

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”*

Виконала:

студентка ІII курсу

групи КВ-02

Майстренко Ольга Олексіївна

Перевірив: Павловський Володимир Ілліч

Київ – 2022

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

*Загальне 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. 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:

1. 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.

2. 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!

3. 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.

4. 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).

Зміст звіту:

1. Нормалізована логічна модель даних;
2. Середовище розробки та налаштування програмної системи;
3. Методологія побудови ПС та шаблон проектування;
4. Структура програми та її опис;
5. Структура меню програми - концептуальну модель програми;
6. Лістинг модуля "Model" згідно із шаблоном MVC. Лістинг має бути пояснений;
7. Лістинги фрагментів програм з моделлю даних, директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив. Лістинги мають бути пояснені;
8. Лістинги фрагментів програм зі статичними та динамічними запитами пошуку. Лістинги мають бути пояснені;
9. Лістинги фрагментів програм генерування випадкових даних в таблицях БД. Лістинги мають бути пояснені;
10. Лістинги та скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних. Причини помилок мають бути пояснені;
11. Лістинги та скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці нема відповідного запису. Лістинги мають бути пояснені;
12. Посилання для навігації по тексту програми;
13. Текст програми та посилання на Git з текстом програми.

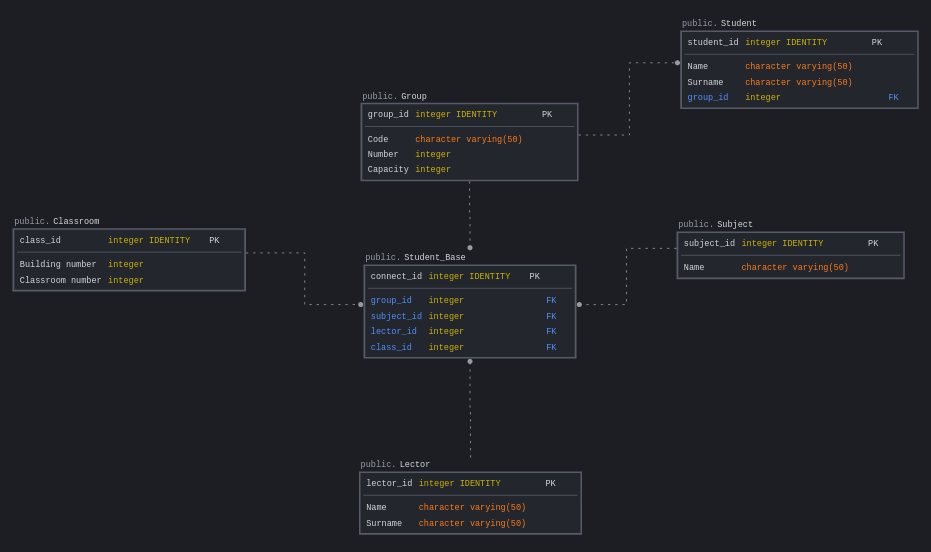
**Нормалізована логічна модель даних**

Рисунок 1 - Нормалізована логічна модель даних (інструмент: sqldbm.com)

**Середовище розробки та налаштування програмної системи**

Для виконання лабораторної роботи використовувалась мова програмування C++ та середовище розробки VS Code.

Для підключення до серверу бази даних PostgreSQL використано бібліотеку Libpq.

**Шаблон проектування програмного додатку**

MVC – шаблон проектування, що використовується у програмі.

*Модель (baseModel.h /.cpp)* – по суті сховище даних та описує логіку використовуваних даних.

*Подання (baseView.h /.cpp)* – грубо кажучи частина паттерну – обробник даних.

*Контролер (baseController.h /.cpp)* – консольний інтерфейс, у який користувачем вводяться дані, інтерфейс у консолі.

**Структура програми та її зміст**

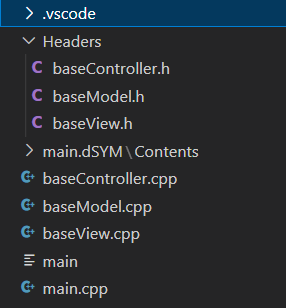


Рисунок 2 – Структура програми

1. main.cpp – файл з якого починається виконання програми, викликає власне меню і опісля запускає потрібну команду на виконання. Також тут задаються дані для з’єднання із базою;
2. baseModel.h/.cpp – модуль, що відповідає за збереження даних та регулювання підключення до бази;
3. baseController.h/.cpp – модуль, відповідає за взаємодію користувача і бази даних. Якщо модель відповідає за збереження, то тут відбувається запитів користувача та зв'язок View з Model;
4. baseView.h/.cpp – модуль що відповідає за відображення дій із базою, аби користувач бачив результати своїх дій та за прийом даних, які опісля обробляться у контролері.

**Структура меню програми**

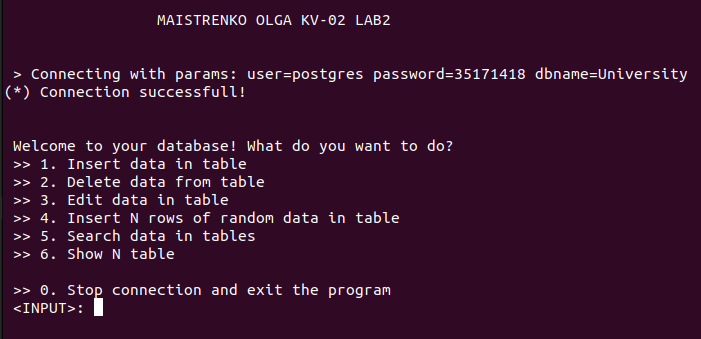


Рисунок 3 – Початкове меню програми, де користувач може обрати дію, яку він хоче виконати з базою даних

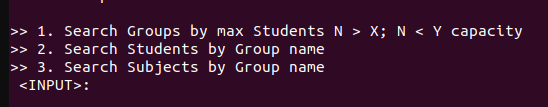


Рисунок 4 – Меню, яке бачить користувач при переході у вкладку запитів

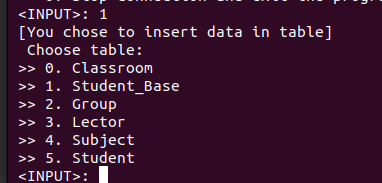


Рисунок 5 – Типове меню, яке бачить користувач при виклику функціі вставки, видалення, редагування тощо.

**Лістинг модуля "Model" згідно із шаблоном MVC**

Модуль складається з .h та .cpp файлів. Відповідає він за зберігання та опис логіки використовуваних даних.

Детальний опис наведено у коментарях у самому коді.

*baseModel.h – хедер-файл у якому зберігаються прототипи класу Model*

#ifndef \_BASE\_MODEL\_H\_

#define \_BASE\_MODEL\_H\_

#include <iostream>

#include <vector>

#include "libpq-fe.h"

typedef void baseSQLlink;

namespace Model

{

class baseModel

{

private:

*/\* Змінні, що містять дані про PG \*/*

PGconn \*baseConnection;

std::string baseUsername;

std::string basePassword;

std::string baseName;

std::string query;

*/\* Збережені значення для використання у функціях \*/*

int valMenuChoice;

int valTableChoice;

int valColumnChoice;

int valSearchChoice;

int valCountRow;

int valCountColumn;

*/\* Вектори для зберігання даних із бази даних \*/*

std::vector<std::string> vertorTableNames;

std::vector<std::string> vectorColumnsNames;

std::vector<std::vector<std::string>> vectorTableData;

std::vector<int> vectorColumnVarTypes;

public:

baseModel(void);

baseModel(const std::string usr, const std::string pswd, const std::string db);

~baseModel();

*/\* Сеттери та геттери \*/*

std::vector<std::string> getListTables(void);

std::vector<std::string> getListColums(void);

std::vector<std::vector<std::string>> getListTableContents(void);

std::string returnTable(int \_ind);

std::string returnColumn(int \_ind);

int returnDataType(int \_ind);

int returnSizeOfListColumn(void);

void setMenuChoice(int \_ch);

int returnMenuChoice(void);

void setTableChoice(int \_ch);

int returnTableChoice(void);

void setColumnChoice(int \_ch);

int returnColumnChoice(void);

void setSearchChoice(int \_ch);

int returnSearchChoice(void);

void setQuery(std::string \_str);

std::string returnQuery(void);

*/\* Робота SQL \*/*

*/\* З'єднання з базою \*/*

baseSQLlink baseConnect(void);

*/\* Розірвати підключення до бази даних \*/*

baseSQLlink baseTerminate(void);

*/\* Отримати список таблиць із бази даних \*/*

baseSQLlink baseGetTablesFromChoice(void);

*/\* Отримати список стовпців з бази даних. Залежить від вибору столу користувачем \*/*

baseSQLlink baseGetColumnsFromChoice(void);

*/\* Надіслати запит до бази даних \*/*

baseSQLlink baseQuerySend(void);

*/\* Отримати список стовпців з бази даних. Залежить від рядка запиту \*/*

int baseGetColumnsFromQuery(void);

*/\* Отримати дані таблиці з бази даних. Залежить від рядка запиту \*/*

int baseGetResultFromQuery(void);

*/\* Отримати дані таблиці з бази даних. Залежить від вибору користувача \*/*

baseSQLlink baseGetTableContents(void);

};

};

#endif //\_BASE\_MODEL\_H\_

*baseModel.cpp – файл у якому зберігається власне реалізація функцій, а точніше методів класу Model*

#include "baseModel.h"

using namespace Model;

baseModel::baseModel(void)

{

}

baseModel::baseModel(const std::string usr, const std::string pswd, const std::string db)

{

baseUsername = usr;

basePassword = pswd;

baseName = db;

baseConnection = nullptr;

