Лабораторна робота №2
По дисципліні “Бази даних”

Студента
Групи КП-02
Литвиненка Артема Сергійовича

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу.

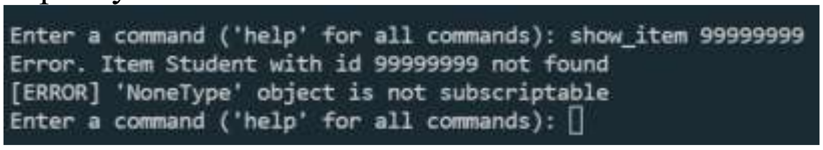
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

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).

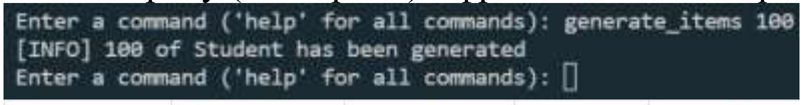
Результати:

- ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних, ілюстрації валідації даних при уведенні користувачем:

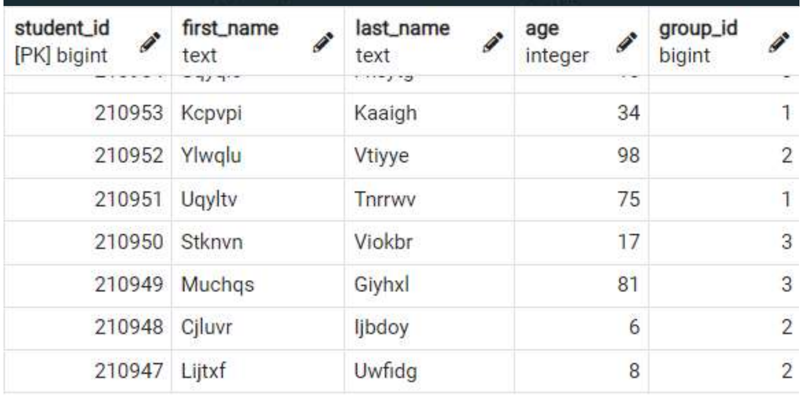


```
Enter a command ('help' for all commands): show_item 99999999
Error. Item Student with id 99999999 not found
[ERROR] 'NoneType' object is not subscriptable
Enter a command ('help' for all commands): 
```

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць:

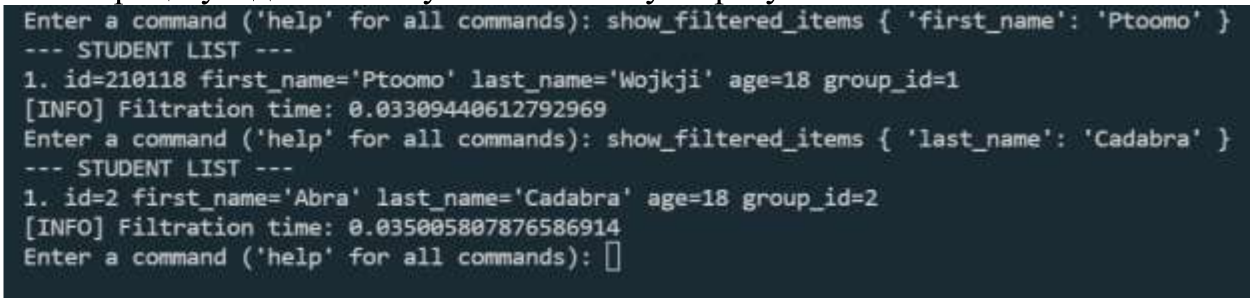


```
Enter a command ('help' for all commands): generate_items 100
[INFO] 100 of Student has been generated
Enter a command ('help' for all commands): 
```



student_id [PK] bigint	first_name text	last_name text	age integer	group_id bigint
210953	Kcpvpi	Kaaigh	34	1
210952	Ylwqlu	Vtiyye	98	2
210951	Uqyltv	Tnrrwv	75	1
210950	Stknvn	Viokbr	17	3
210949	Muchqs	Giyhxl	81	3
210948	Cjluvr	Ijbdoy	6	2
210947	Lijtxf	Uwfidg	8	2

