**Лабораторна робота № 2**

«**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**»

***з дисципліни***

**“БАЗИ ДАНИХ ТА ЗАСОБИ УПРАВЛІННЯ”**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Група: КВ-01**

**Виконала: Бондарчук Марія**

**Завдання роботи полягає у наступному:**

* + - 1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
      2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
      3. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі. 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).

**Деталізоване завдання:**

Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL (без ORM).

**Логічна модель предметної області «Пекарня»**

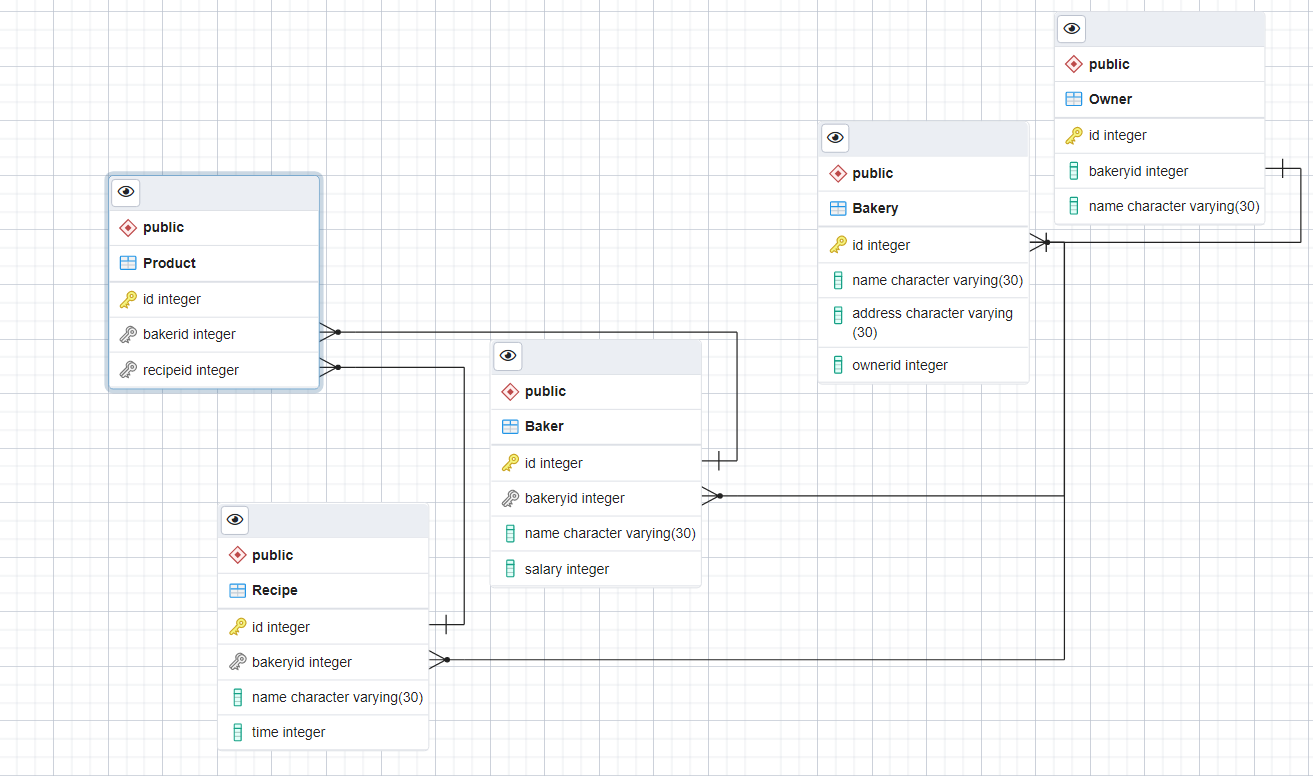


Рисунок 1 – Логічна модель предметної області «Пекарня»

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування JavaScript(Node.js), середовище розробки JetBrains WebStorm, а також сторонній клієнт node-postgres для Node.js, що надає API для доступу до PostgreSQL.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє скрипт, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі це BaseModel.js

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то ій відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє скрипт, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, у моїй програмі це BaseController.js

