

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота № 2**

з дисципліни «Бази даних і засоби управління»

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

Виконала: Садирова Д.А.

Студентка групи КВ-91

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 «IT-компанія»**

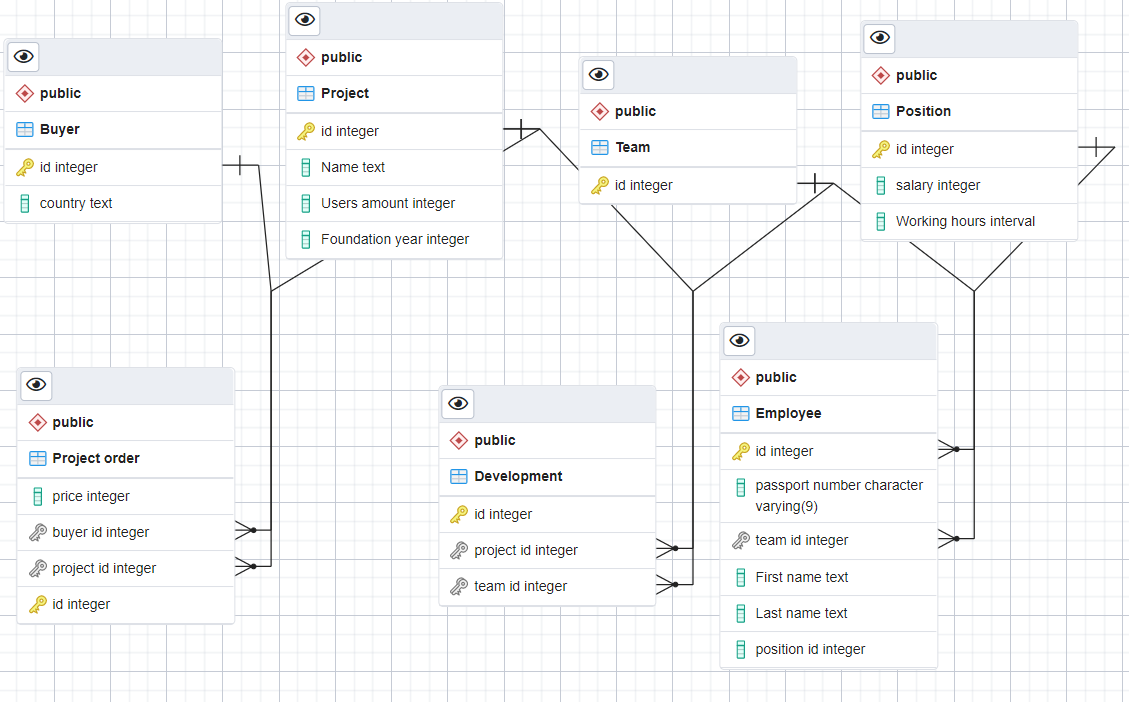


рисунок 1 – логічна модель предметної області «IT-компанія»

**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. Згідно компоненту моделі, у моій програмі відповідають всі компоненти які знаходять у папці Models.

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

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

**Структура програми та її опис**

На рисунку 1 відображено деревовидну структуру програми:

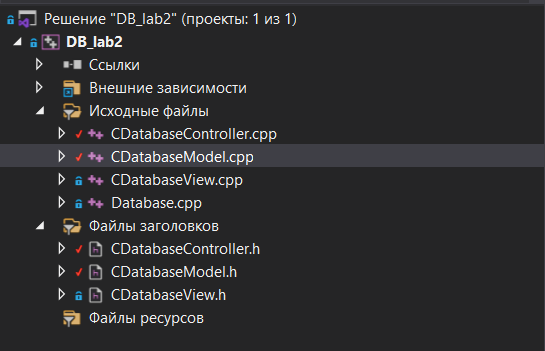


Рисунок 2 – структура програми

Програма умовно поділена на 4 модулі: класс CDatabaseModel, клас CDatabaseController, клас CDatabaseView та головний файл Database.cpp.

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

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

У файлах CDatabaseController.h(.cpp) описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

