A black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Розрахунково-графічна робота**

з дисципліни **Бази даних і засоби управління**

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

Виконала:

студентка ІІI курсу

групи КВ-22

Деркач А. І.

Перевірив:

Павловский В. І.

Київ – 2024

**Мета:** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Виконання роботи**

Нижче будуть наведені сутності предметної області:

**Сутності предметної області**

Для побудови концептуальної моделі обраної предметної області, були виділені такі сутності:

Рахунок з атрибутами: ідентифікатор рахунку, баланс, тип валюти, ідентифікатор власнику рахунку. Призначення: збереження інформації про рахунки які беруть участь в транзакціях.

Клієнт з атрибутами ідентифікатор клієнта, ім’я клієнта, адреса, номер телефону. Призначення: зберігання інформації про клієнтів фінансової системи.

Транзакція з атрибутами: ідентифікатор транзакції, дата транзакції, сума транзакції, ідентифікатор рахунку-відправника, ідентифікатор рахунку-отримувача. Призначення: збереження інформації про транзакції які відбулися.

Нижче будуть наведені зв’язки між сутностями предметної області:

**Зв’язки між сутностями предметної області**

Кожен рахунок має одного клієнта-власника. Інші клієнти також можуть користуватися цим рахунком, якщо їм було надано доступ. Таким чином, між сутностями Клієнт і Рахунок існує зв'язок M:N.

Один рахунок може здійснити багато транзакцій і одна транзакція може стосуватись тільки одного рахунку. Тому між сутностями Рахунок і Транзакція існує зв’язок 1:N.

Графічне подання концептуальної моделі «Сутність-зв’язок» зображено на рисунку 1.

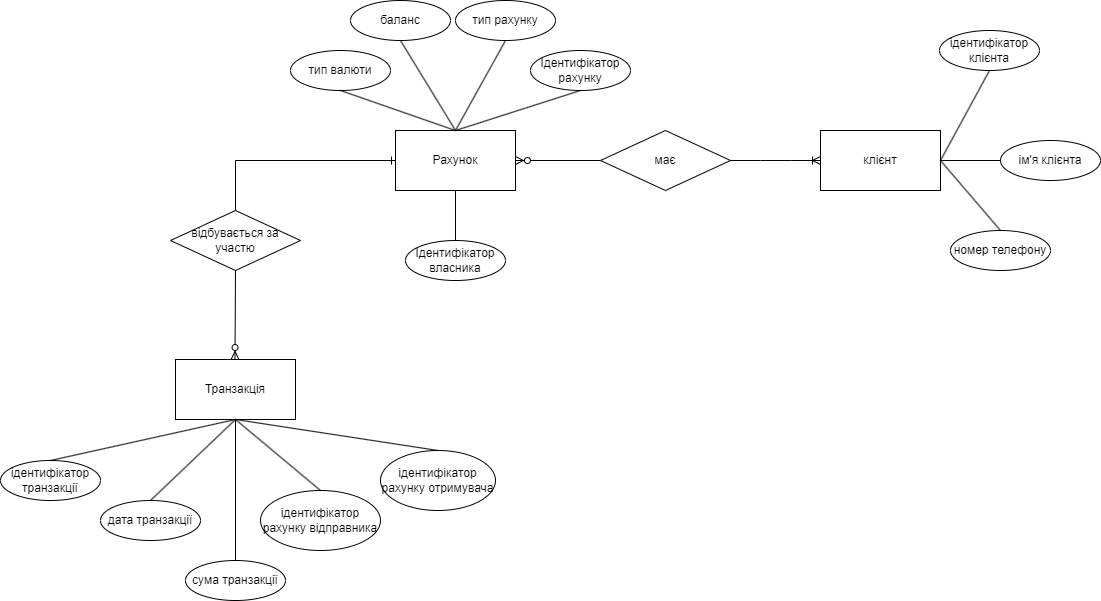


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 2.

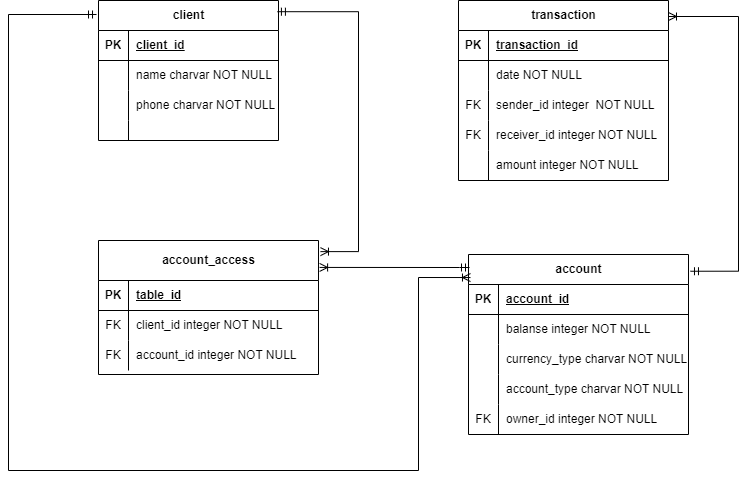


Рисунок 2 – Логічна модель

**Середовище та компоненти розробки**

У процесі розробки була використана мова програмування С++, інтегроване середовище розробки Visual Studio, а також була використана бібліотека libpq-fe.h, яка надає API для взаємодії з базою даних PostgreSQL.