**Структура програми та її опис**

На рисунку 2 відображено деревовидну структуру програми:

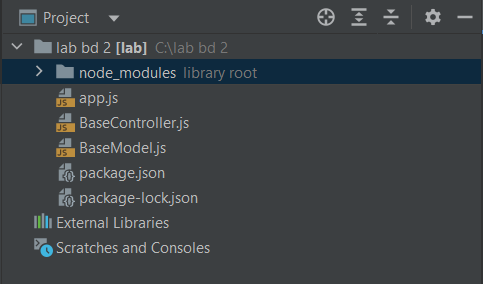


Рисунок 2 – структура програми

Програма умовно поділена на 2 модулі: скрипт BaseModel.js, скрипт BaseController.js та головний файл app.js.

Як видно з їх назв, файли повністю відповідають використаному патерну MVC.

У файлі BaseModel.js описаний скрипт моделі, що займається регулювання підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлі BaseController.js описаний скрипт взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі app.js описаний скрипт, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**

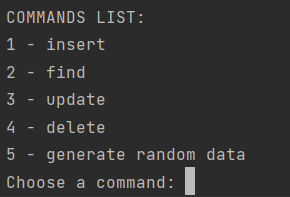


Рисунок 3. Меню для вибору операції над БД.

**Меню операцій складається з 5 пунктів (Рисунок 3)**

1. Додавання нового рядку даних до таблиці.
2. Пошук рядка з таблиці, який має певні параметри з представлених.
3. Оновлення рядку даних у таблиці.
4. Видалення рядку даних з таблиці.
5. Генерація рандомних даних для вибраної таблиці.

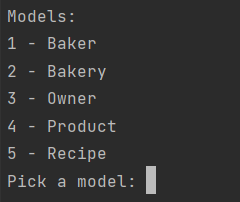


Рисунок 4. Меню для вибору таблиці, над якою проводитиметься обрана операція.

**Меню таблиці складається з 5 пунктів (Рисунок 4)**

1. Виконати обрану операцію над таблицею “baker”.
2. Виконати обрану операцію над таблицею “bakery”.
3. Виконати обрану операцію над таблицею “owner”.
4. Виконати обрану операцію над таблицею “product”.
5. Виконати обрану операцію над таблицею “recipe”.

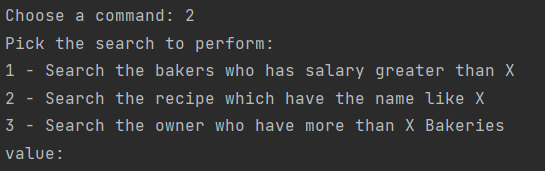


Рисунок 5. Меню для вибору умов пошуку.

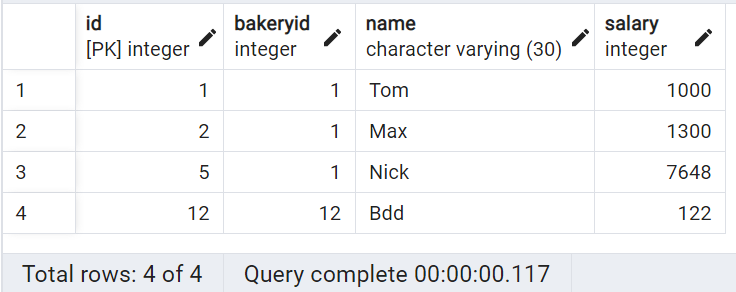
**Меню складається з 3 пунктів (Рисунок 5)**

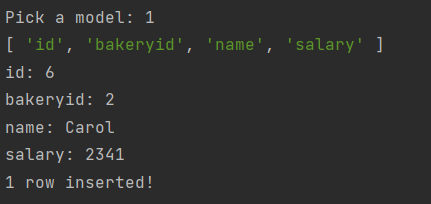
1. Пошук пекаря, який має зарплатню більшу ніж Х, який має ввести користувач.
2. Пошук рецепту за назвою.
3. Пошук власника, який має кількість пекарень більшу за Х, який має ввести користувач.

