Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики
Кафедра спеціалізованих комп’ютерних систем

Лабораторна робота № 3

з дисципліни «Бази даних і засоби управління»
«Ознайомлення з базовими операціями СУБД PostgreSQL»

Виконав:
студент групи КП-73
Литвиненко Антон

Перевірив:
_____
_____
_____

Завдання

Завдання роботи полягає у наступному:

1.Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

2.Створити та проаналізувати різні типи індексів у PostgreSQL.

3.Розробити тригер бази даних PostgreSQL.

4.Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Порядок виконання роботи

В ході роботи розроблено:

1. Логічну модель БД та Діаграму класів;
2. Функціонал програмного додатку;
3. ОО програмний додаток роботи з БД "Система питання-відповідь".

Для взаємодії з БД використано ORM модуль SQLAlchemy.

Логічна модель бази даних наведена на Рис 1.

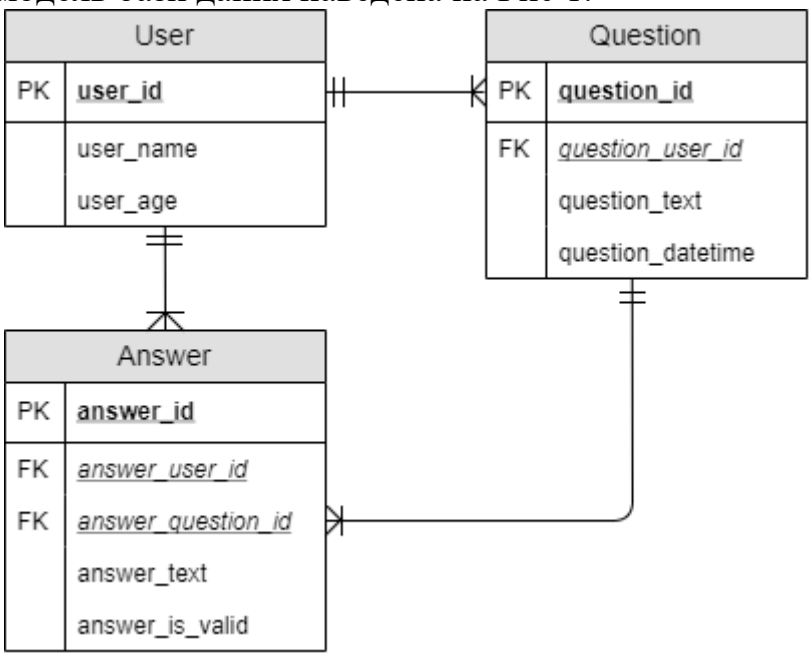

Рис 1. Логічна модель бази даних

Сутнісні класи програми наведені на Рис 2.

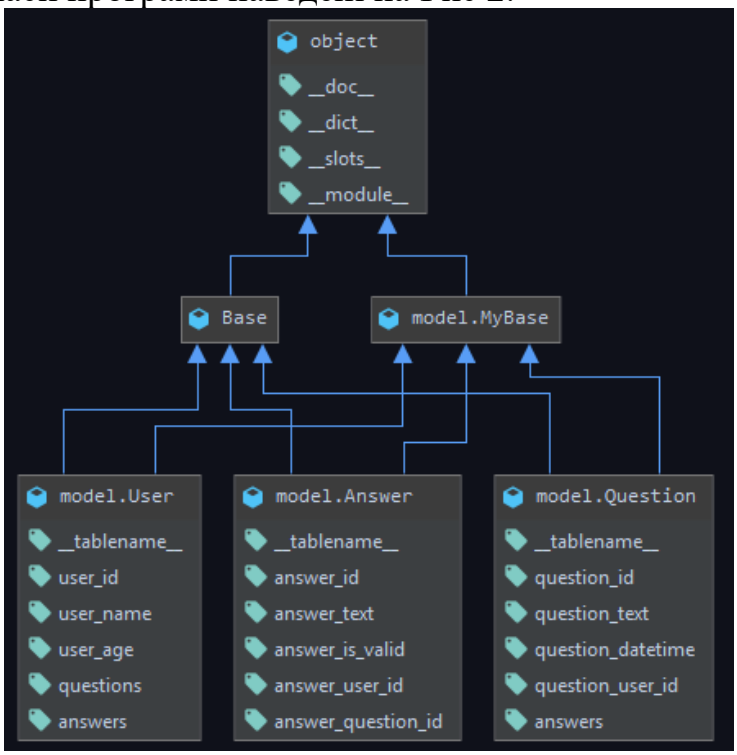

Рис 2. Фрагмент UML діаграми сутнісних класів

Зв’язки між сутнісними класами, сгенеровані за допомогою PyCharm, наведені на Рис 3.