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

**Структура меню програми**

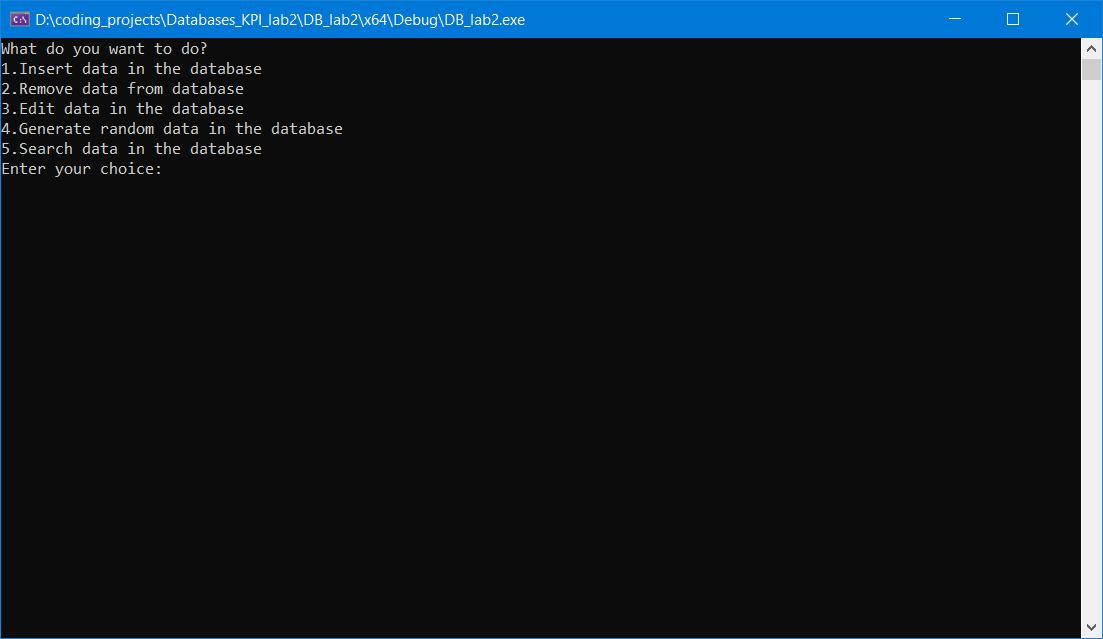


Рисунок 3 – Початкове меню програми

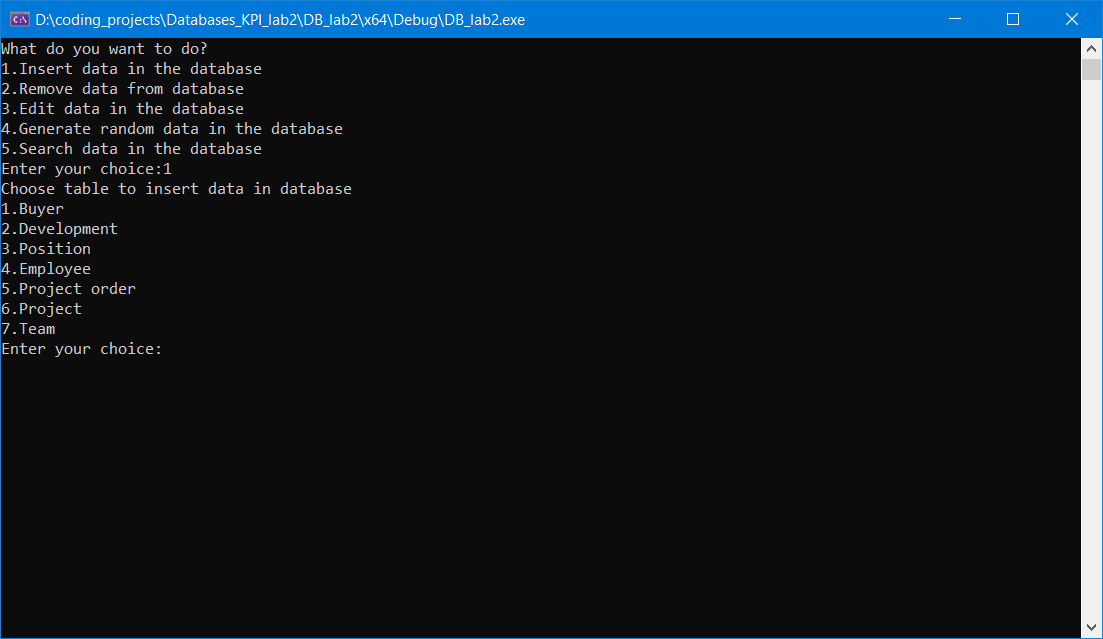


Рисунок 4 – Меню для додавання даних

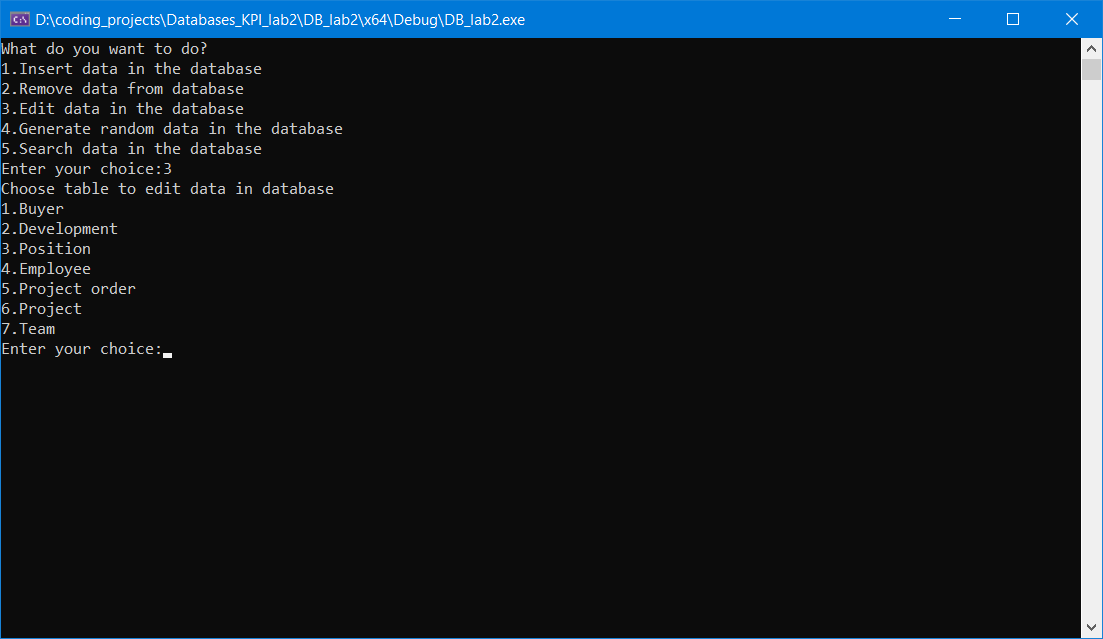


Рисунок 5 – Меню для редагування даних

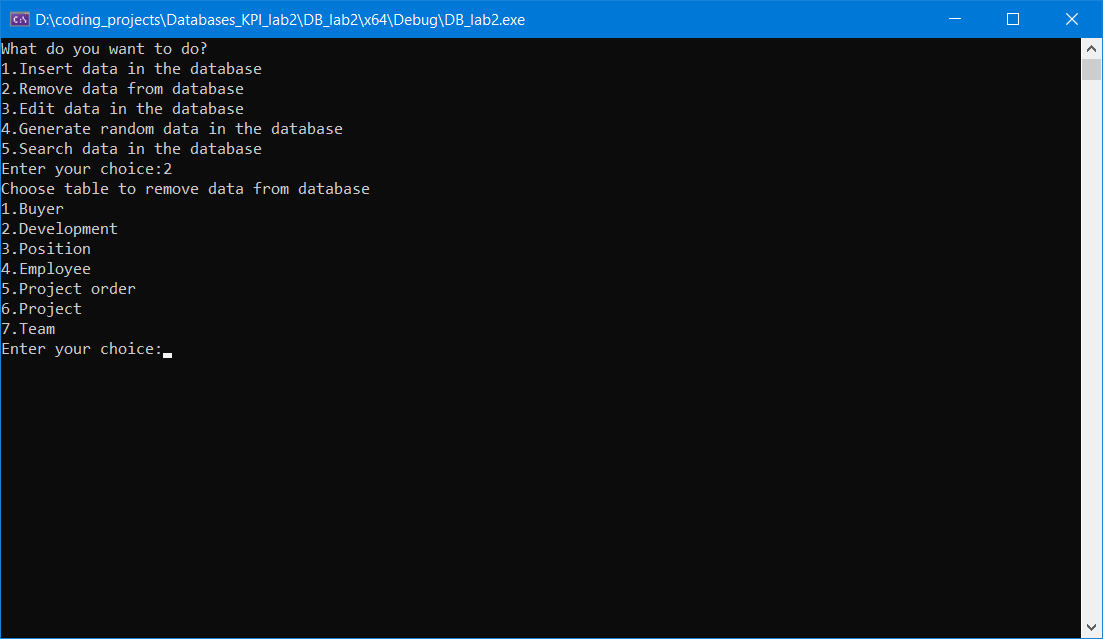


Рисунок 6 – Меню для вилучення даних

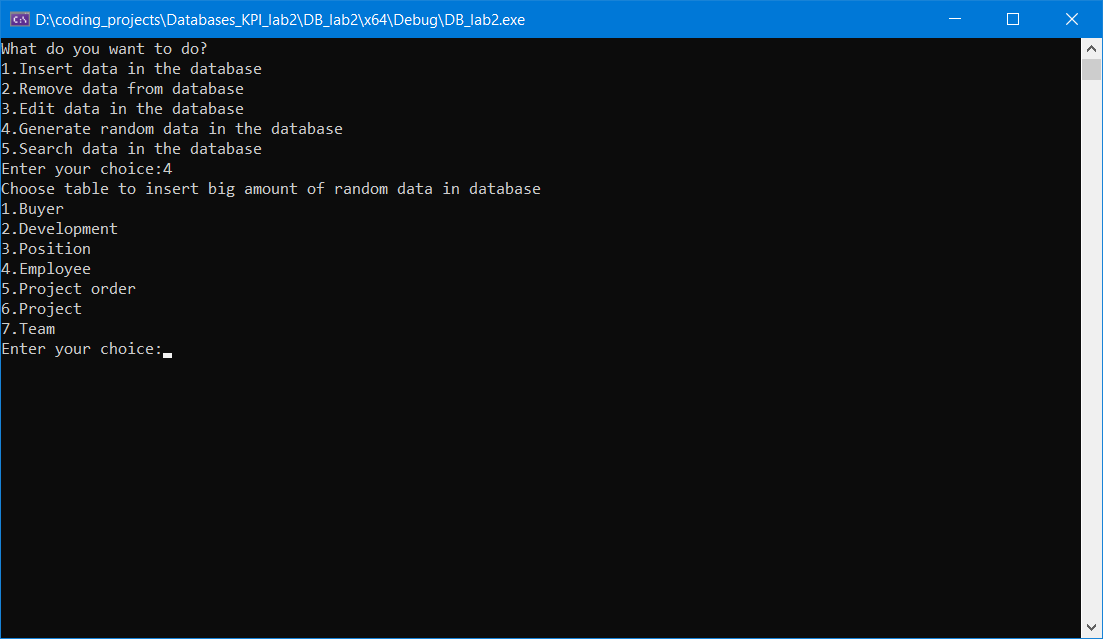


Рисунок 7 – Меню для заповнення даних випадковим чином

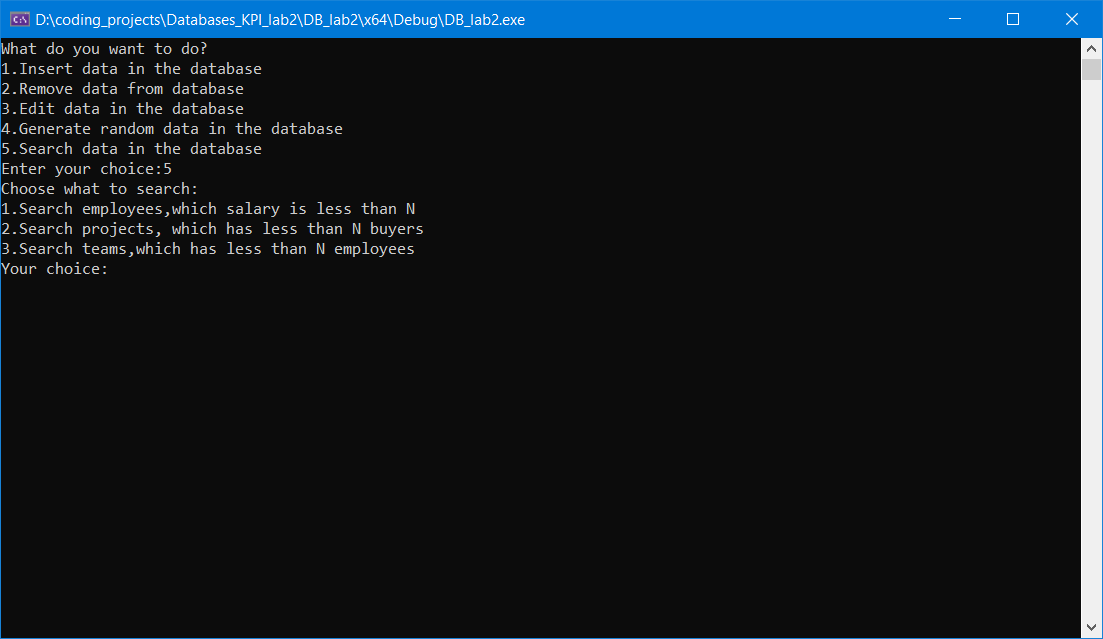


Рисунок 8 – Меню для пошуку даних

Меню для користувача складається з п’яти пунктів(Рисунок 2).

Перший пункт пропонує внесення даних(Рисунок 3). Перед тим як внести дані необхідно ще вибрати для якої таблиці саме буде відбуватись внесення. Тому користувач має вибрати опцію. Після цього користувач може вносити дані.

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

Третій пункт пропонує редагування даних(Рисунок 4). Перед тим як редагувати дані необхідно ще вибрати для якої моделі саме буде відбуватись редагування. Тому користувач має вибрати опцію. Після цього користувач може редагувати дані.

Четвертий пункт пропонує заповнення даних випадковим чином(Рисунок 6). Перед тим як заповнити дані випаковим чином необхідно ще вибрати для якої талиці саме буде відбуватись заповнення. Тому користувач має вибрати опцію.

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

**Фрагменти програм внесення, редагування та вилучення даних у базі даних**

-Фрагмент програми для внесення даних

PGresult\* CDatabaseController::performInsert(int tableIndex) {

auto cols = m\_model->columnsInTable(tableIndex);

auto types = m\_model->dataTyperInTable(tableIndex);

std::vector<std::string> userData;

int indx = 0;

for (auto col : cols) {

auto key = model::dataTypesMap.find(types[indx++]);

if (key != model::dataTypesMap.end()) {

userData.push\_back(requestData(col, key->second));

}

else

std::cout << "key is end!\n";

}

std::string userInputStr;

for (auto input : userData) {

userInputStr += std::string(input) + ",";

}

userInputStr.pop\_back();

userInputStr.push\_back(')');

userInputStr.insert(0, 1, '(');

std::cout << userInputStr;

std::string insertQuery = (boost::format(

"INSERT INTO \"%s\"\n"

"VALUES %s"

) % m\_model->tables()[tableIndex] % userInputStr).str();

auto\* insertQueryRes = m\_model->query(insertQuery.c\_str());

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

std::wcout << reinterpret\_cast<const wchar\_t\*>(m\_model->GetLastError().data());

return insertQueryRes;

}

-Фрагмент програми для редагування даних

PGresult\* CDatabaseController::performEdit(int tableIndex, int rowIndex) {

auto cols = m\_model->columnsInTable(tableIndex);

auto types = m\_model->dataTyperInTable(tableIndex);

auto row = m\_model->rowsInTable(tableIndex)[rowIndex];

std::vector<std::string> userInput;

std::string input;

std::getline(std::cin, input);

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

auto data = requestData(cols[i], typeFromString(types[i]), true);

if (data == "\*")

data = row[i];

userInput.push\_back(data);

}

std::string updateQueryStr = (boost::format("UPDATE public.\"%s\" SET ") % m\_model->tables()[tableIndex]).str();

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

const bool needParentheses =

typeFromString(types[i]) == model::dataTypes::interval ||

typeFromString(types[i]) == model::dataTypes::characterVarying ||

typeFromString(types[i]) == model::dataTypes::text;

const std::string updateVarstr = "\"" + (std::string(cols[i]) + "\"" + " = " +

(needParentheses ?

"'" + userInput[i] + "'" :

userInput[i]) + ", ");

updateQueryStr += updateVarstr;

}

updateQueryStr.pop\_back();

updateQueryStr.pop\_back();

updateQueryStr += "\nWHERE ";

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

const bool needParentheses =

typeFromString(types[i]) == model::dataTypes::interval ||

typeFromString(types[i]) == model::dataTypes::characterVarying ||

typeFromString(types[i]) == model::dataTypes::text;

const std::string currVarStr = "\"" + (std::string(cols[i]) + "\"" + " = " + (needParentheses ? "'" + std::string(row[i]) + "'" : row[i]) + (i == userInput.size() - 1 ? "" : " AND "));

updateQueryStr += currVarStr;

}

std::cout << updateQueryStr << std::endl;

auto res = m\_model->query(updateQueryStr);

return res;

}

-Фрагмент програми для видалення даних

PGresult\* CDatabaseController::performRemove(int tableIndex, int rowIndex) {

auto tables = m\_model->tables();

auto cols = m\_model->columnsInTable(tableIndex);

auto rows = m\_model->rowsInTable(tableIndex);

auto pKey = m\_model->getTablePrimaryKey(tableIndex);

auto pKeyIter = std::find\_if(cols.begin(), cols.end(), [&](const char\* col) {

return std::string(col) == pKey;

});

const int pKeyIndex = pKeyIter - cols.begin();

std::string removeQuery = (boost::format(

"DELETE FROM public.\"%s\"\n"

"WHERE \"%s\" = %s"

) % tables[tableIndex] % cols[pKeyIndex] % rows[rowIndex][pKeyIndex]).str();

std::cout << removeQuery << std::endl;

return m\_model->query(removeQuery);

}

Дані фрагменти програми, які наведені вижче, відповідають за функціонал додвання даних, редагування та вилучення даних у базі даних.  
Взаємодія відбувається через клас CDatabaseModel, який займається підключенням до БД, а самі функції знаходяться в класі CDatabaseController. Спочатку, виходячи з індексу таблиці, та номеру рядка, відбувається генерація SQL-запиту, а потім його виконання за допомогою CDatabaseModel::query().

**Лістинги програм с запитами пошуку**

-Пошук робітників за розміром ЗП

PGresult\* CDatabaseController::searchEmployees(int salary) {

PGresult\* res = nullptr;

auto tables = m\_model->tables();

auto iterEmployees = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Employee";

});

auto iterPositions = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Position";

});

auto positionCols = m\_model->columnsInTable(iterPositions - tables.begin());

auto employeesCols = m\_model->columnsInTable(iterEmployees - tables.begin());

std::string PositionsQuery = "SELECT ";

for (auto& i : positionCols)

PositionsQuery += "\"" + std::string(i) + "\",";

PositionsQuery.pop\_back();

PositionsQuery += "\nFROM public.\"Position\"\n";

PositionsQuery += (boost::format("WHERE \"salary\" < %d") % salary).str();

res = m\_model->query(PositionsQuery);

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

std::cout << "Error search positions!";

return res;

}

auto tpls = m\_model->getTuples(res);

std::vector<int> neededPositionsId;

for (auto& tuple : tpls) {

neededPositionsId.push\_back(std::stoi(tuple[0]));

}

//std::cout << "positions id:\n";

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

// std::cout << neededPositionsId[i] << std::endl;

//}

std::string employeesQuery = "SELECT ";

for (auto& i : employeesCols)

employeesQuery += "\"" + std::string(i) + "\",";

employeesQuery.pop\_back();

employeesQuery += "\nFROM public.\"Employee\"\n";

employeesQuery += "WHERE \"id\" = " + std::to\_string(neededPositionsId[0]);

for (int i = 1; i < neededPositionsId.size(); ++i) {

employeesQuery += " OR \"id\"= " + std::to\_string(neededPositionsId[i]);

}

//std::cout << employeesQuery << std::endl;

res = m\_model->query(employeesQuery);

return res;

}

Спочатку в таблиці Position шукаються робочі місця, зп яких менше ніж аргумент функції. Потім id робітників на цих позиціях використовуються для того щоб сформувати запит для пошуку робітників.