**Результати та виконання операцій**

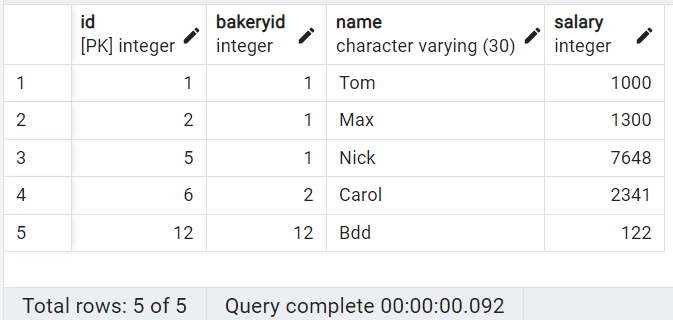
**Результат операції insert для таблиці baker**

Таблиця до



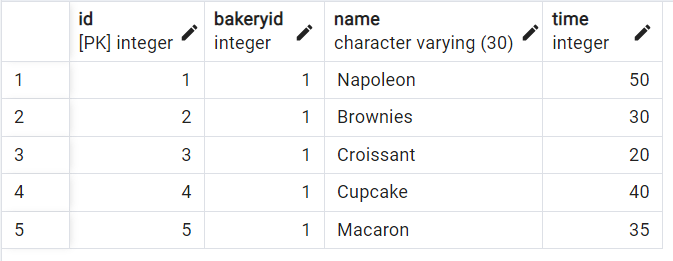


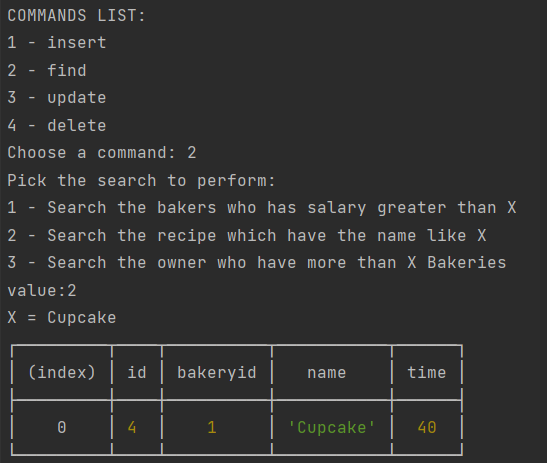
Таблиця після



**Результат операції find для таблиці recipe**

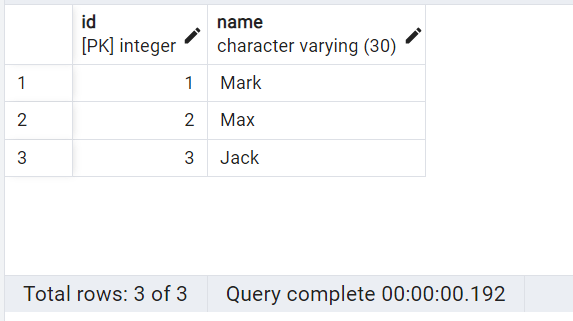
Таблиця

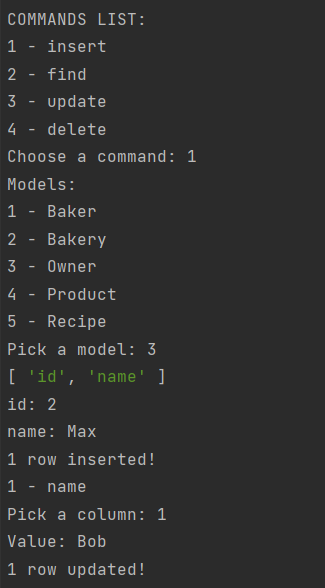




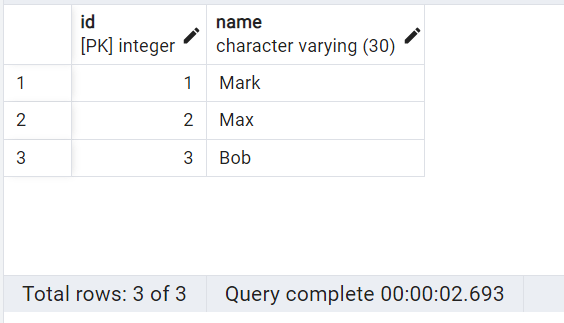
**Результат операції update для таблиці owner**

