

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “* *Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL ”*

Виконав:

студент ІII курсу

групи КВ-91

Сторожук К. В.

Перевірив:

Павловський В. І.

Київ – 2021

**Лабораторна робота №2**

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

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

Загальне завдання роботи полягає у наступному

1. Реалізувати функції 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. 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).

**Логічна модель предметної області «ЄДЕБО»**

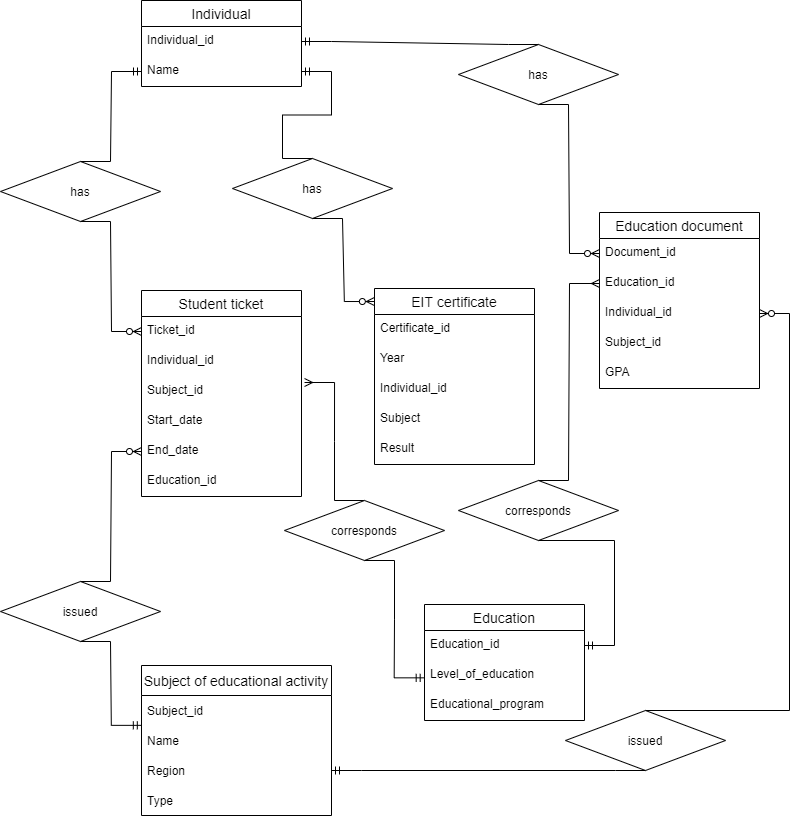


Рисунок 1 – логічна модель предметної області «ЄДЕБО»

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування С++, середовище розробки Visual Studio 2019, а також стороння бібліотека, що надає API для доступу до PostgreSQL – libpq++.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних та забезпечує взаємодію програми з базою даних.

View – представляє клас що дає можливість відобразити результати виконання програми в консольному інтерфейсі.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою за допомогою класів View та Model.

**Структура програми та її опис**

На рисунку 1 відображено деревовидну структуру програми:

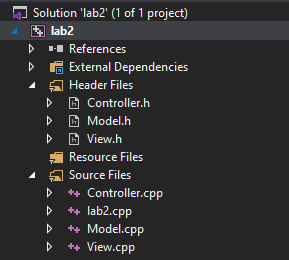


Рисунок 2 – структура програми

Програма поділена на 4 модулі: класс Model, клас Controller, клас View та головний файл lab2.cpp.

Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлах Model.h(.cpp) описаний клас моделі, що займається регулювання підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлах Controller.h(.cpp) описаний інтерфейс взаємодії з користувачем, реалізовано формування запиту та надсилання його в Model, формування результату та надсилання його в View.

У файлах View.h(.cpp) описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**