**Шаблон проектування**

Модель-представлення-контролер (MVC) - це шаблон проектування, який використовується у розробці програм. Кожен компонент відповідає за конкретну функціональну частину програми:

Модель (Model) - це клас, який визначає логіку роботи з даними. Він обробляє всі операції з даними, такі як додавання, оновлення та видалення.

Представлення (View) - це клас, який дозволяє користувачеві взаємодіяти з програмою. В даному випадку це консольний інтерфейс, який відображає дані користувачу та приймає введення з екрану.

Контролер (Controller) - це клас, що відповідає за взаємодію між користувачем та системою. Він приймає дані, введені користувачем, та обробляє їх. Залежно від результатів, він викликає відповідні дії в Model або View.

Цей підхід дозволяє розділити логіку програми на логічні компоненти, що полегшує розробку, тестування та підтримку продукту.

**Структура програми та її опис**

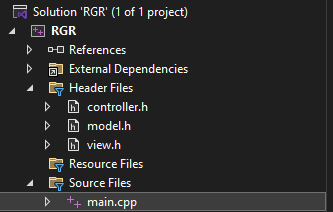
****

Рисунок 3 – Структура програми

З файлу main.cpp відбувається виклик контролера та передача йому управління.

У файлі model.h описаний клас моделі, який відповідає за управління підключенням до бази даних і виконанням низькорівневих запитів до неї.

У файлі controller.h реалізовано інтерфейс взаємодії з користувачем, включаючи обробку запитів користувача, виконання пошуку, а також інші дії, необхідні для взаємодії з моделлю та представленням.

У файлі view.h описаний клас, який відображає результати виконання різних дій користувача на екрані консолі. Цей компонент відповідає за представлення даних користувачу в зручному для сприйняття вигляді.

Отже, структура програми відповідає патерну MVC.

**Меню програми**

На рисунку 4 зображено меню користувача, яке складається з семи пунктів.

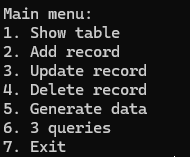


Рисунок 4 – Структура меню користувача

**Фрагмент коду (файл controller.h), в якому наведено головний цикл роботи програми:**

void Start() {

while (true) {

int input = View.MainMenu();

if (input == 1)

OutputTable();

else if (input == 2)

AddRecord();

else if (input == 3)

UpdateData();

else if (input == 4)

DeleteData();

else if (input == 5)

GenerateRandomData();

else if (input == 6)

Queries();

else

cout << "Incorrect input. Try again" << endl;

}

}

**Фрагмент коду (файл model.h), в якому наведено функції перегляду, внесення, редагування, вилучення та генерації у базі даних:**

Функції виведення даних:

PGresult\* ShowTable(string name) {

string query = "select \* from public.\"" + name + "\"";

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

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

{

cerr << PQerrorMessage(connection) << endl;

PQclear(res);

return nullptr;

}

return res;

}

Ця функція перевіряє чи є таблиця, якщо перевірка пройдена то функція виводить повністю таблицю даних.

Функції внесення даних:

bool AddRecordClient(vector<string> values) {

string query = "select \* from public.\"client\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"client\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordAccountAccess(vector<string> values) {

string query = "select \* from public.\"account\_access\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

PGresult\* result2 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[1] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

for (int i = 0; i < PQntuples(result2); i++) {

if (values[2] == PQgetvalue(result2, i, 0)) {

f2 = true;

break;

}

}

if (!f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

PQclear(result1);

PQclear(result2);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"account\_access\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordAccount(vector<string> values) {

string query = "select \* from public.\"account\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

bool f1 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[4] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"account\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordTransaction(vector<string> values) {

string query = "select \* from public.\"transaction\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[2] == PQgetvalue(result1, i, 0))

f1 = true;

if (values[3] == PQgetvalue(result1, i, 0))

f2 = true;

if (f1 && f2)

break;

}

if (!f1 || !f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"transaction\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

Ці функції перевіряють, чи вводить користувач унікальний ідентифікатор для нового запису в таблиці. Також вона перевіряє наявність зовнішнього ключа, якщо такий є в таблиці, щоб упевнитися, що введений ключ існує у відповідній зв’язаній таблиці. Додатково здійснюється перевірка на валідність внесених змін. Якщо всі умови виконуються, дозволяється вставка запису; в іншому випадку користувачу буде запропоновано ввести інші дані.