}

baseModel::~baseModel(void)

{

}

void baseModel::setMenuChoice(int \_ch)

{

valMenuChoice = \_ch;

}

void baseModel::setTableChoice(int \_ch)

{

valTableChoice = \_ch;

baseModel::baseGetColumnsFromChoice();

}

void baseModel::setColumnChoice(int \_ch)

{

valColumnChoice = \_ch;

}

void baseModel::setSearchChoice(int \_ch)

{

valSearchChoice = \_ch;

}

void baseModel::setQuery(std::string \_str)

{

query = \_str;

}

int baseModel::returnTableChoice(void)

{

return valTableChoice;

}

int baseModel::returnMenuChoice(void)

{

return valMenuChoice;

}

int baseModel::returnColumnChoice(void)

{

return valColumnChoice;

}

int baseModel::returnSearchChoice(void)

{

return valSearchChoice;

}

std::string baseModel::returnQuery(void)

{

return query;

}

std::vector<std::string> baseModel::getListTables(void)

{

baseModel::baseGetTablesFromChoice();

return vertorTableNames;

}

std::vector<std::string> baseModel::getListColums(void)

{

return vectorColumnsNames;

}

std::vector<std::vector<std::string>> baseModel::getListTableContents(void)

{

return vectorTableData;

}

std::string baseModel::returnColumn(int \_ind)

{

return vectorColumnsNames.at(\_ind);

}

std::string baseModel::returnTable(int \_ind)

{

return vertorTableNames.at(\_ind);

}

int baseModel::returnDataType(int \_ind)

{

return vectorColumnVarTypes.at(\_ind);

}

int baseModel::returnSizeOfListColumn(void)

{

return vectorColumnsNames.size();

}

baseSQLlink baseModel::baseConnect(void)

{

std::string result = "user=" + baseUsername;

if (basePassword != "0")

{

result = result + " password=" + basePassword;

}

result = result + " dbname=" + baseName;

std::cout << " > Connecting with params: " << result << std::endl;

baseConnection = PQconnectdb(result.c\_str());

if (PQstatus(baseConnection) == CONNECTION\_BAD)

{

std::cout << PQerrorMessage(baseConnection) << std::endl;

std::cout << "(!) Connection failed!" << std::endl;

exit(0);

}

std::cout << "(\*) Connection successfull!" << std::endl;

std::cout << std::endl;

}

baseSQLlink baseModel::baseTerminate(void)

{

PQfinish(baseConnection);

std::cout << "(\*) Connection terminated!" << std::endl;

exit(0);

}

baseSQLlink baseModel::baseGetTablesFromChoice(void)

{

vertorTableNames.clear();

PGresult \*res = PQexec(baseConnection, "SELECT table\_name FROM information\_schema.tables "

"WHERE table\_schema = 'public'");

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << PQresultErrorMessage(res) << std::endl;

PQclear(res);

}

int rows = PQntuples(res);

for (int i = 0; i < PQntuples(res); i++)

{

vertorTableNames.push\_back(PQgetvalue(res, i, 0));

}

}

baseSQLlink baseModel::baseGetColumnsFromChoice(void)

{

vectorColumnsNames.clear();

vectorColumnVarTypes.clear();

std::string path = "SELECT \* FROM public.**\"**" + vertorTableNames.at(valTableChoice) + "**\"**";

PGresult \*res = PQexec(baseConnection, path.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< " >> No data retrieved from " << baseName << ". Abort function << " << std::endl;

PQclear(res);

}

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

{

vectorColumnsNames.push\_back(PQfname(res, i));

vectorColumnVarTypes.push\_back(PQftype(res, i));

}

PQclear(res);

}

int baseModel::baseGetColumnsFromQuery(void)

{

vectorColumnsNames.clear();

vectorColumnVarTypes.clear();

PGresult \*res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << PQresultErrorMessage(res) << std::endl;

PQclear(res);

return 0;

}

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

{

vectorColumnsNames.push\_back(PQfname(res, i));

vectorColumnVarTypes.push\_back(PQftype(res, i));

}

PQclear(res);

return 1;

}

baseSQLlink baseModel::baseQuerySend(void)

{

PGresult \*res = PQexec(baseConnection, "BEGIN");

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

{

printf("BEGIN command failed**\n**");

PQclear(res);

}

res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< PQerrorMessage(baseConnection) << std::endl;

PQclear(res);

}

res = PQexec(baseConnection, "COMMIT");

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

{

printf("COMMIT command failed**\n**");

PQclear(res);

}

}

baseSQLlink baseModel::baseGetTableContents(void)

{

vectorTableData.clear();

baseGetTablesFromChoice();

baseGetColumnsFromChoice();

std::string path = "SELECT \* FROM public.**\"**" + vertorTableNames.at(valTableChoice) + "**\"**";

path += " ORDER BY **\"**" + vectorColumnsNames.at(0) + "**\"**";

PGresult \*res = PQexec(baseConnection, path.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << "No data retrieved. Abort function" << std::endl;

PQclear(res);

}

valCountRow = PQntuples(res);

valCountColumn = PQnfields(res);

std::vector<std::string> lsttmp;

for (uint16\_t i = 0; i < valCountRow; i++)

{

for (uint16\_t j = 0; j < valCountColumn; j++)

{

lsttmp.push\_back(PQgetvalue(res, i, j));

}

vectorTableData.push\_back(lsttmp);

lsttmp.clear();

}

PQclear(res);

}

int baseModel::baseGetResultFromQuery(void)

{

vectorTableData.clear();

baseGetColumnsFromQuery();

PGresult \*res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< PQerrorMessage(baseConnection) << std::endl;

PQclear(res);

return 0;

}

valCountRow = PQntuples(res);

valCountColumn = PQnfields(res);

std::vector<std::string> lsttmp;

for (uint16\_t i = 0; i < valCountRow; i++)

{

for (uint16\_t j = 0; j < valCountColumn; j++)

{

lsttmp.push\_back(PQgetvalue(res, i, j));

}

vectorTableData.push\_back(lsttmp);

lsttmp.clear();

}

return 1;

}

**Лістинг фрагмента програми для внесення даних**

За внесення даних відповідає метод класу Controller:   
*void baseController::performDataInsert(void).*

Спочатку виводиться список доступних таблиць для здійснення вибору, який опісля обробляється і виводиться вже список стовпців, які є у таблиці. Далі – внесення даних та визначення типу даних. Після цих дій відбувається збірка команди, її вивід та подальше надсилання запиту до бази даних.

[Результати виконання за посиланням](#Результати_виконання_вставки)

void baseController::performDataInsert(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Показати список доступних стовпців \*/*

viewObj->viewList(modObj->getListColums());

std::string comSQL = "INSERT INTO public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** (";

*/\* Збірка команди \*/*

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

comSQL += "**\"**";

comSQL += modObj->returnColumn(i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += "**\"**, ";

}

else

{

comSQL += "**\"**) ";

}

}

comSQL += "VALUES (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

std::cout << "Enter value for " << modObj->returnColumn(i) << " column" << std::endl;

std::string inp = viewObj->getString();

comSQL += "'" + inp + "'";

concatVarTypes(&comSQL, i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += ", ";

}

else

{

comSQL += ")";

}

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

**Лістинг фрагмента програми для видалення даних**

За видалення даних відповідає метод класу Controller:   
*void baseController::performDataDelete(void).*

Спочатку виводиться список доступних таблиць для здійснення вибору, який опісля обробляється і отримується вміст обраної таблиці. Після цих дій – ввід бажаного для видалення id та подальше надсилання запиту до бази даних.

[Результати виконання за посиланням](#Результати_виконання_видалення)

void baseController::performDataDelete(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

std::cout << "Choose ID to delete" << std::endl;

std::string comSQL = "DELETE FROM public.**\"**" + modObj->returnTable(modObj->returnTableChoice());

comSQL += "**\"** WHERE **\"**" + modObj->returnColumn(0) + "**\"**";

comSQL += " IN (" + viewObj->getString() + ")";

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

**Лістинг фрагмента програми для редагування даних**

За видалення даних відповідає метод класу Controller:   
*void baseController::performDataEdit(void).*

Спочатку виводиться список доступних таблиць для здійснення вибору, який опісля обробляється і отримується вміст обраної таблиці. Після цих дій ввід бажаного для редагування id і виводиться вже список стовпців, які є у таблиці. Далі – зміна даних, власне саме редагування та визначення типу даних. Після цих дій відбувається збірка команди, її вивід та подальше надсилання запиту до бази даних.

