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”*

Виконав:

студентка ІІI курсу

групи КВ-22

Вакульчук Ярослав

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:

1. Користувач (User)

Атрибути ідентифікатор користувача, ім’я, прізвище, електронна пошта.

Призначення: збереження інформації про зареєстрованих користувачів платформи.

1. Книга (Book)

Атрибути: ідентифікатор книги, назва, автор, жанр, стан.

Призначення: збереження інформації про книги, доступні для обміну.

1. Оголошення (Listing)

Атрибути: ідентифікатор оголошення, статус оголошення, дата створення.

Призначення: збереження інформації про оголошення, створені користувачами, статус оголошення.

Нижче будуть наведені зв’язки між сутностями предметної області:

**Зв’язки між сутностями предметної області**

Зв’язок-сутність «Користувач» - «Книга» є зв’язком 1:N. Один користувач може додати кілька книг для обміну, але кожна книга належить тільки одному користувачу.

Зв’язок-сутність «Книга» - «Оголошеня» є зв’язком 1:1. Одна книга може бути пов’язана лише з одним оголошенням.

Зв’язок-сутність «Користувач» - «Оголошення» є зв’язком M:N. Один користувач може створити кілька оголошень на обмін та кожне оголошення може прийняти декілька користувачів.

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

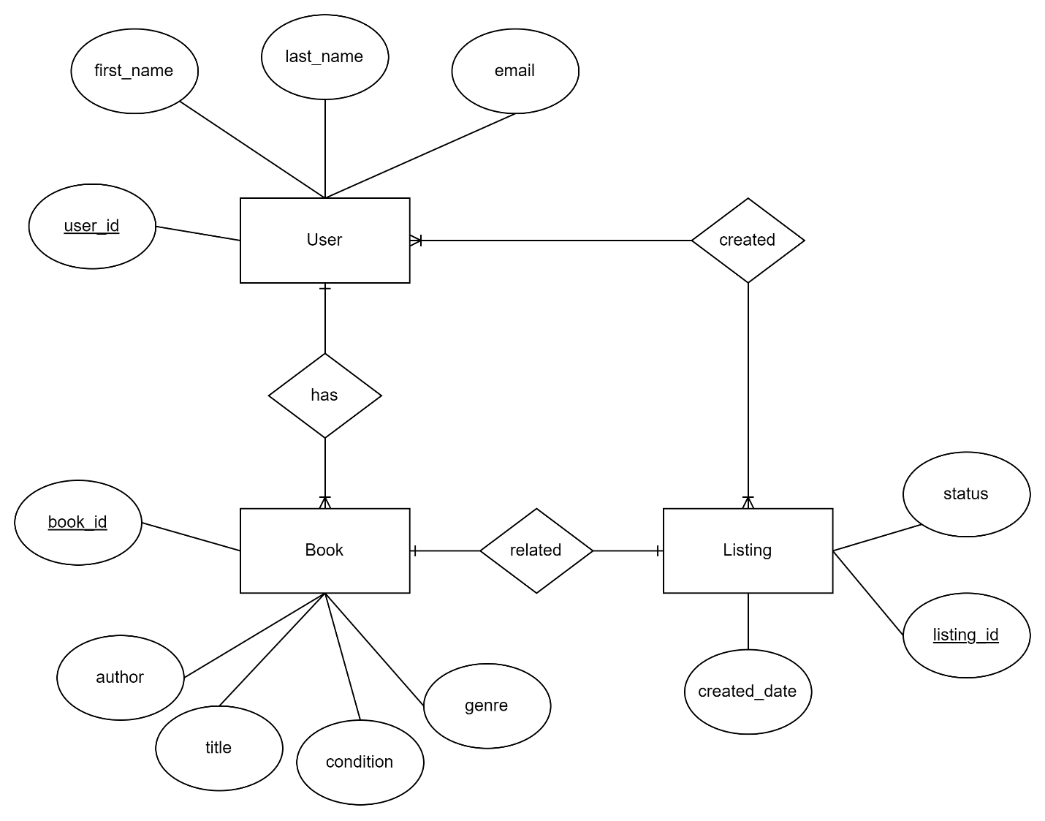


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

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

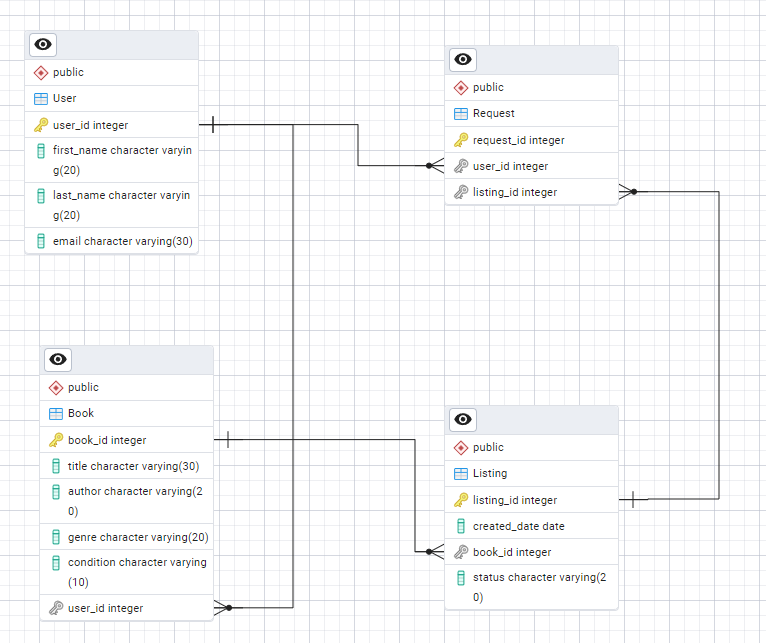


Рисунок 2 – Логічна модель

**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**

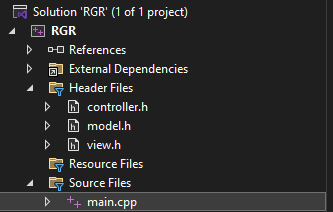
****

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.

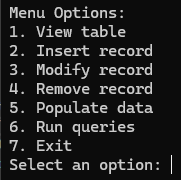


Рисунок 4 – Структура меню користувача

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

void Run() {

while (true) {

int choice = dbView.ShowMainMenu();

if (choice == 1)

ShowTableData();

else if (choice == 2)

InsertNewRecord();

else if (choice == 3)

ModifyRecord();

else if (choice == 4)

RemoveRecord();

else if (choice == 5)

PopulateDataRandomly();

else if (choice == 6)

ExecuteQuery();

else if (choice == 7)

break;

else

dbView.PrintMessage("Invalid option. Please try again.\n");

}

}

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

Функції виведення даних:

PGresult\* GetTable(string table\_name) {

string query = "SELECT \* FROM public.\"" + table\_name + "\"";

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

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

PQclear(res);

return nullptr;

}

return res;

}

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

Функції внесення даних:

void AddRecord\_user(vector<string> values) {

PGresult\* result = GetTable("User");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"User\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_book(vector<string> values) {

PGresult\* result = GetTable("Book");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[4] == PQgetvalue(result\_user, i, 0)) {

f = true;

break;

}

}

PQclear(result\_user);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Book\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_listing(vector<string> values) {

PGresult\* result = GetTable("Listing");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_book = PQexec(connection, "SELECT \"book\_id\" FROM \"Book\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_book); i++) {

if (values[2] == PQgetvalue(result\_book, i, 0)) {

f = true;

break;

}

}

PQclear(result\_book);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"Book\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Listing\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_request(vector<string> values) {