Функції оновлення даних:

bool UpdateRecordClient(vector<string> values, int idx) {

bool f = true;

string query = "select \* from public.\"client\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"client\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordAccountAccess(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

PGresult\* result2 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[1] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

for (int i = 0; i < PQntuples(result2); i++) {

if (values[2] == PQgetvalue(result2, i, 0)) {

f2 = true;

break;

}

}

if (!f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

PQclear(result1);

PQclear(result2);

bool f = true;

string query = "select \* from public.\"account\_access\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"account\_access\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordAccount(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

bool f1 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[4] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

bool f = true;

string query = "select \* from public.\"account\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"account\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordTransaction(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[2] == PQgetvalue(result1, i, 0))

f1 = true;

if (values[3] == PQgetvalue(result1, i, 0))

f2 = true;

if (f1 && f2)

break;

}

if (!f1 || !f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

bool f = true;

string query = "select \* from public.\"transaction\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"transaction\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

Ці функції перевіряють правильність внесених змін. Якщо користувач намагається змінити поле, яке є зовнішнім ключем, функція перевіряє, чи існує введений зовнішній ключ у відповідній зв’язаній таблиці. Якщо всі перевірки проходять успішно, дозволяється виконати вставку. В іншому випадку, користувачу потрібно ввести інші дані.

Функції видалення даних:

bool DeleteRecordClient(int idx) {

string query1, query2;

query1 = "select \"client\_id\" from \"account\_access\"";

query2 = "select \"owner\_id\" from \"account\"";

auto res1 = PQexec(connection, query1.c\_str());

for (int i = 0; i < PQntuples(res1); i++) {

if (PQgetvalue(res1, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\_access\"" << endl;

return false;

}

}

auto res2 = PQexec(connection, query2.c\_str());

for (int i = 0; i < PQntuples(res2); i++) {

if (PQgetvalue(res2, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\"" << endl;

return false;

}

}

PQclear(res1);

PQclear(res2);

string query = "delete from \"client\" where \"client\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordAccountAccess(int idx) {

string query = "delete from \"account\_access\" where \"table\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordAccount(int idx) {

string query1, query2, query3;

query1 = "select \"account\_id\" from \"account\_access\"";

query2 = "select \"sender\_id\" from \"transaction\"";

query3 = "select \"receiver\_id\" from \"transaction\"";

auto res1 = PQexec(connection, query1.c\_str());

for (int i = 0; i < PQntuples(res1); i++) {

if (PQgetvalue(res1, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\_access\"" << endl;

return false;

}

}

auto res2 = PQexec(connection, query2.c\_str());

for (int i = 0; i < PQntuples(res2); i++) {

if (PQgetvalue(res2, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"transaction\"" << endl;

return false;

}

}

auto res3 = PQexec(connection, query3.c\_str());

for (int i = 0; i < PQntuples(res3); i++) {

if (PQgetvalue(res3, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"transaction\"" << endl;

return false;

}

}

PQclear(res1);

PQclear(res2);

PQclear(res3);

string query = "delete from \"account\" where \"table\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordTransaction(int idx) {

string query = "delete from \"transaction\" where \"transaction\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

Ці функції перевіряють, чи є зв’язані таблиці. Якщо такі таблиці існують, вона перевіряє, чи значення, яке користувач намагається видалити, присутнє в зв’язаній таблиці. Якщо це значення є, видалення не дозволяється.

Функції генерування даних:

bool GenerateDataClient(int num) {

string query = "insert into \"client\""

"select\n"

"ROW\_NUMBER() OVER () as client\_id,\n"

"name,\n"

"phone\n"

"from generate\_series(1, 1000) as client\_id\n"

"cross join (select unnest(array['Denis', 'Andrii', 'Oleh', 'Bohdan', 'Oleksii', 'Mykola', 'Dmytro', 'Viktor', 'Ihor', 'Serhii', 'Vitalii', 'Oleksandr', 'Yaroslav', 'Ruslan', 'Roman', 'Taras', 'Artem', 'Valentyn', 'Pavlo', 'Maksym', 'Kyrylo', 'Yevhen', 'Vadym', 'Vladyslav', 'Stepan', 'Volodymyr', 'Yurii', 'Anatolii', 'Petro', 'Mykhailo']) as name)\n"

"cross join (select unnest(array['+380501234567', '+380631234567', '+380671234567', '+380931234567', '+380661234567', '+380981234567', '+380991234567', '+380681234567', '+380501112233', '+380631112233', '+380671112233', '+380931112233', '+380661112233', '+380981112233', '+380991112233', '+380681112233', '+380502223344', '+380632223344', '+380672223344', '+380932223344']) as phone)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataAccountAccess(int num) {

string query = "insert into \"account\_access\""

"select\n"

"ROW\_NUMBER() OVER () as table\_id,\n"

"client\_id,\n"