- ілюстрації уведення пошукового запиту та результатів виконання запитів.



```
Enter a command ('help' for all commands): show_filtered_items { 'first_name': 'Ptoomo' }
--- STUDENT LIST ---
1. id=210118 first_name='Ptoomo' last_name='Wojkji' age=18 group_id=1
[INFO] Filtration time: 0.033094406127929690
Enter a command ('help' for all commands): show_filtered_items { 'last_name': 'Cadabra' }
--- STUDENT LIST ---
1. id=2 first_name='Abra' last_name='Cadabra' age=18 group_id=2
[INFO] Filtration time: 0.035005807876586914
Enter a command ('help' for all commands): 
```

Код:

main.py

```python
import psycopg2
import inspect
from pprint import pprint
from time import time
import os

import models
from config import host, user, password, db_name
from logger import Logger

from repository import Repository
from controller import Controller
from view import View


def get_connection(host, user, password, db_name):
    return psycopg2.connect(
        host=host,
        user=user,
        password=password,
        database=db_name
    )


def start_session():
    connection = get_connection(host, user, password, db_name)
    session = Session(connection)
    session.start()


class Session:

    def __init__(self, connection):
        self.__connection = connection

    def start(self):
        connection = get_connection(host, user, password, db_name)
        Logger.log_info("PostgreSQL connection opened")
        while True:
            try:
                model_type = input('Input model type: ')
                model = self.search_model(model_type)
                controller = Controller(
                    Repository(connection, model), View())
                while True:
                    command = input(
                        'Enter a command (\'help\' for all commands): ')
                    if command == 'switch_model':
                        break

                    try:
                        self.dispatch_command(controller, command)
                    except Exception as _ex:
                        Logger.log_error(_ex)
                    continue
            except Exception as _ex:
                Logger.log_error(_ex)
                continue

    def exit(self):
        self.__connection.close()
        Logger.log_info("PostgreSQL connection closed")
        quit()

    def switch_model(self):
        self.__connection.close()
        self.start()

    def get_commands(self, controller):
        commands = {
            'exit': self.exit,
            'show_items': controller.show_items,
            'show_item': controller.show_item,
            'show_filtered_items': controller.show_filtered_items,
            'insert_item': controller.insert_item,
            'update_item': controller.update_item,
            'delete_item': controller.delete_item,
            'generate_items': controller.generate_items,
            'switch_model': '',
        }
        commands['help'] = pprint
        return commands

    def dispatch_command(self, controller, command):
        command_parts = command.split(' ')
        command = command_parts[0]
        command_param = ''.join(command_parts[1:])
        commands = self.get_commands(controller)
        if command == 'help':
            commands[command](tuple(commands.keys()))
        elif command == 'cls':
            os.system('cls')
        elif command == 'show_filtered_items':
            start_time = time()
            commands[command](command_param)
            end_time = time()
            Logger.log_info(f'Filtration time: {end_time - start_time}')
        else:
            commands[command](command_param)

    def search_model(self, model_name):
        for _, model in inspect.getmembers(models, inspect.isclass):
            if model.__name__.lower() == model_name.lower():
                return model
        raise Exception('Model not found')


if __name__ == '__main__':
    start_session()
```

controller.py

```
from ast import literal_eval


class Controller():

    def __init__(self, repository, view):
        self.repository = repository  # crud api for existing model
        self.view = view

    def show_items(self, bullet_points=False):
        items = self.repository.get_items()
        item_name = self.repository.model.__name__
        if bullet_points:
            self.view.show_bullet_point_list(item_name, items)
        else:
            self.view.show_number_point_list(item_name, items)

    def show_item(self, item_id: int):
        item_name = self.repository.model.__name__
        try:
            item = self.repository.get_item_by_id(item_id)
            self.view.show_item(item, item_name)
        except Exception as _ex:
            self.view.display_missing_item_error(item_name, item_id, _ex)

    def show_filtered_items(self, attrs, bullet_points=False):
        items = self.repository.get_filtered_items(literal_eval(attrs))
        item_name = self.repository.model.__name__
        if bullet_points:
            self.view.show_bullet_point_list(item_name, items)
        else:
            self.view.show_number_point_list(item_name, items)

    def insert_item(self, item_data_tuple):
        item = self.repository.model.from_tuple(literal_eval(item_data_tuple))
        try:
            self.repository.create_item(item)
            self.view.display_item_insertion(self.repository.model.__name__)
        except Exception as _ex:
            self.view.display_insert_item_error(item, _ex)

    def update_item(self, item_data_tuple):
        item_type = self.repository.model.__name__
        item = self.repository.model.from_tuple(literal_eval(item_data_tuple))
        try:
            is_updated = self.repository.update_item(item)
            if not is_updated:
                raise Exception('Item not found exception')

            self.view.display_item_updated(item_type, item.id)
        except Exception as _ex:
            self.view.display_missing_item_error(item_type, item.id, _ex)

    def delete_item(self, item_id: int):
        item_type = self.repository.model.__name__
        try:
            is_deleted = self.repository.delete_item(item_id)
            if not is_deleted:
                raise Exception('Item not found exception')

            self.view.display_item_deletion(item_type, item_id)
        except Exception as _ex:
            self.view.display_missing_item_error(item_type, item_id, _ex)

    def generate_items(self, amount):
        item_type = self.repository.model.__name__
        self.repository.generate_items(int(amount))
        self.view.display_items_generated(amount, item_type)
```