Таблиця до



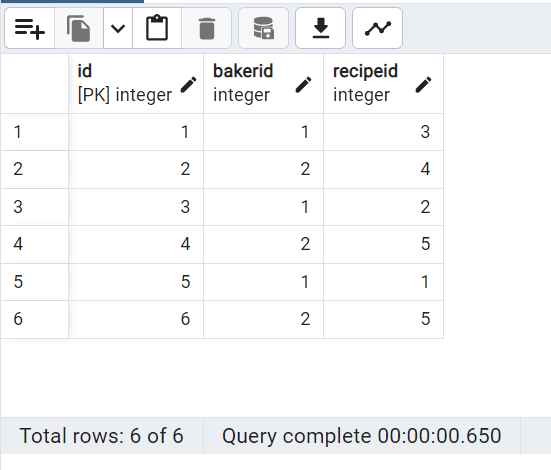


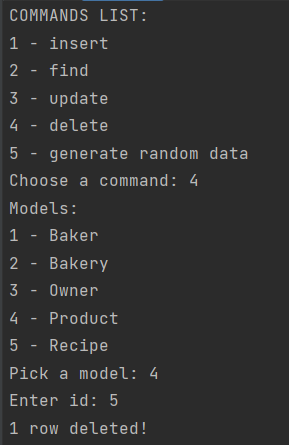
**Таблиця після**



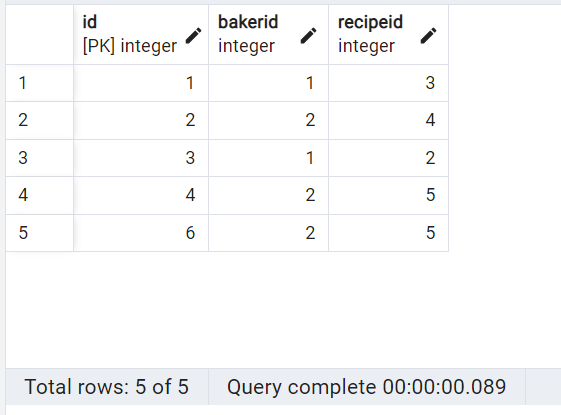
**Результат операції delete для таблиці product**

Таблиця до

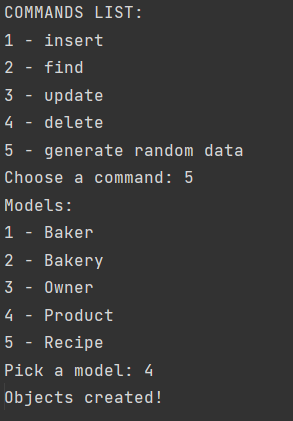




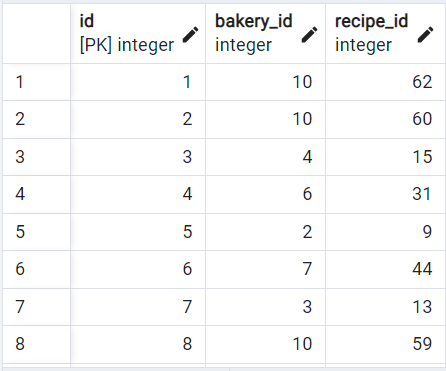
Таблиця після



**Результат операції generate random data для таблиці product**



Таблиця після



**Ілюстрації програмного коду**

Вибір даних з таблиці

const pickModel = async () => {  
 ***console***.log(`Models:\n${***Object***.keys(models).map((m, i) => `${i + 1} - ${m}`).join('\n')}`);  
 const i = +(await rl.question(`Pick a model: `));  
 return ***Object***.values(models)[i - 1];  
};