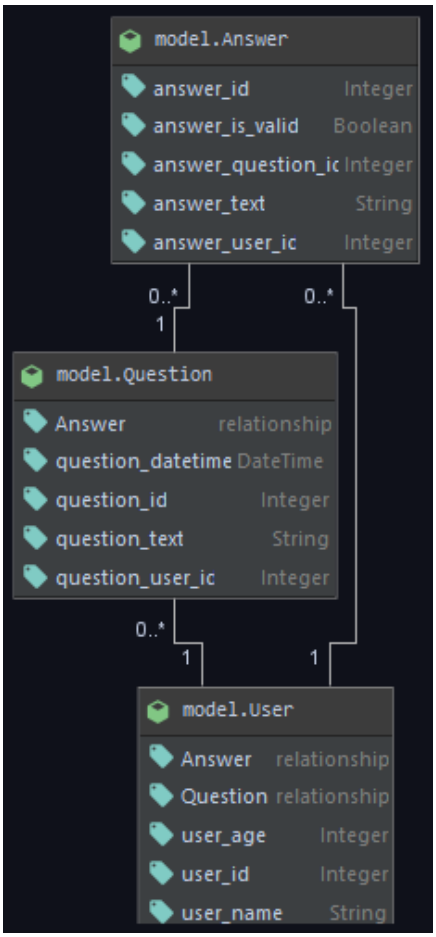

Рис 3. Зв’язки між сутнісними класами

Лістинг програми

```
from controller import display_main_menu

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

controller.py

```
from consolemenu import SelectionMenu

import model
import view


def display_main_menu(err='', table=None):
    tables = list(model.TABLES.keys())

    menu = SelectionMenu(tables + ['Make commit'], subtitle=err,
                         title="Select a table to work with:")
    menu.show()


    index = menu.selected_option
    if index < len(tables):
        table = tables[index]
        display_secondary_menu(table)
    elif index == len(tables):
        model.commit()
        display_main_menu('Commit was made successful')


def display_secondary_menu(table, subtitle=''):
    opts = ['Select', 'Insert', 'Update', 'Delete']
    steps = [select, insert, update, delete, display_main_menu]

    menu = SelectionMenu(
        opts, subtitle=subtitle,
        title=f'Selected table "{table}"', exit_option_text='Go back',)
    menu.show()
    index = menu.selected_option
    steps[index](table=table)


def select(table):
    query = view.multiple_input(table, 'Enter requested fields:')
    data = model.get(table, query)
    view.print_entities(table, data)
    view.press_enter()
    display_secondary_menu(table)


def insert(table):
    data = view.multiple_input(table, 'Enter new fields values:')
    model.insert(table, data)
    display_secondary_menu(table, 'Insertion was made successfully')


def update(table):
    condition = view.single_input(
        table, 'Enter requirement of row to be changed:')
    query = view.multiple_input(table, 'Enter new fields values:')

    model.update(table, condition, query)
```

```
    display_secondary_menu(table, 'Update was made successfully')


def delete(table):
    query = view.multiple_input(
        table, 'Enter requirement of row to be deleted:')

    model.delete(table, query)
    display_secondary_menu(table, 'Deletion was made successfully')
```

model.py

```
from sqlalchemy import Column, Integer, String, DateTime, \
    Boolean, ForeignKey, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.orm import sessionmaker

COLUMN_WIDTH = 25
db_str = 'postgres://anton:password@localhost:5432/labs'
db = create_engine(db_str)
Base = declarative_base()


class MyBase:
    def __init__(self, **kwargs):
        for attr, val in kwargs.items():
            setattr(self, attr, val)

    def __clean_dict(self):
        clean = self.__dict__.copy()
        clean.pop('_sa_instance_state')
        return clean

    def get_columns(self):
        return self.__clean_dict().keys()

    def get_values(self):
        return self.__clean_dict().values()


class User(MyBase, Base):
    __tablename__ = 'user'
    user_id = Column(Integer, primary_key=True)
    user_name = Column(String, nullable=False)
    user_age = Column(Integer)

    questions = relationship('Question')
    answers = relationship('Answer')


class Question(MyBase, Base):
    __tablename__ = 'question'

    question_id = Column(Integer, primary_key=True)
    question_text = Column(String, nullable=False)
    question_datetime = Column(DateTime)
    question_user_id = Column(Integer, ForeignKey('user.user_id'))

    answers = relationship('Answer')


class Answer(MyBase, Base):
```

```
    __tablename__ = 'answer'

    answer_id = Column(Integer, primary_key=True)
    answer_text = Column(String, nullable=False)
    answer_is_valid = Column(Boolean, default=False)
    answer_user_id = Column(Integer, ForeignKey('user.user_id'))
    answer_question_id = Column(Integer, ForeignKey('question.question_id'))


session = sessionmaker(db)()
Base.metadata.create_all(db)