Рисунок 3 – Початкове меню програми

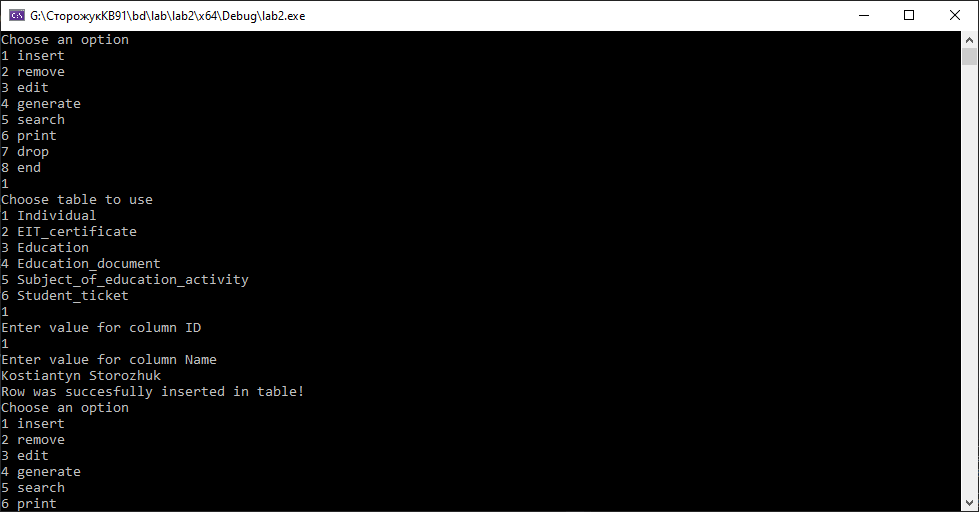


Рисунок 4 – Меню для додавання даних

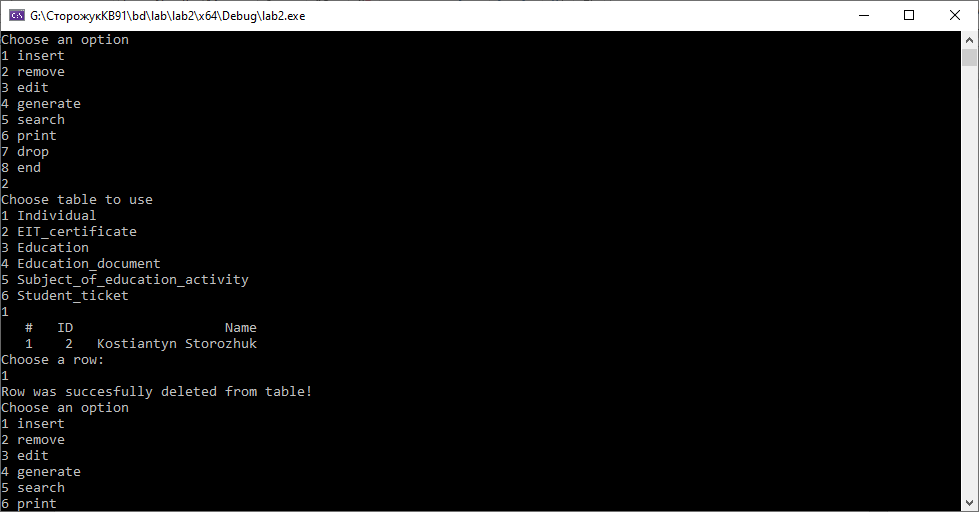


Рисунок 5 – Меню для вилучення даних

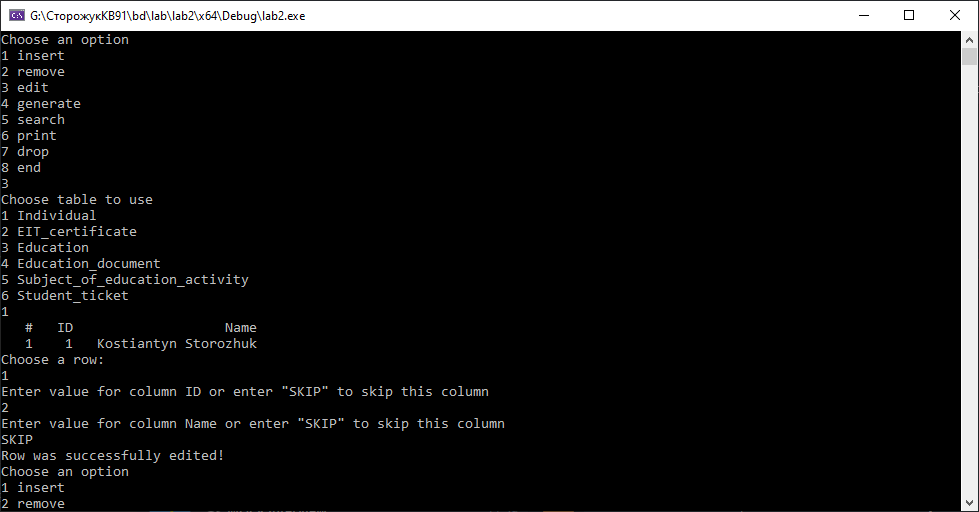


Рисунок 6 – Меню для редагування даних

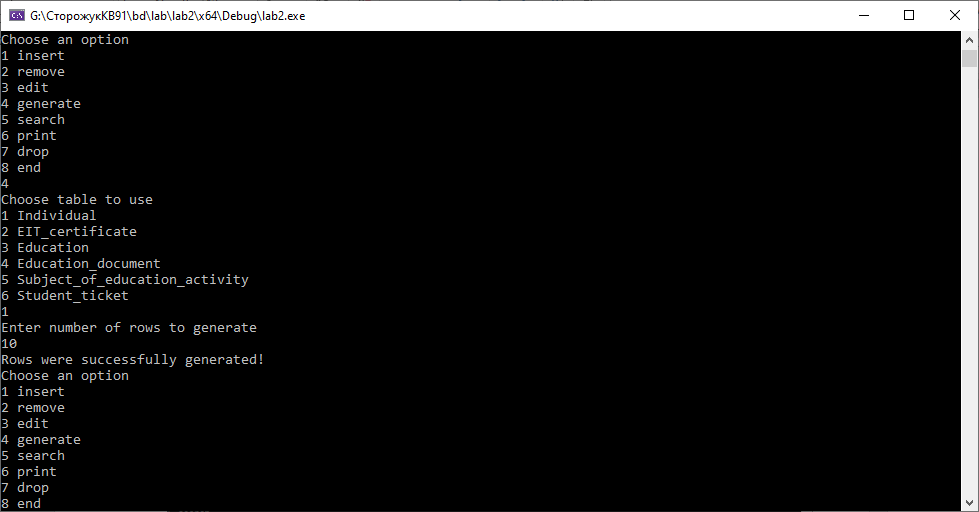


Рисунок 7 – Меню для заповнення даних випадковим чином

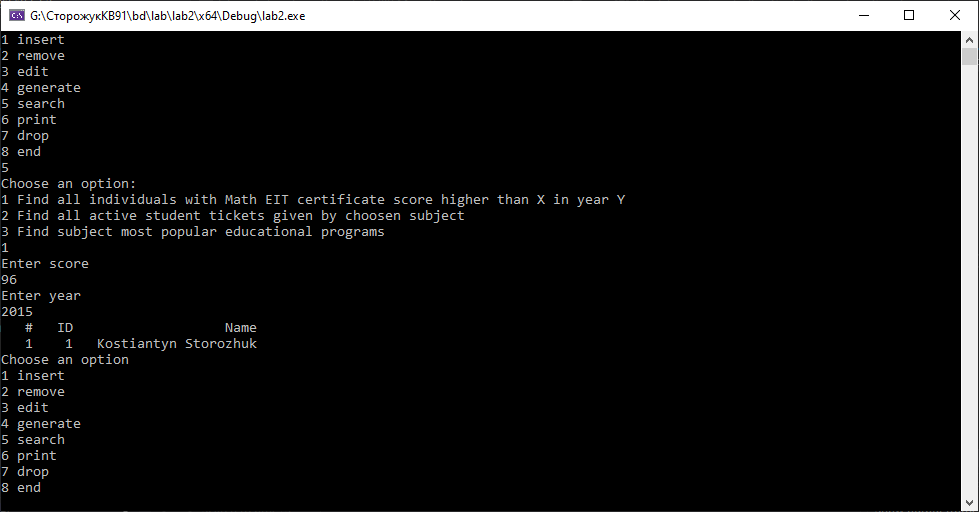


Рисунок 8 – Меню для пошуку даних

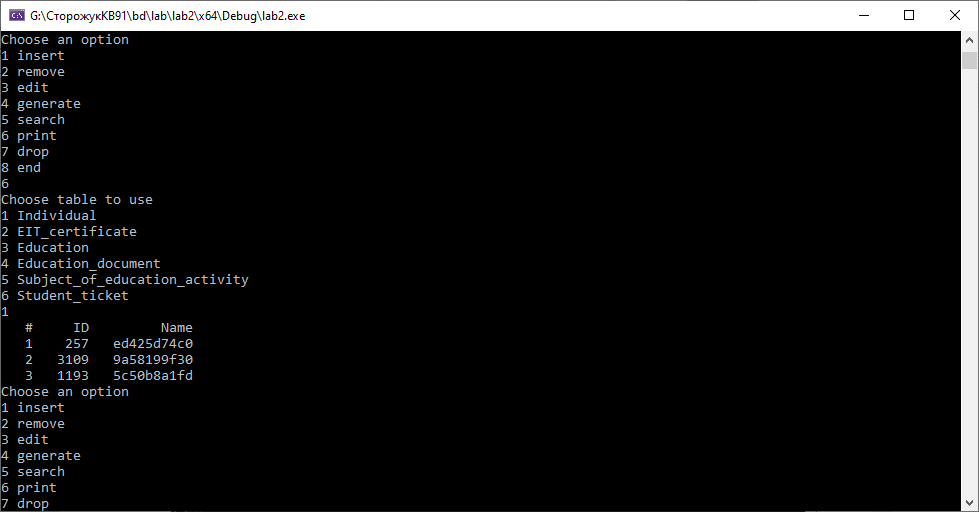


Рисунок 9 – Меню для виведення таблиці

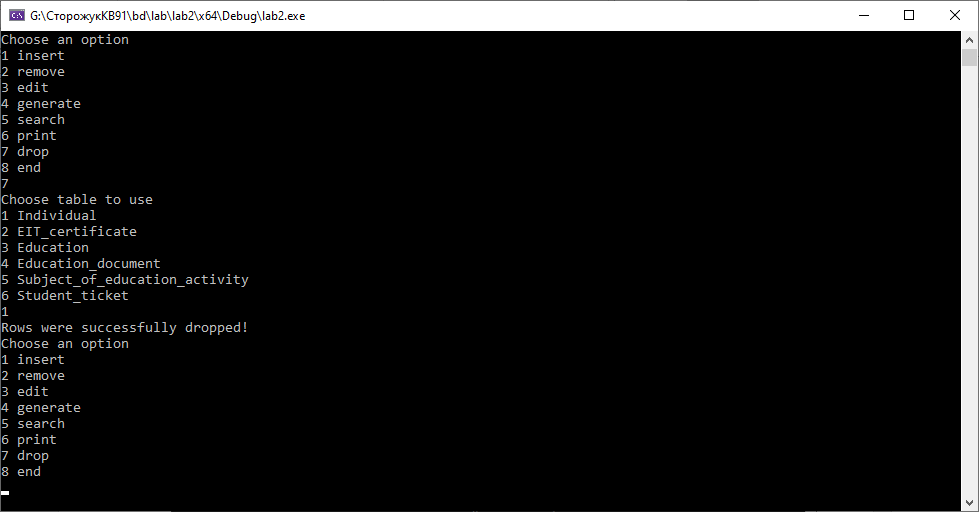


Рисунок 10 – Меню видалення всіх даних таблиці

Меню для користувача складається з восьми пунктів(Рисунок 3).

Перший пункт пропонує внесення даних (Рисунок 4). Перед тим як ввести дані необхідно ще обрати для якої таблиці саме буде відбуватись внесення. Тому користувач має обрати опцію. Після цього користувач може вносити дані.

Другий пункт пропонує видалення даних (Рисунок 5). Перед тим як видалити дані необхідно ще обрати для якої таблиці саме буде відбуватись видалення. Тому користувач має обрати опцію. Після цього користувач може видалити дані.

Третій пункт пропонує редагування даних (Рисунок 6). Перед тим як редагувати дані необхідно ще обрати для якої таблиці саме буде відбуватись редагування. Тому користувач має обрати опцію. Після цього користувач може редагувати дані.

Четвертий пункт пропонує заповнення даних випадковим чином (Рисунок 7). Перед тим як заповнити дані випадковим чином необхідно ще обрати для якої таблиці саме буде відбуватись заповнення. Тому користувач має обрати опцію.

П’ятий пункт пропонує пошук за атрибутами з декількох таблицю (Рисунок 8). Користувач має обрати, який запит він хоче виконувати, а тоді вже здійснювати відповідний пошук. Після введені необхідних атрибутів він побачить результати пошуку.

Шостий пункт пропонує виведення таблиці (Рисунок 9). Перед тим як редагувати дані необхідно ще обрати яка таблиця буде виведена. Після цього відповідна таблиця буде надрукована.

Сьомий пункт пропонує видалення всіх даних з таблиці (Рисунок 10). Перед тим як видалити данні необхідно ще обрати для якої таблиці саме буде відбуватись видалення. Тому користувач має обрати опцію. Після цього дані будуть видалені.

Восьмий пункт завершує роботу програми.

**Фрагменти програми для додавання, редагування, пошуку, генерації та відображення даних з бази даних**

Фрагмент програми для внесення даних

PGresult\* Controller::performInsert(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Insert interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

string tmp;

getline(cin, tmp);

std::vector<string> new\_row;

for (auto& column : columns) {

new\_row.push\_back(getValue(column.first, column.second));

}

string query\_new\_row = "(";

for (int i = 0; i < (int)(new\_row.size()) - 1; i++)

query\_new\_row += new\_row[i] + ", ";

if (new\_row.size())

query\_new\_row += new\_row[new\_row.size() - 1];

query\_new\_row += ")";

string query\_insert = "INSERT INTO \"" + table.second + "\"\n" +

"VALUES " + query\_new\_row + "\n";

auto result = model->query(query\_insert);

if (to\_print)

view->printInsert(result, model->GetLastError());

return result;

}

Фрагмент програми для видалення даних

PGresult\* Controller::performRemove(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Remove interapted" << endl;

return nullptr;

}

int row\_index = chooseRow(table);

if (row\_index == -1) {

cout << "Remove interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto prime\_key = model->getTablePrimaryKey(table.first);

int prime\_key\_index = columns.size();

for (int i = 0; i < columns.size(); i++)

if (columns[i].first == prime\_key) {

prime\_key\_index = i;

break;

}

string k;

string query\_remove = "DELETE FROM public.\"" + table.second + "\"\n"

"WHERE \"" + columns[prime\_key\_index].first + "\" = " + rows[row\_index][prime\_key\_index];

auto result = model->query(query\_remove);

if (to\_print)

view->printRemove(result, model->GetLastError());

return result;

}

Фрагмент програми для редагування даних

PGresult\* Controller::performEdit(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Edit interapted" << endl;

return nullptr;

}

int row\_index = chooseRow(table);

if (row\_index == -1) {

cout << "Edit interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto prime\_key = model->getTablePrimaryKey(table.first);

int prime\_key\_index = columns.size();

for(int i = 0; i < columns.size(); i++)

if (columns[i].first == prime\_key) {

prime\_key\_index = i;

break;

}

string query\_edit = "UPDATE public.\"" + table.second + "\"\n";

query\_edit += "SET ";

string tmp;

getline(cin, tmp);

bool first = true;

for (int i = 0; i < columns.size(); i++) {

string val = getValue(columns[i].first, columns[i].second, true);

if (val != "SKIP\n\n") {

if (!first)

query\_edit += ", ";

first = false;

query\_edit += "\"" + columns[i].first + "\" = " + val;

}

}

if (first) {

cout << "None of the column was modified" << endl;

return nullptr;

}

query\_edit += "\nWHERE \"" + columns[prime\_key\_index].first + "\" = " + rows[row\_index][prime\_key\_index];

auto result = model->query(query\_edit);

if (to\_print)

view->printEdit(result, model->GetLastError());

return result;

}

Фрагмент програми для генерації даних

PGresult\* Controller::performGenerate(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Generate interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto foreign\_keys = model->getTableForeignKeys(table.first);

auto generatingStringFromType = [&](model::Model::dataTypes dt) -> string {

switch (dt) {

case model::Model::dataTypes::integer:

return "(RANDOM() \* 10000)::integer";

case model::Model::dataTypes::numeric:

return "(RANDOM() \* 100)::numeric";

case model::Model::dataTypes::text:

return "SUBSTR(MD5(RANDOM()::text), 3, 10)";

case model::Model::dataTypes::date:

return "timestamp '2000-01-10 20:00:00' + random() \* (timestamp '2030-01-10 20:00:00' - timestamp '2000-01-10 20:00:00')";

case model::Model::dataTypes::EIT\_subject:

return "(enum\_range(NULL::\"EIT\_subject\"))[1 + FLOOR((RANDOM() \* 12))::integer]";

case model::Model::dataTypes::Education\_level:

return "(enum\_range(NULL::\"Education\_level\"))[1 + FLOOR((RANDOM() \* 5))::integer]";

case model::Model::dataTypes::Type\_of\_subject\_of\_educational\_activity:

return "(enum\_range(NULL::\"Type\_of\_subject\_of\_educational\_activity\"))[1 + FLOOR((RANDOM() \* 3))::integer]";

case model::Model::dataTypes::Ukraine\_region:

return "(enum\_range(NULL::\"Ukraine\_region\"))[1 + FLOOR((RANDOM() \* 25))::integer]";

default:

return "";

}

};

std::vector<string> generating\_strings;

for (auto& column : columns) {

if (!foreign\_keys.count(column.first))

generating\_strings.push\_back(generatingStringFromType(column.second));

else

generating\_strings.push\_back(

"(ARRAY ((SELECT \"" + foreign\_keys[column.first].second + "\" FROM public.\"" + foreign\_keys[column.first].first + "\")))[1 + FLOOR((RANDOM() \* "

"(SELECT COUNT(\"" + foreign\_keys[column.first].second + "\") FROM public.\"" + foreign\_keys[column.first].first + "\")"

"))::integer]"

);

}

string generate\_query = "INSERT INTO public.\"" + table.second + "\"\nSelect";

for (int i = 0; i < generating\_strings.size(); i++) {

generate\_query += "\n" + generating\_strings[i];

if(i != generating\_strings.size() - 1)

generate\_query += ",";

}

cout << "Enter number of rows to generate" << endl;

int number;

cin >> number;

generate\_query += "\nFROM generate\_series(1," + std::to\_string(number) + ")\nON CONFLICT DO NOTHING";

auto result = model->query(generate\_query);

if (to\_print)

view->printGenerate(result, model->GetLastError());

return result;