[Результати виконання за посиланням](#Результати_виконання_редагування)

void baseController::performDataEdit(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

std::string comSQL = "UPDATE public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** SET ";

std::cout << "Choose ID to edit" << std::endl;

*/\* Виберіть елемент для редагування \*/*

std::string strID = viewObj->getString();

std::cout << "Choose column to edit" << std::endl;

*/\* Показати список доступних стовпців \*/*

viewObj->viewList(modObj->getListColums());

*/\* Виберіть стовпець для редагування \*/*

modObj->setColumnChoice(viewObj->inputInt());

comSQL += "**\"**" + modObj->returnColumn(modObj->returnColumnChoice()) + "**\"** = ";

std::cout << "Set value: " << std::endl;

comSQL += "'" + viewObj->getString() + "'";

*/\* Визначити тип даних \*/*

concatVarTypes(&comSQL, modObj->returnColumnChoice());

comSQL += " WHERE **\"**" + modObj->returnColumn(0) + "**\"** = " + strID;

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

**Результати виконання фрагментів програм для внесення, видалення та редагування даних.**

Результати виконання вставки:

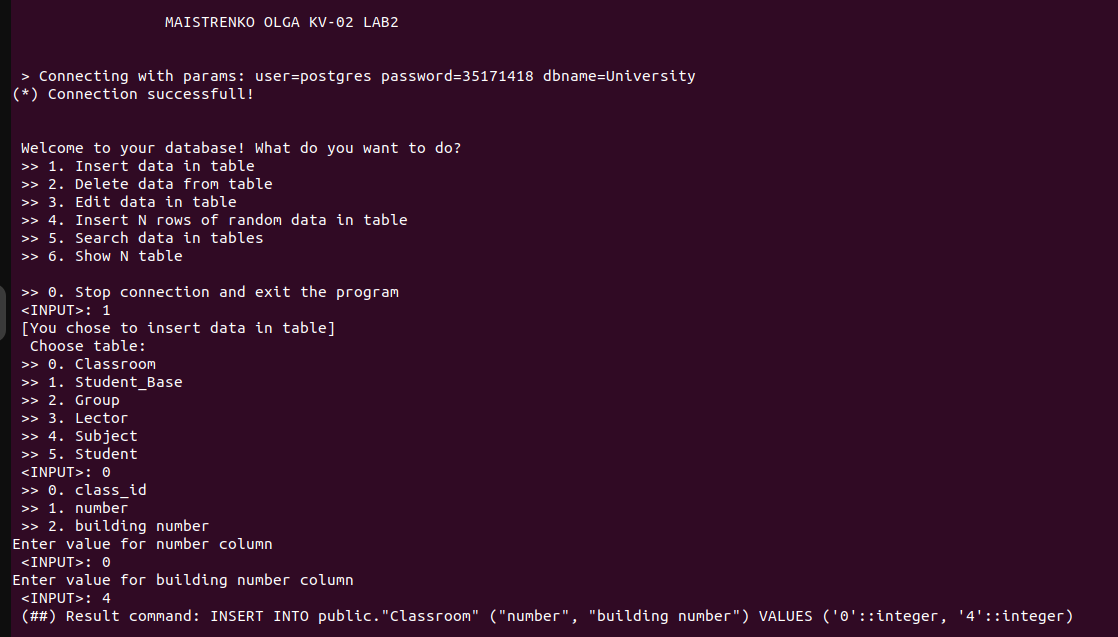


Рисунок 6 – Виконання вставки даних у таблицю

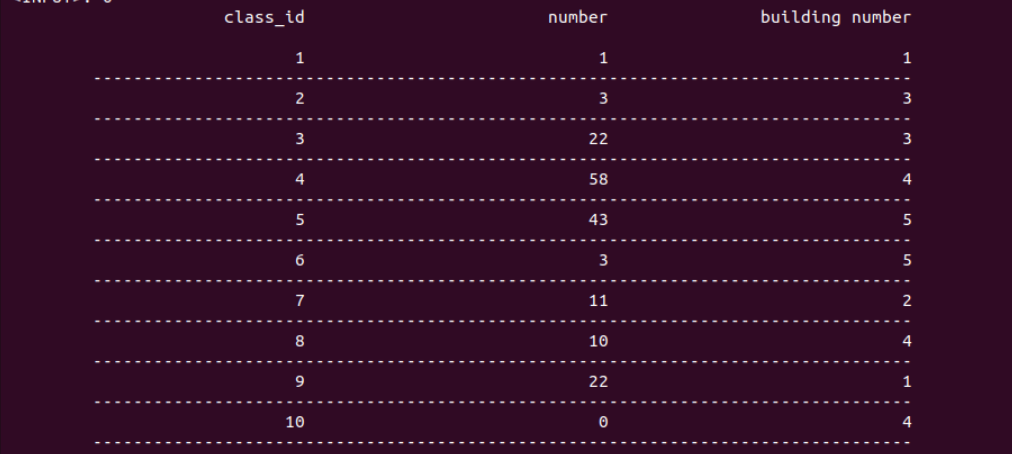


Рисунок 7 – У результаті вставки додався рядок з class\_id 10

Результати виконання видалення:

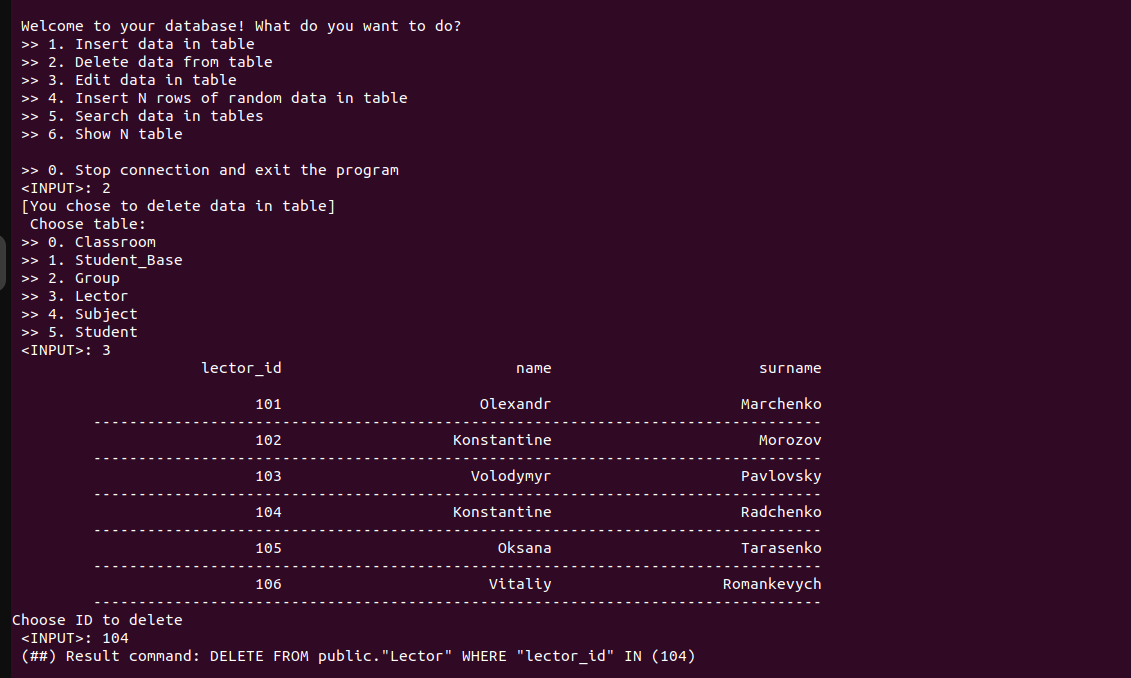


Рисунок 8 – Видалення рядка з lector\_id 104 з таблиці Lector

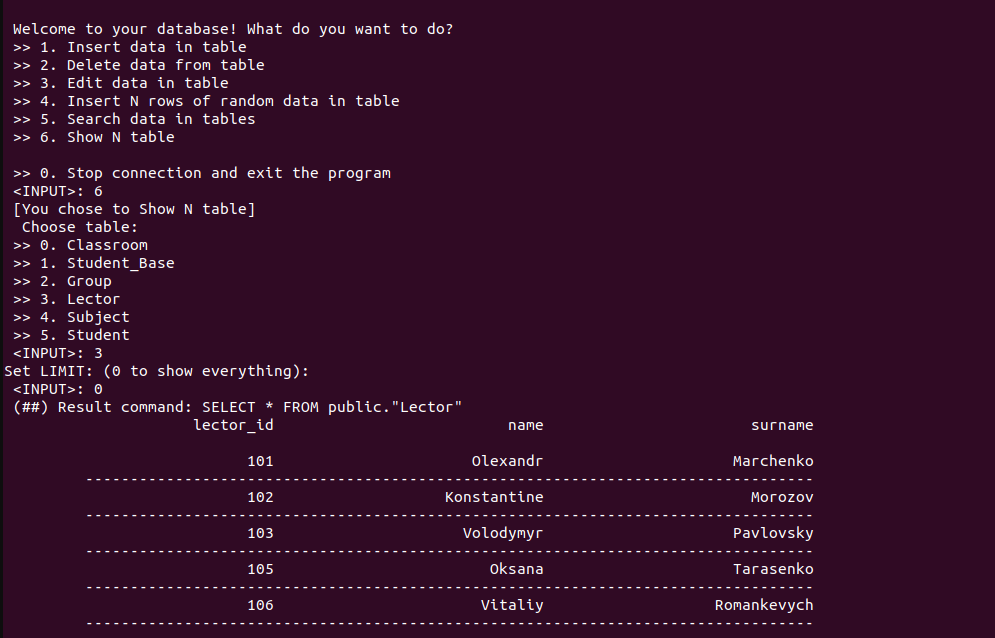


Рисунок 9 – Результат видалення

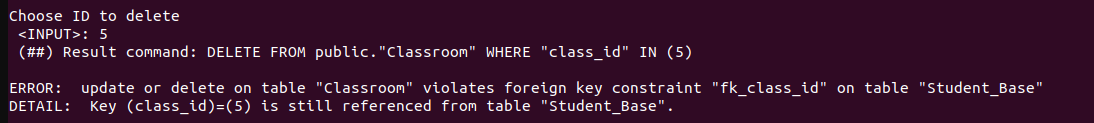


Рисунок 10 – Спроба видалення даних з таблиці, яка має зв’язки з іншою

Результати виконання редагування:

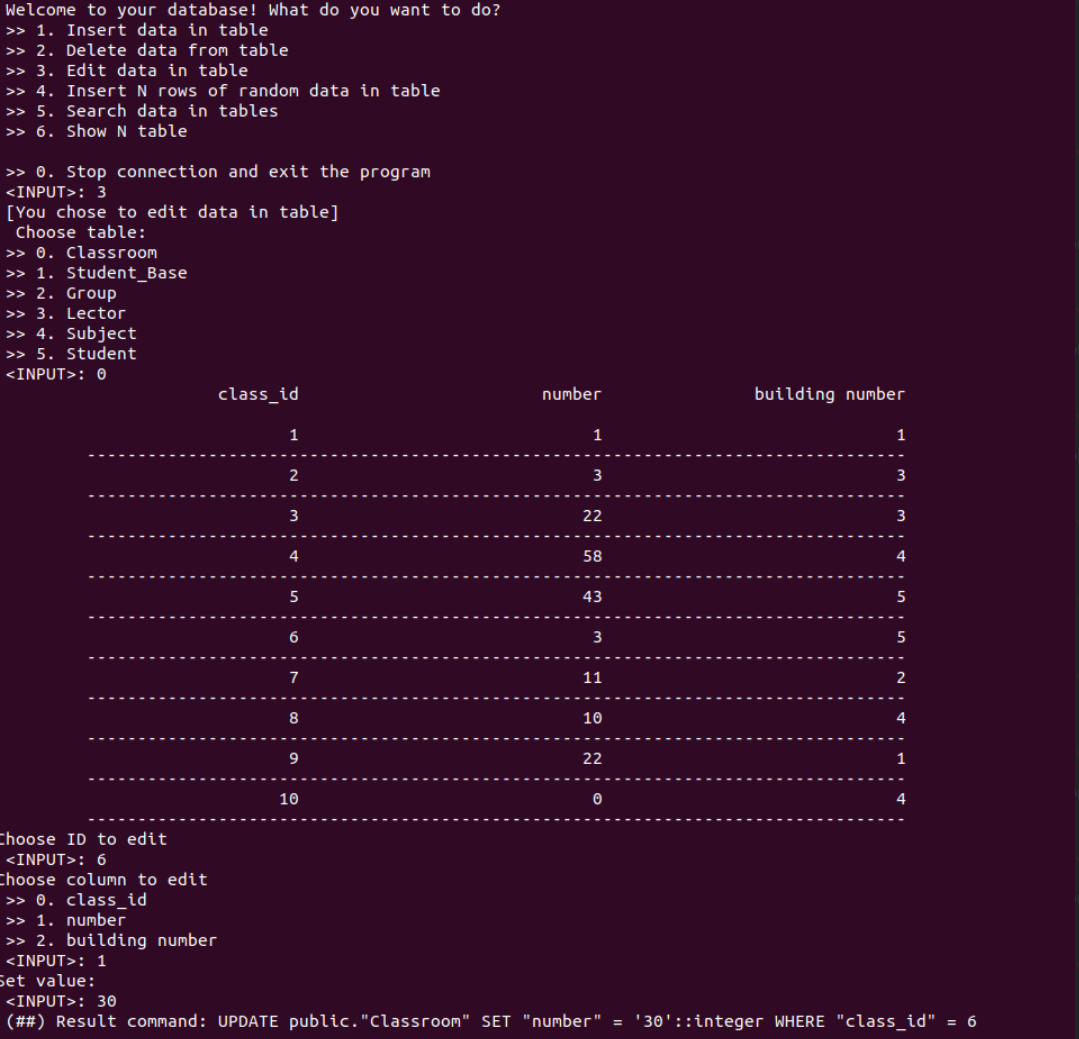


Рисунок 11 – Редагування даних у таблиці Classroom

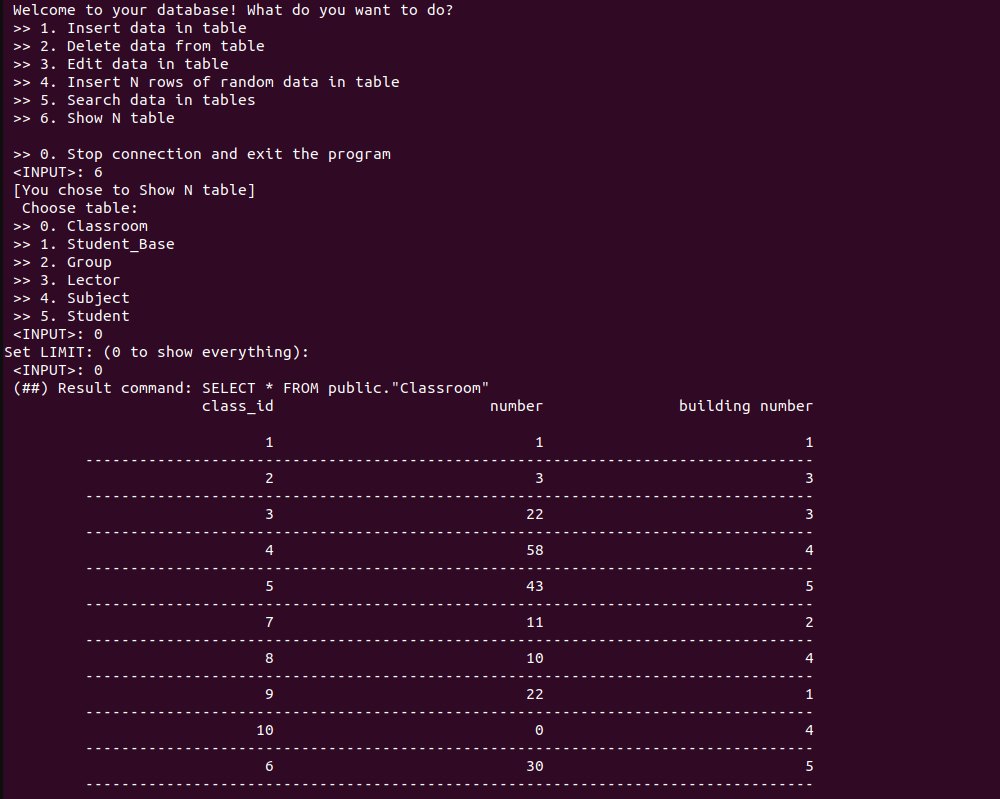


Рисунок 12 – Результат редагування. У останньому рядку змінився номер класу.

**Лістинг фрагмента програми для виконання запитів пошуку**

За виконання запитів пошуку відповідає метод класу Controller:   
*void baseController::performDataSearch(void).*

Спочатку виводиться список доступних варіантів для пошуку та встановлюється опція.

Доступні запити пошуку:

1. Пошук груп з максимальним розміром у діапазоні введеному користувачем з клавіатури;
2. Пошук студентів за кодом групи;
3. Пошук предметів, які вивчаються у групі з певним кодом;

Після цих дій відбувається збірка команди, її вивід та подальше надсилання запиту до бази даних.

void baseController::performDataSearch(void)

{

*/\* Виберіть варіант пошуку \*/*

viewObj->viewSearchOptions();

*/\* Встановити опцію пошуку в моделі \*/*

modObj->setSearchChoice(viewObj->inputInt());

std::string comSQL;

switch (modObj->returnSearchChoice())

{

case 1:

comSQL += "SELECT \* ";

comSQL += "FROM public.**\"**Group**\"** ";

comSQL += "WHERE **\"**capacity**\"** > ";

std::cout << "Set start range: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " AND **\"**capacity**\"** < ";

std::cout << "Set end range: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**capacity**\"**";

break;

case 2:

comSQL += "SELECT **\"**student\_id**\"**, **\"**name**\"**, **\"**surname**\"**, **\"**code**\"**, **\"**number**\"** ";

comSQL += "FROM public.**\"**Student**\"** ";

comSQL += "JOIN public.**\"**Group**\"** ON **\"**Student**\"**.**\"**group\_id**\"** = **\"**Group**\"**.**\"**group\_id**\"** ";

comSQL += "WHERE **\"**code**\"** = ";

std::cout << "Set group code: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**code**\"**";

break;

case 3:

comSQL += "SELECT **\"**Subject**\"**.**\"**name**\"**, **\"**Lector**\"**.**\"**name**\"**, **\"**surname**\"**, **\"**code**\"**, **\"**Group**\"**.**\"**number**\"**, **\"**Classroom**\"**.**\"**number**\"** ";

comSQL += "FROM public.**\"**Student\_Base**\"** ";

comSQL += "JOIN public.**\"**Group**\"** ON **\"**Student\_Base**\"**.**\"**group\_id**\"** = **\"**Group**\"**.**\"**group\_id**\"** ";

comSQL += "JOIN public.**\"**Lector**\"** ON **\"**Student\_Base**\"**.**\"**lector\_id**\"** = **\"**Lector**\"**.**\"**lector\_id**\"** ";

comSQL += "JOIN public.**\"**Subject**\"** ON **\"**Student\_Base**\"**.**\"**subject\_id**\"** = **\"**Subject**\"**.**\"**subject\_id**\"** ";

comSQL += "JOIN public.**\"**Classroom**\"** ON **\"**Student\_Base**\"**.**\"**class\_id**\"** = **\"**Classroom**\"**.**\"**class\_id**\"** ";

comSQL += "WHERE **\"**code**\"** = ";

std::cout << "Set group code: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**Group**\"**.**\"**number**\"**";

break;

default:

break;

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит і записати результати до векторів у моделі \*/*

if(modObj->baseGetResultFromQuery() != 0)