TABLES = {
    'user': ('user_id', 'user_name', 'user_age'),
    'question': ('question_id', 'question_text', 'question_datetime',
'question_user_id'),
    'answer': ('answer_id', 'answer_text', 'answer_is_valid', 'answer_user_id',
'answer_question_id')
}

MODELS = {'user': User, 'question': Question, 'answer': Answer}

def insert(table, opts):
    object_class = MODELS[table]
    obj = object_class(**opts)
    session.add(obj)


def get(table, opts=None):
    objects_class = MODELS[table]
    objects = session.query(objects_class)
    for key, item in opts.items():
        objects = objects.filter(getattr(objects_class, key) == item)

    return list(objects)


def update(table, condition, opts):
    column, value = condition
    object_class = MODELS[table]
    filter_attr = getattr(object_class, column)
    objects = session.query(object_class).filter(filter_attr == value)

    for obj in objects:
        for key, item in opts.items():
            setattr(obj, key, item)


def delete(table, opts):
    objects_class = MODELS[table]
    objects = session.query(objects_class)
    for key, item in opts.items():
        objects = objects.filter(getattr(objects_class, key) == item)

    objects.delete()


def commit():
    session.commit()
```

view.py

```python
import model

COLUMN_WIDTH = 25


def print_entities(table, data):
    entities = data
    if not entities:
        return

    cols = entities[0].get_columns()
    separator_line = '-' * COLUMN_WIDTH * len(cols)

    print(f'Working with table "{table}"', end='\n\n')
    print(separator_line)
    print(''.join([f'{col}     |'.rjust(COLUMN_WIDTH, ' ') for col in cols]))
    print(separator_line)

    for entity in entities:
        print(''.join([f'{val}     |'.rjust(COLUMN_WIDTH, ' ') for val in
entity.get_values()]))
    print(separator_line)


def specified_input(colname=None, msg=None):
    if msg:
        print(msg)
    if colname:
        print(f'{colname}=', end='')
    return input()


def single_input(tname, msg):
    print(msg)
    print('(use format <attribute>=<value>)')
    print(f'({"/".join(model.TABLES[tname])})', end='\n\n')

    while True:
        data = input()
        if not data or data.count('=') != 1:
            print('Invalid input, try one more time')
            continue

        data = data.split('=')
        col, val = data[0].strip(), data[1].strip()
        if col.lower() in [tcol.lower() for tcol in model.TABLES[tname]]:
            return col, val
        else:
            print(f'Invalid column name "{col}" for table "{tname}"')


def multiple_input(tname, msg):
    print(msg)
    print('(use format <attribute>=<value>)')
    print(f'({"/".join(model.TABLES[tname])})', end='\n\n')

    res = {}
    while True:
        data = input()
        if not data:
            break
        if data.count('=') != 1:
            print('Invalid input')
            continue
```

```
        data = data.split('=')
        col, val = data[0].strip(), data[1].strip()
        if col.lower() in [tcol.lower() for tcol in model.TABLES[tname]]:
            res[col] = val
        else:
            print(f'Invalid column name "{col}" for table "{tname}"')

    return res


def multiline_input(msg):
    print(msg, end='\n\n')

    lines = []
    while True:
        line = input()
        if not line:
            break
        lines.append(line)

    return '\n'.join(lines)


def press_enter():
    input()
```

Індекси

Btree індекс:

```
create index btree_index on question using btree(question_id);
```

Порядок звертання до таблиці без використання фільтру по колонці, на яку додано індекс (створений індекс не використовується):

Запит:

```
explain select * from question;
```

Результат:

	QUERY PLAN
1	Seq Scan on question (cost=0.00..4062.10 rows=200010 width=49)

Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):

Запит:

```
explain select * from question where question_id = 1000;
```

Результат:

	QUERY PLAN
1	Index Scan using btree_index on question (cost=0.42..8.44 rows=1 width=49)
2	Index Cond: (question_id = 1000)

BRIN індекс:

```
create index brin_index on question using brin(question_datetime);
```

Порядок звертання до таблиці без використання фільтру по колонці, на яку додано індекс (створений індекс не використовується):

Запит:

```
explain select * from question;
```

Результат:

	QUERY PLAN
1	Seq Scan on question (cost=0.00..4062.10 rows=200010 width=49)

Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):

Запит:

```
explain select * from question
where question_datetime between '2005-01-01 00:00:00' and '2015-01-01 00:00:00';
```

Результат:

	QUERY PLAN
1	Gather (cost=1015.42..27542.37 rows=1 width=49)
2	Workers Planned: 2
3	-> Parallel Bitmap Heap Scan on question (cost=15.42..26542.27 rows=1 width=49)
4	Recheck Cond: ((question_datetime >= '2005-01-01 00:00:00'::timestamp without…
5	-> Bitmap Index Scan on brin_index (cost=0.00..15.42 rows=1324616 width=0)

Тригер

BEFORE INSERT

Якщо текст питання не закінчується знаком питання(?), то таке питання не буде додане.

Код:

```
create or replace function before_insert_question()
returns trigger
language plpgsql
as $$
begin
    if NEW.question_text LIKE '%?' then
        return NEW;
    end if;
    raise exception 'question text should end with `?`';
end;
$$;

create trigger before_insert before insert on question
    for each row execute procedure before_insert_question();
```

Приклади результатів:

```
insert into question(question_text, question_user_id) values ('are you ok', 1);
```

```
[P0001] ERROR: question text should end with `?`
Where: PL/pgSQL function before_insert_question() line 6 at RAISE
```

Або вдале додавання і усі питання після цього:

```
insert into question(question_text, question_user_id) values ('are you ok?', 1);
```

	question_id	question_text	question_datetime	question_user_id
1	1	Who are you?	2015-01-01 01:00:00.000000	1
2	2	Is sky blue?	2016-01-01 01:00:00.000000	2
3	3	Are you ok?	2017-01-01 01:00:00.000000	2
4	4	How are you?	2018-01-01 01:00:00.000000	3
5	5	Are you hungry?	2019-01-01 01:00:00.000000	4
6	1780010	are you ok?	<null>	1

BEFORE DELETE

Якщо таблиці менше 10 питань, видалити питання не вдасться. (реалізовано курсорним циклом через вимогу використання в функції тригера подібного циклу, тим не менш підрахунок кількості питань в таблиці можна було б реалізувати SQL запитом).

Код:

```
create or replace function before_delete_question()
returns trigger
language plpgsql
as $$
declare
    all_questions cursor is select * from question;
    question_count integer = 0;
begin
        for q in all_questions loop
        question_count := question_count + 1;
    end loop;
    if question_count < 10 then
        raise exception 'At least 10 questions should be in table';
    end if;
    return old;
end;
$$;

create trigger before_delete before delete on question
    for each row execute procedure before_delete_question();
```

Приклади результатів:

```
select count(*) from question;
```

	count
1	11

```
delete from question where question_id = 1780010;
select count(*) from question;
```

	count
1	10

```
delete from question where question_id = 1780014;
```

```
[P0001] ERROR: At least 10 questions should be in table
Where: PL/pgSQL function before_delete_question() line 10 at RAISE
```

```
select count(*) from question;
```

	count
1	10

Дослідження рівнів ізоляції

Для перевірки аномалій буде використовуватися розроблений програмний додаток, запущений у двох екземплярах паралельно.

1. **READ COMMITTED**

Перевіримо наявність аномалії “dirty read”, коли транзакція читає дані, які ще не були закомічені паралельною транзакцією.

Транзакція №1

Перевіряє список користувачів, видаляє якогось користувача, перевіряє видалення.

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |           user_age     |            user_id     |
-------------------------------------------------------------------------
            lolkek     |                  20    |                  1     |
              user     |                  30    |                  2     |
          user1234     |                  15    |                  3     |
         nagibator     |                  20    |                  4     |
            pro100     |                  10    |                  5     |
            sample     |                  11    |                  6     |
         to_delete     |                None    |                  9     |
-------------------------------------------------------------------------
```

```
Enter requirement of row to be deleted:
(use format <attribute>=<value>)
(user_id/user_name/user_age)

user_name=to_delete
Working with table "user"

-------------------------------------------------------------------------
         user_name     |           user_age     |            user_id     |
-------------------------------------------------------------------------
            lolkek     |                  20    |                  1     |
              user     |                  30    |                  2     |
          user1234     |                  15    |                  3     |
         nagibator     |                  20    |                  4     |
            pro100     |                  10    |                  5     |
            sample     |                  11    |                  6     |
-------------------------------------------------------------------------
```

Транзакція №2
Отримує список всіх Користувачів.