}

Фрагмент програми для пошуку даних

PGresult\* Controller::performSearch(bool to\_print) {

cout << "Choose an option:" << endl;

cout << "1 Find all individuals with Math EIT certificate score higher than X in year Y" << endl;

cout << "2 Find all active student tickets given by choosen subject" << endl;

cout << "3 Find subject most popular educational programs" << endl;

int input = -1;

while (input == -1) {

cin >> input;

if (input > 3 || input < 1) {

input = -1;

cout << "Invalid option. Choose an option from the list above" << endl;

}

}

int x, y;

string s;

string search\_query;

switch (input)

{

case 1:

cout << "Enter score" << endl;

cin >> x;

cout << "Enter year" << endl;

cin >> y;

search\_query =

"SELECT \* \n"

"FROM public.\"Individual\"\n"

"WHERE \"ID\" IN (\n"

"SELECT \"Individual\_ID\" FROM public.\"EIT\_certificate\"\n"

"WHERE \"Subject\" = 'Math' and \"Result\" >= " + std::to\_string(x) + " and \"Year\" = " + std::to\_string(y) + ")\n"

"ORDER BY \"ID\"\n"

;

break;

case 2:

cout << "Enter subject name" << endl;

getline(cin, s);

getline(cin, s);

search\_query =

"SELECT \* FROM public.\"Student\_ticket\""

"WHERE \"Start\_date\" >= current\_date and \"End\_date\" <= current\_date and \"Subject\_ID\" IN ("

"SELECT \"ID\" FROM public.\"Subject\_of\_education\_activity\""

"WHERE \"Name\" = '" + s + "')"

"ORDER BY \"ID\"\n"

;

break;

case 3:

cout << "Enter subject name" << endl;

getline(cin, s);

getline(cin, s);

cout << "Enter number of programs" << endl;

cin >> x;

search\_query =

"SELECT \"ID\", \"Level\_of\_education\", \"Education\_program\", \"count\" FROM\n"

"public.\"Education\"\n"

"INNER JOIN(\n"

"SELECT \"Education\_ID\", COUNT(\*) FROM(\n"

"SELECT \"Education\_ID\" FROM\n"

"(SELECT \"Subject\_ID\", \"Education\_ID\" FROM public.\"Student\_ticket\"\n"

"UNION ALL\n"

"SELECT \"Subject\_ID\", \"Education\_ID\" FROM public.\"Education\_document\") AS combined\_table\n"

"WHERE \"Subject\_ID\" IN(\n"

"SELECT \"ID\" FROM public.\"Subject\_of\_education\_activity\"\n"

"WHERE \"Name\" = '" + s +"')\n"

") AS countable\_table\n"

"GROUP BY \"Education\_ID\"\n"

"ORDER BY COUNT(\*) DESC\n"

"LIMIT " + std::to\_string(x) + ") as count\_table\n"

"ON \"ID\" = \"Education\_ID\"\n"

"ORDER BY \"count\" DESC";

break;

default:

break;

}

auto start = std::chrono::steady\_clock::now();

auto result = model->query(search\_query);

auto end = std::chrono::steady\_clock::now();

if (to\_print)

view->printSearch(result, model->GetLastError());

cout << "Time spent on search: " << std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() << " ms " << endl;

return result;

}

Фрагмент програми для відображення даних

PGresult\* Controller::performPrint(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Generate interapted" << endl;

return nullptr;

}

auto rows = model->getTableRows(table.first);

int input = -1;

if (rows.size() < 100)

input = 1;

else {

cout << "Choose which rows to display" << endl;

cout << "1 All rows" << endl;

cout << "2 First 100 rows" << endl;

cout << "3 Last 100 rows" << endl;

while (input == -1) {

cin >> input;

if (input > 3 || input < 1) {

input = -1;

cout << "Invalid option. Choose an option from the list above" << endl;

}

}

}

int left = 0;

int right = rows.size();

if (input == 2)

right = std::min(100, (int)(rows.size()));

if (input == 3)

left = std::max(0, (int)(rows.size()) - 100);

auto result = model->query("SELECT \* FROM public.\"" + table.second + "\"");

if(to\_print)

view->showTable(result, left, right, true);

return result;

}

Наведені вище фрагменти програми відповідають за функціонал додавання, редагування, пошуку, генерації та відображення даних з бази даних.

Взаємодія відбувається через клас Model, який займається підключенням до БД. Самі функції знаходяться в класі Controller. Спочатку користувач вводить відповідні атрибути за допомогою яких генерується SQL-запит, а потім запит виконується за допомогою Model::query().