logger.py

```python
class Logger:

    @staticmethod
    def log_info(message):
        print(f'[INFO] {message}')

    @staticmethod
    def log_error(message):
        print(f'[ERROR] {message}')
```

models.py

```python
from pydantic import BaseModel


class Model(BaseModel):
    @classmethod
    def from_tuple(cls, data):
        return [cls(**{key: data[i] for i, key in enumerate(
                cls.__fields__.keys())})][0]


class Student(Model):
    __name__ = 'student'
    __table__ = "students"
    id: int
    first_name: str
    last_name: str
    age: int
    group_id: int


class Group(Model):
    __name__ = 'group'
    __table__ = "groups"
    id: int
    group_name: str


class Subject(Model):
    __name__ = 'subject'
    __table__ = "subjects"
    id: int
    subject_name: str


class Mark(Model):
    __name__ = 'mark'
    __table__ = "marks"
    id: int
    student_id: int
    subject_id: int
    mark: int
```

repository.py

```
import inspect

import models
from models import Model
from pprint import pprint


class Repository():

    def __init__(self, connection, model, autocommit=True):
        connection.autocommit = autocommit
        self.__connection = connection
        self.model = model  # __table__, __name__, __fields__

    def __get_queries(self):
        return {
            'get_items': """
                SELECT * FROM {0}
            """.format(self.model.__table__),
            'get_item_by_id': """
                SELECT * FROM {0} WHERE {1}_id = %(id)s
            """
            .format(self.model.__table__, self.model.__name__.lower()),
            'insert_items': """
              INSERT INTO {0} {1}
              VALUES {2}
            """,
            'update_item': """
              UPDATE {0} SET {1}
              WHERE {2}_id = %(id)s
            """.format(self.model.__table__, ', '.join([f"{field} =
%({field})s" for field in tuple(self.model.__fields__)[1:]]),
self.model.__name__.lower()),
            'delete_item': """
                DELETE FROM {0} WHERE {1}_id = %(id)s
            """.format(self.model.__table__, self.model.__name__.lower()),
            'generate_random_name_series': """
                SELECT chr(trunc(65+random() * 25)::int)
                || chr(trunc(97+random() * 25)::int)
                || chr(trunc(97+random() * 25)::int)
                || chr(trunc(97+random() * 25)::int)
                || chr(trunc(97+random() * 25)::int)
                || chr(trunc(97+random() * 25)::int)
                FROM generate_series(1, %(amount)s)
```

```python
            """,
            'generate_random_integer_series': """
                SELECT trunc(random() * 100)::int
                FROM generate_series(1, %(amount)s)
            """,
            'get_random_id': """
                SELECT {0}_id FROM {1}
                ORDER BY RANDOM()
                LIMIT 1
            """.format(self.model.__name__, self.model.__table__),
            'filter_by_attribute': """
                SELECT * FROM {0}
                WHERE {1}
            """
        }

    def get_items(self):
        cursor = self.__connection.cursor()
        cursor.execute(self.__get_queries()['get_items'])

        items_data = cursor.fetchall()
        items = [self.model.from_tuple(item_data)
                 for item_data in items_data]

        cursor.close()
        return items

    def get_item_by_id(self, id: int):
        cursor = self.__connection.cursor()
        cursor.execute(
            self.__get_queries()['get_item_by_id'], {'id': str(id)}
        )

        items_data = cursor.fetchone()
        item = self.model.from_tuple(items_data)

        cursor.close()
        return item

    def create_item(self, item: Model):
        cursor = self.__connection.cursor()
        cursor.execute(
            self.__get_queries()['insert_item'], item.dict()
        )

        cursor.close()

    def create_items(self, items: list[Model]):
        cursor = self.__connection.cursor()
        values = [str(tuple(item.dict().values())[1:]) for item in items]
        cursor.execute(
            self.__get_queries()['insert_items']
                .format(self.model.__table__,
str(tuple(self.model.__fields__)[1:]).replace("'", ""),
                        ','.join(values))
        )

        cursor.close()

    def update_item(self, item: Model):
        cursor = self.__connection.cursor()
        item_dict = item.dict()
        cursor.execute(
            self.__get_queries()['update_item'], item_dict