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |          user_age     |           user_id     |
-------------------------------------------------------------------------
            lolkek     |                 20    |                 1     |
              user     |                 30    |                 2     |
          user1234     |                 15    |                 3     |
         nagibator     |                 20    |                 4     |
            pro100     |                 10    |                 5     |
            sample     |                 11    |                 6     |
         to_delete     |               None    |                 9     |
-------------------------------------------------------------------------
```

Транзакція №1
Робить коміт.

```
Commit was made successful
```

Транзакція №2
Отримує список всіх Клієнтів.

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |          user_age     |           user_id     |
-------------------------------------------------------------------------
            lolkek     |                 20    |                 1     |
              user     |                 30    |                 2     |
          user1234     |                 15    |                 3     |
         nagibator     |                 20    |                 4     |
            pro100     |                 10    |                 5     |
            sample     |                 11    |                 6     |
-------------------------------------------------------------------------
```

Як бачимо Транзація №2 не бачила зміни, внесені до таблиці Транзакцією №1, до ти пір поки остання не закомітила свої зміни. Отже, було доведено, що рівень ізоляції READ COMMITTED захищає від аномалії “брудного читання”.

2. **REPEATABLE READ**

Перевіримо наявність аномалії “nonrepeatable read”, коли транзакція повторно зчитує дані і вони виявляються модифіковані комітом паралельної транзакції.

Транзакція №1
Отримує список усіх Користувачів.

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |          user_age     |            user_id     |
-------------------------------------------------------------------------
            lolkek     |                 20    |                 1     |
              user     |                 30    |                 2     |
          user1234     |                 15    |                 3     |
         nagibator     |                 20    |                 4     |
            pro100     |                 10    |                 5     |
            sample     |                 11    |                 6     |
-------------------------------------------------------------------------
```

Транзакція №2

Створює нового Клієнта, комітить зміни і перевіряє його наявність в списку усіх клієнтів.

```
Enter new fields values:
(use format <attribute>=<value>)
(user_id/user_name/user_age)

user_name=new_user
```

```
Commit was made successful
```

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |          user_age     |            user_id     |
-------------------------------------------------------------------------
            lolkek     |                 20    |                 1     |
              user     |                 30    |                 2     |
          user1234     |                 15    |                 3     |
         nagibator     |                 20    |                 4     |
            pro100     |                 10    |                 5     |
            sample     |                 11    |                 6     |
          new_user     |               None    |                10     |
-------------------------------------------------------------------------
```

Транзакція №1

Заново отримує список усіх клієнтів.

```
Working with table "user"

-------------------------------------------------------------------------
         user_name     |          user_age     |            user_id     |
-------------------------------------------------------------------------
            lolkek     |                 20    |                 1     |
              user     |                 30    |                 2     |
          user1234     |                 15    |                 3     |
         nagibator     |                 20    |                 4     |
            pro100     |                 10    |                 5     |
            sample     |                 11    |                 6     |
-------------------------------------------------------------------------
```

Як бачимо, Транзакція №1 так і не побачила закомічені зміни Транзакції №2, так як перша з них зчитувала дані з таблиці ще до коміту. Отже, було доведено, що рівень ізоляції REPEATBLE READ захищає від аномалії “nonrepeatable read”.

3. **SERIALIZABLE**

Перевіримо наявність аномалії “serialization anomaly”, коли дві паралельні транзакції хочуть закомітити свої результати і при цьому є різниця, в якому порядку виконувати команди, виконані кожною з транзакцій.

Транзакція №1

Створює нове питання з текстом ‘serialize me’.

```
Enter new fields values:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=serialize me
question_user_id=1
```

Оновлює всі дописи з текстом ‘dont serialize me’.

```
Enter requirement of row to be changed:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=dont serialize me
Enter new fields values:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=serialize me
```

Транзакція №2

Створює допис з текстом ‘dont serialize me’.

```
Enter new fields values:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=dont serialize me
question_user_id=1
```

Оновлює всі дописи з текстом ‘serialize me’.

```
Enter requirement of row to be changed:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=serialize me
Enter new fields values:
(use format <attribute>=<value>)
(question_id/question_text/question_datetime/question_user_id)

question_text=dont serialize me
```

Транзакція №1
Намагається закомітити зміни

```
Commit was made successful
```

Транзакція №2
Намагається закомітити зміни

```
sqlalchemy.exc.OperationalError: (psycopg2.errors.SerializationFailure) could not serialize
 access due to read/write dependencies among transactions
DETAIL:  Reason code: Canceled on identification as a pivot, during commit attempt.
HINT:  The transaction might succeed if retried.
```

Як бачимо Транзація №2 не змогла закомітити зміни бо порядок виконання команд з обох транзакцій змінює вихідний результат. Отже, було доведено, що рівень ізоляції SERIALIZABLE захищає від “serialization anomaly”.