**Результати роботи програми**

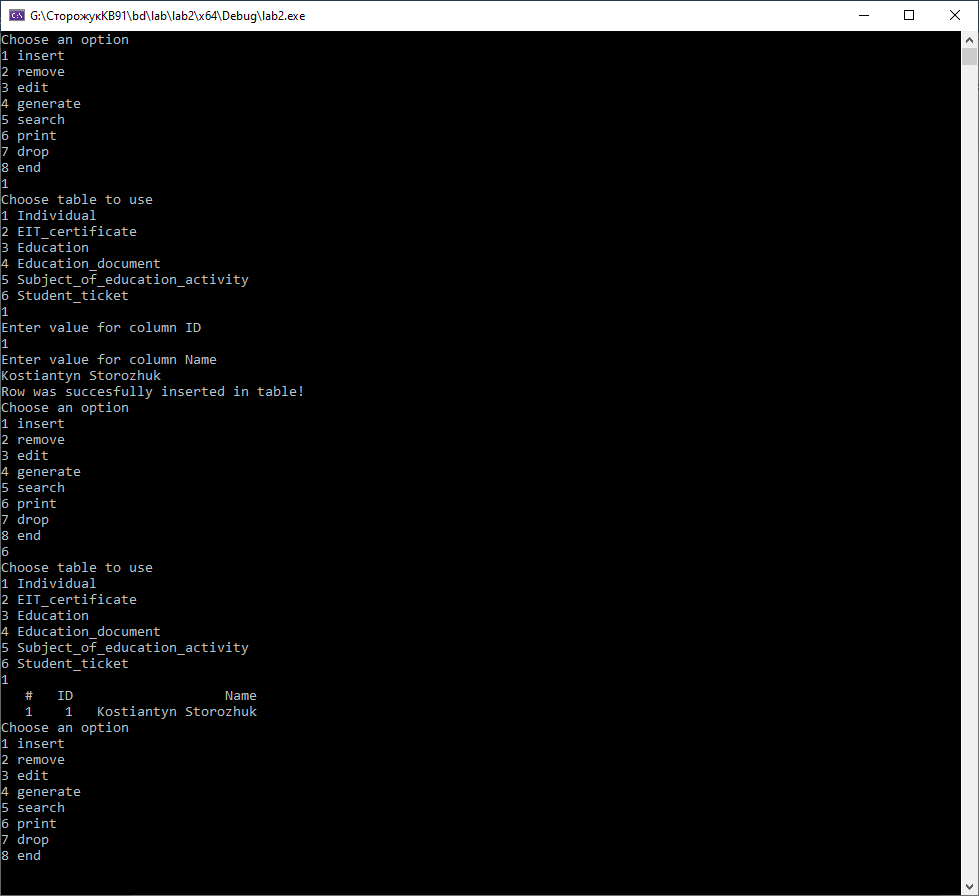


Рисунок 11 – результат додавання даних

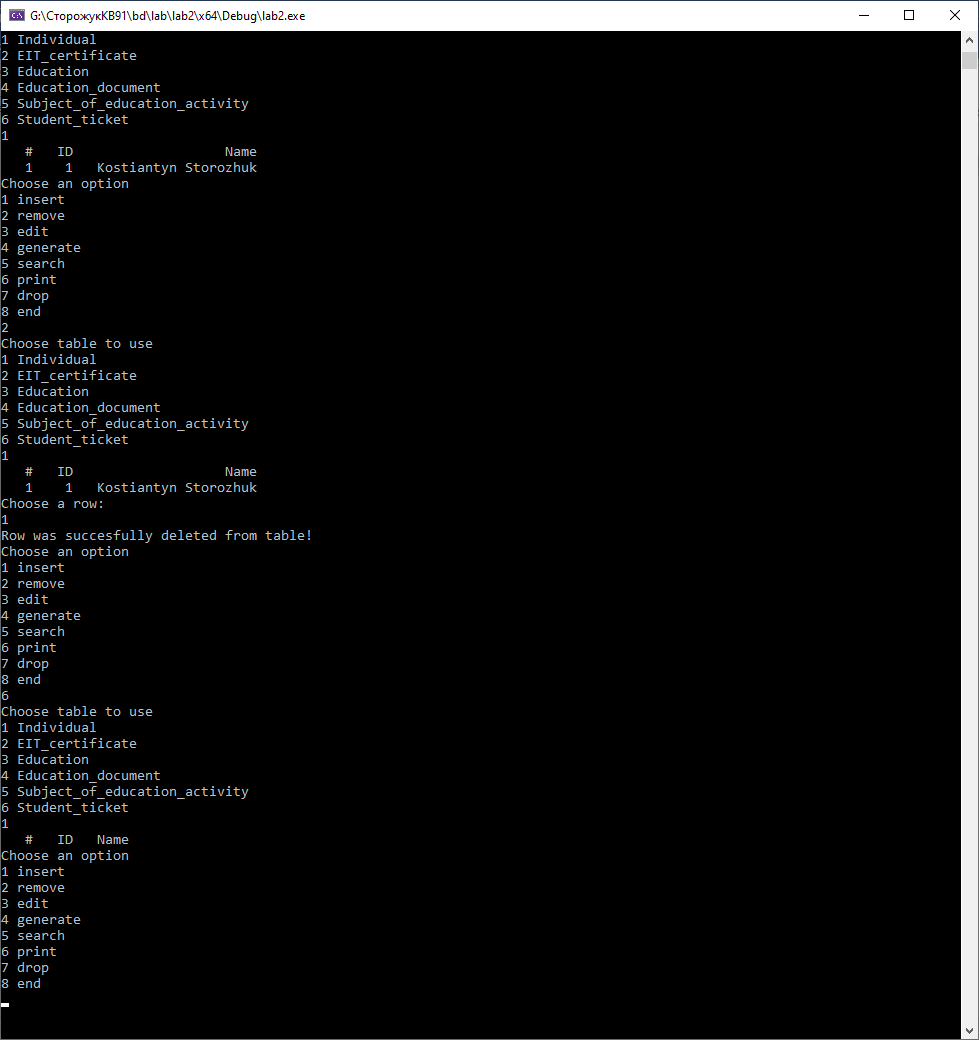


Рисунок 12 – результат вилучення даних

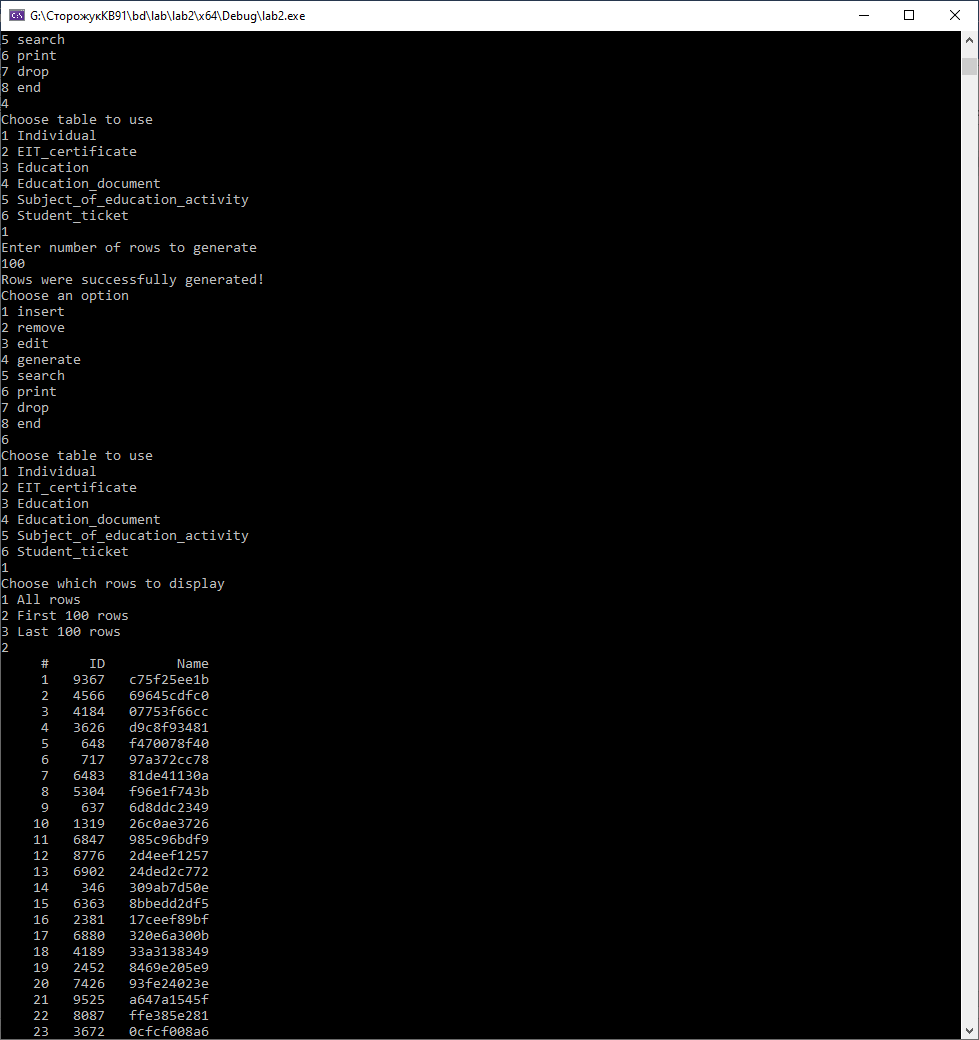


Рисунок 13 – результат генерації даних

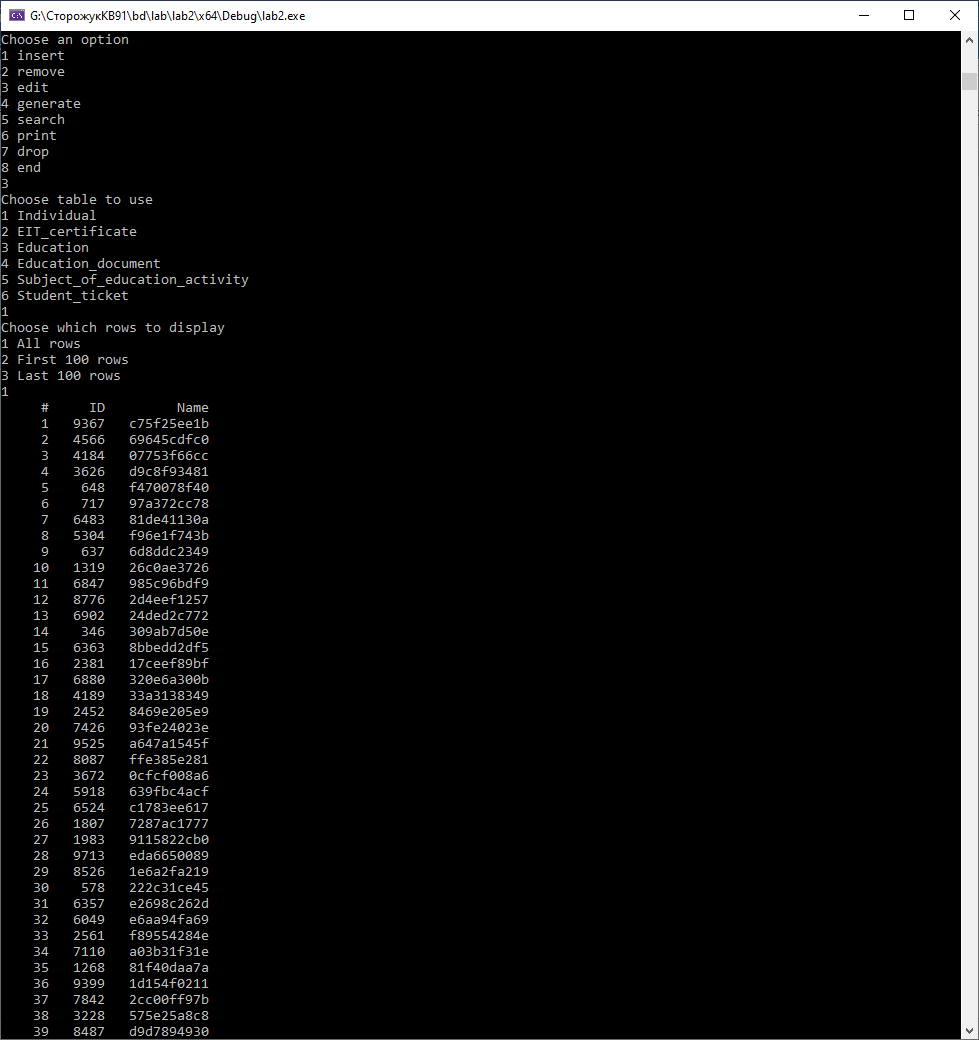


Рисунок 14 – результат редагування даних (1)

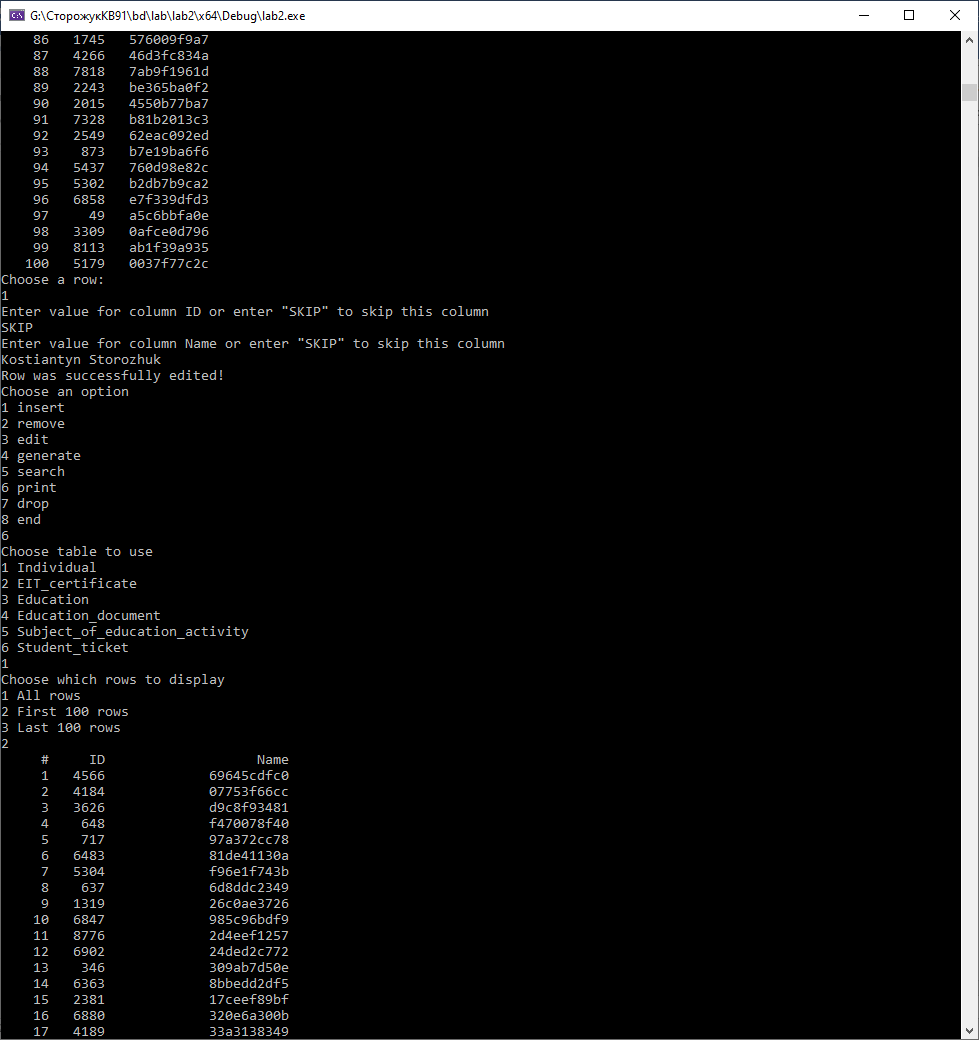


Рисунок 15 – результат редагування даних (2)

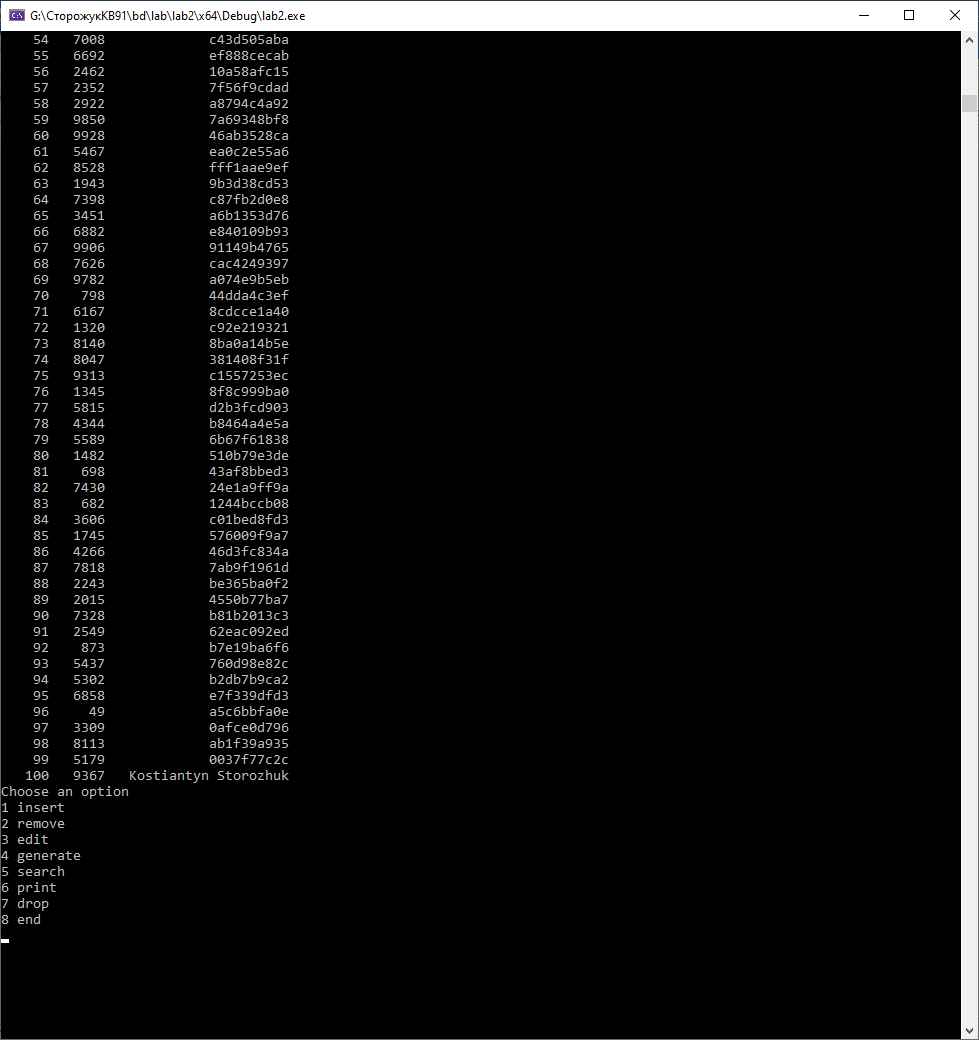


Рисунок 16 – результат редагування даних (3)

Далі для демонстрації пошуку було згенеровано по 100000 записів для таблиць Individual, EIT\_certificate, Education\_document, Student\_ticket, та 100 записів для таблиць Education, Subject\_of\_education\_activity. Далі один запис з таблиці Subject\_of\_education\_activity було переіменовано в “KPI” для зручності.

