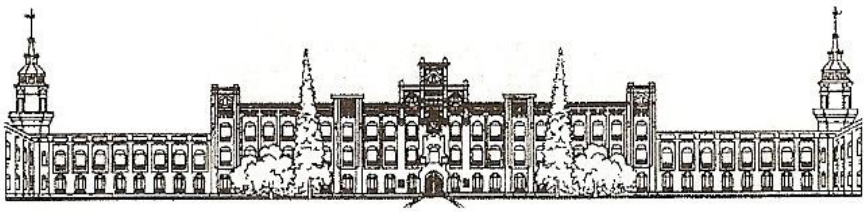

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Розрахункова-графічна робота

з дисципліни «Бази даних»

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

Виконав студент групи: КВ-33

ПІБ: Поляков Єгор Олегович

Перевірив:

Київ 2025

Постановка задачі:

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQT, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Опис предметної галузі

Тема бази даних: **Електронний довідник для зберігання технічної документації** - це база даних, призначена для централізованого збереження, упорядкування та швидкого доступу до технічних матеріалів. Вона дозволить накопичувати, структуризувати та оновлювати документацію, забезпечує зручний пошук і перегляд даних, а також сприяє ефективній організації роботи з технічною інформацією відповідно до вимог підприємства чи навчального закладу.

При створенні даної бази даних було виділено такі сутності:

1. **Працівник (Employee)**

Атрибути: ім’я, прізвище, електронна пошта, відділ.

Призначення: зберігає дані про тих, хто працює з документами.

2. **Документ (Document)**

Атрибути: назва, опис, дата створення, дата оновлення, версія.

Призначення: зберігає основну інформацію про технічну документацію.

3. **Категорія (Category)**

Атрибути: назва, опис, тип.

Призначення: групує документи за темами чи напрямами.

4. **Файл (File)**

Атрибути: назва, формат, розмір, дата завантаження.

Призначення: зберігає електронні файли, прикріплені до документа.

5. **Доступ (Access)**

Атрибути: рівень доступу(читання / редагування / видалення), дата надання доступу, строк дії доступу.

Призначення: регулює права доступу працівників до документів.

Зв’язки між сутностями:

Зв’язок «Працівник – Документ» є зв’язком 1:N. Один документ має лише одного автора, але один працівник може створювати багато документів.

Зв’язок «Категорія – Документ» є зв’язком 1:N. Один документ належить лише до однієї категорії, але одна категорія може містити багато документів.

Зв’язок «Документ – Файл» є зв’язком 1:N. Один файл належить лише одному документу, але один документ може мати багато файлів.

Зв’язок «Працівник – Доступ» є зв’язком 1:N. Один працівник може мати багато доступів, але один доступ відноситься лише до одного працівника.

Зв’язок «Доступ – Документ» є зв’язком 1:N. Один доступ може поширюватися на багато документів, але один документ пов'язаний лише з одним доступом.

Графічне подання концептуальної моделі «Сутність-зв’язок» (*ER-діаграма побудована за нотацією “Пташина лапка”*) зображено на рис. 1.