-Пошук проектів за кількістю покупців

PGresult\* CDatabaseController::searchProjects(int buyersCount) {

PGresult\* res = nullptr;

const auto tables = m\_model->tables();

const auto iterProjectOrders = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Project order";

});

const auto iterProjects = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Project";

});

auto projectOrderCols = m\_model->columnsInTable(iterProjectOrders - tables.begin());

constexpr int idIndexProject = 0;

std::unordered\_map<int, int> buyersForProject;

std::vector<std::pair<int, int>> usedPairs;

auto rowsOrders = m\_model->rowsInTable(iterProjectOrders - tables.begin());

auto rowsProjects = m\_model->rowsInTable(iterProjects - tables.begin());

for (auto& proj : rowsProjects) {

const int id = std::stoi(proj[0]);

buyersForProject[id] = 0;

for (auto& order : rowsOrders) {

const int currProjectId = std::stoi(order[3]);

if (currProjectId == id) {

const int currBuyerId = std::stoi(order[1]);

if (std::find(usedPairs.begin(), usedPairs.end(), std::pair<int,int>(currProjectId,currBuyerId)) == usedPairs.end()) {

buyersForProject[currProjectId]++;

usedPairs.push\_back(std::pair<int, int>(currProjectId, currBuyerId));

}

}

}

}

//std::cout << "projects and unique buyers:\n";

//for (auto& pair : buyersForProject) {

// std::cout << "project id: " << pair.first << "buyers: " << pair.second << std::endl;

//}

std::string projectsArray = "ANY(ARRAY[";

for (auto& pair : buyersForProject) {

if (pair.second < buyersCount)

projectsArray += std::to\_string(pair.first) + ",";

}

projectsArray.pop\_back();

projectsArray += "])";

std::string projectsQuery = "SELECT \* FROM public.\"Project\" WHERE \"id\" = " + projectsArray;

//std::cout << projectsQuery << std::endl;

res = m\_model->query(projectsQuery);

return res;

}

Спочатку в таблиці Project order шукаються замовлення, у яких покупців менше ніж число, яке передається як аргумент функції пошуку. Потім id проектів цих замовлень використовуються для того щоб сформувати запит пошуку відповідних рядків з таблиці Project.

-Пошук команд за кількістю працівників

PGresult\* CDatabaseController::searchTeams(int employeesCount) {

PGresult\* res = nullptr;

auto tables = m\_model->tables();

auto iterEmployees = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Employee";

});

auto iterTeams = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Team";

});

using team\_id = int;

std::unordered\_map<team\_id, int> teamEmployees;

auto rows = m\_model->rowsInTable(iterEmployees - tables.begin());

constexpr int teamIdIndex = 2;

for (auto& row : rows) {

teamEmployees[std::stoi(row[teamIdIndex])]++;

}

for (auto& pair : teamEmployees) {

std::cout << "team id: " << pair.first << " count: " << pair.second << std::endl;

}

std::string teamsArray = "ANY(ARRAY[";

for (auto& pair : teamEmployees) {

if (pair.second < employeesCount)

teamsArray += std::to\_string(pair.first) + ",";

}

teamsArray.pop\_back();

teamsArray += "])";

std::string teamsQuery = "SELECT \* FROM public.\"Team\" WHERE \"id\" = " + teamsArray;

//std::cout << teamsQuery << std::endl;

res = m\_model->query(teamsQuery);

return res;

}

Спочатку виконується прохід по таблиці робітників, і формується хеш-таблиця, де ключем є id команди, а значенням є кількість робітників. Коли хеш-таблиця сформована, звідти видобуваютсья команди з потрібною кількістю робітників для формування запиту.

**Лістинг фрагменту програми генерування випадкових даних в таблицях БД**

PGresult\* CDatabaseController::performGeneratingRandomData(int tableIndex) {

auto randomStrByType = [&](model::dataTypes dt) -> std::string {

using namespace model;

switch (dt) {

case dataTypes::characterVarying:

return "substr(md5(random()::text), 1, 9)";

case dataTypes::integer:

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

case dataTypes::interval:

return "INTERVAL '00:00:00' \* (random()\*10)::integer";

case dataTypes::text:

return "random()::text";

case dataTypes::unk:

return "";

}

};

auto types = m\_model->dataTyperInTable(tableIndex);

auto cols = m\_model->columnsInTable(tableIndex);

auto tables = m\_model->tables();

std::vector<std::string> generatingStrings;

for (auto& t : types) {

auto type = typeFromString(t);

generatingStrings.push\_back(randomStrByType(type));

}

std::string generateQuery = "INSERT INTO public.\"" + std::string(tables[tableIndex]) + "\"(";

for (auto& col : cols)

generateQuery += "\"" + std::string(col) + "\",";

generateQuery.pop\_back();

generateQuery += ")";

generateQuery += "\nSELECT";

for (auto& randStr : generatingStrings)

generateQuery += "\n" + randStr + ",";

generateQuery.pop\_back();

generateQuery += (boost::format("\nFROM generate\_series(1,%d)\nON CONFLICT DO NOTHING") % model::randomDataCount).str();

std::cout << generateQuery << std::endl;

return m\_model->query(generateQuery);

}

Лістинг функції для генерування випадкових даних в обраній таблиці. Спочатку, для кожного типу даних в обраній таблиці, за спеціальним шаблоном генерується рядок, який буде генерувати набір даних певного типу. Потім рядки для кожного стовпчика поєднуються, і на виході маємо готовий SQL-запит для генерації даних.

**Лістинг модуля "СDatabaseModel"**

*CDatabaseModel.h*

#pragma once

#include <libpq-fe.h>

#include <string>

#include <vector>

#include <fstream>

#include <unordered\_map>

namespace model {

using childTableRelation = std::pair<std::string, std::string>; //table name and relation field

using parentTableRelation = std::pair<std::string, std::string>; //table name and relation field

struct relation {

parentTableRelation parent;

childTableRelation child;

};

const std::vector<relation> relations = {

{{"Project order","buyer id"},{"Buyer","id"}},

{{"Project order","project id"},{"Project","id"}},

{{"Development","project id"},{"Project","id"}},

{{"Development","team id"},{"Team","id"}},

{{"Employee","team id"},{"Team","id"}},

{{"Employee","position id"},{"Position","id"}},

};

enum class dataTypes {

text,

integer,

characterVarying,

interval,

unk

};

constexpr int randomDataCount = 100000;

const std::unordered\_map<std::string, dataTypes> dataTypesMap = {

{"text",dataTypes::text},

{"integer",dataTypes::integer},

{"interval",dataTypes::interval},

{"character varying",dataTypes::characterVarying},

};

class CDatabaseModel {

public:

CDatabaseModel();

CDatabaseModel(const std::string& username, const std::string& db, const std::string& passw);

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

std::string GetLastError() const;

std::vector<const char\*> tables() const;

std::vector<const char\*> columnsInTable(int tableIndex);

std::vector<std::vector<const char\*>> rowsInTable(int tableIndex);

std::vector<const char\*> dataTyperInTable(int tableIndex);

std::vector<std::vector<std::string>> getTuples(PGresult\* res);

std::string getTablePrimaryKey(int tableIndex);

void SetUserName(const std::string& username);

void SetPassword(const std::string& passw);

void SetDbName(const std::string& db);

std::string userName() const;

std::string dbName() const;

std::string password() const;

private:

std::vector<const char\*> m\_tables;

std::vector<const char\*> retrieveTables(PGresult \*res);

void connect();

void terminate();

PGconn\* m\_connection = nullptr;

std::string m\_lastError;

std::string m\_userName;

std::string m\_dbName;

std::string m\_password;

};

}

*CDatabaseModel.cpp*

#include "CDatabaseModel.h"

#include <exception>

#include <iostream>

#include <locale>

#include <utility>

#include <boost/format.hpp>

using namespace model;

CDatabaseModel::CDatabaseModel() {}

CDatabaseModel::CDatabaseModel(const std::string& username, const std::string& db, const std::string& passw)

: m\_dbName(db),m\_password(passw),m\_userName(username) {

connect();

const char\* querySearchTables = "SELECT table\_name\n"

"FROM information\_schema.tables\n"

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

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

auto tablesQueryResult = query(querySearchTables);

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

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

std::cout << "Select failed: " << errMsg << std::endl;

}

else

m\_tables = retrieveTables(tablesQueryResult);

auto langQuery = "SET lc\_messages TO 'en\_US.UTF-8';";

auto langResult = query(langQuery);

}

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

//std::ofstream out("1.txt", std::fstream::out);

auto msg = PQerrorMessage(m\_connection);

//out << msg;

return msg;

}

void CDatabaseModel::SetUserName(const std::string& username) {

m\_userName = username;

}

void CDatabaseModel::SetPassword(const std::string& passw) {

m\_password = passw;

}

void CDatabaseModel::SetDbName(const std::string& db) {

m\_dbName = db;

}

std::string CDatabaseModel::userName() const {

return m\_userName;

}

std::string CDatabaseModel::dbName() const {

return m\_dbName;

}

std::string CDatabaseModel::password() const {

return m\_password;

}

void CDatabaseModel::connect() {

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

const std::string connection\_string =

"user=" + m\_userName +

" password=" + m\_password +

" host = 127.0.0.1 dbname=" + m\_dbName +

" client\_encoding = UTF8";

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

if (m\_connection == nullptr) {

terminate();

}

}

std::vector<const char\*> CDatabaseModel::retrieveTables(PGresult\* res) {

std::vector<const char\*> resVec;

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

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

auto name = PQgetvalue(res, i, j);

resVec.push\_back(name);

}

}

return resVec;

}

void CDatabaseModel::terminate() {

throw std::runtime\_error(GetLastError());

}

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

return PQexec(m\_connection, query.c\_str());

}

std::vector<const char\*> CDatabaseModel::tables() const {

return m\_tables;

}

std::vector<const char\*> CDatabaseModel::columnsInTable(int tableIndex) {

std::vector<const char\*> res;

std::string columnsQuery = (boost::format(

"SELECT column\_name\n"

"FROM information\_schema.columns\n"

"WHERE table\_schema = '%s'\n"

"AND table\_name = '%s';") % "public" % m\_tables[tableIndex]).str();

auto columnsQueryResult = query(columnsQuery);

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

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

auto name = PQgetvalue(columnsQueryResult, i, j);

res.push\_back(name);

}

}

return res;

}

std::vector<const char\*> CDatabaseModel::dataTyperInTable(int tableIndex) {

std::vector<const char\*> res;

std::string dataTyperQuery = (boost::format(

"SELECT data\_type\n"

"FROM information\_schema.columns\n"

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

"AND table\_name = '%s'") % m\_tables[tableIndex]).str();

auto dataTypesQueryResult = query(dataTyperQuery.c\_str());

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

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

auto name = PQgetvalue(dataTypesQueryResult, i, j);

res.push\_back(name);

}

}

//std::string charlength = (boost::format(

// "select character\_maximum\_length\n"

// "from INFORMATION\_SCHEMA.COLUMNS\n"

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

// "AND table\_name = '%s'") % m\_tables[tableIndex]).str();

//dataTypesQueryResult = query(dataTyperQuery.c\_str());

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

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

auto name = PQgetvalue(dataTypesQueryResult, i, j);

//std::cout << name << std::endl;

}

}

return res;

}

std::vector<std::vector<const char\*>> CDatabaseModel::rowsInTable(int tableIndex) {

std::vector<std::vector<const char\*>> res;

std::string dataQuery = (boost::format(

"select \*\n"

"from \"%s\";"

) % m\_tables[tableIndex]).str();

auto\* queryRes = query(dataQuery.c\_str());

//std::cout << dataQuery << std::endl;

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

std::cout << GetLastError() << "-";

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

res.push\_back(std::vector<const char\*>(PQnfields(queryRes)));

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

auto name = PQgetvalue(queryRes, i, j);

//std::cout << name << " ";

res[i][j] = name;

}

//std::cout << std::endl;

}

return res;

}

std::vector<std::vector<std::string>> CDatabaseModel::getTuples(PGresult\* res) {

std::vector<std::vector<std::string>> resVec;

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

resVec.push\_back(std::vector<std::string>(PQnfields(res)));

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

auto name = PQgetvalue(res, i, j);

//std::cout << name << " ";

resVec[i][j] = name;

}

}

return resVec;

}

std::string CDatabaseModel::getTablePrimaryKey(int tableIndex) {

auto\* tablename = m\_tables[tableIndex];

std::string queryPKey = (boost::format(

"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.\"%s\"'::regclass\n"

"AND i.indisprimary;") % tablename).str();

auto\* res = query(queryPKey);

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

return getTuples(res)[0][0];

}

return std::string();