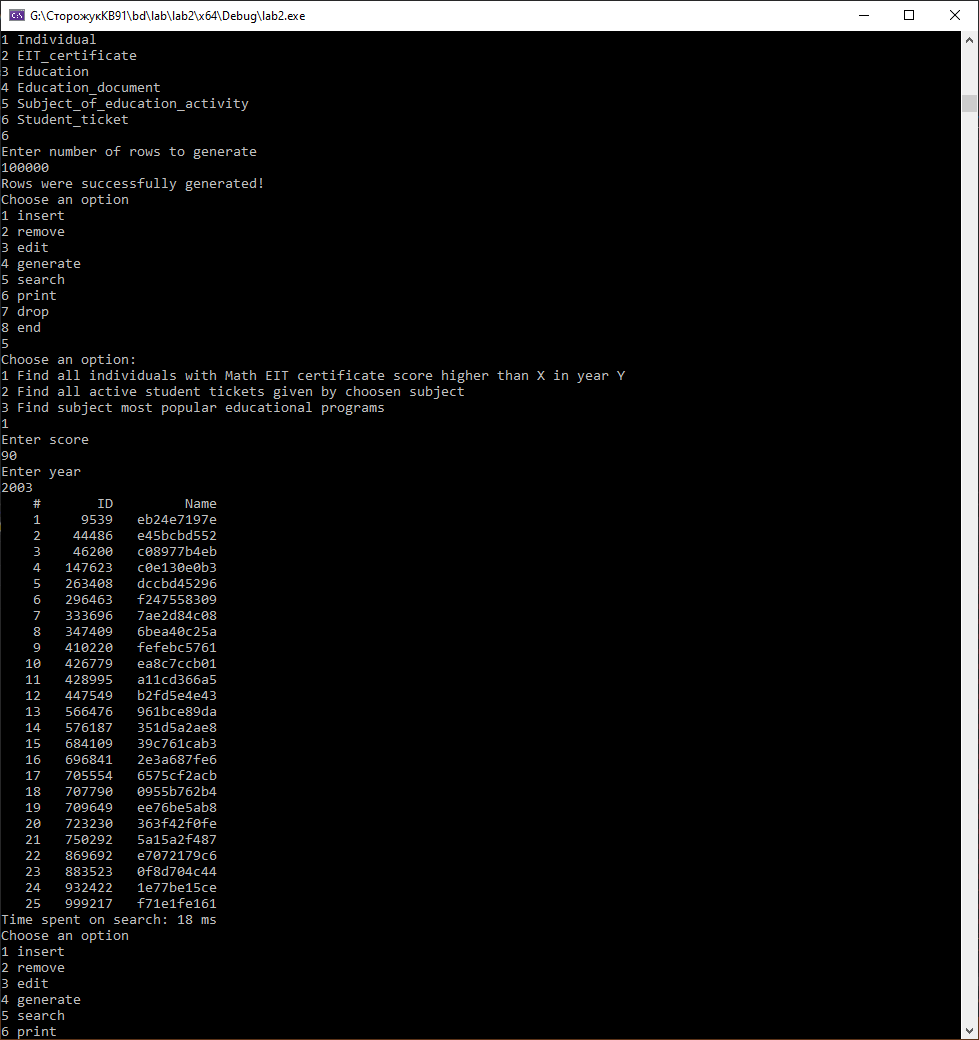


Рисунок 17 – результат пошуку осіб що здали ЗНО з математики у 2003 році на оцінку вищу або рівну за 90 балів

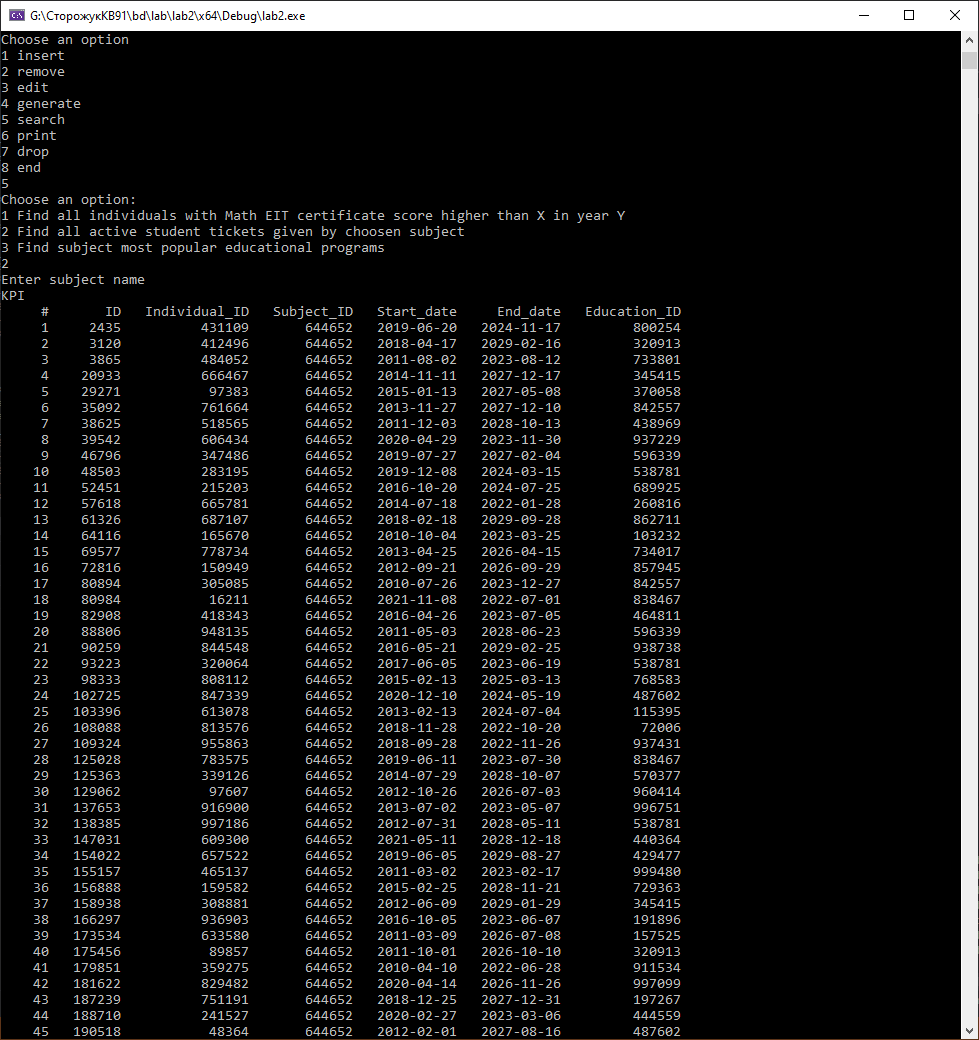


Рисунок 18 – результат пошуку всіх активних студентських виданих “KPI”

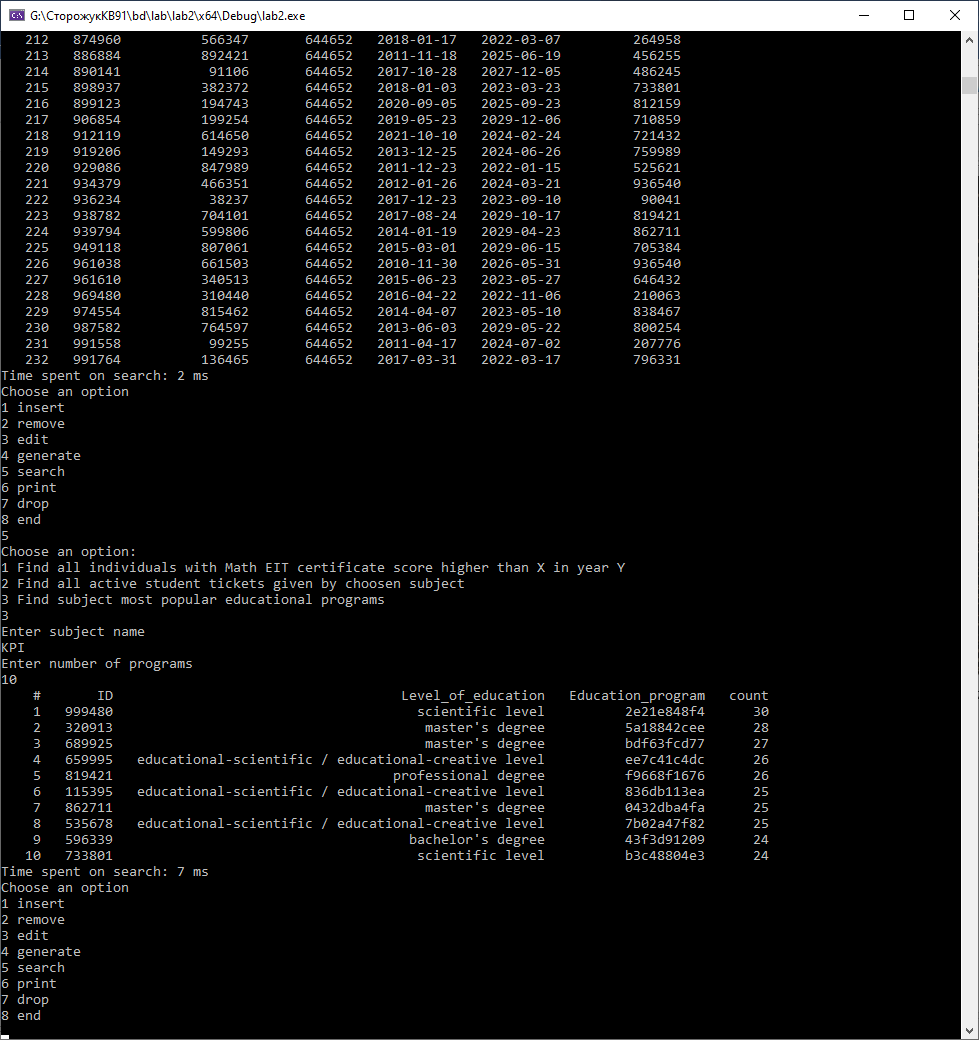


Рисунок 19 – пошук 10 найпопулярніших освітніх програм в “KPI” (сумарно між виданими документами про освіту та студентськими квитками)

