None,
  'last_name': None}]
```

Приклад пошукового запиту у БД

Аналіз швидкодії запиту на пошук до БД(EXPLAIN ANALYZE):
https://explain.dalibo.com/plan/pQe

Висновок

Виконавши дану лабораторну роботу було відпрацьовано навички створення прикладних додатків з використанням PostgreSQL як СКБД .