PGresult\* result = GetTable("Request");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f\_user = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[1] == PQgetvalue(result\_user, i, 0)) {

f\_user = true;

break;

}

}

PQclear(result\_user);

if (!f\_user)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result\_listing = PQexec(connection, "SELECT \"listing\_id\" FROM \"Listing\"");

bool f\_listing = false;

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (values[2] == PQgetvalue(result\_listing, i, 0)) {

f\_listing = true;

break;

}

}

PQclear(result\_listing);

if (!f\_listing)

throw runtime\_error("The foreign key does not exist in the parent table \"Listing\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Request\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

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:

void UpdateRecord\_user(vector<string> values, int idx) {

PGresult\* result = GetTable("User");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"User\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_book(vector<string> values, int idx) {

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[4] == PQgetvalue(result\_user, i, 0)) {

f = true;

break;

}

}

PQclear(result\_user);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result = GetTable("Book");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Book\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_listing(vector<string> values, int idx) {

PGresult\* result\_book = PQexec(connection, "SELECT \"book\_id\" FROM \"Book\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_book); i++) {

if (values[2] == PQgetvalue(result\_book, i, 0)) {

f = true;

break;

}

}

PQclear(result\_book);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"Book\"");

PGresult\* result = GetTable("Listing");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Listing\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_request(vector<string> values, int idx) {

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f\_user = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[1] == PQgetvalue(result\_user, i, 0)) {

f\_user = true;

break;

}

}

PQclear(result\_user);

if (!f\_user)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result\_listing = PQexec(connection, "SELECT \"listing\_id\" FROM \"Listing\"");

bool f\_listing = false;

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (values[2] == PQgetvalue(result\_listing, i, 0)) {

f\_listing = true;

break;

}

}

PQclear(result\_listing);

if (!f\_listing)

throw runtime\_error("The foreign key does not exist in the parent table \"Listing\"");

PGresult\* result = GetTable("Request");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Request\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

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:

void DeleteRecord\_user(int idx) {

PGresult\* result\_book = PQexec(connection, "SELECT \"user\_id\" from \"Book\"");

for (int i = 0; i < PQntuples(result\_book); i++) {

if (PQgetvalue(result\_book, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Book\"");

}

PGresult\* result\_request = PQexec(connection, "SELECT \"user\_id\" from \"Request\"");

for (int i = 0; i < PQntuples(result\_request); i++) {

if (PQgetvalue(result\_request, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Request\"");

}

PQclear(result\_book);

PQclear(result\_request);

string query = "DELETE FROM \"User\" WHERE \"user\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_book(int idx) {

PGresult\* result\_listing = PQexec(connection, "SELECT \"book\_id\" from \"Listing\"");

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (PQgetvalue(result\_listing, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Listing\"");