Функція додавання нових даних у таблицю

const insertFlow = async () => {  
 const model = await pickModel();  
 ***console***.log(model.keys);  
  
 const entries = [];  
 for (const key of model.keys) {  
 const value = await rl.question(`${key}: `);  
 entries.push([key, value]);  
 }  
  
 const obj = ***Object***.fromEntries(entries);  
 return performQuery(model.insert(obj))  
 .then(() => ***console***.log('1 row inserted!'));  
};  
  
const formatEnteredValue = (val) => {  
 if (Number.isFinite(Number.parseInt(val))) return val;  
 return val;  
};

Функція оновлення даних у рядках таблиця

const updateFlow = async () => {  
 const model = await pickModel();  
 const id = +(await rl.question(`Enter id: `));  
 ***console***.log(`Keys:\n${model.keys.map((key, i) => `${i} - ${key}`).join('\n')}`);  
 const i = +(await rl.question(`Pick a column: `));  
 const col = model.keys[i];  
 if (!col) return ***console***.info('Error: Wrong column!');  
 const value = formatEnteredValue(await rl.question('Value: '));  
 return performQuery(model.update(id, col, value))  
 .then(() => ***console***.log('1 row updated!'));  
};

Функція видалення даних з таблиці

const deleteFlow = async () => {  
 const model = await pickModel();  
 const id = +(await rl.question(`Enter id: `));  
 return performQuery(model.delete(id))  
 .then(() => ***console***.log('1 row deleted!'));  
};

Функція пошуку даних з обраної таблиці

const findFlow = async () => {  
 const searches = [  
 'Search the bakers who has salary greater than X',  
 'Search the recipe which have the name like X',  
 'Search the owner who have more than X Bakeries'  
 ]  
 .map((f, i) => `${i + 1} - ${f}`)  
 .join('\n');  
  
 ***console***.log(`Pick the search to perform:\n${searches}`);  
  
 const searchID = await rl.question('value:');  
  
 if (searchID === '1') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT \*  
 FROM public."Baker"  
 WHERE salary > ${X}  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
  
 if (searchID === '2') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT \*  
 FROM public."Recipe"  
 WHERE name ILIKE '%${X}%'  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
  
 if (searchID === '3') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT id, name  
 FROM public."Owner" o  
 JOIN (  
 SELECT ownerid, count(\*)  
 FROM public."Bakery"  
 GROUP BY ownerid  
 ) g on o.id = g.ownerid  
 WHERE g.count > ${X}  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
};

Функція генерація випадкових даних

const getRandomflow = async () => {  
 if (count.equals("")) count = "20";  
 if (!count.matches("\\d+")) {  
 return "Incorret data ";  
 }  
 let sql = "";  
  
 switch ({sql}) {  
 case 1:  
 sql = "INSERT INTO public.baker (['id', 'bakeryid', 'name', 'salary']) VALUES ((select max(id) from baker) + 1, (select bakeryid from baker order by random() limit 1), substr(md5(random()::text), 0, 25), FLOOR(RANDOM() \* 100))";  
 break;  
 case 2:  
 sql = "INSERT INTO public.owner (['id', 'name']) VALUES ((select max(id) from owner) + 1, substr(md5(random()::text), 0, 25)";  
 break;  
 case 3:  
 sql = "INSERT INTO public.bakery (['id', 'name', 'address', 'ownerid']) VALUES ((select max(id) from bakery) + 1, substr(md5(random()::text), 0, 25), substr(md5(random()::text), 0, 25), (select ownerid from film bakery by random() limit 1)";  
 break;  
 case 4:  
 sql = "INSERT INTO public.product ['id', 'bakerid', 'recipeid']) VALUES ((select max(id) from product) + 1,(select bakerid from product order by random() limit 1),(select recipeid from product order by random() limit 1) + 1";  
 break;  
 case 5:  
 sql = "INSERT INTO public.recipe (['id', 'bakeryid', 'name', 'time']) VALUES ((select max(id) from recipe) + 1, (select bakeryid from recipe order by random() limit 1), substr(md5(random()::text), 0, 25), to\_timestamp(random()\*2147483647)::time)";  
 break;  
 default:  
 break;  
 }  
  
 const result = await performQuery(sql);  
 ***console***.table(result.rows);  
 return "Objects created!";  
}