{

*/\* Показати дані з векторної таблиці в моделі \*/*

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

}

}

Результати виконання запитів пошуку

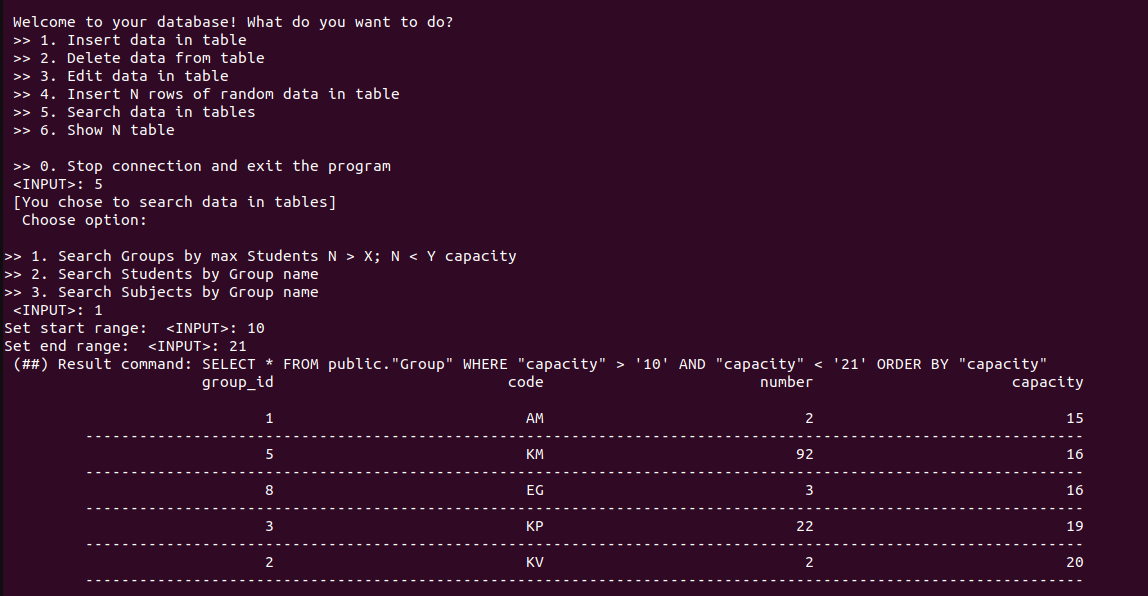


Рисунок 13 – Результат 1-го запиту в діапазоні з 10-ти до 21-го студента у групі.

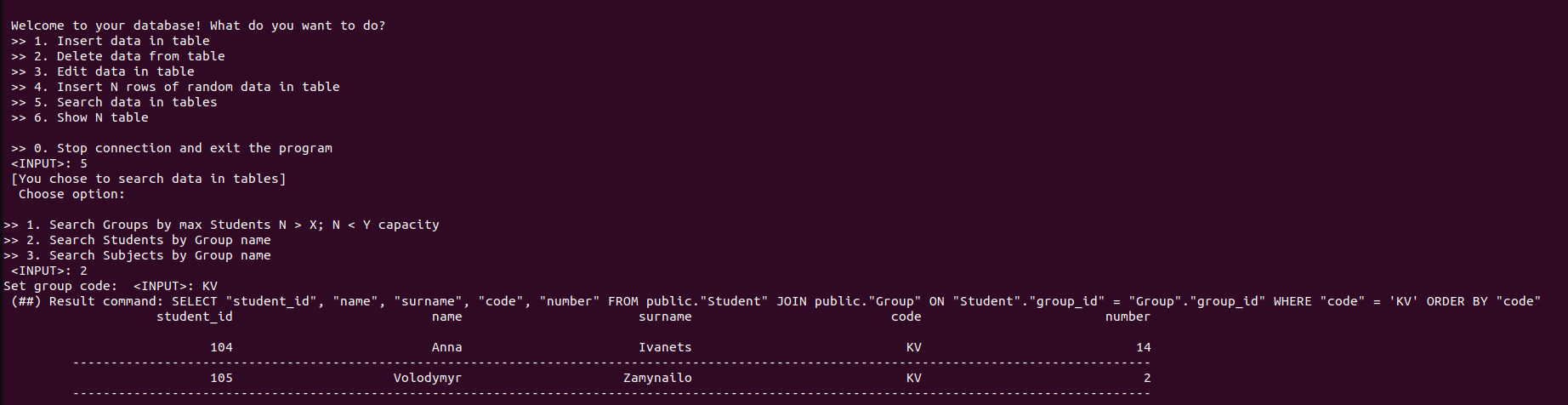


Рисунок 14 – Результат 2-го запиту. Знайдено всіх студентів, що навчаються у групах із кодом КВ

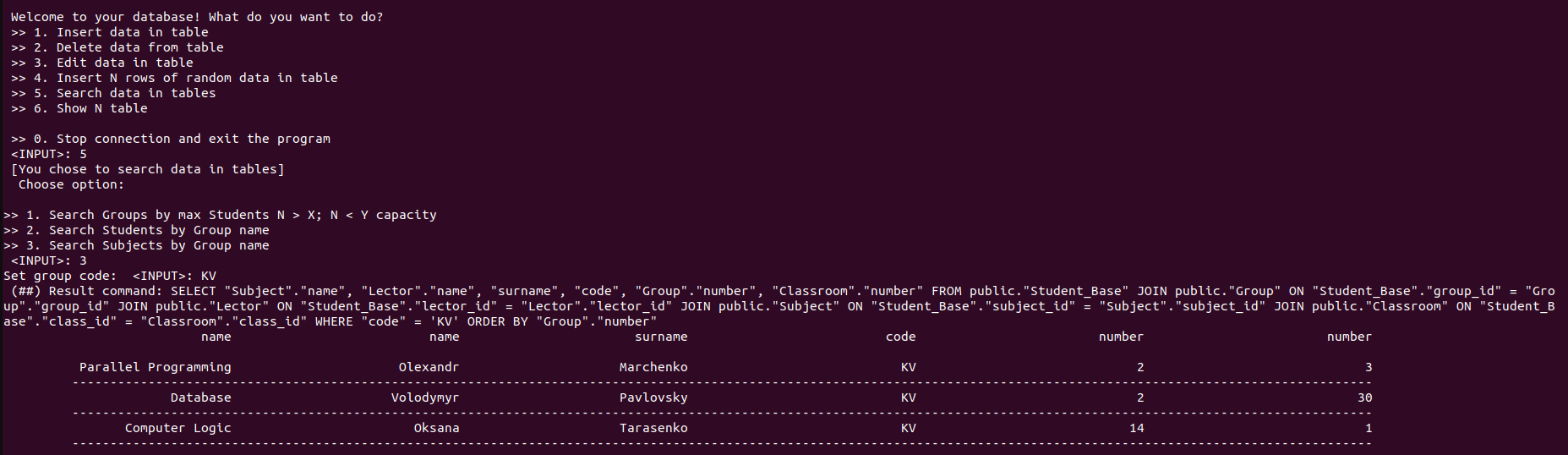


Рисунок 15 – Результат 3-го запиту. Виведено результат пошуку предметів у групах із кодом КВ

**Лістинг фрагмента програми для генерування випадкових даних у таблицях БД**

За внесення даних відповідає метод класу Controller:   
*void baseController::performDataInsertRandom(void).*

Спочатку виводиться список доступних таблиць для здійснення вибору, який опісля обробляється і отримується вміст обраної таблиці. Після цих дій – ввід бажаної кількості рандомізованих рядків та подальше надсилання запиту до бази даних.

void baseController::performDataInsertRandom(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

std::string comSQL = "INSERT INTO public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

comSQL += "**\"**";

comSQL += modObj->returnColumn(i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += "**\"**, ";

}

else

{

comSQL += "**\"**) ";

}

}

comSQL += "VALUES (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

switch (modObj->returnDataType(i))

{

case 23:

comSQL += "::integer";

break;

case 1043:

comSQL += "substr(md5(random()::text), 0, 10)";

break;

case 1082:

comSQL += "now() - random() \* INTERVAL '20000 days'";

break;

default:

break;

}

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += ", ";

}

else

{

comSQL += ")";

}

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

std::cout << " Choose how many rows of random data will be generated: ";

int \_quantity = viewObj->inputInt();

for (int i = 0; i < \_quantity; i++)

{

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

}

Результати виконання запиту на внесення великої кількості рандомізованих даних до обраної таблиці