}

PQclear(result\_listing);

string query = "DELETE FROM \"Book\" WHERE \"book\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_listing(int idx) {

PGresult\* result\_request = PQexec(connection, "SELECT \"listing\_id\" from \"Request\"");

for (int i = 0; i < PQntuples(result\_request); i++) {

if (PQgetvalue(result\_request, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Request\"");

}

PQclear(result\_request);

string query = "DELETE FROM \"Listing\" WHERE \"listing\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_request(int idx) {

string query = "DELETE FROM \"Request\" WHERE \"request\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

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:

void PopulateData\_user(int limit) {

string query = "INSERT INTO \"User\" (user\_id, first\_name, last\_name, email)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS user\_id,\n"

"first\_name,\n"

"last\_name,\n"

"Lower(first\_name || '.' || last\_name) || '@example.com' as email\n"

"FROM (SELECT unnest(array['Mykola', 'Andrii', 'Oleh', 'Oleksandr', 'Ihor', 'Vasyl', 'Yaroslav', 'Dmytro', 'Bohdan', 'Petro', 'Volodymyr', 'Serhii', 'Yurii', 'Roman', 'Artem', 'Pavlo', 'Taras', 'Stepan', 'Vitalii', 'Oleksii', 'Vadym', 'Hryhorii', 'Leonid', 'Anatolii', 'Kyrylo', 'Ivan', 'Oksana', 'Natalia', 'Liudmyla', 'Halyna']) AS first\_name) AS first\_name\_table,\n"

"(SELECT unnest (array['Shevchenko', 'Kovalchuk', 'Bondarenko', 'Tkachenko', 'Kravchenko', 'Melnychenko', 'Lysenko', 'Tymoshenko', 'Chernenko', 'Hrytsenko', 'Ponomarenko', 'Martynenko', 'Levchenko', 'Yaremchenko', 'Vasylenko', 'Kozachenko', 'Polishchuk', 'Kulychenko', 'Savchenko', 'Zavhorodko', 'Horodko', 'Ivanenko', 'Karpenko', 'Novychenko', 'Radchenko', 'Shapovalenko', 'Verbychenko', 'Dovzhenko', 'Kalchenko', 'Kostenko']) AS last\_name) AS last\_name\_table\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);

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

throw runtime\_error(PQerrorMessage(connection));

}

void PopulateData\_book(int limit) {

string query = "INSERT INTO \"Book\" (book\_id, title, author, genre, condition, user\_id)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS book\_id,\n"

"title,\n"

"author,\n"

"genre,\n"

"condition,\n"

"user\_id\n"

"FROM (SELECT unnest(array['Vita et Mysterium', 'Sparks of Eternity', 'Shadows in the Night', 'The Silent Path', 'Whispers of the Forest', 'Beneath the Moonlight', 'Echoes of the Past', 'Wings of Freedom', 'Through the Storm', 'The Lost Symphony', 'Rays of Hope', 'Tales of the Forgotten', 'Into the Horizon', 'The Last Journey', 'Beyond the Veil', 'A Dance with Time', 'Reflections of the Soul', 'The Secret Garden', 'Chronicles of a Dreamer', 'The Hearts Song']) AS title),\n"

"(SELECT unnest(array['Shakespeare', 'Hemingway', 'Austen', 'Tolstoy', 'Dickens', 'Hemingway', 'Fitzgerald', 'Poe', 'Kafka', 'Orwell', 'Dostoevsky', 'Huxley', 'Hawthorne', 'Melville', 'Mann', 'Joyce', 'Tolkien', 'Woolf', 'Hemingway', 'Chekhov']) AS author),\n"

"(SELECT unnest(array['Fiction', 'Mystery', 'Fantasy', 'Romance', 'Science Fiction', 'Thriller', 'Horror', 'Historical Fiction', 'Adventure', 'Drama', 'Poetry', 'Dystopian', 'Biography', 'Memoir', 'Non-fiction', 'Psychological Thriller', 'Young Adult', 'Crime', 'Humor', 'Magical Realism']) AS genre),\n"

"(SELECT unnest(array['Brand New', 'Like New', 'Very Good', 'Good', 'Acceptable', 'Worn', 'Used', 'Damaged', 'Tattered', 'Well-Read']) AS condition),\n"

"(SELECT user\_id FROM \"User\" AS user\_id)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);;

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

throw runtime\_error(PQerrorMessage(connection));

}

void PopulateData\_listing(int limit) {

string query = "INSERT INTO \"Listing\" (listing\_id, created\_date, book\_id, status)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS listing\_id,\n"

"(date '2018-01-10' + (random() \* (date '2023-05-20' - date '2018-01-10'))::int) AS created\_date,\n"

"book\_id,\n"

"status\n"

"FROM "

"(SELECT book\_id FROM \"Book\" AS book\_id)\n"

"CROSS JOIN (SELECT unnest(array['Pending', 'Approved', 'Rejected', 'In Progress', 'Completed', 'Cancelled', 'Awaiting Confirmation', 'On Hold', 'Transferred', 'Returned']) AS status)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);;

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

throw runtime\_error(PQerrorMessage(connection));

}

}

void PopulateData\_request(int limit) {

string query = "INSERT INTO \"Request\" (request\_id, user\_id, listing\_id)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS request\_id,\n"