}

**Результати роботи програми**

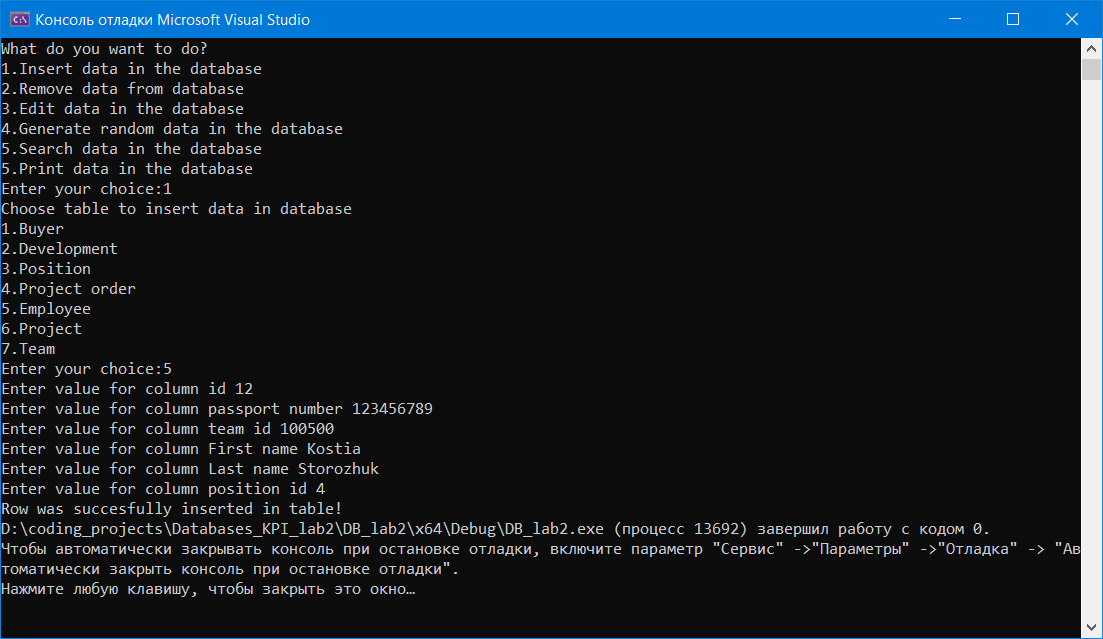


Рисунок 9 - Інтерфейс для введення даних

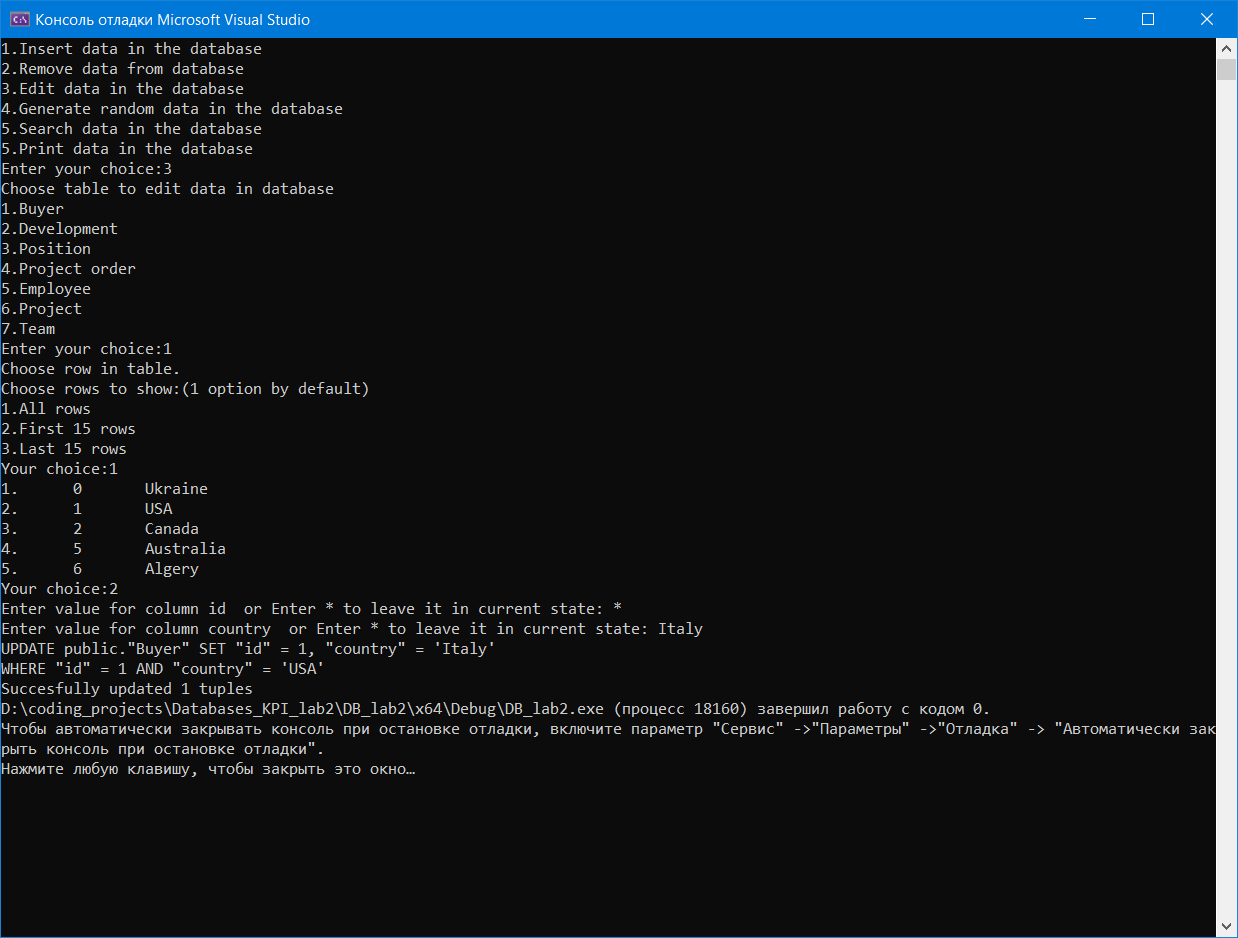


Рисунок 10 - Інтерфейс для редагування даних

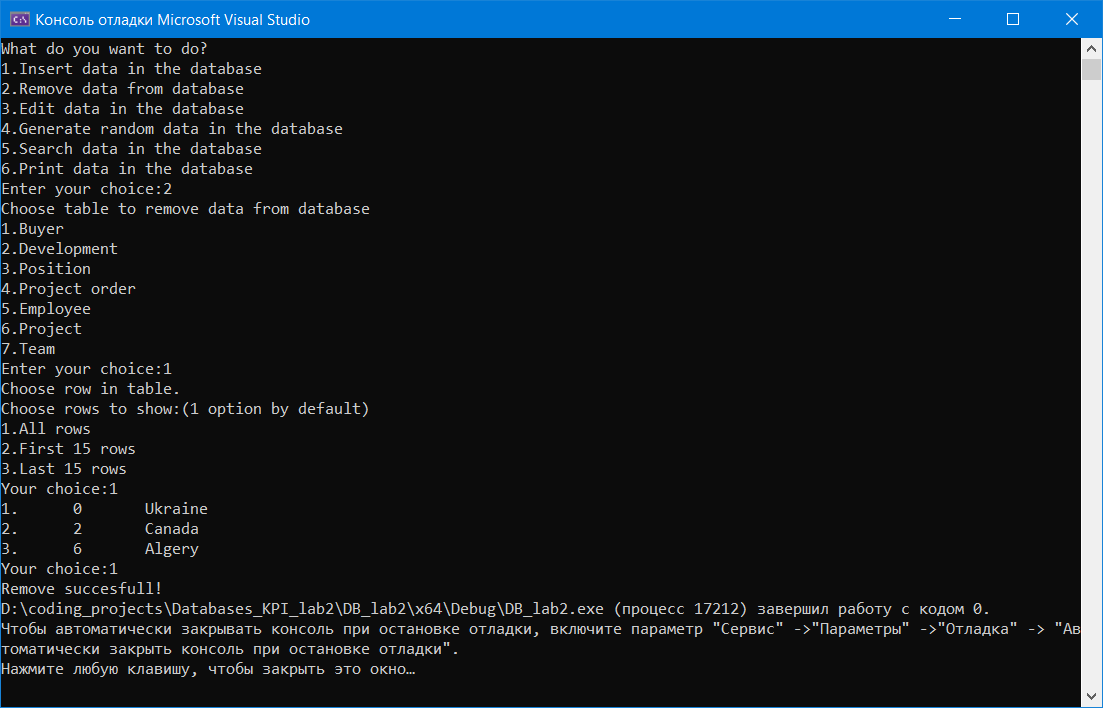


Рисунок 11 - Інтерфейс для видалення даних

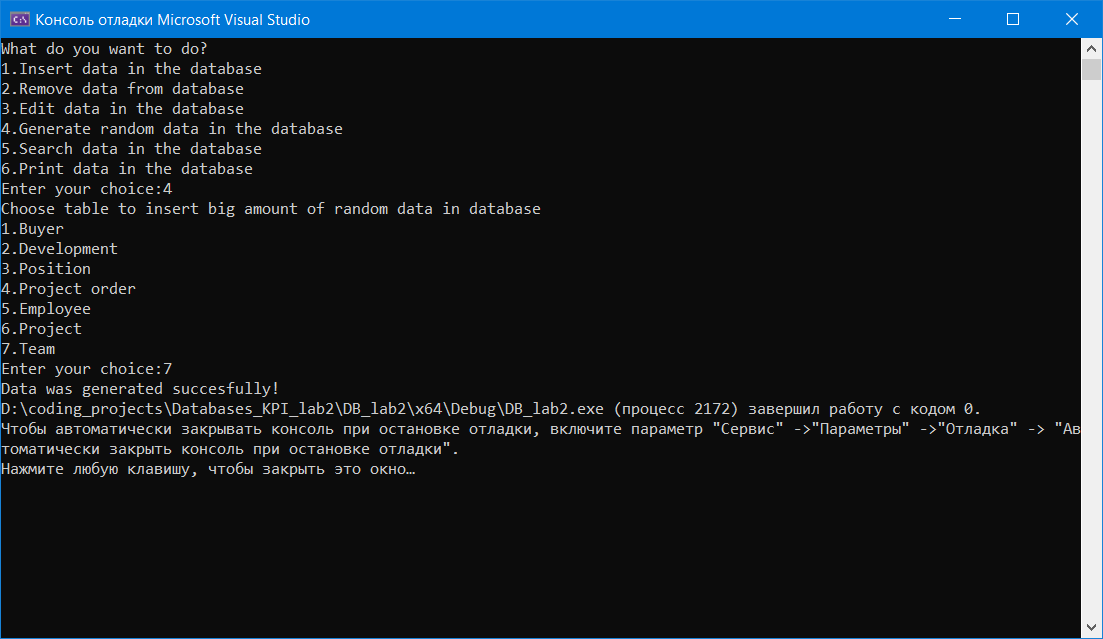


Рисунок 12 - Інтерфейс для заповнення даних випадковим чином

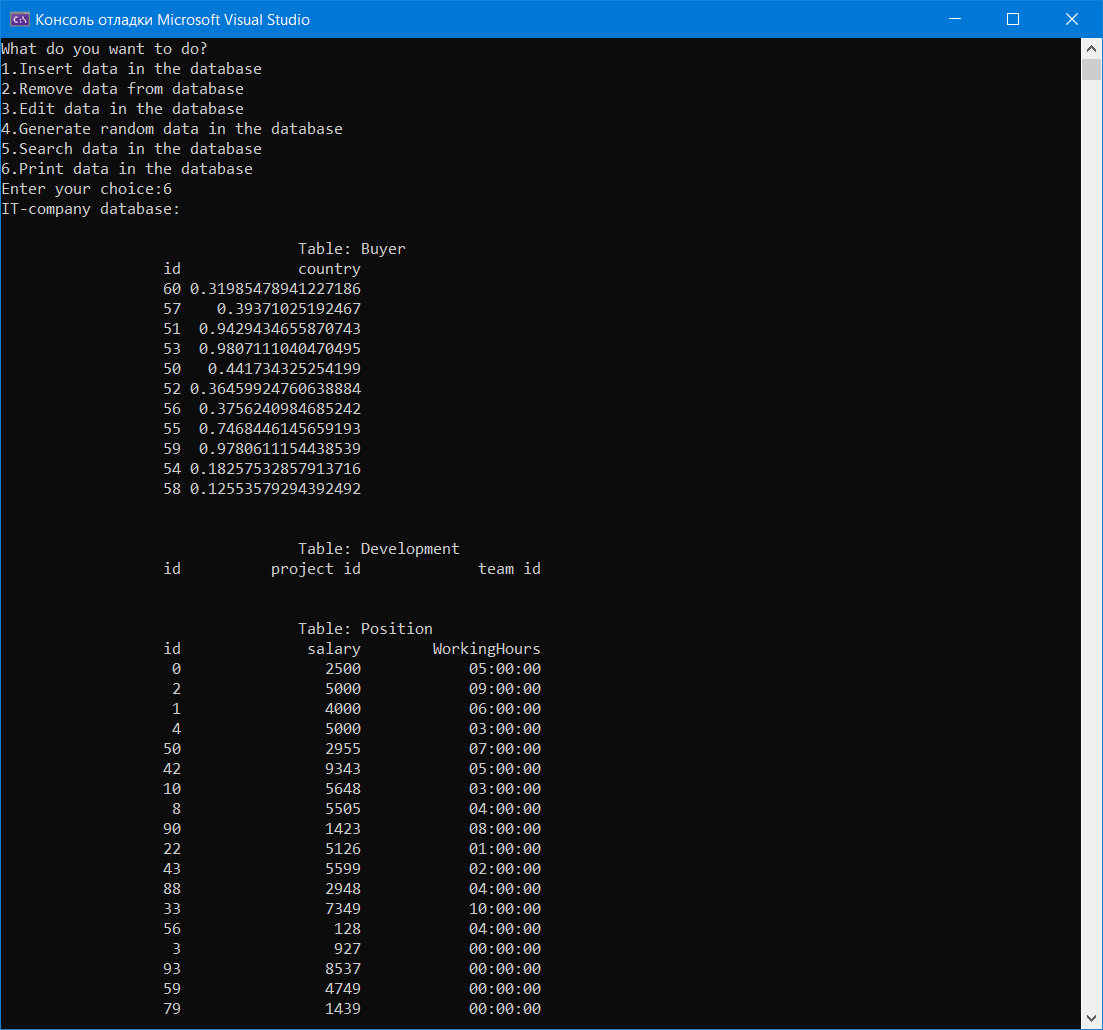


Рисунок 13.1: виведення таблиці, частина 1

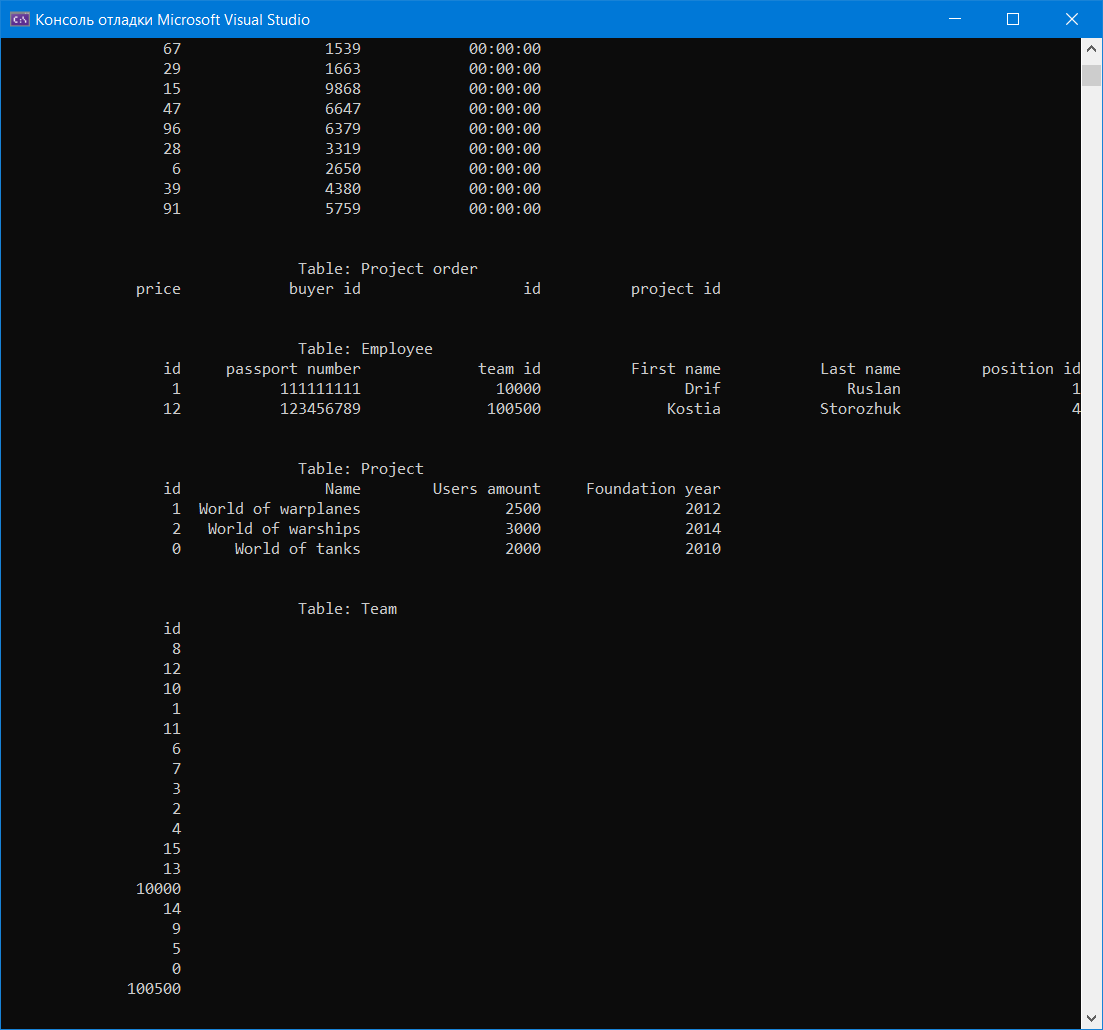


Рисунок 13.2: виведення таблиці, частина 2

**Ілюстрації валідації даних при уведені користувачем**

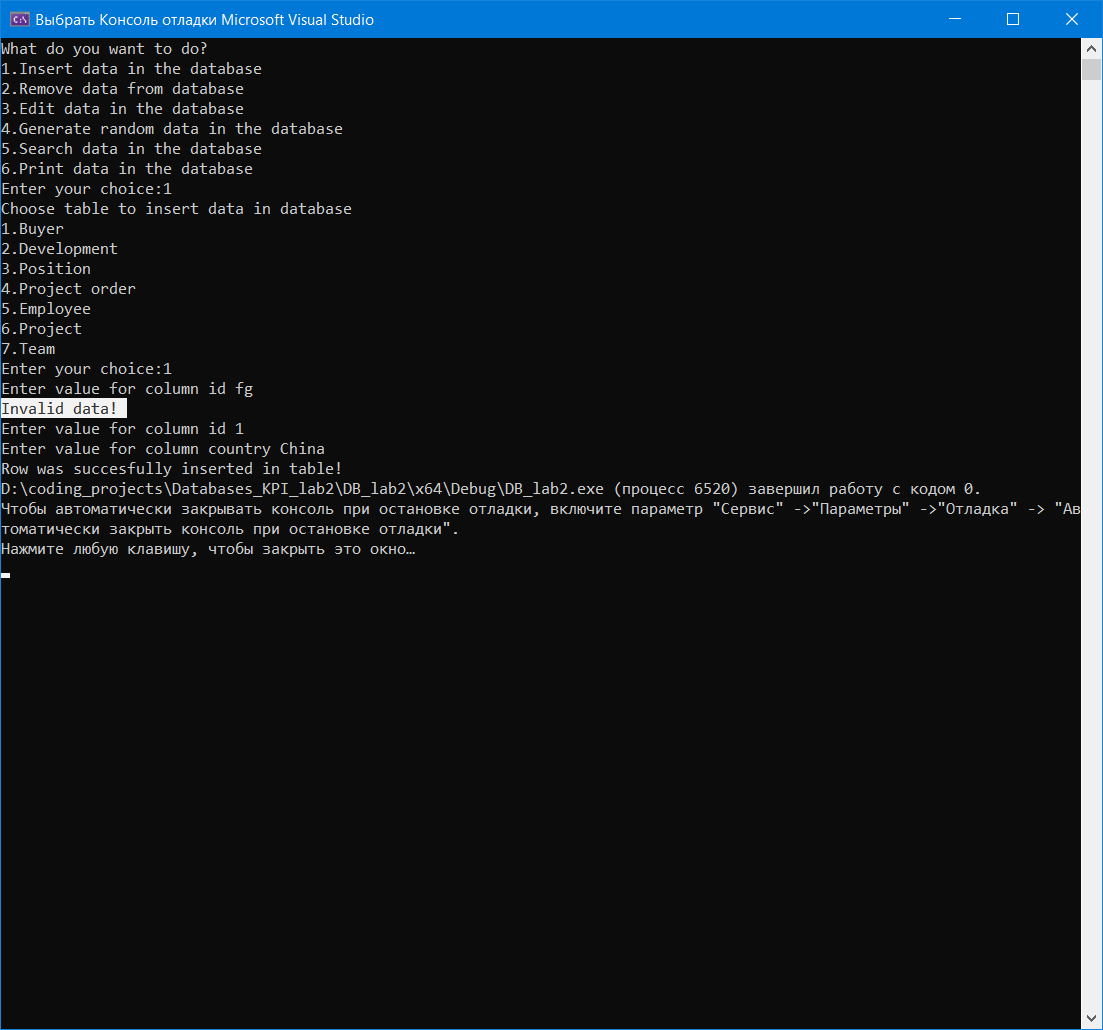


Рисунок 14: Приклад валідації цілочисельних даних