"account\_id\n"

"from generate\_series(1, 1000) as table\_id\n"

"cross join (select client\_id from \"client\" as client\_id)\n"

"cross join (select account\_id from \"account\" as account\_id)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataAccount(int num) {

string query = "insert into \"account\""

"select\n"

"ROW\_NUMBER() OVER () as account\_id,\n"

"balanse,\n"

"currency\_type,\n"

"account\_type,\n"

"client\_id as owner\_id\n"

"from generate\_series(1, 1000) as account\_id\n"

"cross join (select unnest(array[200, 900, 1600, 2300, 3000, 3700, 4400, 5100, 5800, 6500, 7200, 7900, 8600, 9300, 10000, 10700, 11400, 12100, 12800, 13500, 14200, 14900, 15600, 16300, 17000, 17700, 18400, 19100, 19800, 20000]) as balanse)\n"

"cross join (select unnest(array['USD', 'EUR', 'UAH', 'GBP', 'JPY', 'CHF', 'CAD', 'AUD', 'CNY', 'PLN']) as currency\_type)\n"

"cross join (select unnest(array['personal', 'corporative']) as account\_type)\n"

"cross join (select client\_id from \"client\")\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataTransaction(int num) {

string query = "insert into \"transaction\""

"select\n"

"ROW\_NUMBER() OVER () as transaction\_id,\n"

"date,\n"

"account\_id as sender\_id,\n"

"account\_id as receiver\_id,\n"

"amount\n"

"from generate\_series(1, 1000) as transaction\_id\n"

"cross join (select ('2019-03-18'::timestamp + RANDOM() \* ('2024-12-12'::timestamp - '2019-03-18'::timestamp))::date as date)\n"

"cross join (select account\_id from \"account\" as sender\_id)\n"

"cross join (select unnest(array[1400, 3600, 5800, 8000, 10200, 12400, 14600, 16800, 19000, 21200, 23400, 25600, 27800, 30000, 32200, 34400, 36600, 38800, 41000, 43200, 45400, 47600, 49800, 52000, 54200, 56400, 58600, 60800, 63000, 65200, 67400, 69600, 71800, 74000, 76200, 78400, 80600, 82800, 85000, 87200, 89400, 91600, 93800, 96000, 98200]) as amount)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

Ці функції створює визначену кількість нових записів у вказаній таблиці, при цьому первинний ключ завжди буде унікальним. Коли здійснюється вставка у поле зовнішніх ключів, для цього вибирається значення, яке вже існує в зв’язаній таблиці. Це забезпечує зв’язок між таблицями через зовнішні ключі. Для всіх інших полів вставка відбувається відповідно до їх типів.

Функції пошуку даних:

PGresult\* Query1(string param) {

string query = "select a.account\_type as account\_type, a.currency\_type as currency\_type, c.name as name "

"from \"account\" a "

"join \"client\" c on a.owner\_id = c.client\_id "

"where a.account\_type like $1 "

"group by a.account\_type, a.currency\_type, c.name";

param = "%" + param + "%";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

PGresult\* Query2(string param) {

string query = "select a.balanse as balance, c.name as name, c.phone as phone "

"from \"account\" a "

"join \"client\" c on a.owner\_id = c.client\_id "

"where a.balanse >= $1 "

"group by a.balanse, c.name, c.phone";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

PGresult\* Query3(string param) {

string query = "select aa.table\_id as table\_id, a.balanse as balanse, c.name as name "

"from \"account\_access\" aa "

"join \"client\" c on aa.client\_id = c.client\_id "

"join \"account\" a on aa.account\_id = a.account\_id "

"where aa.table\_id = $1 "

"group by aa.table\_id, a.balanse, c.name";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

Ці функції виконують за заданими параметрами 3 запити: імені клієнта за типом аккаунта, імені клієнта який має баланс більший за заданий параметр, баланс та ім’я клієнта за table\_id з таблиці account\_access.

**Повний код програми**

Файл main.cpp:

#include "controller.h"

#include <locale>

int main() {

setlocale(LC\_ALL, "en\_US.UTF-8");

controller Controller;

Controller.Start();

return 0;

}

Файл model.h:

#pragma once

#include <iostream>

#include <libpq-fe.h>

#include <string>

#include <vector>

#include <sstream>

#include <chrono>

#include <iomanip>

using namespace std;

class model {

public:

model() {

dur = static\_cast<chrono::milliseconds>(0);

const char\* info = "dbname=database1 user=postgres password=1234 host=localhost port=5432";

connection = PQconnectdb(info);

if (PQstatus(connection) != CONNECTION\_OK) {

cerr << PQerrorMessage(connection) << endl;

PQfinish(connection);

exit(1);

}

}

~model() {

PQfinish(connection);

}

PGresult\* ShowTable(string name) {

string query = "select \* from public.\"" + name + "\"";

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

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

{

cerr << PQerrorMessage(connection) << endl;