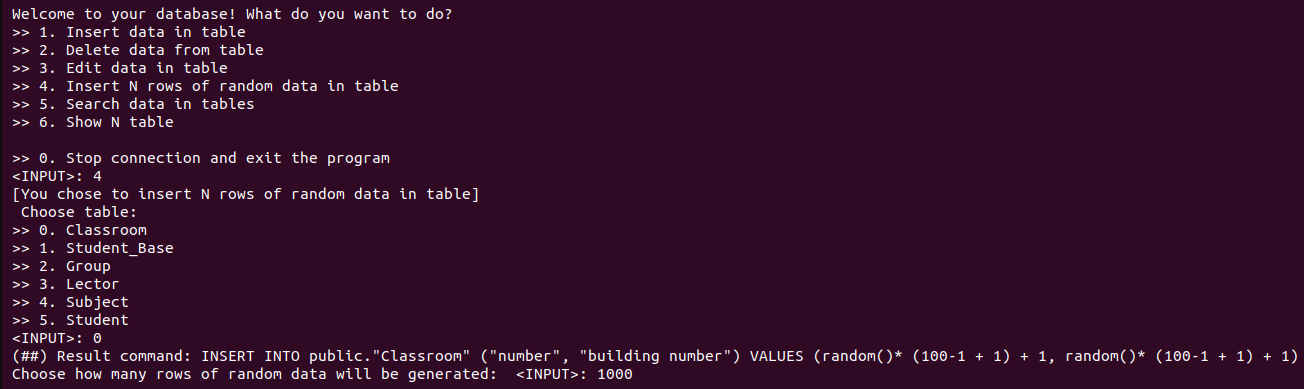


Рисунок 16 – Введення 1000 рандомних рядків у таблицю Classroom

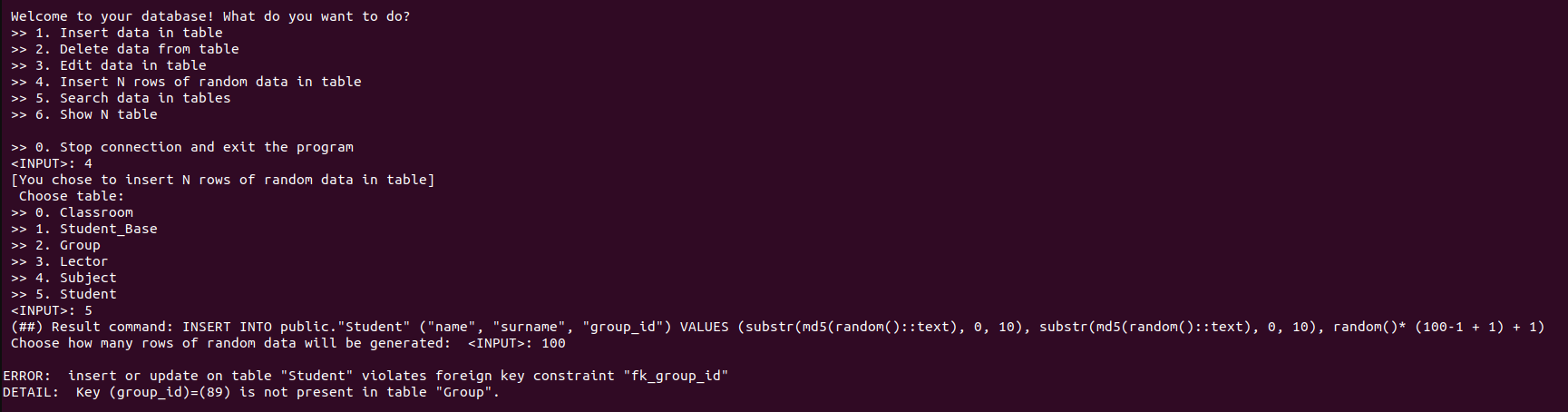


Рисунок 17 – Невдала спроба вставити рандомні дані з невідомим id.

**Ілюстрації програмного коду з репозиторію Git**

**Посилання для навігації по тексту програми**

* + [Main](#Main)
  + [Model](#Model)
  + [Controller](#Controller)
  + [View](#View)

*main.cpp*

#include <iostream>

#include "baseController.h"

int main()

{

std::string username = "postgres";

std::string password = "35171418";

std::string databasename = "University";

Model::baseModel mdl(username, password, databasename);

View::baseView vw;

Controller::baseController ctrl(&mdl, &vw);

ctrl.stepEstablishConnection();

while (1)

{

ctrl.stepMenu();

ctrl.stepDBOperations();

}

return 0;

}

*baseModel.h*

#ifndef \_BASE\_MODEL\_H\_

#define \_BASE\_MODEL\_H\_

#include <iostream>

#include <vector>

#include "libpq-fe.h"

typedef void baseSQLlink;

namespace Model

{

class baseModel

{

private:

*/\* Змінні, що містять дані про PG \*/*

PGconn \*baseConnection;

std::string baseUsername;

std::string basePassword;

std::string baseName;

std::string query;

*/\* Збережені значення для використання у функціях \*/*

int valMenuChoice;

int valTableChoice;

int valColumnChoice;

int valSearchChoice;

int valCountRow;

int valCountColumn;

*/\* Вектори для зберігання даних із бази даних \*/*

std::vector<std::string> vertorTableNames;

std::vector<std::string> vectorColumnsNames;

std::vector<std::vector<std::string>> vectorTableData;

std::vector<int> vectorColumnVarTypes;

public:

baseModel(void);

baseModel(const std::string usr, const std::string pswd, const std::string db);

~baseModel();

*/\* Сеттери та геттери \*/*

std::vector<std::string> getListTables(void);

std::vector<std::string> getListColums(void);

std::vector<std::vector<std::string>> getListTableContents(void);

std::string returnTable(int \_ind);

std::string returnColumn(int \_ind);

int returnDataType(int \_ind);

int returnSizeOfListColumn(void);

void setMenuChoice(int \_ch);

int returnMenuChoice(void);

void setTableChoice(int \_ch);

int returnTableChoice(void);

void setColumnChoice(int \_ch);

int returnColumnChoice(void);

void setSearchChoice(int \_ch);

int returnSearchChoice(void);

void setQuery(std::string \_str);

std::string returnQuery(void);

*/\* Робота SQL \*/*

*/\* З'єднання з базою \*/*

baseSQLlink baseConnect(void);

*/\* Розірвати підключення до бази даних \*/*

baseSQLlink baseTerminate(void);

*/\* Отримати список таблиць із бази даних \*/*

baseSQLlink baseGetTablesFromChoice(void);

*/\* Отримати список стовпців з бази даних. Залежить від вибору столу користувачем \*/*

baseSQLlink baseGetColumnsFromChoice(void);

*/\* Надіслати запит до бази даних \*/*

baseSQLlink baseQuerySend(void);

*/\* Отримати список стовпців з бази даних. Залежить від рядка запиту \*/*

int baseGetColumnsFromQuery(void);

*/\* Отримати дані таблиці з бази даних. Залежить від рядка запиту \*/*

int baseGetResultFromQuery(void);

*/\* Отримати дані таблиці з бази даних. Залежить від вибору користувача \*/*

baseSQLlink baseGetTableContents(void);

};

};

#endif //\_BASE\_MODEL\_H\_

*baseModel.cpp*

#include "baseModel.h"

using namespace Model;

baseModel::baseModel(void)

{

}

baseModel::baseModel(const std::string usr, const std::string pswd, const std::string db)

{

baseUsername = usr;

basePassword = pswd;

baseName = db;

baseConnection = nullptr;

}

baseModel::~baseModel(void)

{

}

void baseModel::setMenuChoice(int \_ch)

{

valMenuChoice = \_ch;

}

void baseModel::setTableChoice(int \_ch)

{

valTableChoice = \_ch;

baseModel::baseGetColumnsFromChoice();

}

void baseModel::setColumnChoice(int \_ch)

{

valColumnChoice = \_ch;

}

void baseModel::setSearchChoice(int \_ch)

{

valSearchChoice = \_ch;

}

void baseModel::setQuery(std::string \_str)

{

query = \_str;

}

int baseModel::returnTableChoice(void)

{

return valTableChoice;

}

int baseModel::returnMenuChoice(void)

{

return valMenuChoice;

}

int baseModel::returnColumnChoice(void)

{

return valColumnChoice;

}

int baseModel::returnSearchChoice(void)

{

return valSearchChoice;

}

std::string baseModel::returnQuery(void)

{

return query;

}

std::vector<std::string> baseModel::getListTables(void)

{

baseModel::baseGetTablesFromChoice();

return vertorTableNames;

}

std::vector<std::string> baseModel::getListColums(void)

{

return vectorColumnsNames;

}

std::vector<std::vector<std::string>> baseModel::getListTableContents(void)

{

return vectorTableData;

}

std::string baseModel::returnColumn(int \_ind)

{

return vectorColumnsNames.at(\_ind);

}

std::string baseModel::returnTable(int \_ind)

{

return vertorTableNames.at(\_ind);

}

int baseModel::returnDataType(int \_ind)

{

return vectorColumnVarTypes.at(\_ind);

}

int baseModel::returnSizeOfListColumn(void)

{

return vectorColumnsNames.size();

}

baseSQLlink baseModel::baseConnect(void)

{

std::string result = "user=" + baseUsername;

if (basePassword != "0")

{

result = result + " password=" + basePassword;

}

result = result + " dbname=" + baseName;

std::cout << " > Connecting with params: " << result << std::endl;

baseConnection = PQconnectdb(result.c\_str());

if (PQstatus(baseConnection) == CONNECTION\_BAD)

{

std::cout << PQerrorMessage(baseConnection) << std::endl;

std::cout << "(!) Connection failed!" << std::endl;

exit(0);

}

std::cout << "(\*) Connection successfull!" << std::endl;

std::cout << std::endl;

}

baseSQLlink baseModel::baseTerminate(void)

{

PQfinish(baseConnection);

std::cout << "(\*) Connection terminated!" << std::endl;

exit(0);

}

baseSQLlink baseModel::baseGetTablesFromChoice(void)

{

vertorTableNames.clear();

PGresult \*res = PQexec(baseConnection, "SELECT table\_name FROM information\_schema.tables "

"WHERE table\_schema = 'public'");

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << PQresultErrorMessage(res) << std::endl;

PQclear(res);

}

int rows = PQntuples(res);

for (int i = 0; i < PQntuples(res); i++)

{

vertorTableNames.push\_back(PQgetvalue(res, i, 0));

}

}

baseSQLlink baseModel::baseGetColumnsFromChoice(void)

{

vectorColumnsNames.clear();

vectorColumnVarTypes.clear();

std::string path = "SELECT \* FROM public.**\"**" + vertorTableNames.at(valTableChoice) + "**\"**";

PGresult \*res = PQexec(baseConnection, path.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< " >> No data retrieved from " << baseName << ". Abort function << " << std::endl;

PQclear(res);

}

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

{

vectorColumnsNames.push\_back(PQfname(res, i));

vectorColumnVarTypes.push\_back(PQftype(res, i));

}

PQclear(res);

}

int baseModel::baseGetColumnsFromQuery(void)

{

vectorColumnsNames.clear();

vectorColumnVarTypes.clear();

PGresult \*res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << PQresultErrorMessage(res) << std::endl;

PQclear(res);

return 0;

}

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

{

vectorColumnsNames.push\_back(PQfname(res, i));

vectorColumnVarTypes.push\_back(PQftype(res, i));

}

PQclear(res);

return 1;

}

baseSQLlink baseModel::baseQuerySend(void)

{

PGresult \*res = PQexec(baseConnection, "BEGIN");