**Копія екрану (ілюстрація) з фрагментом згенерованих даних таблиці**

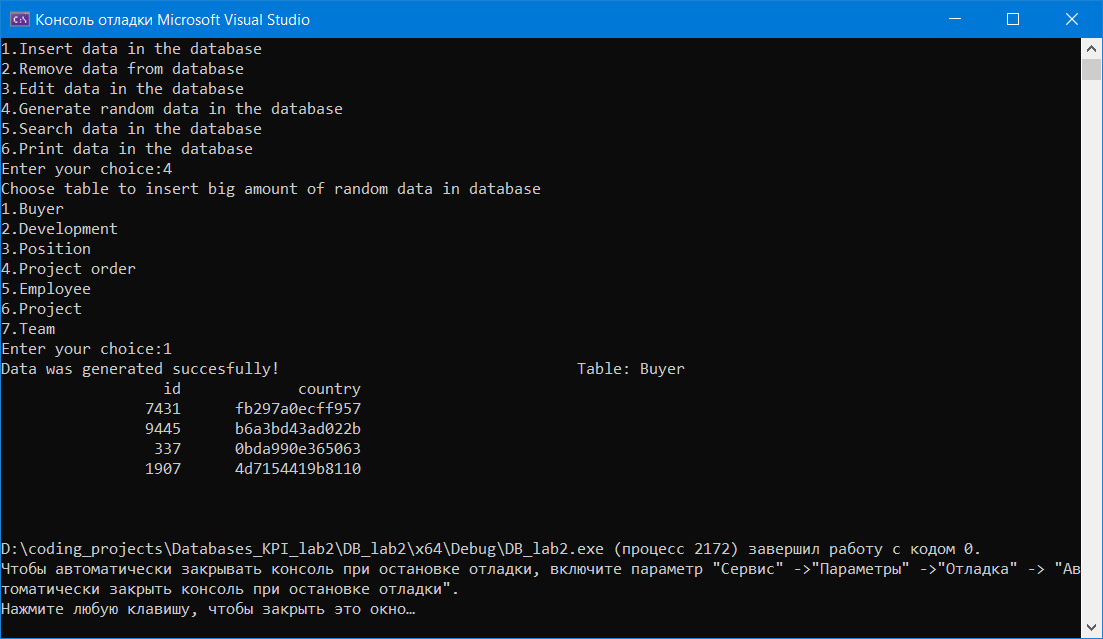


Рисунок 15: Приклад випадкового заповнення даних в одній з таблиць

**Ілюстрація уведення пошукового запиту та результатів виконання запитів**

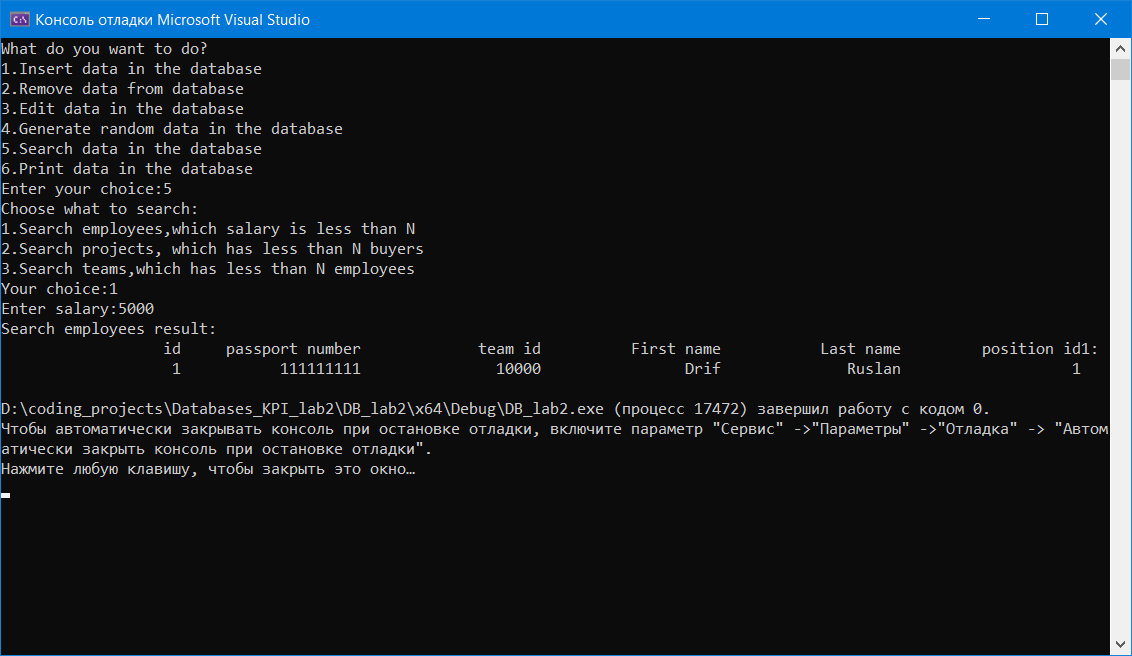


Рисунок 16: Приклад пошуку робітників, ЗП яких менше 5000$

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

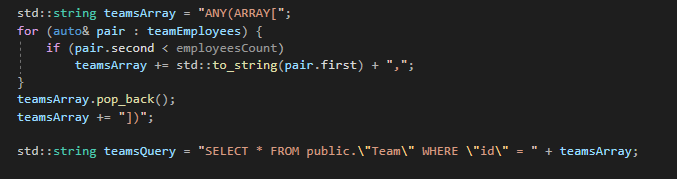


Рисунок 17: Формування запиту для пошуку робітників, зп яких меньше ніж N

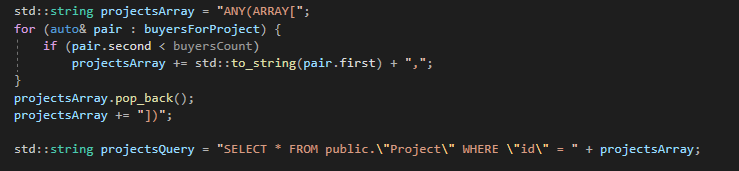


Рисунок 18: Формування запиту для пошуку проектів, в яких менше ніж N покупців

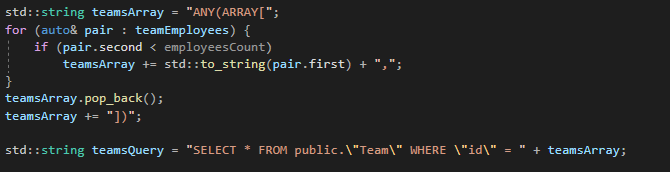


Рисунок 19: Формування запиту для пошуку команд, в яких меньше ніж N робітників.

**Скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці нема відповідного запису**

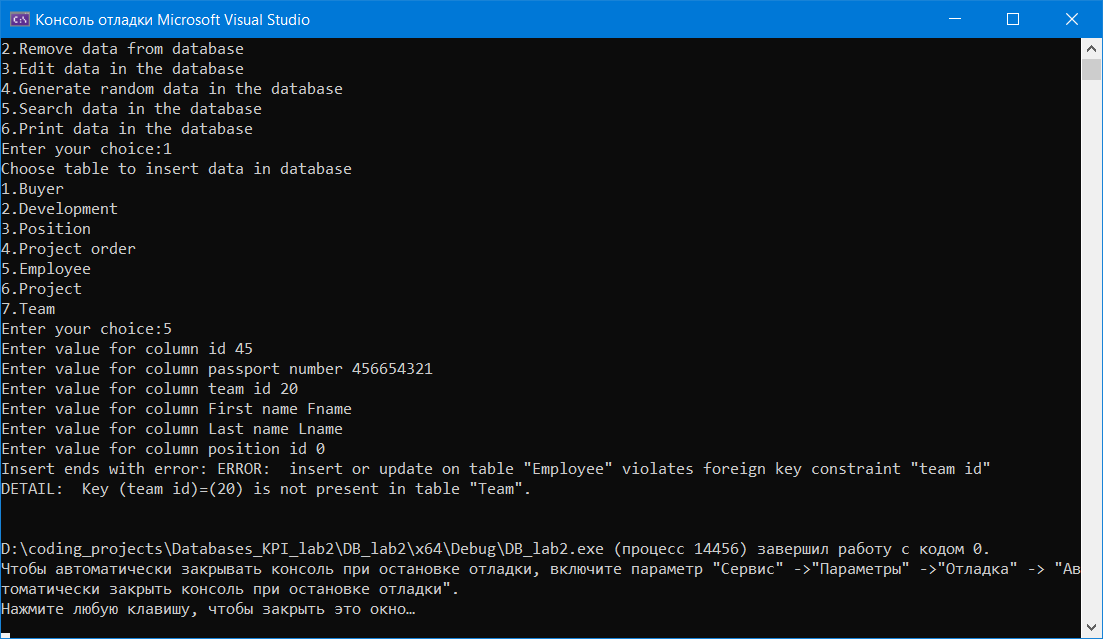


Рисунок 20 – виведення помилки про неможливість вставки

**Скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення**

Візьмемо для прикладу таблицю “Employee”, у якої наявне поле team id, та таблицю “Team”.

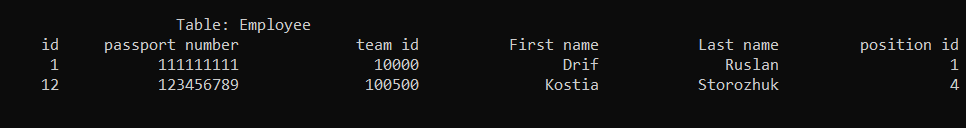


Рисунок 21 – таблиця “Employee” до видалення

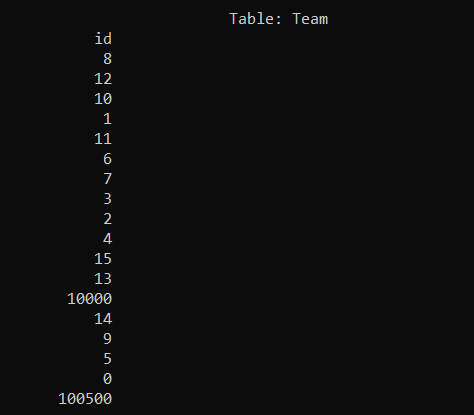


Рисунок 22 – таблиця “Team” до видалення

Видалимо з таблиці Team запис з id = 10000, відповідно з таблиці Employee потрібно буде видалити запис з id = 1.

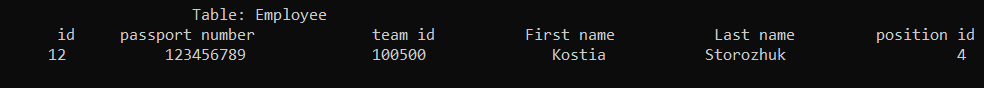


Рисунок 23 – таблиця “Employee” після видалення

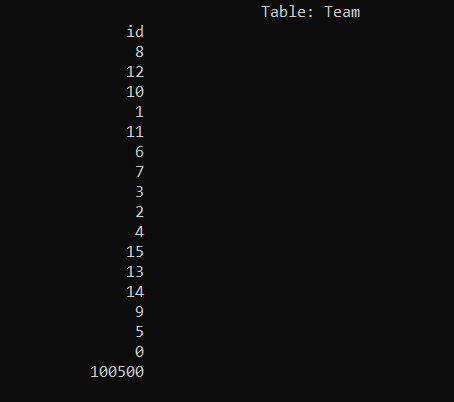


Рисунок 24 – таблиця “Team” після видалення

**Ілюстрації програмного коду з репозиторію Git**

*CDatabaseModel.h*

#pragma once

#include <libpq-fe.h>

#include <string>

#include <vector>

#include <fstream>

#include <unordered\_map>

namespace model {

using childTableRelation = std::pair<std::string, std::string>; //table name and relation field

using parentTableRelation = std::pair<std::string, std::string>; //table name and relation field

struct relation {

parentTableRelation parent;

childTableRelation child;

};

const std::vector<relation> relations = {

{{"Project order","buyer id"},{"Buyer","id"}},

{{"Project order","project id"},{"Project","id"}},

{{"Development","project id"},{"Project","id"}},

{{"Development","team id"},{"Team","id"}},

{{"Employee","team id"},{"Team","id"}},

{{"Employee","position id"},{"Position","id"}},

};

enum class dataTypes {

text,

integer,

characterVarying,

interval,

unk

};

constexpr int randomDataCount = 4;

const std::unordered\_map<std::string, dataTypes> dataTypesMap = {

{"text",dataTypes::text},

{"integer",dataTypes::integer},

{"interval",dataTypes::interval},

{"character varying",dataTypes::characterVarying},

};

class CDatabaseModel {

public:

CDatabaseModel();

CDatabaseModel(const std::string& username, const std::string& db, const std::string& passw);

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

std::string GetLastError() const;

std::vector<const char\*> tables() const;

std::vector<const char\*> columnsInTable(int tableIndex);

std::vector<std::vector<const char\*>> rowsInTable(int tableIndex);

std::vector<const char\*> dataTyperInTable(int tableIndex);

std::vector<std::vector<std::string>> getTuples(PGresult\* res);

std::string getTablePrimaryKey(int tableIndex);

void SetUserName(const std::string& username);

void SetPassword(const std::string& passw);

void SetDbName(const std::string& db);

std::string userName() const;

std::string dbName() const;

std::string password() const;

private:

std::vector<const char\*> m\_tables;

std::vector<const char\*> retrieveTables(PGresult \*res);

void connect();

void terminate();

PGconn\* m\_connection = nullptr;

std::string m\_lastError;

std::string m\_userName;

std::string m\_dbName;

std::string m\_password;

};

}

*CDatabaseModel.cpp*

#include "CDatabaseModel.h"

#include <exception>

#include <iostream>

#include <locale>

#include <utility>

#include <boost/format.hpp>

using namespace model;

CDatabaseModel::CDatabaseModel() {}

CDatabaseModel::CDatabaseModel(const std::string& username, const std::string& db, const std::string& passw)

: m\_dbName(db),m\_password(passw),m\_userName(username) {

connect();

const char\* querySearchTables = "SELECT table\_name\n"

"FROM information\_schema.tables\n"

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

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

auto tablesQueryResult = query(querySearchTables);

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

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

std::cout << "Select failed: " << errMsg << std::endl;

}

else

m\_tables = retrieveTables(tablesQueryResult);

auto langQuery = "SET lc\_messages TO 'en\_US.UTF-8';";

auto langResult = query(langQuery);

}

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

//std::ofstream out("1.txt", std::fstream::out);

auto msg = PQerrorMessage(m\_connection);

//out << msg;

return msg;

}

void CDatabaseModel::SetUserName(const std::string& username) {

m\_userName = username;

}

void CDatabaseModel::SetPassword(const std::string& passw) {

m\_password = passw;

}

void CDatabaseModel::SetDbName(const std::string& db) {

m\_dbName = db;

}

std::string CDatabaseModel::userName() const {

return m\_userName;

}

std::string CDatabaseModel::dbName() const {

return m\_dbName;

}

std::string CDatabaseModel::password() const {

return m\_password;

}

void CDatabaseModel::connect() {

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

const std::string connection\_string =

"user=" + m\_userName +

" password=" + m\_password +

" host = 127.0.0.1 dbname=" + m\_dbName +

" client\_encoding = UTF8";

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

if (m\_connection == nullptr) {

terminate();

}

}

std::vector<const char\*> CDatabaseModel::retrieveTables(PGresult\* res) {

std::vector<const char\*> resVec;

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

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

auto name = PQgetvalue(res, i, j);

resVec.push\_back(name);

}

}

return resVec;

}

void CDatabaseModel::terminate() {

throw std::runtime\_error(GetLastError());

}

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

return PQexec(m\_connection, query.c\_str());

}

std::vector<const char\*> CDatabaseModel::tables() const {

return m\_tables;

}

std::vector<const char\*> CDatabaseModel::columnsInTable(int tableIndex) {

std::vector<const char\*> res;

std::string columnsQuery = (boost::format(

"SELECT column\_name\n"

"FROM information\_schema.columns\n"

"WHERE table\_schema = '%s'\n"

"AND table\_name = '%s';") % "public" % m\_tables[tableIndex]).str();

auto columnsQueryResult = query(columnsQuery);

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

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

auto name = PQgetvalue(columnsQueryResult, i, j);

res.push\_back(name);

}

}

return res;

}

std::vector<const char\*> CDatabaseModel::dataTyperInTable(int tableIndex) {

std::vector<const char\*> res;

std::string dataTyperQuery = (boost::format(

"SELECT data\_type\n"

"FROM information\_schema.columns\n"

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

"AND table\_name = '%s'") % m\_tables[tableIndex]).str();

auto dataTypesQueryResult = query(dataTyperQuery.c\_str());

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

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

auto name = PQgetvalue(dataTypesQueryResult, i, j);

res.push\_back(name);

}

}

//std::string charlength = (boost::format(

// "select character\_maximum\_length\n"

// "from INFORMATION\_SCHEMA.COLUMNS\n"

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

// "AND table\_name = '%s'") % m\_tables[tableIndex]).str();

//dataTypesQueryResult = query(dataTyperQuery.c\_str());

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

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

auto name = PQgetvalue(dataTypesQueryResult, i, j);

//std::cout << name << std::endl;

}

}

return res;

}

std::vector<std::vector<const char\*>> CDatabaseModel::rowsInTable(int tableIndex) {

std::vector<std::vector<const char\*>> res;

std::string dataQuery = (boost::format(

"select \*\n"

"from \"%s\";"

) % m\_tables[tableIndex]).str();

auto\* queryRes = query(dataQuery.c\_str());

//std::cout << dataQuery << std::endl;

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

std::cout << GetLastError() << "-";

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

res.push\_back(std::vector<const char\*>(PQnfields(queryRes)));

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

auto name = PQgetvalue(queryRes, i, j);

//std::cout << name << " ";

res[i][j] = name;

}

//std::cout << std::endl;

}

return res;