"user\_id,\n"

"listing\_id\n"

"FROM (SELECT user\_id FROM \"User\" AS user\_id)\n"

"CROSS JOIN (SELECT listing\_id FROM \"Listing\" AS listing\_id)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);

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

throw runtime\_error(PQerrorMessage(connection));

}

}

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\* QueryStatus(string status) {

string query = "SELECT u.first\_name AS first\_name, u.last\_name AS last\_name, b.title AS title, l.status AS status\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"JOIN \"Book\" b ON l.book\_id = b.book\_id\n"

"WHERE l.status LIKE $1\n"

"ORDER BY u.first\_name, u.last\_name, b.title, l.status;";

status = "%" + status + "%";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

PGresult\* QueryDate(string date) {

string query = "SELECT u.first\_name AS first\_name, u.last\_name AS last\_name, b.title AS title, l.created\_date AS created\_date\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"JOIN \"Book\" b ON l.book\_id = b.book\_id\n"

"WHERE l.created\_date > $1\n"

"ORDER BY u.first\_name, u.last\_name, b.title, l.created\_date;";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

PGresult\* QueryID(string date) {

string query = "SELECT u.user\_id AS user\_id, l.listing\_id AS listing\_id, l.created\_date AS created\_date, l.status AS status\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"WHERE u.user\_id = $1\n"

"ORDER BY u.user\_id, l.listing\_id, l.created\_date, l.status;";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

Ці функції виконують за заданими параметрами 3 запити: назва книжки та ім’я користувача за статусом оголошення, назва книжки та ім’я користувача оголошення якого молодше за дату, дані про оголошення за ID користувача.

**Повний код програми**

Файл main.cpp:

#include "controller.h"

#include <locale>

int main() {

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

Manager dbController;

dbController.Run();

return 0;

}

Файл model.h:

#pragma once

#include <iostream>

#include <libpq-fe.h>

#include <string>

#include <vector>

#include <sstream>

#include <chrono>

#include <iomanip>

#include <stdexcept>

using namespace std;

class Database {

public:

Database() {

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

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

connection = PQconnectdb(info);

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

throw runtime\_error(PQerrorMessage(connection));

PQfinish(connection);

exit(1);

}

}

~Database() {

PQfinish(connection);

}

PGresult\* GetTable(string table\_name) {

string query = "SELECT \* FROM public.\"" + table\_name + "\"";

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

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

PQclear(res);

return nullptr;

}

return res;

}

void AddRecord\_user(vector<string> values) {

PGresult\* result = GetTable("User");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"User\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_book(vector<string> values) {

PGresult\* result = GetTable("Book");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[4] == PQgetvalue(result\_user, i, 0)) {

f = true;

break;

}

}

PQclear(result\_user);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Book\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_listing(vector<string> values) {

PGresult\* result = GetTable("Listing");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_book = PQexec(connection, "SELECT \"book\_id\" FROM \"Book\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_book); i++) {

if (values[2] == PQgetvalue(result\_book, i, 0)) {

f = true;

break;

}

}

PQclear(result\_book);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"Book\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Listing\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void AddRecord\_request(vector<string> values) {

PGresult\* result = GetTable("Request");

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

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

throw runtime\_error("Index " + values[0] + " is already in use");

PQclear(result);

}

}

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f\_user = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[1] == PQgetvalue(result\_user, i, 0)) {

f\_user = true;

break;

}

}

PQclear(result\_user);

if (!f\_user)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result\_listing = PQexec(connection, "SELECT \"listing\_id\" FROM \"Listing\"");

bool f\_listing = false;

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (values[2] == PQgetvalue(result\_listing, i, 0)) {

f\_listing = true;

break;

}

}

PQclear(result\_listing);

if (!f\_listing)

throw runtime\_error("The foreign key does not exist in the parent table \"Listing\"");

string vals = "'";

for (auto& val : values)

vals += val + "', '";

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

string query = "INSERT INTO \"Request\" VALUES(" + vals + ")";

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_user(vector<string> values, int idx) {

PGresult\* result = GetTable("User");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"User\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_book(vector<string> values, int idx) {

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[4] == PQgetvalue(result\_user, i, 0)) {

f = true;

break;

}

}

PQclear(result\_user);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result = GetTable("Book");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Book\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_listing(vector<string> values, int idx) {

PGresult\* result\_book = PQexec(connection, "SELECT \"book\_id\" FROM \"Book\"");

bool f = false;

for (int i = 0; i < PQntuples(result\_book); i++) {

if (values[2] == PQgetvalue(result\_book, i, 0)) {

f = true;

break;

}

}

PQclear(result\_book);

if (!f)

throw runtime\_error("The foreign key does not exist in the parent table \"Book\"");

PGresult\* result = GetTable("Listing");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Listing\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void UpdateRecord\_request(vector<string> values, int idx) {

PGresult\* result\_user = PQexec(connection, "SELECT \"user\_id\" FROM \"User\"");

bool f\_user = false;

for (int i = 0; i < PQntuples(result\_user); i++) {

if (values[1] == PQgetvalue(result\_user, i, 0)) {

f\_user = true;

break;

}

}

PQclear(result\_user);

if (!f\_user)

throw runtime\_error("The foreign key does not exist in the parent table \"User\"");