**Код програми**

**app.js**

const readline = require('readline/promises');  
const PGClient = require('pg').Client;  
const Model = require('./BaseModel');  
const Controller = require('./BaseModel');  
const { stdin: input, stdout: output } = process;  
  
const rl = readline.createInterface({ input, output });  
  
const client = new PGClient({  
 user: 'postgres',  
 host: 'localhost',  
 password: '7387'  
});  
  
const COMMANDS = {  
 1: 'insert',  
 2: 'find',  
 3: 'update',  
 4: 'delete',  
 5: 'generate random data'  
};  
  
const commandsList = ***Object***.entries(COMMANDS).map(([key, value]) => `${key} - ${value}`).join('\n');  
  
// create models  
const models = {  
 Baker: new Model('Baker', ['id', 'bakeryid', 'name', 'salary']),  
 Bakery: new Model('Bakery', ['id', 'name', 'address', 'ownerid']),  
 Owner: new Model('Owner', ['id', 'name']),  
 Product: new Model('Product', ['id', 'bakerid', 'recipeid']),  
 Recipe: new Model('Recipe', ['id', 'bakeryid', 'name', 'time'])  
};  
  
// create controllers for each model  
const controllers = ***Object***.keys(models)  
 .map(modelName => new Controller(modelName));  
  
const pickModel = async () => {  
 ***console***.log(`Models:\n${***Object***.keys(models).map((m, i) => `${i + 1} - ${m}`).join('\n')}`);  
 const i = +(await rl.question(`Pick a model: `));  
 return ***Object***.values(models)[i - 1];  
};  
  
(async () => {  
 await client.connect();  
  
 const performQuery = async (query) => {  
 try {  
 return client.query(query);  
 } catch (e) {  
 ***console***.log(e.message);  
 }  
 };  
  
 const insertFlow = async () => {  
 const model = await pickModel();  
 ***console***.log(model.keys);  
  
 const entries = [];  
 for (const key of model.keys) {  
 const value = await rl.question(`${key}: `);  
 entries.push([key, value]);  
 }  
  
 const obj = ***Object***.fromEntries(entries);  
 return performQuery(model.insert(obj))  
 .then(() => ***console***.log('1 row inserted!'));  
 };  
  
 const formatEnteredValue = (val) => {  
 if (Number.isFinite(Number.parseInt(val))) return val;  
 return val;  
 };  
  
 const updateFlow = async () => {  
 const model = await pickModel();  
 const id = +(await rl.question(`Enter id: `));  
 ***console***.log(`Keys:\n${model.keys.map((key, i) => `${i} - ${key}`).join('\n')}`);  
 const i = +(await rl.question(`Pick a column: `));  
 const col = model.keys[i];  
 if (!col) return ***console***.info('Error: Wrong column!');  
 const value = formatEnteredValue(await rl.question('Value: '));  
 return performQuery(model.update(id, col, value))  
 .then(() => ***console***.log('1 row updated!'));  
 };  
  
 const deleteFlow = async () => {  
 const model = await pickModel();  
 const id = +(await rl.question(`Enter id: `));  
 return performQuery(model.delete(id))  
 .then(() => ***console***.log('1 row deleted!'));  
 };  
  
 const findFlow = async () => {  
 const searches = [  
 'Search the bakers who has salary greater than X',  
 'Search the recipe which have the name like X',  
 'Search the owner who have more than X Bakeries'  
 ]  
 .map((f, i) => `${i + 1} - ${f}`)  
 .join('\n');  
  
 ***console***.log(`Pick the search to perform:\n${searches}`);  
  
 const searchID = await rl.question('value:');  
  
 if (searchID === '1') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT \*  
 FROM public."Baker"  
 WHERE salary > ${X}  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
  
 if (searchID === '2') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT \*  
 FROM public."Recipe"  
 WHERE name ILIKE '%${X}%'  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
  
 if (searchID === '3') {  
 const X = await rl.question('X = ');  
 const query = `  
 SELECT id, name  
 FROM public."Owner" o  
 JOIN (  
 SELECT ownerid, count(\*)  
 FROM public."Bakery"  
 GROUP BY ownerid  
 ) g on o.id = g.ownerid  
 WHERE g.count > ${X}  
 `;  
 const result = await performQuery(query);  
 ***console***.table(result.rows);  
 }  
 };  
  