PQclear(res);

return nullptr;

}

return res;

}

bool AddRecordClient(vector<string> values) {

string query = "select \* from public.\"client\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"client\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordAccountAccess(vector<string> values) {

string query = "select \* from public.\"account\_access\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

PGresult\* result2 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[1] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

for (int i = 0; i < PQntuples(result2); i++) {

if (values[2] == PQgetvalue(result2, i, 0)) {

f2 = true;

break;

}

}

if (!f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

PQclear(result1);

PQclear(result2);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"account\_access\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordAccount(vector<string> values) {

string query = "select \* from public.\"account\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

bool f1 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[4] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"account\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool AddRecordTransaction(vector<string> values) {

string query = "select \* from public.\"transaction\"";

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

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

if (values[0] == PQgetvalue(res, i, 0)) {

cerr << "Index " << values[0] << " is already in use" << endl;

PQclear(res);

return false;

}

}

PGresult\* result1 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[2] == PQgetvalue(result1, i, 0))

f1 = true;

if (values[3] == PQgetvalue(result1, i, 0))

f2 = true;

if (f1 && f2)

break;

}

if (!f1 || !f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

string vals = "'";

for (auto& val : values)

vals += val + "', '";

vals.erase(vals.length() - 3);

query = "insert into \"transaction\" values(" + vals + ")";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordClient(vector<string> values, int idx) {

bool f = true;

string query = "select \* from public.\"client\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"client\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordAccountAccess(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

PGresult\* result2 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[1] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

for (int i = 0; i < PQntuples(result2); i++) {

if (values[2] == PQgetvalue(result2, i, 0)) {

f2 = true;

break;

}

}

if (!f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

PQclear(result2);

return false;

}

PQclear(result1);

PQclear(result2);

bool f = true;

string query = "select \* from public.\"account\_access\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"account\_access\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordAccount(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"client\_id\" from \"client\"");

bool f1 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[4] == PQgetvalue(result1, i, 0)) {

f1 = true;

break;

}

}

if (!f1) {

cerr << "The foreign key does not exist in the parent table \"client\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

bool f = true;

string query = "select \* from public.\"account\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"account\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool UpdateRecordTransaction(vector<string> values, int idx) {

PGresult\* result1 = PQexec(connection, "select \"account\_id\" from \"account\"");

bool f1 = false, f2 = false;

for (int i = 0; i < PQntuples(result1); i++) {

if (values[2] == PQgetvalue(result1, i, 0))

f1 = true;

if (values[3] == PQgetvalue(result1, i, 0))

f2 = true;

if (f1 && f2)

break;

}

if (!f1 || !f2) {

cerr << "The foreign key does not exist in the parent table \"account\"" << endl;

PQclear(result1);

return false;

}

PQclear(result1);

bool f = true;

string query = "select \* from public.\"transaction\"";

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

ostringstream oss;

for (int i = 1; i < PQnfields(res); i++) {

oss << "\"" << PQfname(res, i) << "\"='" << values[i - 1] << "'";

if (i < PQnfields(res) - 1)

oss << ", ";

}

query = "update \"transaction\" set " + oss.str() + " where " + PQfname(res, 0) + "=" + to\_string(idx);

PQclear(res);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordClient(int idx) {

string query1, query2;

query1 = "select \"client\_id\" from \"account\_access\"";

query2 = "select \"owner\_id\" from \"account\"";

auto res1 = PQexec(connection, query1.c\_str());

for (int i = 0; i < PQntuples(res1); i++) {

if (PQgetvalue(res1, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\_access\"" << endl;

return false;

}

}

auto res2 = PQexec(connection, query2.c\_str());

for (int i = 0; i < PQntuples(res2); i++) {

if (PQgetvalue(res2, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\"" << endl;

return false;

}

}

PQclear(res1);

PQclear(res2);

string query = "delete from \"client\" where \"client\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordAccountAccess(int idx) {

string query = "delete from \"account\_access\" where \"table\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordAccount(int idx) {

string query1, query2, query3;

query1 = "select \"account\_id\" from \"account\_access\"";

query2 = "select \"sender\_id\" from \"transaction\"";

query3 = "select \"receiver\_id\" from \"transaction\"";

auto res1 = PQexec(connection, query1.c\_str());

for (int i = 0; i < PQntuples(res1); i++) {

if (PQgetvalue(res1, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"account\_access\"" << endl;

return false;

}

}

auto res2 = PQexec(connection, query2.c\_str());

for (int i = 0; i < PQntuples(res2); i++) {

if (PQgetvalue(res2, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"transaction\"" << endl;

return false;

}

}

auto res3 = PQexec(connection, query3.c\_str());

for (int i = 0; i < PQntuples(res3); i++) {

if (PQgetvalue(res3, i, 0) == to\_string(idx)) {

cerr << "The table is linked to a child table \"transaction\"" << endl;

return false;