}

std::vector<std::vector<std::string>> CDatabaseModel::getTuples(PGresult\* res) {

std::vector<std::vector<std::string>> resVec;

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

resVec.push\_back(std::vector<std::string>(PQnfields(res)));

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

auto name = PQgetvalue(res, i, j);

//std::cout << name << " ";

resVec[i][j] = name;

}

}

return resVec;

}

std::string CDatabaseModel::getTablePrimaryKey(int tableIndex) {

auto\* tablename = m\_tables[tableIndex];

std::string queryPKey = (boost::format(

"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.\"%s\"'::regclass\n"

"AND i.indisprimary;") % tablename).str();

auto\* res = query(queryPKey);

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

return getTuples(res)[0][0];

}

return std::string();

}

*CDatabaseController.h*

#pragma once

#include <string>

#include <libpq-fe.h>

#include <memory>

#include <unordered\_map>

namespace model {

class CDatabaseModel;

}

namespace view {

class CDatabaseView;

}

namespace controller {

enum class searchType {

employeesBySalary,

projectsByBuyers,

teamsByEmployeesCount,

unknown

};

const std::unordered\_map<searchType, std::string> searchModeMap = {

{searchType::employeesBySalary,"Search employees,which salary is less than N"},

{searchType::projectsByBuyers,"Search projects, which has less than N buyers"},

{searchType::teamsByEmployeesCount,"Search teams,which has less than N employees"},

};

enum class userAction : int {

insert = 1,

remove,

edit,

generateRandomData,

search,

print,

unknown

};

const std::unordered\_map<userAction, std::string> userActionMap = {

{userAction::edit,"edit data in database"},

{userAction::remove,"remove data from database"},

{userAction::insert,"insert data in database"},

{userAction::generateRandomData,"insert big amount of random data in database"},

{userAction::search,"search data in database"},

{userAction::unknown,"UNKNOWN\_ACTION"}

};

class CDatabaseController {

public:

CDatabaseController(std::shared\_ptr<model::CDatabaseModel> model, std::shared\_ptr<view::CDatabaseView> view);

userAction requestAction();

void performAction(userAction ua);

private:

bool checkValidInput(const std::string& input,model::dataTypes dataType);

int GetCountElemsForSearch(searchType st);

int chooseTableForAction(userAction ua);

int chooseColumnInTable(int tableNum);

int chooseRowInTable(int tableIndex);

PGresult\* searchEmployees(int salary);

PGresult\* searchTeams(int employeesCount);

PGresult\* searchProjects(int buyersCount);

std::string requestData(const char\* colName, model::dataTypes type,bool canBeLeaved = false);

model::dataTypes typeFromString(const char\* str);

PGresult\* performInsert(int tableIndex);

PGresult\* performRemove(int tableIndex, int rowIndex);

PGresult\* performEdit(int tableIndex, int rowIndex);

PGresult\* performGeneratingRandomData(int tableIndex);

PGresult\* performSearch();

std::shared\_ptr<model::CDatabaseModel> m\_model;

std::shared\_ptr<view::CDatabaseView> m\_view;

};

}

*CDatabaseController.cpp*

#include "CDatabaseModel.h"

#include "CDatabaseController.h"

#include "CDatabaseView.h"

#include <boost/format.hpp>

#include <iostream>

#include <iomanip>

#include <regex>

using namespace controller;

userAction CDatabaseController::requestAction() {

using std::cout;

cout << "What do you want to do?\n";

cout << "1.Insert data in the database\n";

cout << "2.Remove data from database\n";

cout << "3.Edit data in the database\n";

cout << "4.Generate random data in the database\n";

cout << "5.Search data in the database\n";

cout << "6.Print data in the database\n";

cout << "Enter your choice:";

int choose = -1;

std::cin >> choose;

constexpr int max\_userAction\_val = 7;

constexpr int min\_userAction\_val = 1;

if (choose > max\_userAction\_val && choose < min\_userAction\_val)

return userAction::unknown;

return static\_cast<userAction>(choose);

}

void CDatabaseController::performAction(userAction ua) {

int tableIndex = -1;

int tableRow = -1;

//int tableIndex = tableNum - 1;

switch (ua) {

case userAction::edit:

tableIndex = chooseTableForAction(ua);

tableRow = chooseRowInTable(tableIndex);

m\_view->printOperationResult(ua,performEdit(tableIndex, tableRow));

//std::cout << "chosen edit" << std::endl;

break;

case userAction::remove:

tableIndex = chooseTableForAction(ua);

tableRow = chooseRowInTable(tableIndex);

m\_view->printOperationResult(ua, performRemove(tableIndex, tableRow));

//std::cout << "chosen remove" << std::endl;

break;

case userAction::insert:

tableIndex = chooseTableForAction(ua);

m\_view->printOperationResult(ua, performInsert(tableIndex));

//std::cout << "chosen insert" << std::endl;

break;

case userAction::generateRandomData:

tableIndex = chooseTableForAction(ua);

m\_view->printOperationResult(ua, performGeneratingRandomData(tableIndex));

m\_view->printDB(tableIndex);

//std::cout << "chosen generateRandomData" << std::endl;

break;

case userAction::search:

m\_view->printOperationResult(ua, performSearch());

//std::cout << "chosen search" << std::endl;

break;

case userAction::print:

m\_view->printOperationResult(ua,nullptr);

//std::cout << "chosen search" << std::endl;

break;

case userAction::unknown:

default:

//std::cout << "chosen unk" << std::endl;

break;

}

}

PGresult\* CDatabaseController::performInsert(int tableIndex) {

auto cols = m\_model->columnsInTable(tableIndex);

auto types = m\_model->dataTyperInTable(tableIndex);

std::vector<std::string> userData;

int indx = 0;

for (auto col : cols) {

auto key = model::dataTypesMap.find(types[indx++]);

if (key != model::dataTypesMap.end()) {

userData.push\_back(requestData(col, key->second));

}

else

std::cout << "key is end!\n";

}

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

const bool needParentheses =

typeFromString(types[i]) == model::dataTypes::interval ||

typeFromString(types[i]) == model::dataTypes::characterVarying ||

typeFromString(types[i]) == model::dataTypes::text;

if (needParentheses) {

userData[i] = "'" + userData[i] + "'";

}

}

std::string userInputStr;

for (auto input : userData) {

userInputStr += std::string(input) + ",";

}

userInputStr.pop\_back();

userInputStr.push\_back(')');

userInputStr.insert(0, 1, '(');

std::string insertQuery = (boost::format(

"INSERT INTO \"%s\"\n"

"VALUES %s"

) % m\_model->tables()[tableIndex] % userInputStr).str();

auto\* insertQueryRes = m\_model->query(insertQuery.c\_str());

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

std::wcout << reinterpret\_cast<const wchar\_t\*>(m\_model->GetLastError().data());

return insertQueryRes;

}

PGresult\* CDatabaseController::performRemove(int tableIndex, int rowIndex) {

auto tables = m\_model->tables();

auto cols = m\_model->columnsInTable(tableIndex);

auto rows = m\_model->rowsInTable(tableIndex);

auto pKey = m\_model->getTablePrimaryKey(tableIndex);

auto pKeyIter = std::find\_if(cols.begin(), cols.end(), [&](const char\* col) {

return std::string(col) == pKey;

});

const int pKeyIndex = pKeyIter - cols.begin();

std::string removeQuery = (boost::format(

"DELETE FROM public.\"%s\"\n"

"WHERE \"%s\" = %s"

) % tables[tableIndex] % cols[pKeyIndex] % rows[rowIndex][pKeyIndex]).str();

for(auto it = model::relations.begin();it != model::relations.end();++it)

if (it->child.first == tables[tableIndex]) {

auto parentTableIndex = std::find\_if(tables.begin(), tables.end(), [&](const char\* t) {

return std::string(t) == it->parent.first;

}) - tables.begin();

auto parentTableCols = m\_model->columnsInTable(parentTableIndex);

auto parentFieldIndex = std::find\_if(parentTableCols.begin(), parentTableCols.end(), [&](const char\* col) {

return std::string(col) == it->parent.second;

}) - parentTableCols.begin();

auto childFieldIndex = std::find\_if(cols.begin(), cols.end(), [&](const char\* col) {

return std::string(col) == it->child.second;

}) - cols.begin();

auto parentRows = m\_model->rowsInTable(parentTableIndex);

bool found = false;

int indx = 0;

for (auto& r : parentRows) {

if (std::string(r[parentFieldIndex]) == std::string(rows[rowIndex][childFieldIndex])) {

found = true;

performRemove(parentTableIndex, indx);

//break;

}

indx++;

}

}

return m\_model->query(removeQuery);

}

PGresult\* CDatabaseController::performEdit(int tableIndex, int rowIndex) {

auto cols = m\_model->columnsInTable(tableIndex);

auto types = m\_model->dataTyperInTable(tableIndex);

auto row = m\_model->rowsInTable(tableIndex)[rowIndex];

std::vector<std::string> userInput;

std::string input;

std::getline(std::cin, input);

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

auto data = requestData(cols[i], typeFromString(types[i]), true);

if (data == "\*")

data = row[i];

userInput.push\_back(data);

}

std::string updateQueryStr = (boost::format("UPDATE public.\"%s\" SET ") % m\_model->tables()[tableIndex]).str();

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

const bool needParentheses =

typeFromString(types[i]) == model::dataTypes::interval ||

typeFromString(types[i]) == model::dataTypes::characterVarying ||

typeFromString(types[i]) == model::dataTypes::text;

const std::string updateVarstr = "\"" + (std::string(cols[i]) + "\"" + " = " +

(needParentheses ?

"'" + userInput[i] + "'" :

userInput[i]) + ", ");

updateQueryStr += updateVarstr;

}

updateQueryStr.pop\_back();

updateQueryStr.pop\_back();

updateQueryStr += "\nWHERE ";

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

const bool needParentheses =

typeFromString(types[i]) == model::dataTypes::interval ||

typeFromString(types[i]) == model::dataTypes::characterVarying ||

typeFromString(types[i]) == model::dataTypes::text;

const std::string currVarStr = "\"" + (std::string(cols[i]) + "\"" + " = " + (needParentheses ? "'" + std::string(row[i]) + "'" : row[i]) + (i == userInput.size() - 1 ? "" : " AND "));

updateQueryStr += currVarStr;

}

std::cout << updateQueryStr << std::endl;

auto res = m\_model->query(updateQueryStr);

return res;

}

PGresult\* CDatabaseController::performGeneratingRandomData(int tableIndex) {

auto randomStrByType = [&](model::dataTypes dt) -> std::string {

//std::cout << "type: " << static\_cast<int>(dt) << std::endl;

using namespace model;

switch (dt) {

case dataTypes::characterVarying:

return "substr(md5(random()::text), 1, 9)";

case dataTypes::integer:

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

case dataTypes::interval:

return "INTERVAL '00:00:00' \* (random()\*10)::integer";

case dataTypes::text:

return "substr(md5(random()::text), 0, 15)";

case dataTypes::unk:

return "";

}

};

auto types = m\_model->dataTyperInTable(tableIndex);

auto cols = m\_model->columnsInTable(tableIndex);

auto tables = m\_model->tables();

std::vector<std::string> generatingStrings;

for (auto& t : types) {

auto type = typeFromString(t);

generatingStrings.push\_back(randomStrByType(type));

}

std::string generateQuery = "INSERT INTO public.\"" + std::string(tables[tableIndex]) + "\"(";

for (auto& col : cols)

generateQuery += "\"" + std::string(col) + "\",";

generateQuery.pop\_back();

generateQuery += ")";

generateQuery += "\nSELECT";

for (auto& randStr : generatingStrings)

generateQuery += "\n" + randStr + ",";

generateQuery.pop\_back();

generateQuery += (boost::format("\nFROM generate\_series(1,%d)\nON CONFLICT DO NOTHING") % model::randomDataCount).str();

return m\_model->query(generateQuery);

}

PGresult\* CDatabaseController::performSearch() {

std::cout << "Choose what to search:\n";

int count = 0;

for (auto& pair : searchModeMap) {

std::cout << ++count << "." << pair.second << std::endl;

}

std::cout << "Your choice:";

int choose = -1;

std::cin >> choose;

constexpr int max\_searchMode\_val = 3;

constexpr int min\_searchMode\_val = 1;

if (choose < min\_searchMode\_val || choose > max\_searchMode\_val) {

std::cout << "wrong input!";

return nullptr;

}

auto mode = static\_cast<searchType>(choose - 1);

const int cnt = GetCountElemsForSearch(mode);

std::unordered\_map<int, int> searchMap;

auto tables = m\_model->tables();

PGresult\* res = nullptr;

switch (mode) {

case searchType::employeesBySalary:

res = searchEmployees(cnt);

break;

case searchType::projectsByBuyers:

res = searchProjects(cnt);

break;

case searchType::teamsByEmployeesCount:

res = searchTeams(cnt);

break;

default:

break;

}

return res;