PGresult\* result\_listing = PQexec(connection, "SELECT \"listing\_id\" FROM \"Listing\"");

bool f\_listing = false;

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (values[2] == PQgetvalue(result\_listing, i, 0)) {

f\_listing = true;

break;

}

}

PQclear(result\_listing);

if (!f\_listing)

throw runtime\_error("The foreign key does not exist in the parent table \"Listing\"");

PGresult\* result = GetTable("Request");

ostringstream oss;

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

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

if (i < PQnfields(result) - 1)

oss << ", ";

}

string query = "UPDATE \"Request\" SET " + oss.str() + " WHERE " + PQfname(result, 0) + "=" + to\_string(idx);

PQclear(result);

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_user(int idx) {

PGresult\* result\_book = PQexec(connection, "SELECT \"user\_id\" from \"Book\"");

for (int i = 0; i < PQntuples(result\_book); i++) {

if (PQgetvalue(result\_book, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Book\"");

}

PGresult\* result\_request = PQexec(connection, "SELECT \"user\_id\" from \"Request\"");

for (int i = 0; i < PQntuples(result\_request); i++) {

if (PQgetvalue(result\_request, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Request\"");

}

PQclear(result\_book);

PQclear(result\_request);

string query = "DELETE FROM \"User\" WHERE \"user\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_book(int idx) {

PGresult\* result\_listing = PQexec(connection, "SELECT \"book\_id\" from \"Listing\"");

for (int i = 0; i < PQntuples(result\_listing); i++) {

if (PQgetvalue(result\_listing, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Listing\"");

}

PQclear(result\_listing);

string query = "DELETE FROM \"Book\" WHERE \"book\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_listing(int idx) {

PGresult\* result\_request = PQexec(connection, "SELECT \"listing\_id\" from \"Request\"");

for (int i = 0; i < PQntuples(result\_request); i++) {

if (PQgetvalue(result\_request, i, 0) == to\_string(idx))

throw runtime\_error("The table is linked to a child table \"Request\"");

}

PQclear(result\_request);

string query = "DELETE FROM \"Listing\" WHERE \"listing\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void DeleteRecord\_request(int idx) {

string query = "DELETE FROM \"Request\" WHERE \"request\_id\"='" + to\_string(idx) + "'";

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

throw runtime\_error(PQerrorMessage(connection));

}

void PopulateData\_user(int limit) {

string query = "INSERT INTO \"User\" (user\_id, first\_name, last\_name, email)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS user\_id,\n"

"first\_name,\n"

"last\_name,\n"

"Lower(first\_name || '.' || last\_name) || '@example.com' as email\n"

"FROM (SELECT unnest(array['Mykola', 'Andrii', 'Oleh', 'Oleksandr', 'Ihor', 'Vasyl', 'Yaroslav', 'Dmytro', 'Bohdan', 'Petro', 'Volodymyr', 'Serhii', 'Yurii', 'Roman', 'Artem', 'Pavlo', 'Taras', 'Stepan', 'Vitalii', 'Oleksii', 'Vadym', 'Hryhorii', 'Leonid', 'Anatolii', 'Kyrylo', 'Ivan', 'Oksana', 'Natalia', 'Liudmyla', 'Halyna']) AS first\_name) AS first\_name\_table,\n"

"(SELECT unnest (array['Shevchenko', 'Kovalchuk', 'Bondarenko', 'Tkachenko', 'Kravchenko', 'Melnychenko', 'Lysenko', 'Tymoshenko', 'Chernenko', 'Hrytsenko', 'Ponomarenko', 'Martynenko', 'Levchenko', 'Yaremchenko', 'Vasylenko', 'Kozachenko', 'Polishchuk', 'Kulychenko', 'Savchenko', 'Zavhorodko', 'Horodko', 'Ivanenko', 'Karpenko', 'Novychenko', 'Radchenko', 'Shapovalenko', 'Verbychenko', 'Dovzhenko', 'Kalchenko', 'Kostenko']) AS last\_name) AS last\_name\_table\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);

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

throw runtime\_error(PQerrorMessage(connection));

}

void PopulateData\_book(int limit) {

string query = "INSERT INTO \"Book\" (book\_id, title, author, genre, condition, user\_id)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS book\_id,\n"

"title,\n"

"author,\n"

"genre,\n"

"condition,\n"

"user\_id\n"

"FROM (SELECT unnest(array['Vita et Mysterium', 'Sparks of Eternity', 'Shadows in the Night', 'The Silent Path', 'Whispers of the Forest', 'Beneath the Moonlight', 'Echoes of the Past', 'Wings of Freedom', 'Through the Storm', 'The Lost Symphony', 'Rays of Hope', 'Tales of the Forgotten', 'Into the Horizon', 'The Last Journey', 'Beyond the Veil', 'A Dance with Time', 'Reflections of the Soul', 'The Secret Garden', 'Chronicles of a Dreamer', 'The Hearts Song']) AS title),\n"

"(SELECT unnest(array['Shakespeare', 'Hemingway', 'Austen', 'Tolstoy', 'Dickens', 'Hemingway', 'Fitzgerald', 'Poe', 'Kafka', 'Orwell', 'Dostoevsky', 'Huxley', 'Hawthorne', 'Melville', 'Mann', 'Joyce', 'Tolkien', 'Woolf', 'Hemingway', 'Chekhov']) AS author),\n"