}

}

PQclear(res1);

PQclear(res2);

PQclear(res3);

string query = "delete from \"account\" where \"table\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool DeleteRecordTransaction(int idx) {

string query = "delete from \"transaction\" where \"transaction\_id\"='" + to\_string(idx) + "'";

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataClient(int num) {

string query = "insert into \"client\""

"select\n"

"ROW\_NUMBER() OVER () as client\_id,\n"

"name,\n"

"phone\n"

"from generate\_series(1, 1000) as client\_id\n"

"cross join (select unnest(array['Denis', 'Andrii', 'Oleh', 'Bohdan', 'Oleksii', 'Mykola', 'Dmytro', 'Viktor', 'Ihor', 'Serhii', 'Vitalii', 'Oleksandr', 'Yaroslav', 'Ruslan', 'Roman', 'Taras', 'Artem', 'Valentyn', 'Pavlo', 'Maksym', 'Kyrylo', 'Yevhen', 'Vadym', 'Vladyslav', 'Stepan', 'Volodymyr', 'Yurii', 'Anatolii', 'Petro', 'Mykhailo']) as name)\n"

"cross join (select unnest(array['+380501234567', '+380631234567', '+380671234567', '+380931234567', '+380661234567', '+380981234567', '+380991234567', '+380681234567', '+380501112233', '+380631112233', '+380671112233', '+380931112233', '+380661112233', '+380981112233', '+380991112233', '+380681112233', '+380502223344', '+380632223344', '+380672223344', '+380932223344']) as phone)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataAccountAccess(int num) {

string query = "insert into \"account\_access\""

"select\n"

"ROW\_NUMBER() OVER () as table\_id,\n"

"client\_id,\n"

"account\_id\n"

"from generate\_series(1, 1000) as table\_id\n"

"cross join (select client\_id from \"client\" as client\_id)\n"

"cross join (select account\_id from \"account\" as account\_id)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataAccount(int num) {

string query = "insert into \"account\""

"select\n"

"ROW\_NUMBER() OVER () as account\_id,\n"

"balanse,\n"

"currency\_type,\n"

"account\_type,\n"

"client\_id as owner\_id\n"

"from generate\_series(1, 1000) as account\_id\n"

"cross join (select unnest(array[200, 900, 1600, 2300, 3000, 3700, 4400, 5100, 5800, 6500, 7200, 7900, 8600, 9300, 10000, 10700, 11400, 12100, 12800, 13500, 14200, 14900, 15600, 16300, 17000, 17700, 18400, 19100, 19800, 20000]) as balanse)\n"

"cross join (select unnest(array['USD', 'EUR', 'UAH', 'GBP', 'JPY', 'CHF', 'CAD', 'AUD', 'CNY', 'PLN']) as currency\_type)\n"

"cross join (select unnest(array['personal', 'corporative']) as account\_type)\n"

"cross join (select client\_id from \"client\")\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

bool GenerateDataTransaction(int num) {

string query = "insert into \"transaction\""

"select\n"

"ROW\_NUMBER() OVER () as transaction\_id,\n"

"date,\n"

"account\_id as sender\_id,\n"

"account\_id as receiver\_id,\n"

"amount\n"

"from generate\_series(1, 1000) as transaction\_id\n"

"cross join (select ('2019-03-18'::timestamp + RANDOM() \* ('2024-12-12'::timestamp - '2019-03-18'::timestamp))::date as date)\n"

"cross join (select account\_id from \"account\" as sender\_id)\n"

"cross join (select unnest(array[1400, 3600, 5800, 8000, 10200, 12400, 14600, 16800, 19000, 21200, 23400, 25600, 27800, 30000, 32200, 34400, 36600, 38800, 41000, 43200, 45400, 47600, 49800, 52000, 54200, 56400, 58600, 60800, 63000, 65200, 67400, 69600, 71800, 74000, 76200, 78400, 80600, 82800, 85000, 87200, 89400, 91600, 93800, 96000, 98200]) as amount)\n"

"order by random()\n"

"limit " + to\_string(num);

if (PQresultStatus(PQexec(connection, query.c\_str())) != PGRES\_TUPLES\_OK) {

cerr << PQerrorMessage(connection) << endl;

return false;

}

return true;

}

PGresult\* Query1(string param) {

string query = "select a.account\_type as account\_type, a.currency\_type as currency\_type, c.name as name "

"from \"account\" a "

"join \"client\" c on a.owner\_id = c.client\_id "

"where a.account\_type like $1 "

"group by a.account\_type, a.currency\_type, c.name";

param = "%" + param + "%";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

PGresult\* Query2(string param) {

string query = "select a.balanse as balance, c.name as name, c.phone as phone "

"from \"account\" a "

"join \"client\" c on a.owner\_id = c.client\_id "

"where a.balanse >= $1 "