}

int CDatabaseController::chooseTableForAction(userAction ua) {

using std::cout;

cout << "Choose table to " << userActionMap.at(ua) << std::endl;

auto tables = m\_model->tables();

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

cout << i + 1 << "." << tables[i] << std::endl;

cout << "Enter your choice:";

int res = -1;

std::cin >> res;

return (res >= 1 && res <= tables.size()) ? res - 1 : -1;

}

CDatabaseController::CDatabaseController(std::shared\_ptr<model::CDatabaseModel> model, std::shared\_ptr<view::CDatabaseView> view)

: m\_model(model), m\_view(view)

{

}

int CDatabaseController::chooseColumnInTable(int tableNum) {

auto tables = m\_model->tables();

auto cols = m\_model->columnsInTable(tableNum);

std::cout << "Choose column number in table " << tables[tableNum] << std::endl;

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

std::cout << i + 1 << "." << cols[i] << std::endl;

}

std::cout << "Enter number:";

int choose = -1;

std::cin >> choose;

return (choose >= 1 && choose <= cols.size() ? choose - 1 : -1);

}

int CDatabaseController::chooseRowInTable(int tableIndex) {

auto rows = m\_model->rowsInTable(tableIndex);

auto cols = m\_model->columnsInTable(tableIndex);

if (rows.empty())

return -1;

std::cout << "Choose row in table.\n";

std::cout

<< "Choose rows to show:(1 option by default)\n"

<< "1.All rows\n"

<< "2.First 15 rows\n"

<< "3.Last 15 rows\n";

int showChoose = -1;

std::cout << "Your choice:";

std::cin >> showChoose;

if (showChoose < 1 || showChoose > 3 || rows.size() <= 15)

showChoose = 1;

int left = -1, right = -1;

switch (showChoose) {

case 1:

left = 0;

right = rows.size();

break;

case 2:

left = 0;

right = 15;

break;

case 3:

left = 15;

right = rows.size();

break;

}

for (int i = left; i < right; ++i) {

std::cout << i + 1 << ".";

for (int j = 0; j < rows[i].size(); ++j) {

std::cout << "\t" << rows[i][j];

}

std::cout << std::endl;

}

std::cout << "Your choice:";

int choose = -1;

std::cin >> choose;

return choose >= 1 && choose <= rows.size() ? choose - 1 : -1;

}

std::string CDatabaseController::requestData(const char\* colName, model::dataTypes type, bool canBeLeaved) {

std::string userInput;

bool dataValid = true;

do {

if (!dataValid) {

std::cout << "Invalid data!\n";

}

std::cout << "Enter value for column " << colName << " ";

if (canBeLeaved) {

std::cout << " or Enter \* to leave it in current state: ";

}

std::cin >> userInput;

if (canBeLeaved && userInput == "\*")

return userInput;

dataValid = false;

} while (!checkValidInput(userInput,type));

switch (type) {

case model::dataTypes::characterVarying:

return userInput;

case model::dataTypes::integer:

return (userInput);

case model::dataTypes::interval:

//std::cout << "\nInput format: hh:mm:ss";

return userInput;

case model::dataTypes::text:

return userInput;

default:

return "";

}

return "";

}

model::dataTypes CDatabaseController::typeFromString(const char\* str) {

auto key = model::dataTypesMap.find(std::string(str));

if (key != model::dataTypesMap.end())

return key->second;

return model::dataTypes::unk;

}

int CDatabaseController::GetCountElemsForSearch(searchType st) {

int count = 0;

switch (st) {

case searchType::employeesBySalary:

std::cout << "Enter salary:";

break;

case searchType::projectsByBuyers:

std::cout << "Enter count of buyers:";

break;

case searchType::teamsByEmployeesCount:

std::cout << "Enter count of employees:";

break;

default:

break;

}

std::cin >> count;

return count;

}

PGresult\* CDatabaseController::searchEmployees(int salary) {

PGresult\* res = nullptr;

auto tables = m\_model->tables();

auto iterEmployees = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Employee";

});

auto iterPositions = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Position";

});

auto positionCols = m\_model->columnsInTable(iterPositions - tables.begin());

auto employeesCols = m\_model->columnsInTable(iterEmployees - tables.begin());

std::string PositionsQuery = "SELECT ";

for (auto& i : positionCols)

PositionsQuery += "\"" + std::string(i) + "\",";

PositionsQuery.pop\_back();

PositionsQuery += "\nFROM public.\"Position\"\n";

PositionsQuery += (boost::format("WHERE \"salary\" < %d") % salary).str();

res = m\_model->query(PositionsQuery);

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

std::cout << "Error search positions!";

return res;

}

std::cout << "Search employees result:\n";

for (auto& col : employeesCols) {

std::cout << std::setw(20) << col;

}

std::cout << std::endl;

auto tpls = m\_model->getTuples(res);

std::vector<int> neededPositionsId;

for (auto& tuple : tpls) {

neededPositionsId.push\_back(std::stoi(tuple[0]));

}

std::string employeesQuery = "SELECT ";

for (auto& i : employeesCols)

employeesQuery += "\"" + std::string(i) + "\",";

employeesQuery.pop\_back();

employeesQuery += "\nFROM public.\"Employee\"\n";

employeesQuery += "WHERE \"id\" = " + std::to\_string(neededPositionsId[0]);

for (int i = 1; i < neededPositionsId.size(); ++i) {

employeesQuery += " OR \"id\"= " + std::to\_string(neededPositionsId[i]);

}

res = m\_model->query(employeesQuery);

return res;

}

PGresult\* CDatabaseController::searchTeams(int employeesCount) {

PGresult\* res = nullptr;

auto tables = m\_model->tables();

auto iterEmployees = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Employee";

});

auto iterTeams = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Team";

});

using team\_id = int;

std::unordered\_map<team\_id, int> teamEmployees;

auto rows = m\_model->rowsInTable(iterEmployees - tables.begin());

constexpr int teamIdIndex = 2;

for (auto& row : rows) {

teamEmployees[std::stoi(row[teamIdIndex])]++;

}

for (auto& pair : teamEmployees) {

std::cout << "team id: " << pair.first << " count: " << pair.second << std::endl;

}

std::string teamsArray = "ANY(ARRAY[";

for (auto& pair : teamEmployees) {

if (pair.second < employeesCount)

teamsArray += std::to\_string(pair.first) + ",";

}

teamsArray.pop\_back();

teamsArray += "])";

std::string teamsQuery = "SELECT \* FROM public.\"Team\" WHERE \"id\" = " + teamsArray;

//std::cout << teamsQuery << std::endl;

res = m\_model->query(teamsQuery);

return res;

}

PGresult\* CDatabaseController::searchProjects(int buyersCount) {

PGresult\* res = nullptr;

const auto tables = m\_model->tables();

const auto iterProjectOrders = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Project order";

});

const auto iterProjects = std::find\_if(tables.begin(), tables.end(), [this](const char\* tableName) {

return std::string(tableName) == "Project";

});

auto projectOrderCols = m\_model->columnsInTable(iterProjectOrders - tables.begin());

constexpr int idIndexProject = 0;

std::unordered\_map<int, int> buyersForProject;

std::vector<std::pair<int, int>> usedPairs;

auto rowsOrders = m\_model->rowsInTable(iterProjectOrders - tables.begin());

auto rowsProjects = m\_model->rowsInTable(iterProjects - tables.begin());

for (auto& proj : rowsProjects) {

const int id = std::stoi(proj[0]);

buyersForProject[id] = 0;

for (auto& order : rowsOrders) {

const int currProjectId = std::stoi(order[3]);

if (currProjectId == id) {

const int currBuyerId = std::stoi(order[1]);

if (std::find(usedPairs.begin(), usedPairs.end(), std::pair<int,int>(currProjectId,currBuyerId)) == usedPairs.end()) {

buyersForProject[currProjectId]++;

usedPairs.push\_back(std::pair<int, int>(currProjectId, currBuyerId));

}

}

}

}

std::string projectsArray = "ANY(ARRAY[";

for (auto& pair : buyersForProject) {

if (pair.second < buyersCount)

projectsArray += std::to\_string(pair.first) + ",";

}

projectsArray.pop\_back();

projectsArray += "])";

std::string projectsQuery = "SELECT \* FROM public.\"Project\" WHERE \"id\" = " + projectsArray;

res = m\_model->query(projectsQuery);

return res;

}

bool CDatabaseController::checkValidInput(const std::string& input, model::dataTypes dataType) {

using namespace model;

const std::regex timeRegex("([01]?[0-9]|2[0-3]):[0-5][0-9]");

switch (dataType) {

case dataTypes::characterVarying:

return input.size() == 9; //in that database, the only field which is character varying is passport number, it must have 9 characters

case dataTypes::integer:

for (auto& c : input) {

if (!std::isdigit(c))

return false;

}

return true;

case dataTypes::interval:

return std::regex\_match(input, timeRegex);

case dataTypes::text:

return true;

case dataTypes::unk:

return false;

}

}

*CDatabaseView.h*

#pragma once

#include <memory>

#include <libpq-fe.h>

namespace model {

class CDatabaseModel;

}

namespace controller {

enum class userAction;

class CDatabaseController;

}

namespace view {

class CDatabaseView

{

public:

CDatabaseView(std::shared\_ptr<model::CDatabaseModel> model\_);

void printOperationResult(controller::userAction userAct,PGresult\* res) const;

void printDB() const;

void printDB(int tableIndex) const;

private:

void printInsert(PGresult\* res) const;

void printEdit(PGresult\* res) const;

void printGenerate(PGresult\* res) const;

void printSearch(PGresult\* res) const;

void printRemove(PGresult\* res) const;

std::shared\_ptr<model::CDatabaseModel> m\_model;

};

}

*CDatabaseView.cpp*

#include "CDatabaseView.h"

#include "CDatabaseModel.h"

#include "CDatabaseController.h"

#include <iostream>

#include<iomanip>

using namespace view;

CDatabaseView::CDatabaseView(std::shared\_ptr<model::CDatabaseModel> model)

: m\_model(model) {}

void CDatabaseView::printOperationResult(controller::userAction userAct, PGresult\* res) const {

switch (userAct) {

case controller::userAction::edit:

printEdit(res);

break;

case controller::userAction::generateRandomData:

printGenerate(res);

break;

case controller::userAction::insert:

printInsert(res);

break;

case controller::userAction::remove:

printRemove(res);

break;

case controller::userAction::search:

printSearch(res);

break;

case controller::userAction::print:

printDB();

break;

case controller::userAction::unknown:

break;

}

}

void CDatabaseView::printInsert(PGresult\* res) const {

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

std::cout << "Insert ends with error: " << m\_model->GetLastError() << std::endl;

return;

}

std::cout << "Row was succesfully inserted in table!";

}

void CDatabaseView::printEdit(PGresult\* res) const {

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

std::cout << "Update ends with error: " << m\_model->GetLastError() << std::endl;

return;

}

std::cout << "Succesfully updated " << PQcmdTuples(res) << " tuples";

}

void CDatabaseView::printSearch(PGresult\* res) const {

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

std::cout << "Search ends with error: " << m\_model->GetLastError() << std::endl;

return;

}

//std::cout << "Search result:\n";

auto tpls = m\_model->getTuples(res);

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

std::cout << i + 1 << ": ";

for (int j = 0; j < tpls[i].size(); ++j) {

std::cout << std::setw(20) << tpls[i][j];

}

std::cout << std::endl;

}

}

void CDatabaseView::printRemove(PGresult\* res) const {

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

std::cout << "Remove ends with error: " << m\_model->GetLastError() << std::endl;

return;

}

std::cout << "Remove succesfull!";

}

void CDatabaseView::printGenerate(PGresult\* res) const {

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

std::cout << "Generating data ends with error: " << m\_model->GetLastError() << std::endl;

return;

}

std::cout << "Data was generated succesfully!";

}

void CDatabaseView::printDB() const {

std::cout << "IT-company database:\n";

const auto tables = m\_model->tables();

//std::cout << std::setw(30);

std::cout << std::endl;

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

printDB(tbl);

}

}

void CDatabaseView::printDB(int tableIndex) const {

const auto tables = m\_model->tables();

std::cout << std::setw(40) << "Table: " << tables[tableIndex] << std::endl;

const auto cols = m\_model->columnsInTable(tableIndex);

for (int col = 0; col < cols.size(); ++col) {

std::cout << std::setw(20) << cols[col];

}

std::cout << std::endl;

const auto rows = m\_model->rowsInTable(tableIndex);

for (int row = 0; row < rows.size(); ++row) {

for (int col = 0; col < rows[row].size(); ++col) {

std::cout << std::setw(20) << rows[row][col];

}

std::cout << std::endl;

}

std::cout << "\n\n";

}

*Database.cpp*

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <libpq-fe.h>

#include "CDatabaseModel.h"

#include "CDatabaseView.h"

#include "CDatabaseController.h"

#include <memory>

int main()

{

auto model = std::make\_shared<model::CDatabaseModel>("postgres", "IT-company", "passw");

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

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

auto choose = controller->requestAction();

controller->performAction(choose);

return 0;

}