"(SELECT unnest(array['Fiction', 'Mystery', 'Fantasy', 'Romance', 'Science Fiction', 'Thriller', 'Horror', 'Historical Fiction', 'Adventure', 'Drama', 'Poetry', 'Dystopian', 'Biography', 'Memoir', 'Non-fiction', 'Psychological Thriller', 'Young Adult', 'Crime', 'Humor', 'Magical Realism']) AS genre),\n"

"(SELECT unnest(array['Brand New', 'Like New', 'Very Good', 'Good', 'Acceptable', 'Worn', 'Used', 'Damaged', 'Tattered', 'Well-Read']) AS condition),\n"

"(SELECT user\_id FROM \"User\" AS user\_id)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);;

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

throw runtime\_error(PQerrorMessage(connection));

}

void PopulateData\_listing(int limit) {

string query = "INSERT INTO \"Listing\" (listing\_id, created\_date, book\_id, status)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS listing\_id,\n"

"(date '2018-01-10' + (random() \* (date '2023-05-20' - date '2018-01-10'))::int) AS created\_date,\n"

"book\_id,\n"

"status\n"

"FROM "

"(SELECT book\_id FROM \"Book\" AS book\_id)\n"

"CROSS JOIN (SELECT unnest(array['Pending', 'Approved', 'Rejected', 'In Progress', 'Completed', 'Cancelled', 'Awaiting Confirmation', 'On Hold', 'Transferred', 'Returned']) AS status)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);;

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

throw runtime\_error(PQerrorMessage(connection));

}

}

void PopulateData\_request(int limit) {

string query = "INSERT INTO \"Request\" (request\_id, user\_id, listing\_id)\n"

"SELECT\n"

"ROW\_NUMBER() OVER () AS request\_id,\n"

"user\_id,\n"

"listing\_id\n"

"FROM (SELECT user\_id FROM \"User\" AS user\_id)\n"

"CROSS JOIN (SELECT listing\_id FROM \"Listing\" AS listing\_id)\n"

"ORDER BY RANDOM()\n"

"LIMIT " + to\_string(limit);

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

throw runtime\_error(PQerrorMessage(connection));

}

}

PGresult\* QueryStatus(string status) {

string query = "SELECT u.first\_name AS first\_name, u.last\_name AS last\_name, b.title AS title, l.status AS status\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"JOIN \"Book\" b ON l.book\_id = b.book\_id\n"

"WHERE l.status LIKE $1\n"

"ORDER BY u.first\_name, u.last\_name, b.title, l.status;";

status = "%" + status + "%";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

PGresult\* QueryDate(string date) {

string query = "SELECT u.first\_name AS first\_name, u.last\_name AS last\_name, b.title AS title, l.created\_date AS created\_date\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"JOIN \"Book\" b ON l.book\_id = b.book\_id\n"

"WHERE l.created\_date > $1\n"

"ORDER BY u.first\_name, u.last\_name, b.title, l.created\_date;";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

PGresult\* QueryID(string date) {

string query = "SELECT u.user\_id AS user\_id, l.listing\_id AS listing\_id, l.created\_date AS created\_date, l.status AS status\n"

"FROM \"Request\" r\n"

"JOIN \"User\" u ON r.user\_id = u.user\_id\n"

"JOIN \"Listing\" l ON r.listing\_id = l.listing\_id\n"

"WHERE u.user\_id = $1\n"

"ORDER BY u.user\_id, l.listing\_id, l.created\_date, l.status;";

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

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

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

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

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

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

{

PQclear(result);

throw runtime\_error(PQerrorMessage(connection));

}

return result;

}

chrono::milliseconds GetTime() const {

return time;

}

private:

PGconn\* connection;

chrono::milliseconds time;

};

Файл view.h:

#pragma once

#include <iostream>

#include <string>

#include <libpq-fe.h>

#include <iomanip>

using namespace std;

class Display {

public:

void PrintMessage(const string& msg) {

cout << msg << endl;

}

void PrintTable(PGresult\* result) {

cout << endl;

int rowCount = PQntuples(result);

int colCount = PQnfields(result);

cout << left;

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

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

cout << endl;

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

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

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

cout << endl;

}

cout << endl;

}

int ShowMainMenu() {

cout << "\nMenu Options:" << endl;

cout << "1. View table\n2. Insert record\n3. Modify record\n4. Remove record\n5. Populate data\n6. Run queries\n7. Exit" << endl;

int choice;

cout << "Select an option: ";

cin >> choice;

return choice;

}

int ShowQueryMenu() {

cout << "\nQuery Options:" << endl;

cout << "1. Display the book title and username by listing status\n2. Display the book title and username whose listing is younger than the date\n3. Display listing data by user ID" << endl;

int choice;

cout << "Select a query: ";

cin >> choice;

return choice;

}

int PromptID() {

int id;

cout << "Enter the ID: ";

cin >> id;

return id;

}

int PromptCount() {

int count;

cout << "Enter the number of entries to generate: ";

cin >> count;

return count;