        )

        rows_updated = cursor.rowcount
        cursor.close()

        return bool(rows_updated)

    def delete_item(self, id: int):
        cursor = self.__connection.cursor()
        cursor.execute(
            self.__get_queries()['delete_item'], {'id': str(id)}
        )

        rows_deleted = cursor.rowcount
        cursor.close()

        return bool(rows_deleted)

    def get_random_id(self):
        cursor = self.__connection.cursor()
        cursor.execute(
            self.__get_queries()['get_random_id']
        )

        random_id = cursor.fetchone()
        cursor.close()

        return random_id

    def generate_random_id_series(self, amount: int):
        random_ids = []
        for _ in range(int(amount)):
            random_ids.append(self.get_random_id())

        return random_ids

    def __get_model_of_foreign_key(self, field):
        model_name = field.split('_')[0]
        for _, model in inspect.getmembers(models, inspect.isclass):
            if model.__name__.lower() == model_name:
                return model
        return None

    def __get_tuples_from_fields_series(self, fields_series):
        return [
            tuple([field_series[j][0] for field_series in fields_series]) for j
in range(len(fields_series[0]))
        ]

    def generate_items(self, amount: int):
        cursor = self.__connection.cursor()

        fields_series = []
        for field in list(self.model.__fields__):
            if (field.endswith('id') and field != 'id'):
                model = self.__get_model_of_foreign_key(field)
                modelRepo = Repository(self.__connection, model)
                fields_series.append(
                    modelRepo.generate_random_id_series(amount))
                continue

            is_int = self.model.__annotations__[field] == int
            query_type = 'generate_random_integer_series' if is_int else
'generate_random_name_series'

            cursor.execute(
                self.__get_queries()[query_type],
                {
                    'amount': str(amount)
                }
            )

            random_data = cursor.fetchall()
            fields_series.append(random_data)

        items = [self.model.from_tuple(item_data)
                 for item_data in
self.__get_tuples_from_fields_series(fields_series)]
        self.create_items(items)

        cursor.close()

    def get_filtered_items(self, attrs: dict):
        cursor = self.__connection.cursor()

        cursor.execute(
            self.__get_queries()['filter_by_attribute']
                .format(self.model.__table__, ' AND '.join([f'{key} =
%({key})s' for key in attrs])),
            attrs
        )

        items_data = cursor.fetchall()
        items = [self.model.from_tuple(item_data)
                 for item_data in items_data]

        cursor.close()
        return items
```

view.py

```
from logger import Logger


class View():

    @staticmethod
    def show_bullet_point_list(item_type, items):
        print('--- {} LIST ---'.format(item_type.upper()))
        for item in items:
            print('* {}'.format(item))

    @staticmethod
    def show_number_point_list(item_type, items):
        print('--- {} LIST ---'.format(item_type.upper()))
        for i, item in enumerate(items):
            print('{}. {}'.format(i+1, item))

    @staticmethod
    def show_item(item, item_name):
        print('Item of {} with id {} found'.format(item_name, item.id))
        Logger.log_info(item)

    @staticmethod
    def display_missing_item_error(item_type, id, err):
        print('Error. Item {} with id {} not found'.format(item_type, id))
        Logger.log_error(err)

    @staticmethod
    def display_insert_item_error(item, err):
        print('Error. Could not insert {} into database'
              .format(item))
        Logger.log_error(err)

    @staticmethod
    def display_item_updated(item_type, item_id):
        Logger.log_info(f'Updated item {item_type} with id {item_id}')

    @staticmethod
    def display_item_deletion(item_name, item_id):
```

```python
        Logger.log_info(
            f'{item_name} with id {item_id} has been removed from database'
        )

    @staticmethod
    def display_item_insertion(item_type):
        Logger.log_info(f'{item_type} has been inserted to database')

    @staticmethod
    def display_items_generated(count, item_type):
        Logger.log_info(
            f'{count} of {item_type} has been generated'
        )
```