 const getRandomflow = async () => {  
 if (count.equals("")) count = "20";  
 if (!count.matches("\\d+")) {  
 return "Incorret data ";  
 }  
 let sql = "";  
  
 switch ({sql}) {  
 case 1:  
 sql = "INSERT INTO public.baker (['id', 'bakeryid', 'name', 'salary']) VALUES ((select max(id) from baker) + 1, (select bakeryid from baker order by random() limit 1), substr(md5(random()::text), 0, 25), FLOOR(RANDOM() \* 100))";  
 break;  
 case 2:  
 sql = "INSERT INTO public.owner (['id', 'name']) VALUES ((select max(id) from owner) + 1, substr(md5(random()::text), 0, 25)";  
 break;  
 case 3:  
 sql = "INSERT INTO public.bakery (['id', 'name', 'address', 'ownerid']) VALUES ((select max(id) from bakery) + 1, substr(md5(random()::text), 0, 25), substr(md5(random()::text), 0, 25), (select ownerid from film bakery by random() limit 1)";  
 break;  
 case 4:  
 sql = "INSERT INTO public.product ['id', 'bakerid', 'recipeid']) VALUES ((select max(id) from product) + 1,(select bakerid from product order by random() limit 1),(select recipeid from product order by random() limit 1) + 1";  
 break;  
 case 5:  
 sql = "INSERT INTO public.recipe (['id', 'bakeryid', 'name', 'time']) VALUES ((select max(id) from recipe) + 1, (select bakeryid from recipe order by random() limit 1), substr(md5(random()::text), 0, 25), to\_timestamp(random()\*2147483647)::time)";  
 break;  
 default:  
 break;  
 }  
  
 const result = await performQuery(sql);  
 ***console***.table(result.rows);  
 return "Objects created!";  
 }  
  
  
 while (true) {  
 ***console***.log(`COMMANDS LIST:\n${commandsList}`);  
 const command = +(await rl.question(`Choose a command: `));  
  
 if (command === 1) await insertFlow();  
 if (command === 2) await findFlow();  
 if (command === 3) await updateFlow();  
 if (command === 4) await deleteFlow();  
 if (command === 5) await getRandomflow();  
 }  
  
})()

**BaseController.js**

module.exports = class BaseController {  
 constructor (name, model) {  
 this.name = name;  
 this.model = model;  
 }  
  
 insert (data) {  
 this.model.insert(data);  
 }  
  
 update (id, key, value) {  
 this.model.update(id, key, value);  
 }  
  
 delete (id) {  
 this.model.delete(id);  
 }  
  
 findAll (id) {  
 this.model.findAll(id);  
 }  
};

**BaseModel.js**

module.exports = class BaseModel {  
 constructor (tableName, keys) {  
 this.tableName = tableName;  
 this.keys = keys;  
 }  
  
 findAll (id = null) {  
 return [  
 `SELECT \*  
 FROM public."${this.tableName}"`,  
 id ? `WHERE id = ${id}` : null  
 ].join('\n');  
 }  
  
 insert (data) {  
 const keys = ***Object***.keys(data);  
 const values = ***Object***.values(data).map(x => (typeof x) === 'string' ? `'${x}'` : x);  
  
 return [  
 `INSERT INTO public."${this.tableName}"(${keys.join(', ')})`,  
 `VALUES (${values.join(', ')})`  
 ].join('\n');  
 }  
  
 update (id, column, value) {  
 value = typeof value === 'string' ? `'${value}'` : value;  
  
 return [  
 `UPDATE public."${this.tableName}"`,  
 `SET ${column} = ${value}`,  
 `WHERE id = ${id}`  
 ].join('\n');  
 }  
  
 delete (id) {  
 return [  
 `DELETE  
 FROM public."${this.tableName}"`,  
 `WHERE id = ${id}`  
 ].join('\n');  
 }

**Посилання на GitHub:**

<https://github.com/mariabondarchuk/DB_lab_2>