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

{

printf("BEGIN command failed**\n**");

PQclear(res);

}

res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< PQerrorMessage(baseConnection) << std::endl;

PQclear(res);

}

res = PQexec(baseConnection, "COMMIT");

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

{

printf("COMMIT command failed**\n**");

PQclear(res);

}

}

baseSQLlink baseModel::baseGetTableContents(void)

{

vectorTableData.clear();

baseGetTablesFromChoice();

baseGetColumnsFromChoice();

std::string path = "SELECT \* FROM public.**\"**" + vertorTableNames.at(valTableChoice) + "**\"**";

path += " ORDER BY **\"**" + vectorColumnsNames.at(0) + "**\"**";

PGresult \*res = PQexec(baseConnection, path.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << "No data retrieved. Abort function" << std::endl;

PQclear(res);

}

valCountRow = PQntuples(res);

valCountColumn = PQnfields(res);

std::vector<std::string> lsttmp;

for (uint16\_t i = 0; i < valCountRow; i++)

{

for (uint16\_t j = 0; j < valCountColumn; j++)

{

lsttmp.push\_back(PQgetvalue(res, i, j));

}

vectorTableData.push\_back(lsttmp);

lsttmp.clear();

}

PQclear(res);

}

int baseModel::baseGetResultFromQuery(void)

{

vectorTableData.clear();

baseGetColumnsFromQuery();

PGresult \*res = PQexec(baseConnection, query.c\_str());

if (PQresultStatus(res) != PGRES\_TUPLES\_OK)

{

std::cout << std::endl

<< PQerrorMessage(baseConnection) << std::endl;

PQclear(res);

return 0;

}

valCountRow = PQntuples(res);

valCountColumn = PQnfields(res);

std::vector<std::string> lsttmp;

for (uint16\_t i = 0; i < valCountRow; i++)

{

for (uint16\_t j = 0; j < valCountColumn; j++)

{

lsttmp.push\_back(PQgetvalue(res, i, j));

}

vectorTableData.push\_back(lsttmp);

lsttmp.clear();

}

return 1;

}

*baseController.h*

#ifndef \_BASE\_CONTROLLER\_H\_

#define \_BASE\_CONTROLLER\_H\_

#include <iostream>

#include <unordered\_map>

#include "baseModel.h"

#include "baseView.h"

namespace optionsMenu

{

enum userChoice : int

{

MENU\_INSERT = 1,

MENU\_REMOVE,

MENU\_EDIT,

MENU\_RAND,

MENU\_SEARCH,

MENU\_SHOWTABLE,

MENU\_TERMINATE = 0,

MENU\_DEFAULT = 10

};

};

namespace Controller

{

class baseController

{

private:

*/\* Перегляд вказівника об’єкта \*/*

View::baseView \*viewObj;

*/\* Вказівник об’єкта моделі \*/*

Model::baseModel \*modObj;

public:

baseController(void);

baseController(Model::baseModel \*\_mdl, View::baseView \*\_vw);

~baseController();

*/\* Функція вибору меню \*/*

int menuChoiceHandler(void);

*/\* Функція вибору таблиці \*/*

int tableChoiceHandler(void);

*/\* Визначення типів даних, отриманих для бази даних \*/*

void concatVarTypes(std::string \*\_str, int \_iter);

*/\* Логічні кроки \*/*

*/\* На цьому кроці ви встановлюєте з’єднання з базою даних \*/*

void stepEstablishConnection(void);

*/\* На цьому кроці ви вибираєте, яка операція буде виконана з базою даних \*/*

void stepMenu(void);

*/\* На цьому кроці вибрана операція буде виконана з базою даних\*/*

void stepDBOperations(void);

*/\* Операції з базою даних \*/*

*/\* Виконати вставку даних в базу даних \*/*

void performDataInsert(void);

*/\* Виконати видалення даних в базі даних \*/*

void performDataDelete(void);

*/\* Виконати довільну вставку даних у базу даних\*/*

void performDataInsertRandom(void);

*/\* Виконати редагування даних у базі даних \*/*

void performDataEdit(void);

*/\* Виконати пошук даних у базі даних \*/*

void performDataSearch(void);

*/\* Відображення даних у базі даних \*/*

void performDataShow(void);

*/\* Виконати від'єднання від бази даних \*/*

void performTerminateConnection(void);

};

};

#endif //\_BASE\_CONTROLLER\_H\_

*baseController.cpp*

#include "baseController.h"

using namespace Controller;

baseController::baseController(void)

{

}

baseController::baseController(Model::baseModel \*\_mdl, View::baseView \*\_vw)

{

viewObj = \_vw;

modObj = \_mdl;

}

baseController::~baseController(void)

{

}

int baseController::menuChoiceHandler(void)

{

while (1)

{

int input = viewObj->inputIntMenu();

switch (input)

{

case optionsMenu::MENU\_INSERT:

case optionsMenu::MENU\_REMOVE:

case optionsMenu::MENU\_EDIT:

case optionsMenu::MENU\_RAND:

case optionsMenu::MENU\_SEARCH:

case optionsMenu::MENU\_SHOWTABLE:

case optionsMenu::MENU\_TERMINATE:

return input;

break;

default:

std::cout << " Wrong action. Try again." << std::endl;

break;

}

}

return optionsMenu::MENU\_DEFAULT;

}

int baseController::tableChoiceHandler(void)

{

std::vector<std::string> lst = modObj->getListTables();

while (1)

{

int input = viewObj->inputInt();

if (input >= 0 && input < lst.size())

{

return input;

}

}

}

void baseController::concatVarTypes(std::string \*\_str, int \_iter)

{

switch (modObj->returnDataType(\_iter))

{

case 23:

\*\_str += "::integer";

break;

case 1043:

\*\_str += "::character varying";

break;

default:

break;

}

}

void baseController::stepEstablishConnection(void)

{

std::cout << std::endl

<< "**\t\t** MAISTRENKO OLGA KV-02 LAB2**\n**"

<< std::endl

<< std::endl;

modObj->baseConnect();

}

void baseController::stepMenu(void)

{

modObj->setMenuChoice(menuChoiceHandler());

}

void baseController::stepDBOperations(void)

{

*/\* Показати на екрані те, що вибрав користувач \*/*

viewObj->viewMenubaseOperations(modObj->returnMenuChoice());

*/\* Виконати дію на вибір користувача \*/*

switch (modObj->returnMenuChoice())

{

case optionsMenu::MENU\_INSERT:

*/\* Початок вставки \*/*

performDataInsert();

break;

case optionsMenu::MENU\_REMOVE:

*/\* Початок видалення \*/*

performDataDelete();

break;

case optionsMenu::MENU\_EDIT:

*/\* Початок редагування \*/*

performDataEdit();

break;

case optionsMenu::MENU\_RAND:

*/\* Початок рандомної вставки \*/*

performDataInsertRandom();

break;

case optionsMenu::MENU\_SEARCH:

*/\* Початок пошуку \*/*

performDataSearch();

break;

case optionsMenu::MENU\_SHOWTABLE:

*/\* Початок виводу таблиці\*/*

performDataShow();

break;

case optionsMenu::MENU\_TERMINATE:

*/\* Перервати з'єднання \*/*

performTerminateConnection();

break;

default:

break;

}

}

void baseController::performDataInsert(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Показати список доступних стовпців \*/*

viewObj->viewList(modObj->getListColums());

std::string comSQL = "INSERT INTO public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** (";

*/\* Збірка команди \*/*

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

comSQL += "**\"**";

comSQL += modObj->returnColumn(i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += "**\"**, ";

}

else

{

comSQL += "**\"**) ";

}

}

comSQL += "VALUES (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

std::cout << "Enter value for " << modObj->returnColumn(i) << " column" << std::endl;

std::string inp = viewObj->getString();

comSQL += "'" + inp + "'";

concatVarTypes(&comSQL, i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += ", ";

}

else

{

comSQL += ")";

}

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

void baseController::performDataDelete(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

std::cout << "Choose ID to delete" << std::endl;

std::string comSQL = "DELETE FROM public.**\"**" + modObj->returnTable(modObj->returnTableChoice());

comSQL += "**\"** WHERE **\"**" + modObj->returnColumn(0) + "**\"**";

comSQL += " IN (" + viewObj->getString() + ")";

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

void baseController::performDataInsertRandom(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

std::string comSQL = "INSERT INTO public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

comSQL += "**\"**";

comSQL += modObj->returnColumn(i);

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += "**\"**, ";

}

else

{

comSQL += "**\"**) ";

}

}

comSQL += "VALUES (";

for (int i = 1; i < modObj->returnSizeOfListColumn(); i++)

{

switch (modObj->returnDataType(i))

{

case 23:

comSQL += "random()\* (100-1 + 1) + 1";

break;

case 1043:

comSQL += "substr(md5(random()::text), 0, 10)";

break;

default:

break;

}

if (i != modObj->returnSizeOfListColumn() - 1)

{

comSQL += ", ";

}

else

{

comSQL += ")";

}

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

std::cout << " Choose how many rows of random data will be generated: ";

int \_quantity = viewObj->inputInt();

for (int i = 0; i < \_quantity; i++)

{

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

}

void baseController::performDataEdit(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Отримати вміст таблиці з бази даних \*/*

modObj->baseGetTableContents();

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

std::string comSQL = "UPDATE public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"** SET ";

std::cout << "Choose ID to edit" << std::endl;

*/\* Виберіть елемент для редагування \*/*

std::string strID = viewObj->getString();

std::cout << "Choose column to edit" << std::endl;

*/\* Показати список доступних стовпців \*/*

viewObj->viewList(modObj->getListColums());