}

string PromptTableName() {

string tableName;

cout << "Provide the table name: ";

cin >> tableName;

return tableName;

}

string PromptValues() {

string entries;

cout << "Enter values (comma-separated): ";

cin.ignore();

getline(cin, entries);

return entries;

}

string PromptCondition() {

string filter;

cout << "Specify the condition for the query: ";

cin >> filter;

return filter;

}

};

Файл controller.h:

#pragma once

#include "model.h"

#include "view.h"

#include <chrono>

using namespace std;

class Manager {

public:

Manager() {

Database dbModel;

Display dbView;

dbView.PrintMessage("Successfully connected to the database.");

}

void Run() {

while (true) {

int choice = dbView.ShowMainMenu();

if (choice == 1)

ShowTableData();

else if (choice == 2)

InsertNewRecord();

else if (choice == 3)

ModifyRecord();

else if (choice == 4)

RemoveRecord();

else if (choice == 5)

PopulateDataRandomly();

else if (choice == 6)

ExecuteQuery();

else if (choice == 7)

break;

else

dbView.PrintMessage("Invalid option. Please try again.\n");

}

}

private:

Database dbModel;

Display dbView;

vector<string> ParseInputString(const string& input) {

vector<string> parsedValues;

string value;

stringstream ss(input);

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

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

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

parsedValues.push\_back(value);

}

return parsedValues;

}

void ShowTableData() {

string tableName = dbView.PromptTableName();

auto result = dbModel.GetTable(tableName);

if (result == nullptr)

dbView.PrintMessage("Table does not exist.");

else

dbView.PrintTable(result);

}

void InsertNewRecord() {

string tableName = dbView.PromptTableName();

string inputValues = dbView.PromptValues();

vector<string> values = ParseInputString(inputValues);

try {

if (tableName == "User")

dbModel.AddRecord\_user(values);

else if (tableName == "Book")

dbModel.AddRecord\_book(values);

else if (tableName == "Listing")

dbModel.AddRecord\_listing(values);

else if (tableName == "Request")

dbModel.AddRecord\_request(values);

else

dbView.PrintMessage("Table does not exist.");

dbView.PrintMessage("Record was inserted successfully");

}

catch (const exception& e) {

dbView.PrintMessage(e.what());

}

}

void ModifyRecord() {

string tableName = dbView.PromptTableName();

int recordID = dbView.PromptID();

string inputValues = dbView.PromptValues();

vector<string> values = ParseInputString(inputValues);

try {

if (tableName == "User")

dbModel.UpdateRecord\_user(values, recordID);

else if (tableName == "Book")

dbModel.UpdateRecord\_book(values, recordID);

else if (tableName == "Listing")

dbModel.UpdateRecord\_listing(values, recordID);

else if (tableName == "Request")

dbModel.UpdateRecord\_request(values, recordID);

else

dbView.PrintMessage("Table does not exist.");

dbView.PrintMessage("Record was modified successfully");

}

catch (const exception& e) {

dbView.PrintMessage(e.what());

}

}

void RemoveRecord() {

string tableName = dbView.PromptTableName();

int recordID = dbView.PromptID();

try {

if (tableName == "User")

dbModel.DeleteRecord\_user(recordID);

else if (tableName == "Book")

dbModel.DeleteRecord\_book(recordID);

else if (tableName == "Listing")

dbModel.DeleteRecord\_listing(recordID);

else if (tableName == "Request")

dbModel.DeleteRecord\_request(recordID);

else

dbView.PrintMessage("Table does not exist.");

dbView.PrintMessage("Record was removed successfully");

}

catch (const exception& e) {

dbView.PrintMessage(e.what());

}

}

void PopulateDataRandomly() {

string tableName = dbView.PromptTableName();

int count = dbView.PromptCount();

try {

if (tableName == "User")

dbModel.PopulateData\_user(count);

else if (tableName == "Book")

dbModel.PopulateData\_book(count);

else if (tableName == "Listing")

dbModel.PopulateData\_listing(count);

else if (tableName == "Request")

dbModel.PopulateData\_request(count);

else

dbView.PrintMessage("Table does not exist.");

dbView.PrintMessage("Data was populated successfully");

}

catch (const exception& e) {

dbView.PrintMessage(e.what());

}

}

void ExecuteQuery() {

int queryChoice = dbView.ShowQueryMenu();

string parameter = dbView.PromptCondition();

try {

if (queryChoice == 1) {

PGresult\* result = dbModel.QueryStatus(parameter);

dbView.PrintTable(result);

}

else if (queryChoice == 2) {

PGresult\* result = dbModel.QueryDate(parameter);

dbView.PrintTable(result);

}

else if (queryChoice == 3) {

PGresult\* result = dbModel.QueryID(parameter);

dbView.PrintTable(result);

}

else {

dbView.PrintMessage("Invalid query choice.\n");

}

long long duration = dbModel.GetTime().count();

dbView.PrintMessage("Query duration: " + to\_string(duration) + " ms");

}

catch (const exception& e) {

dbView.PrintMessage(e.what());