"group by a.balanse, c.name, c.phone";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

PGresult\* Query3(string param) {

string query = "select aa.table\_id as table\_id, a.balanse as balanse, c.name as name "

"from \"account\_access\" aa "

"join \"client\" c on aa.client\_id = c.client\_id "

"join \"account\" a on aa.account\_id = a.account\_id "

"where aa.table\_id = $1 "

"group by aa.table\_id, a.balanse, c.name";

const char\* paramValues[1] = { param.c\_str() };

auto start = chrono::high\_resolution\_clock::now();

PGresult\* res = PQexecParams(connection, query.c\_str(), 1, nullptr, paramValues, nullptr, nullptr, 0);

auto end = chrono::high\_resolution\_clock::now();

dur = chrono::duration\_cast<chrono::milliseconds>(end - start);

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

cerr << PQerrorMessage(connection) << endl;

return res;

}

return res;

}

long long GetDur() const {

return dur.count();

}

private:

PGconn\* connection;

chrono::milliseconds dur;

};

Файл view.h:

#pragma once

#include <iostream>

#include <string>

#include <libpq-fe.h>

using namespace std;

class view {

public:

void ShowMessage(string message) {

cout << message << endl;

}

void ShowTable(PGresult\* res) {

int rows = PQntuples(res);

int cols = PQnfields(res);

cout << left;

for (int i = 0; i < cols; i++)

cout << setw(30) << PQfname(res, i);

cout << endl;

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

for (int j = 0; j < cols; j++)

cout << setw(30) << PQgetvalue(res, i, j);

cout << endl;

}

cout << endl;

}

int MainMenu() {

cout << "Main menu:" << endl;

cout << "1. Show table\n2. Add record\n3. Update record\n4. Delete record\n5. Generate data\n6. 3 queries\n7. Exit" << endl;

int input;

cout << "Enter your choise: ";

cin >> input;

return input;

}

int QueriesMenu()

{

cout << "\nQueries menu:" << endl;

cout << "1. Client name by account\_type\n2. Client name by balanse\n3. Client name and account balanse by table\_id" << endl;

int input;

cout << "Enter your choise: ";

cin >> input;

return input;

}

int GetID() {

int id;

cout << "Enter ID: ";

cin >> id;

return id;

}

int GetNum() {

int num;

cout << "Enter number of random fields: ";

cin >> num;

return num;

}

string GetTableName() {

string name;

cout << "Enter table name: ";

cin >> name;

return name;

}

string GetValues() {

string values;

cout << "Enter values (separated by commas): ";

cin.ignore();

getline(cin, values);

return values;

}

string GetCondition() {

string condition;

cout << "Enter the required data to search: ";

cin >> condition;

return condition;

}

};

Файл controller.h:

#pragma once

#include "model.h"

#include "view.h"

class controller {

public:

controller() {

model Model;

view View;

View.ShowMessage("Database connected successfully\n");

}

void Start() {

while (true) {

int input = View.MainMenu();

if (input == 1)

OutputTable();

else if (input == 2)

AddRecord();

else if (input == 3)

UpdateData();

else if (input == 4)

DeleteData();

else if (input == 5)

GenerateRandomData();

else if (input == 6)

Queries();

else

cout << "Incorrect input. Try again" << endl;

}

}

private:

model Model;

view View;

vector<string> SplitString(string input) {

vector<string> words;

string word;

stringstream ss(input);

while (getline(ss, word, ',')) {

word.erase(0, word.find\_first\_not\_of(" \t"));

word.erase(word.find\_last\_not\_of(" \t") + 1);

words.push\_back(word);

}

return words;

}

void OutputTable()

{

string name = View.GetTableName();

auto res = Model.ShowTable(name);

if (res == nullptr)

View.ShowMessage("No such table.\n");

else

View.ShowTable(res);

}

void AddRecord() {

string name = View.GetTableName();

string vals = View.GetValues();

vector<string> values = SplitString(vals);

if (name == "client")

Model.AddRecordClient(values);

else if (name == "account\_access")

Model.AddRecordAccountAccess(values);

else if (name == "account")

Model.AddRecordAccount(values);

else if (name == "transaction")

Model.AddRecordTransaction(values);

else

View.ShowMessage("No such table.\n");

}

void UpdateData() {

string name = View.GetTableName();

int idx = View.GetID();

string vals = View.GetValues();

vector<string> values = SplitString(vals);

if (name == "client")

Model.UpdateRecordClient(values, idx);

else if (name == "account\_access")

Model.UpdateRecordAccountAccess(values, idx);

else if (name == "account")

Model.UpdateRecordAccount(values, idx);

else if (name == "transaction")

Model.UpdateRecordTransaction(values, idx);

else

View.ShowMessage("No such table.\n");

}

void DeleteData() {

string name = View.GetTableName();

int idx = View.GetID();

if (name == "client")

Model.DeleteRecordClient(idx);

else if (name == "account\_access")