*/\* Виберіть стовпець для редагування \*/*

modObj->setColumnChoice(viewObj->inputInt());

comSQL += "**\"**" + modObj->returnColumn(modObj->returnColumnChoice()) + "**\"** = ";

std::cout << "Set value: " << std::endl;

comSQL += "'" + viewObj->getString() + "'";

*/\* Визначити тип даних \*/*

concatVarTypes(&comSQL, modObj->returnColumnChoice());

comSQL += " WHERE **\"**" + modObj->returnColumn(0) + "**\"** = " + strID;

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит до бази даних \*/*

modObj->baseQuerySend();

}

void baseController::performDataSearch(void)

{

*/\* Виберіть варіант пошуку \*/*

viewObj->viewSearchOptions();

*/\* Встановити опцію пошуку в моделі \*/*

modObj->setSearchChoice(viewObj->inputInt());

std::string comSQL;

switch (modObj->returnSearchChoice())

{

case 1:

comSQL += "SELECT \* ";

comSQL += "FROM public.**\"**Group**\"** ";

comSQL += "WHERE **\"**capacity**\"** > ";

std::cout << "Set start range: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " AND **\"**capacity**\"** < ";

std::cout << "Set end range: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**capacity**\"**";

break;

case 2:

comSQL += "SELECT **\"**student\_id**\"**, **\"**name**\"**, **\"**surname**\"**, **\"**code**\"**, **\"**number**\"** ";

comSQL += "FROM public.**\"**Student**\"** ";

comSQL += "JOIN public.**\"**Group**\"** ON **\"**Student**\"**.**\"**group\_id**\"** = **\"**Group**\"**.**\"**group\_id**\"** ";

comSQL += "WHERE **\"**code**\"** = ";

std::cout << "Set group code: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**code**\"**";

break;

case 3:

comSQL += "SELECT **\"**Subject**\"**.**\"**name**\"**, **\"**Lector**\"**.**\"**name**\"**, **\"**surname**\"**, **\"**code**\"**, **\"**Group**\"**.**\"**number**\"**, **\"**Classroom**\"**.**\"**number**\"** ";

comSQL += "FROM public.**\"**Student\_Base**\"** ";

comSQL += "JOIN public.**\"**Group**\"** ON **\"**Student\_Base**\"**.**\"**group\_id**\"** = **\"**Group**\"**.**\"**group\_id**\"** ";

comSQL += "JOIN public.**\"**Lector**\"** ON **\"**Student\_Base**\"**.**\"**lector\_id**\"** = **\"**Lector**\"**.**\"**lector\_id**\"** ";

comSQL += "JOIN public.**\"**Subject**\"** ON **\"**Student\_Base**\"**.**\"**subject\_id**\"** = **\"**Subject**\"**.**\"**subject\_id**\"** ";

comSQL += "JOIN public.**\"**Classroom**\"** ON **\"**Student\_Base**\"**.**\"**class\_id**\"** = **\"**Classroom**\"**.**\"**class\_id**\"** ";

comSQL += "WHERE **\"**code**\"** = ";

std::cout << "Set group code: ";

comSQL += "'" + viewObj->getString() + "'";

comSQL += " ORDER BY **\"**Group**\"**.**\"**number**\"**";

break;

default:

break;

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит і записати результати до векторів у моделі \*/*

if (modObj->baseGetResultFromQuery() != 0)

{

*/\* Показати дані з векторної таблиці в моделі \*/*

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

}

}

void baseController::performDataShow(void)

{

*/\* Показати список доступних таблиць \*/*

viewObj->viewList(modObj->getListTables());

*/\* Вибір між таблицями \*/*

modObj->setTableChoice(tableChoiceHandler());

*/\* Початок створення запиту \*/*

std::string comSQL = "SELECT \* FROM public.**\"**" + modObj->returnTable(modObj->returnTableChoice()) + "**\"**";

std::cout << "Set LIMIT: (0 to show everything): " << std::endl;

*/\* Встановити обмеження для рядків \*/*

int LIMIT = viewObj->inputInt();

switch (LIMIT)

{

case 0:

break;

default:

comSQL += " LIMIT " + std::to\_string(LIMIT);

break;

}

*/\* Показати складену команду \*/*

std::cout << " (##) Result command: ";

viewObj->viewString(comSQL);

*/\* Встановити рядок запиту \*/*

modObj->setQuery(comSQL);

*/\* Надіслати запит і записати результати до векторів у моделі \*/*

if (modObj->baseGetResultFromQuery() != 0)

{

*/\* Показати дані з векторної таблиці в моделі \*/*

viewObj->viewTableContents(modObj->getListTableContents(), modObj->getListColums());

}

}

void baseController::performTerminateConnection(void)

{

modObj->baseTerminate();

}

*baseView.h*

#ifndef \_BASE\_VIEW\_H\_

#define \_BASE\_VIEW\_H\_

#include <iostream>

#include <string>

#include <iomanip>

#include <vector>

namespace View

{

class baseView

{

private:

public:

baseView();

~baseView();

*/\* Користувацький інтерфейс \*/*

*/\* Показати головне меню \*/*

void viewMenuMain(void);

*/\* Показати, що користувач вибрав у меню \*/*

void viewMenubaseOperations(const int \_ch);

*/\* Показати текст для параметрів пошуку \*/*

void viewSearchOptions(void);

*/\* Показати змінні \*/*

*/\* Показати рядок у консолі\*/*

void viewString(const std::string \_str);

*/\* Показати вміст вектора в консолі \*/*

void viewList(const std::vector<std::string> \_lst);

*/\* Показати вміст двовимірного вектора в консолі \*/*

void viewTableContents(const std::vector<std::vector<std::string>> \_lst1, const std::vector<std::string> \_lst2);

*/\* Взаємодія з користувачем \*/*

*/\* Отримати введені рядки від користувача \*/*

std::string getString(void);

*/\* Отримати введення числа від користувача \*/*

int inputInt(void);

*/\* Отримати введення числа від користувача у меню \*/*

int inputIntMenu(void);

};

};

#endif //\_BASE\_VIEW\_H\_

*baseView.cpp*

#include "baseView.h"

#include "baseController.h"

using namespace View;

baseView::baseView(void)

{

}

baseView::~baseView(void)

{

}

void baseView::viewString(const std::string \_str)

{

std::cout << \_str << std::endl;

}

void baseView::viewMenuMain(void)

{

std::cout << std::endl;

std::cout << " Welcome to your database! What do you want to do?" << std::endl;

std::cout << " >> 1. Insert data in table" << std::endl;

std::cout << " >> 2. Delete data from table" << std::endl;

std::cout << " >> 3. Edit data in table" << std::endl;

std::cout << " >> 4. Insert N rows of random data in table" << std::endl;

std::cout << " >> 5. Search data in tables" << std::endl;

std::cout << " >> 6. Show N table" << std::endl;

std::cout << std::endl

<< " >> 0. Stop connection and exit the program" << std::endl;

}

void baseView::viewList(const std::vector<std::string> \_lst)

{

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

{

std::cout << " >> " << i << ". " << \_lst.at(i) << std::endl;

}

}

std::string baseView::getString(void)

{

std::string str;

std::cout << " <INPUT>: ";

std::getline(std::cin >> std::ws, str);

return str;

}

int baseView::inputInt(void)

{

int inp = 0;

std::cout << " <INPUT>: ";

std::cin >> inp;

return inp;

}

int baseView::inputIntMenu(void)

{

viewMenuMain();

return inputInt();

}

void baseView::viewMenubaseOperations(const int \_ch)

{

switch (\_ch)

{

case optionsMenu::MENU\_INSERT:

std::cout << " [You chose to insert data in table]" << std::endl;

std::cout << " Choose table:" << std::endl;

break;

case optionsMenu::MENU\_REMOVE:

std::cout << " [You chose to delete data in table]" << std::endl;

std::cout << " Choose table:" << std::endl;

break;

case optionsMenu::MENU\_EDIT:

std::cout << " [You chose to edit data in table]" << std::endl;

std::cout << " Choose table:" << std::endl;

break;

case optionsMenu::MENU\_RAND:

std::cout << " [You chose to insert N rows of random data in table]" << std::endl;

std::cout << " Choose table:" << std::endl;

break;

case optionsMenu::MENU\_SEARCH:

std::cout << " [You chose to search data in tables]" << std::endl;

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

break;

case optionsMenu::MENU\_SHOWTABLE:

std::cout << " [You chose to Show N table]" << std::endl;

std::cout << " Choose table:" << std::endl;

break;

case optionsMenu::MENU\_TERMINATE:

std::cout << " [You chose to top connection and exit the program]" << std::endl;

break;

default:

std::cout << " There is no such option. Try again!" << std::endl;

break;

}

}

void baseView::viewSearchOptions(void)

{

std::cout << std::endl;

std::cout << ">> 1. Search Groups by max Students N > X; N < Y capacity" << std::endl;

std::cout << ">> 2. Search Students by Group name" << std::endl;

std::cout << ">> 3. Search Subjects by Group name" << std::endl;

}

void baseView::viewTableContents(const std::vector<std::vector<std::string>> \_lst1, const std::vector<std::string> \_lst2)

{

int spaces = 30;

for (int i = 0; i < \_lst2.size(); i++)

{

std::cout << std::setw(spaces) << \_lst2.at(i);

}

std::cout << std::endl

<< std::endl;

for (int i = 0; i < \_lst1.size(); i++)

{

for (int j = 0; j < \_lst1[i].size(); j++)

{

std::cout << std::setw(spaces) << \_lst1[i][j];

}

std::cout << std::endl;

for (int k = 0; k < \_lst2.size() \* spaces; k++)

{

if (k > 8)

{

std::cout << "-";

}

else

{

std::cout << " ";

}

}

std::cout << std::endl;

}

}