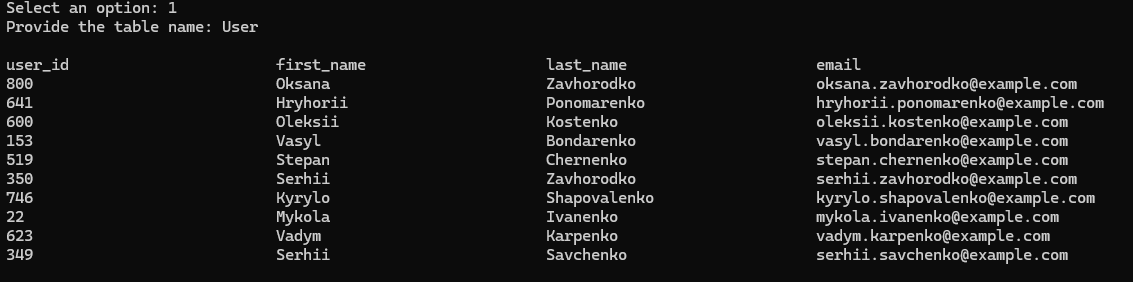
}

}

};

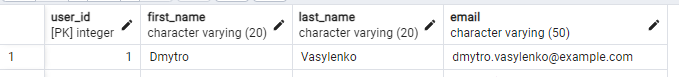
**Результат виконання програми**

Вивід даних:

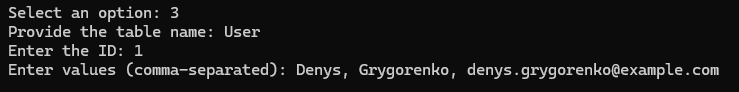
****

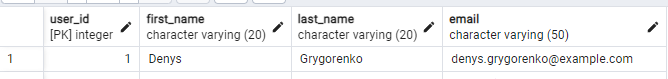
Вставка даних:

****

****

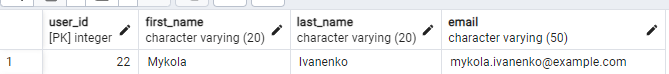
Редагування даних:

****

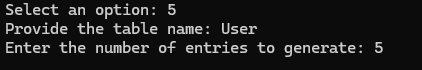
****

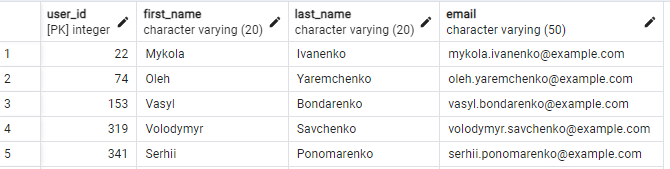
Видалення даних:

****

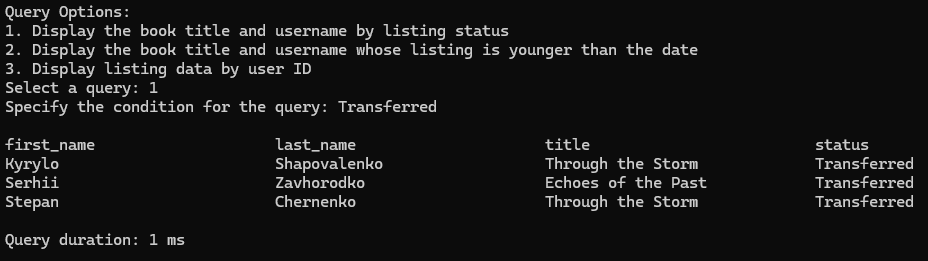
****

Генерування даних:

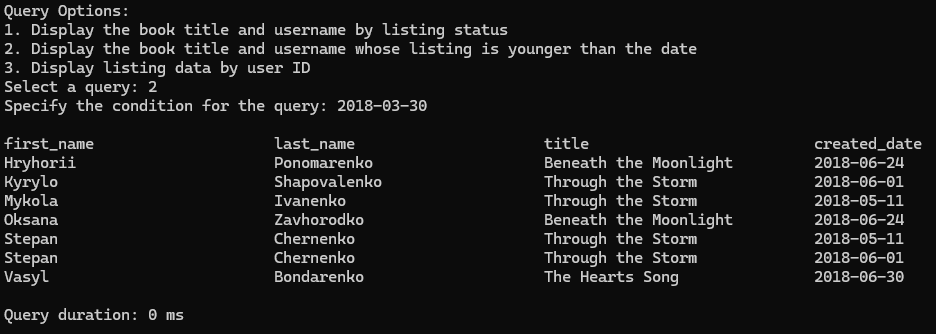


****

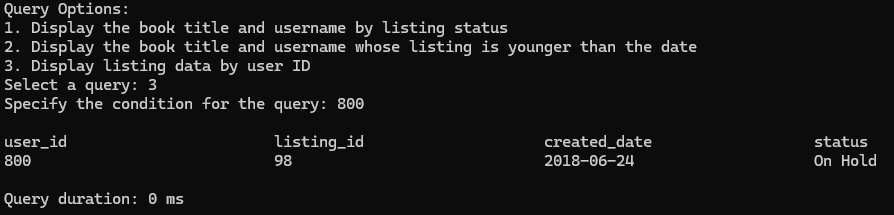
Пошуковий запит 1:

****

Пошуковий запит 2:

****

Пошуковий запит 3:

****

**Репозиторій Git: https://github.com/Yaroslavvak/rgr**