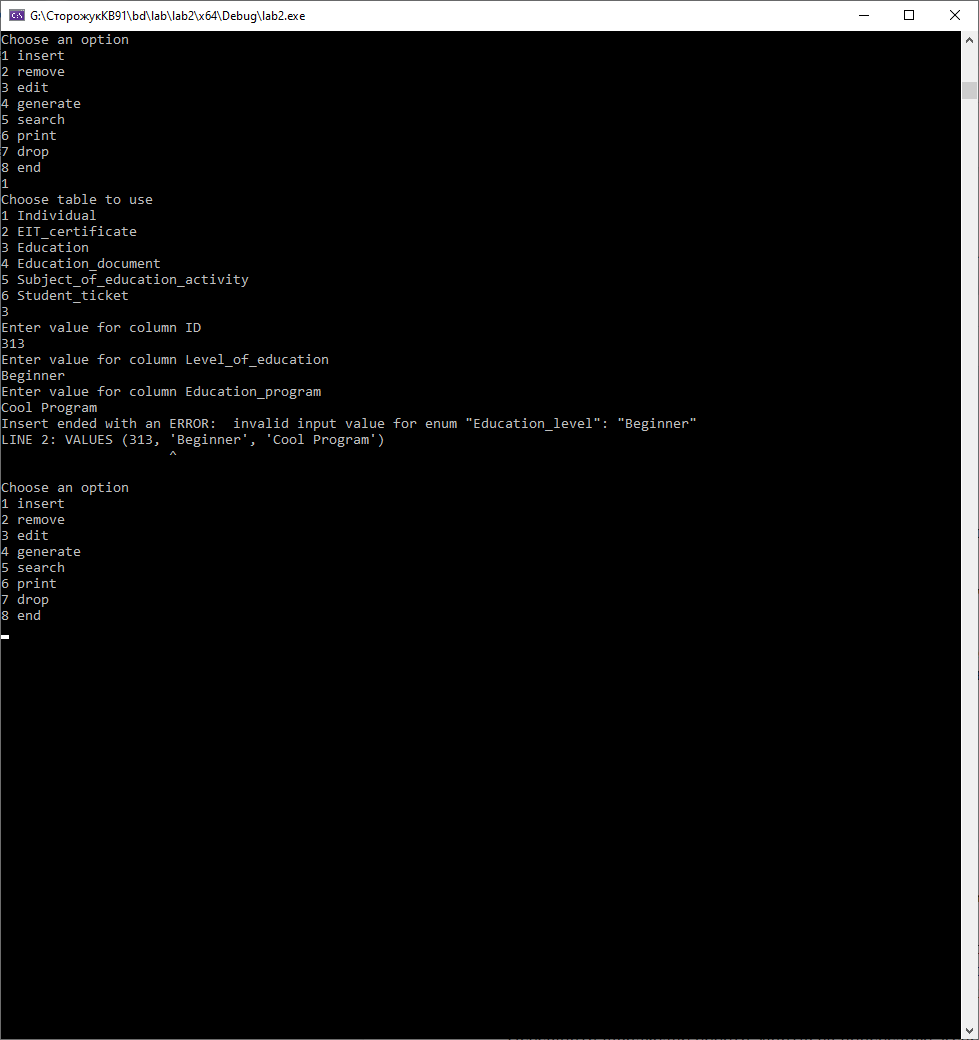
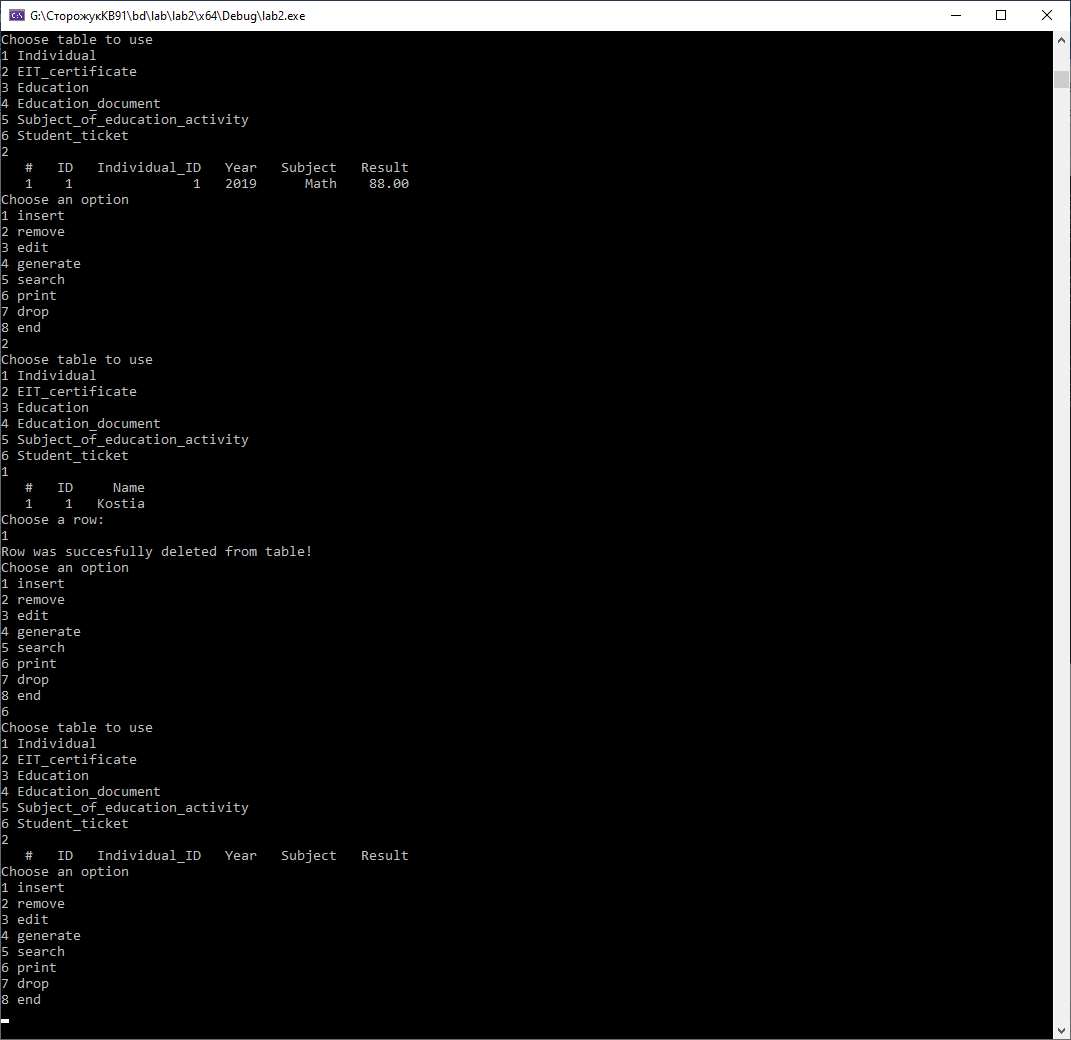


Рисунок 20 – приклад обробки неправильних вхідних даних

 Рисунок 21 – приклад видалення зв’язаних даних

**Копії SQL-запитів, що ілюструють пошук з зазначеними початковими параметрами**

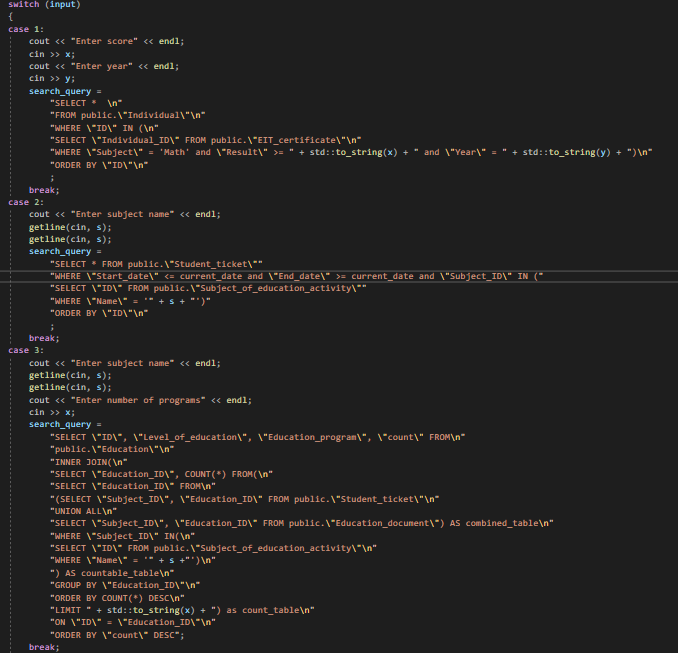


Рисунок 22 – формування запитів пошуку

**Програмний код проекту (GIT)**

*Controller.h*

#pragma once

#include <map>

#include <string>

#include "libpq-fe.h"

#include "Model.h"

#include "View.h"

namespace controller {

using std::string;

using std::map;

class Controller

{

private:

enum class action : int {

insert = 1,

remove,

edit,

generate,

search,

print,

drop,

end

};

using action\_function = PGresult \* (Controller::\*)(bool to\_print);

std::map<int, std::pair<string, action\_function>> action\_map;

public:

Controller(std::shared\_ptr<model::Model> \_model, std::shared\_ptr<view::View> \_view);

void interactionLoop();

private:

std::string getValue(string& column\_name, model::Model::dataTypes type, bool skippable = false);

std::pair<int, string> chooseTable();

int chooseRow(std::pair<int, string> table);

PGresult\* performInsert(bool to\_print = true);

PGresult\* performRemove(bool to\_print = true);

PGresult\* performEdit(bool to\_print = true);

PGresult\* performGenerate(bool to\_print = true);

PGresult\* performSearch(bool to\_print = true);

PGresult\* performPrint(bool to\_print = true);

PGresult\* performDrop(bool to\_print = true);

std::shared\_ptr<model::Model> model;

std::shared\_ptr<view::View> view;

};

}

*Controller.cpp*

#include "Controller.h"

#include <chrono>

#include <iostream>

#include <vector>

using namespace controller;

using std::cin;

using std::cout;

using std::endl;

Controller::Controller(std::shared\_ptr<model::Model> \_model, std::shared\_ptr<view::View> \_view) : model(\_model), view(\_view) {

action\_map[static\_cast<int>(action::insert)] = { "insert", &Controller::performInsert };

action\_map[static\_cast<int>(action::remove)] = { "remove", &Controller::performRemove };

action\_map[static\_cast<int>(action::edit)] = { "edit", &Controller::performEdit };

action\_map[static\_cast<int>(action::generate)] = { "generate", &Controller::performGenerate };

action\_map[static\_cast<int>(action::search)] = { "search", &Controller::performSearch };

action\_map[static\_cast<int>(action::print)] = { "print", &Controller::performPrint };

action\_map[static\_cast<int>(action::drop)] = { "drop", &Controller::performDrop };

action\_map[static\_cast<int>(action::end)] = { "end", nullptr };

};

void Controller::interactionLoop() {

bool go\_on = true;

try {

while (go\_on) {

cout << "Choose an option" << endl;

for (auto& action : action\_map) {

cout << static\_cast<int>(action.first) << " " << action.second.first << endl;

}

int input;

cin >> input;

auto action = action\_map.find(input);

if (action == action\_map.end()) {

cout << "Invalid input! Please, input valid option" << endl;

continue;

}

if (input == static\_cast<int>(action::end)) {

go\_on = false;

cout << "Goodbye!" << endl;

}

else {

(this->\*(action->second.second))(true);

}

}

}

catch (const std::exception& e) {

cout << "Exception occured: " << e.what() << endl;

}

}

std::pair<int, string> Controller::chooseTable() {

auto tables = model->getTables();

if (tables.empty()) {

cout << "There are no tables, nothing to choose from" << endl;

return { -1, "" };

}

cout << "Choose table to use" << endl;

for (int i = 0; i < tables.size(); ++i)

cout << i + 1 << " " << tables[i] << endl;

int input = -1;

while (input == -1) {

cin >> input;

if (input > tables.size() || input < 0) {

input = -1;

cout << "Invalid table. Choose a table from the list above or enter 0 to break" << endl;