Model.DeleteRecordAccountAccess(idx);

else if (name == "account")

Model.DeleteRecordAccount(idx);

else if (name == "transaction")

Model.DeleteRecordTransaction(idx);

else

View.ShowMessage("No such table.\n");

}

void GenerateRandomData() {

string name = View.GetTableName();

int num = View.GetNum();

if (name == "client")

Model.GenerateDataClient(num);

else if (name == "account\_access")

Model.GenerateDataAccountAccess(num);

else if (name == "account")

Model.GenerateDataAccount(num);

else if (name == "transaction")

Model.GenerateDataTransaction(num);

else

View.ShowMessage("No such table.\n");

}

void Queries()

{

int num = View.QueriesMenu();

string param = View.GetCondition();

if (num == 1) {

auto res = Model.Query1(param);

View.ShowTable(res);

string mess = "dur = " + Model.GetDur();

View.ShowMessage(mess);;

}

else if (num == 2) {

auto res = Model.Query2(param);

View.ShowTable(res);

string mess = "dur = " + Model.GetDur();

View.ShowMessage(mess);

}

else if (num == 3) {

auto res = Model.Query3(param);

View.ShowTable(res);

string mess = "dur = " + Model.GetDur();

View.ShowMessage(mess);

}

else

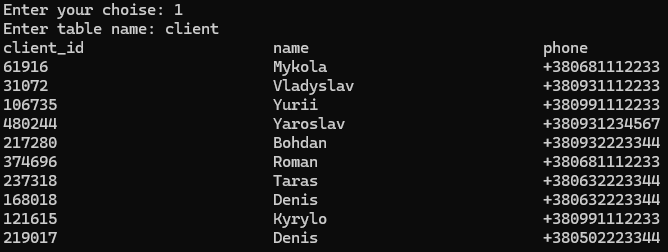
View.ShowMessage("Incorrect num\n");

}

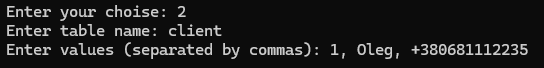
};

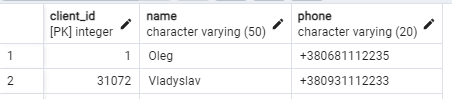
**Результат виконання програми**

Вивід даних:

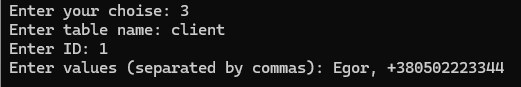
****

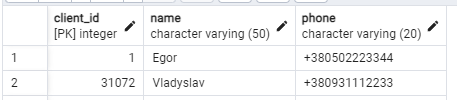
Вставка даних:

****

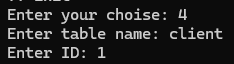
****

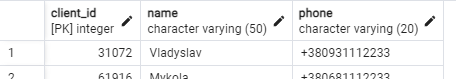
Редагування даних:

****

****

Видалення даних:

****

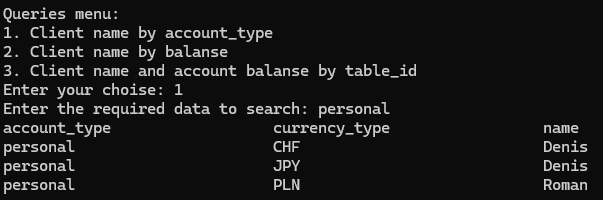
****

Генерування даних:

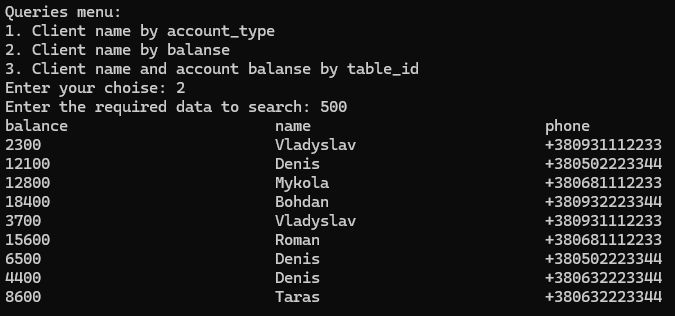


****

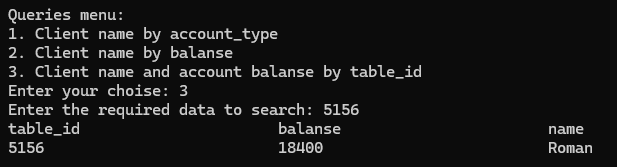
Пошуковий запит 1:

****

Пошуковий запит 2:

****

Пошуковий запит 3:

****Репозиторій Github**:**

[**https://github.com/andriyderkach1/KV22\_Derkach\_SQL\_RGR1**](https://github.com/andriyderkach1/KV22_Derkach_SQL_RGR1)