}

}

return { input - 1, input?tables[input - 1]:"" };

}

int Controller::chooseRow(std::pair<int, string> table) {

auto rows = model->getTableRows(table.first);

auto cols = model->getTableColumns(table.first);

if (rows.empty()) {

cout << "There are no rows, nothing to choose from" << endl;

return -1;

}

int input = -1;

if (rows.size() < 100)

input = 1;

else {

cout << "Choose which rows to display" << endl;

cout << "1 All rows" << endl;

cout << "2 First 100 rows" << endl;

cout << "3 Last 100 rows" << endl;

while (input == -1) {

cin >> input;

if (input > 3 || input < 1) {

input = -1;

cout << "Invalid option. Choose an option from the list above" << endl;

}

}

}

int left = 0;

int right = rows.size();

if(input == 2)

right = std::min(100, (int)(rows.size()));

if(input == 3)

left = std::max(0, (int)(rows.size()) - 100);

auto result = model->query("SELECT \* FROM public.\"" + table.second + "\"");

view->showTable(result, left, right, true);

cout << "Choose a row:" << endl;

input = -1;

while (input == -1) {

std::cin >> input;

if (input > rows.size() || input < 0) {

input = -1;

cout << "Invalid row. Choose a row from the list above or enter 0 to break";

}

}

return input - 1;

}

std::string Controller::getValue(string& column\_name, model::Model::dataTypes type, bool skippable) {

cout << "Enter value for column ";

cout << column\_name;

if (skippable)

cout << " or enter \"SKIP\" to skip this column";

cout << endl;

string value = "";

string add = "";

if (type == model::Model::dataTypes::text ||

type == model::Model::dataTypes::EIT\_subject ||

type == model::Model::dataTypes::Type\_of\_subject\_of\_educational\_activity ||

type == model::Model::dataTypes::Ukraine\_region ||

type == model::Model::dataTypes::Education\_level)

{

add += "'";

}

getline(cin, value);

if (skippable && value == "SKIP")

return "SKIP\n\n";

return add + value + add;

}

PGresult\* Controller::performInsert(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Insert interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

string tmp;

getline(cin, tmp);

std::vector<string> new\_row;

for (auto& column : columns) {

new\_row.push\_back(getValue(column.first, column.second));

}

string query\_new\_row = "(";

for (int i = 0; i < (int)(new\_row.size()) - 1; i++)

query\_new\_row += new\_row[i] + ", ";

if (new\_row.size())

query\_new\_row += new\_row[new\_row.size() - 1];

query\_new\_row += ")";

string query\_insert = "INSERT INTO \"" + table.second + "\"\n" +

"VALUES " + query\_new\_row + "\n";

auto result = model->query(query\_insert);

if (to\_print)

view->printInsert(result, model->GetLastError());

return result;

}

PGresult\* Controller::performRemove(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Remove interapted" << endl;

return nullptr;

}

int row\_index = chooseRow(table);

if (row\_index == -1) {

cout << "Remove interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto prime\_key = model->getTablePrimaryKey(table.first);

int prime\_key\_index = columns.size();

for (int i = 0; i < columns.size(); i++)

if (columns[i].first == prime\_key) {

prime\_key\_index = i;

break;

}

string k;

string query\_remove = "DELETE FROM public.\"" + table.second + "\"\n"

"WHERE \"" + columns[prime\_key\_index].first + "\" = " + rows[row\_index][prime\_key\_index];

auto result = model->query(query\_remove);

if (to\_print)

view->printRemove(result, model->GetLastError());

return result;

}

PGresult\* Controller::performEdit(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Edit interapted" << endl;

return nullptr;

}

int row\_index = chooseRow(table);

if (row\_index == -1) {

cout << "Edit interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto prime\_key = model->getTablePrimaryKey(table.first);

int prime\_key\_index = columns.size();

for(int i = 0; i < columns.size(); i++)

if (columns[i].first == prime\_key) {

prime\_key\_index = i;

break;

}

string query\_edit = "UPDATE public.\"" + table.second + "\"\n";

query\_edit += "SET ";

string tmp;

getline(cin, tmp);

bool first = true;

for (int i = 0; i < columns.size(); i++) {

string val = getValue(columns[i].first, columns[i].second, true);

if (val != "SKIP\n\n") {

if (!first)

query\_edit += ", ";

first = false;

query\_edit += "\"" + columns[i].first + "\" = " + val;

}

}

if (first) {

cout << "None of the column was modified" << endl;

return nullptr;

}

query\_edit += "\nWHERE \"" + columns[prime\_key\_index].first + "\" = " + rows[row\_index][prime\_key\_index];

auto result = model->query(query\_edit);

if (to\_print)

view->printEdit(result, model->GetLastError());

return result;

}

PGresult\* Controller::performGenerate(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Generate interapted" << endl;

return nullptr;

}

auto columns = model->getTableColumns(table.first);

auto rows = model->getTableRows(table.first);

auto foreign\_keys = model->getTableForeignKeys(table.first);

auto prime\_key = model->getTablePrimaryKey(table.first);

auto generatingStringFromType = [&](model::Model::dataTypes dt) -> string {

switch (dt) {

case model::Model::dataTypes::integer:

return "(RANDOM() \* 40)::integer + 1990";

case model::Model::dataTypes::numeric:

return "(RANDOM() \* 100)::numeric";

case model::Model::dataTypes::text:

return "SUBSTR(MD5(RANDOM()::text), 3, 10)";

case model::Model::dataTypes::date:

return "timestamp '2010-01-10 20:00:00' + random() \* (timestamp '2030-01-10 20:00:00' - timestamp '2010-01-10 20:00:00')";

case model::Model::dataTypes::EIT\_subject:

return "(enum\_range(NULL::\"EIT\_subject\"))[1 + FLOOR((RANDOM() \* 12))::integer]";

case model::Model::dataTypes::Education\_level:

return "(enum\_range(NULL::\"Education\_level\"))[1 + FLOOR((RANDOM() \* 5))::integer]";

case model::Model::dataTypes::Type\_of\_subject\_of\_educational\_activity:

return "(enum\_range(NULL::\"Type\_of\_subject\_of\_educational\_activity\"))[1 + FLOOR((RANDOM() \* 3))::integer]";

case model::Model::dataTypes::Ukraine\_region:

return "(enum\_range(NULL::\"Ukraine\_region\"))[1 + FLOOR((RANDOM() \* 25))::integer]";

default:

return "";

}

};

std::vector<string> generating\_strings;

for (auto& column : columns) {

if (foreign\_keys.count(column.first))

generating\_strings.push\_back(

"(ARRAY ((SELECT \"" + foreign\_keys[column.first].second + "\" FROM public.\"" + foreign\_keys[column.first].first + "\")))[1 + FLOOR((RANDOM() \* "

"(SELECT COUNT(\"" + foreign\_keys[column.first].second + "\") FROM public.\"" + foreign\_keys[column.first].first + "\")"

"))::integer]"

);

else if (prime\_key == column.first)

generating\_strings.push\_back("(RANDOM() \* 1000000)::integer");

else

generating\_strings.push\_back(generatingStringFromType(column.second));

}

string generate\_query = "INSERT INTO public.\"" + table.second + "\"\nSelect";

for (int i = 0; i < generating\_strings.size(); i++) {

generate\_query += "\n" + generating\_strings[i];

if(i != generating\_strings.size() - 1)

generate\_query += ",";

}

cout << "Enter number of rows to generate" << endl;

int number;

cin >> number;

generate\_query += "\nFROM generate\_series(1," + std::to\_string(number) + ")\nON CONFLICT DO NOTHING";

auto result = model->query(generate\_query);

if (to\_print)

view->printGenerate(result, model->GetLastError());

return result;

}

PGresult\* Controller::performSearch(bool to\_print) {

cout << "Choose an option:" << endl;

cout << "1 Find all individuals with Math EIT certificate score higher than X in year Y" << endl;

cout << "2 Find all active student tickets given by choosen subject" << endl;

cout << "3 Find subject most popular educational programs" << endl;

int input = -1;

while (input == -1) {

cin >> input;

if (input > 3 || input < 1) {

input = -1;

cout << "Invalid option. Choose an option from the list above" << endl;

}

}

int x, y;

string s;

string search\_query;

switch (input)

{

case 1:

cout << "Enter score" << endl;

cin >> x;

cout << "Enter year" << endl;

cin >> y;

search\_query =

"SELECT \* \n"

"FROM public.\"Individual\"\n"

"WHERE \"ID\" IN (\n"

"SELECT \"Individual\_ID\" FROM public.\"EIT\_certificate\"\n"

"WHERE \"Subject\" = 'Math' and \"Result\" >= " + std::to\_string(x) + " and \"Year\" = " + std::to\_string(y) + ")\n"

"ORDER BY \"ID\"\n"

;

break;

case 2:

cout << "Enter subject name" << endl;

getline(cin, s);

getline(cin, s);

search\_query =

"SELECT \* FROM public.\"Student\_ticket\""

"WHERE \"Start\_date\" <= current\_date and \"End\_date\" >= current\_date and \"Subject\_ID\" IN ("

"SELECT \"ID\" FROM public.\"Subject\_of\_education\_activity\""

"WHERE \"Name\" = '" + s + "')"

"ORDER BY \"ID\"\n"

;

break;

case 3:

cout << "Enter subject name" << endl;

getline(cin, s);

getline(cin, s);

cout << "Enter number of programs" << endl;

cin >> x;

search\_query =

"SELECT \"ID\", \"Level\_of\_education\", \"Education\_program\", \"count\" FROM\n"

"public.\"Education\"\n"

"INNER JOIN(\n"

"SELECT \"Education\_ID\", COUNT(\*) FROM(\n"

"SELECT \"Education\_ID\" FROM\n"

"(SELECT \"Subject\_ID\", \"Education\_ID\" FROM public.\"Student\_ticket\"\n"

"UNION ALL\n"

"SELECT \"Subject\_ID\", \"Education\_ID\" FROM public.\"Education\_document\") AS combined\_table\n"

"WHERE \"Subject\_ID\" IN(\n"

"SELECT \"ID\" FROM public.\"Subject\_of\_education\_activity\"\n"

"WHERE \"Name\" = '" + s +"')\n"

") AS countable\_table\n"

"GROUP BY \"Education\_ID\"\n"

"ORDER BY COUNT(\*) DESC\n"

"LIMIT " + std::to\_string(x) + ") as count\_table\n"

"ON \"ID\" = \"Education\_ID\"\n"

"ORDER BY \"count\" DESC";

break;

default:

break;

}

auto start = std::chrono::steady\_clock::now();

auto result = model->query(search\_query);

auto end = std::chrono::steady\_clock::now();

if (to\_print)

view->printSearch(result, model->GetLastError());

cout << "Time spent on search: " << std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() << " ms " << endl;

return result;

}

PGresult\* Controller::performPrint(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Generate interapted" << endl;

return nullptr;

}

auto rows = model->getTableRows(table.first);

int input = -1;

if (rows.size() < 100)

input = 1;

else {

cout << "Choose which rows to display" << endl;

cout << "1 All rows" << endl;

cout << "2 First 100 rows" << endl;

cout << "3 Last 100 rows" << endl;

while (input == -1) {

cin >> input;

if (input > 3 || input < 1) {

input = -1;

cout << "Invalid option. Choose an option from the list above" << endl;

}

}

}

int left = 0;

int right = rows.size();

if (input == 2)

right = std::min(100, (int)(rows.size()));

if (input == 3)

left = std::max(0, (int)(rows.size()) - 100);

auto result = model->query("SELECT \* FROM public.\"" + table.second + "\"");

if(to\_print)

view->showTable(result, left, right, true);

return result;

}

PGresult\* Controller::performDrop(bool to\_print) {

std::pair<int, string> table = chooseTable();

if (table.first == -1) {

cout << "Generate interapted" << endl;

return nullptr;

}

string drop\_query = "DELETE FROM public.\"" + table.second + "\"";

auto result = model->query(drop\_query);

if (to\_print)

view->printDrop(result, model->GetLastError());

return nullptr;

}

*Model.h*

#pragma once

#include <vector>

#include <string>

#include <unordered\_map>

#include "libpq-fe.h"

#include "Model.h"

namespace model {

class Model

{

public:

enum class dataTypes {

text,

integer,

numeric,

date,

EIT\_subject,

Education\_level,

Type\_of\_subject\_of\_educational\_activity,

Ukraine\_region,

};

static std::unordered\_map<std::string, dataTypes> data\_types;

Model(const std::string& \_database\_name, const std::string& \_username, const std::string& \_password);

PGresult\* query(const std::string& query) const;

std::vector<std::string> getTables() const;

std::string getTablePrimaryKey(int tableIndex) const;

std::unordered\_map<std::string, std::pair<std::string, std::string>> getTableForeignKeys(int tableIndex) const;

std::vector<std::vector<std::string>> getTableRows(int table\_index) const;

std::vector<std::pair<std::string, model::Model::dataTypes>> getTableColumns(int tableIndex) const;

std::string GetLastError() const;

private:

void connect();

PGconn\* connection = nullptr;

std::vector<std::string> tables;

std::string username;

std::string database\_name;

std::string password;

};

}

*Model.cpp*

#include "Model.h"

#include <iostream>

using namespace model;

std::unordered\_map<std::string, Model::dataTypes> Model::data\_types = {

{"text",dataTypes::text},

{"integer",dataTypes::integer},

{"numeric",dataTypes::numeric},

{"date",dataTypes::date},

{"EIT\_subject",dataTypes::EIT\_subject},

{"Education\_level",dataTypes::Education\_level},

{"Type\_of\_subject\_of\_educational\_activity",dataTypes::Type\_of\_subject\_of\_educational\_activity},

{"Ukraine\_region",dataTypes::Ukraine\_region},

};

Model::Model(const std::string& \_database\_name, const std::string& \_username, const std::string& \_password)

: database\_name(\_database\_name), username(\_username), password(\_password) {

connect();

auto tablesQueryResult = query(

"SELECT table\_name\n"

"FROM information\_schema.tables\n"

"WHERE table\_type = 'BASE TABLE'\n"

"AND table\_schema = 'public';\n"

);

if (PQresultStatus(tablesQueryResult) != PGRES\_TUPLES\_OK) {

auto error = std::string(PQresultErrorMessage(tablesQueryResult));

throw std::logic\_error(error);

}

for (int i = 0; i < PQntuples(tablesQueryResult); i++) {

for (int j = 0; j < PQnfields(tablesQueryResult); j++) {

tables.push\_back({ PQgetvalue(tablesQueryResult, i, j) });

}

}

}

void Model::connect() {

//"user=postgres password= host=127.0.0.1 dbname=postgres"

const std::string connection\_string =

"user=" + username + "\n"

"password=" + password + "\n"

"host = 127.0.0.1 dbname=" + database\_name + "\n"

"client\_encoding = UTF8";

connection = PQconnectdb(connection\_string.c\_str());

if (connection == nullptr) {

throw std::logic\_error("Can't connect to database!");

}

}

PGresult\* Model::query(const std::string& query) const {

return PQexec(connection, query.c\_str());

}

std::vector<std::string> Model::getTables() const {

return tables;

}

std::string Model::GetLastError() const {

auto msg = PQerrorMessage(connection);

return msg;

}

std::vector<std::pair<std::string, model::Model::dataTypes>> Model::getTableColumns(int table\_index) const {

std::vector<std::pair<std::string, model::Model::dataTypes>> columns;

auto columns\_name\_result = query(

"SELECT column\_name\n"

"FROM information\_schema.columns\n"

"WHERE table\_schema = 'public'\n"

"AND table\_name = '" + tables[table\_index] + "';"

);

auto columns\_types\_result = query(

"SELECT data\_type\n"

"FROM information\_schema.columns\n"

"WHERE table\_schema = 'public'\n"

"AND table\_name = '" + tables[table\_index] +"';"

);

auto columns\_custom\_types\_result = query(

"SELECT udt\_name\n"

"FROM information\_schema.columns\n"

"WHERE table\_schema = 'public'\n"

"AND table\_name = '" + tables[table\_index] + "';"

);

for (int i = 0; i < PQntuples(columns\_name\_result); i++) {

for (int j = 0; j < PQnfields(columns\_name\_result); j++) {

auto name = PQgetvalue(columns\_name\_result, i, j);

std::string type = PQgetvalue(columns\_types\_result, i, j);

if(type == "USER-DEFINED")

type = PQgetvalue(columns\_custom\_types\_result, i, j);

if(data\_types.count(type))

columns.push\_back({ name, data\_types[type] });

else

throw std::logic\_error("Unrecognizable data type: " + type + "!");

}

}

return columns;

}

std::vector<std::vector<std::string>> Model::getTableRows(int table\_index) const {

std::vector<std::vector<std::string>> rows;

auto rows\_result = query(

"SELECT \*\n"

"FROM \"" + tables[table\_index] + "\";"

);

for (int i = 0; i < PQntuples(rows\_result); i++) {

rows.push\_back({});

for (int j = 0; j < PQnfields(rows\_result); j++) {

auto value = PQgetvalue(rows\_result, i, j);

rows[i].push\_back(value);

}

}

return rows;

}

std::string Model::getTablePrimaryKey(int tableIndex) const {

std::string table\_name = tables[tableIndex];

auto\* primary\_result = query(

"SELECT a.attname, format\_type(a.atttypid, a.atttypmod) AS data\_type\n"

"FROM pg\_index i\n"

"JOIN pg\_attribute a ON a.attrelid = i.indrelid\n"

"AND a.attnum = ANY(i.indkey)\n"

"WHERE i.indrelid = 'public.\"" + table\_name +

"\"'::regclass\n"

"AND i.indisprimary;"

);

if (PQresultStatus(primary\_result) == PGRES\_TUPLES\_OK) {

return PQgetvalue(primary\_result, 0, 0);

}

return "";

}

std::unordered\_map<std::string, std::pair<std::string, std::string>> Model::getTableForeignKeys(int tableIndex) const {

std::string table\_name = tables[tableIndex];

std::unordered\_map<std::string, std::pair<std::string, std::string>> res;

auto\* foreign\_result = query(

"SELECT kcu.column\_name, ccu.table\_name AS foreign\_table\_name, ccu.column\_name AS foreign\_column\_name\n"

"FROM information\_schema.table\_constraints AS tc\n"

"JOIN information\_schema.key\_column\_usage AS kcu\n"

"ON tc.constraint\_name = kcu.constraint\_name\n"

"AND tc.table\_schema = kcu.table\_schema\n"

"JOIN information\_schema.constraint\_column\_usage AS ccu\n"

"ON ccu.constraint\_name = tc.constraint\_name\n"

"AND ccu.table\_schema = tc.table\_schema\n"

"WHERE tc.constraint\_type = 'FOREIGN KEY' AND tc.table\_name = '" + table\_name + "';"

);

for (int i = 0; i < PQntuples(foreign\_result); i++) {

auto column = PQgetvalue(foreign\_result, i, 0);

auto foreign\_table = PQgetvalue(foreign\_result, i, 1);

auto foreign\_column = PQgetvalue(foreign\_result, i, 2);

res[column] = { foreign\_table, foreign\_column };

}

if (PQresultStatus(foreign\_result) != PGRES\_TUPLES\_OK) {

std::cout << "Error : " << GetLastError() << std::endl;

}

return res;

}

*View.h*

#pragma once

#include <libpq-fe.h>

#include <memory>

#include <string>

#include "Model.h"

namespace view {

class View

{

public:

View();

void printInsert(PGresult\* res, const std::string& error) const;

void printRemove(PGresult\* res, const std::string& error) const;

void printEdit(PGresult\* res, const std::string& error) const;

void printGenerate(PGresult\* res, const std::string& error) const;

void printDrop(PGresult\* res, const std::string& error) const;

void printSearch(PGresult\* res, const std::string& error) const;

void showTable(PGresult\* res, int left, int right, bool range = false) const;

private:

};

}

*View.cpp*

#include "View.h"

#include <iostream>

#include <iomanip>

using namespace view;

using std::cout;

using std::endl;

View::View() {

}

void View::printInsert(PGresult\* res, const std::string& error) const {

if (PQresultStatus(res) != PGRES\_COMMAND\_OK) {

cout << "Insert ended with an " << error << endl;

}

else {

cout << "Row was succesfully inserted in table!" << endl;

}

}

void View::printRemove(PGresult\* res, const std::string& error) const {

if (PQresultStatus(res) != PGRES\_COMMAND\_OK) {

cout << "Remove ended with an " << error << endl;

}

else {

cout << "Row was succesfully deleted from table!" << endl;

}

}

void View::printEdit(PGresult\* res, const std::string& error) const {

if (PQresultStatus(res) != PGRES\_COMMAND\_OK) {

cout << "Edit ended with an " << error << endl;

}

else {

cout << "Row was successfully edited!" << endl;

}

}

void View::printGenerate(PGresult\* res, const std::string& error) const {

if (PQresultStatus(res) != PGRES\_COMMAND\_OK) {

cout << "Generating ended with an " << error << endl;

}

else {

cout << "Rows were successfully generated!" << endl;

}

}

void View::printDrop(PGresult\* res, const std::string& error) const {

if (PQresultStatus(res) != PGRES\_COMMAND\_OK) {

cout << "Drop ended with an " << error << endl;

}

else {

cout << "Rows were successfully dropped!" << endl;

}

}

void View::printSearch(PGresult\* res, const std::string& error) const {

if (error != "") {

cout << "Search ended with an " << error << endl;

}

else {

showTable(res, 0, 0);

}

}

void View::showTable(PGresult\* res, int left, int right, bool range) const {

std::vector<unsigned long long> width;

int l = 0;

int r = PQntuples(res);

if (range) {

l = std::max(l, left);

r = std::min(r, right);

}

width.push\_back(std::to\_string(r).size() + 3);

for (int j = 0; j < PQnfields(res); j++)

width.push\_back(strlen(PQfname(res, j)) + 3);

for (int i = l; i < r; i++)

for (int j = 0; j < PQnfields(res); j++)

width[j + 1] = std::max(width[j + 1], strlen(PQgetvalue(res, i, j)) + 3);

cout << std::setw(width[0]) << "#";

for (int j = 0; j < PQnfields(res); j++)

cout << std::setw(width[j + 1]) << PQfname(res, j);

cout << endl;

for (int i = l; i < r; i++) {

cout << std::setw(width[0]) << std::to\_string(i + 1);

for (int j = 0; j < PQnfields(res); j++) {

cout << std::setw(width[j + 1]) << PQgetvalue(res, i, j);

}

cout << endl;

}

}

*lab2.cpp*

#include <iostream>

#include "Model.h"

#include "Controller.h"

#include "View.h"

int main()

{

try {

auto model = std::make\_shared<model::Model>("edbo", "postgres", "--");

auto view = std::make\_shared<view::View>();

auto controller = std::make\_shared<controller::Controller>(model, view);

controller->interactionLoop();

}

catch (const std::exception& e){

std::cout << "Exception occured: " << e.what() << std::endl;

}

}