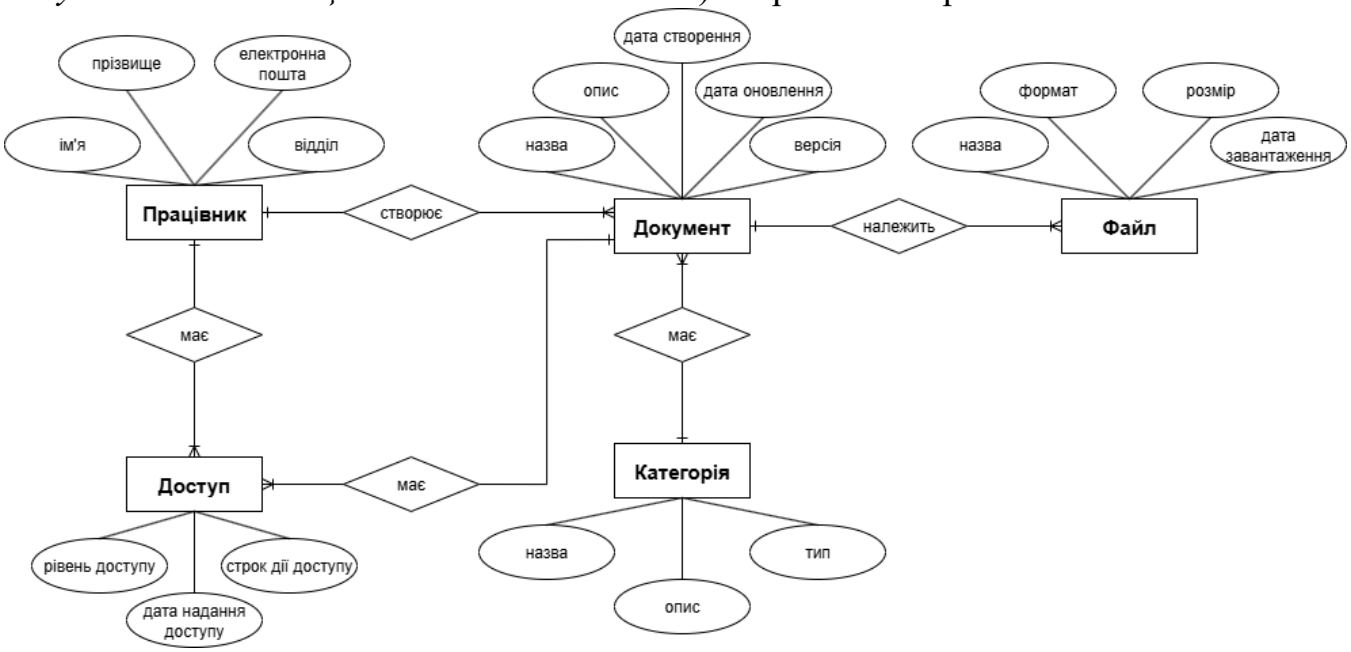

Рис. 1 – Концептуальна модель «Сутність-зв’язок»

Перетворення концептуальної моделі у логічну модель та схему бази даних

Сутність «Працівник» перетворено в таблицю Employee з первинним ключем employee_id та атрибутами first_name, last_name, email, department.

Сутність «Категорія» перетворено в таблицю Category з первинним ключем category_id та атрибутами category_name, description, type.

Сутність «Документ» перетворено в таблицю Document з первинним ключем document_id та атрибутами title, description, date_created, date_updated, version. У таблиці є зовнішні ключі employee_id та category_id, які пов’язані з таблицями Employee та Category відповідно.

Сутність «Файл» перетворено в таблицю File з первинним ключем file_id та атрибутами file_name, format, size, upload_date. У таблиці є зовнішній ключ document_id, який пов’язаний з таблицею Document.

Сутність «Доступ» перетворено в таблицю Access з первинним ключем access_id та атрибутами access_level, date_granted, expiry_date. У таблиці є зовнішні ключі employee_id та document_id, які пов’язані з таблицями Employee та Document відповідно.

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

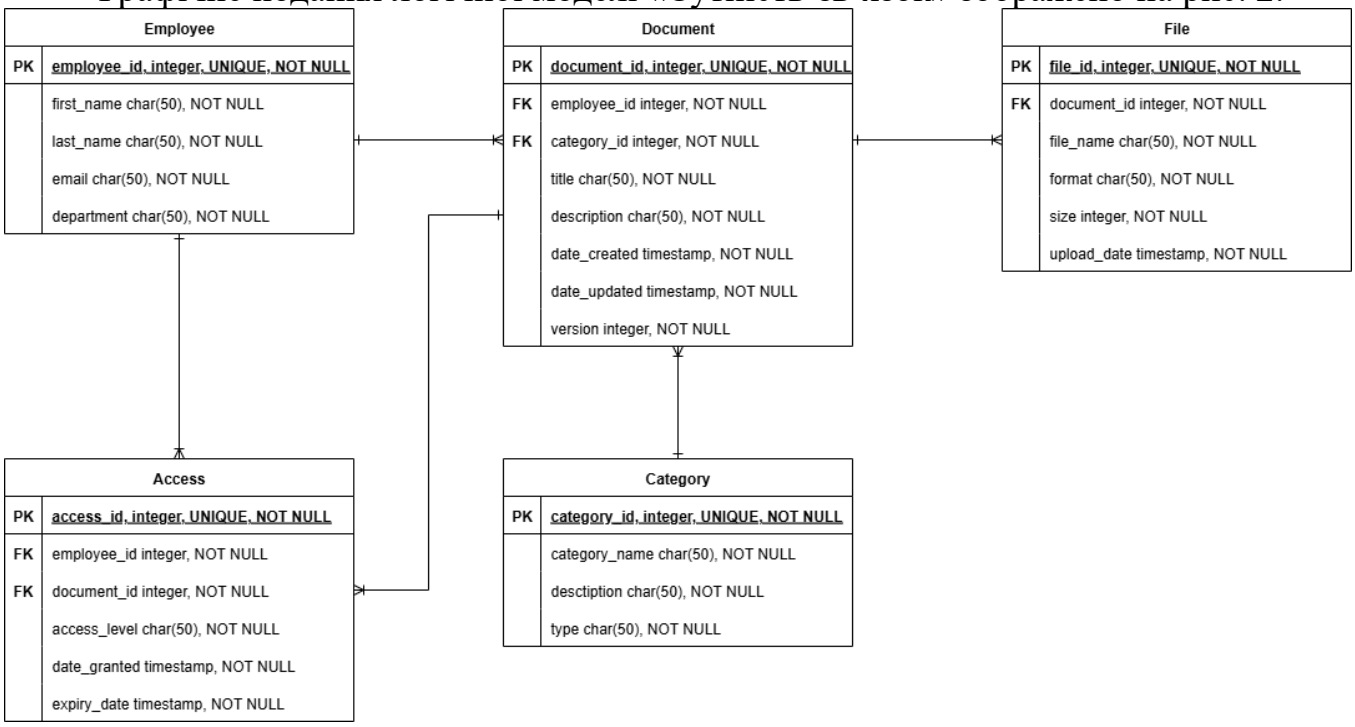

Рис. 2 – Логічна модель «Сутність-зв’язок»

Графічне подання схеми бази даних у pgAdmin 4 зображено на рисунку 3.

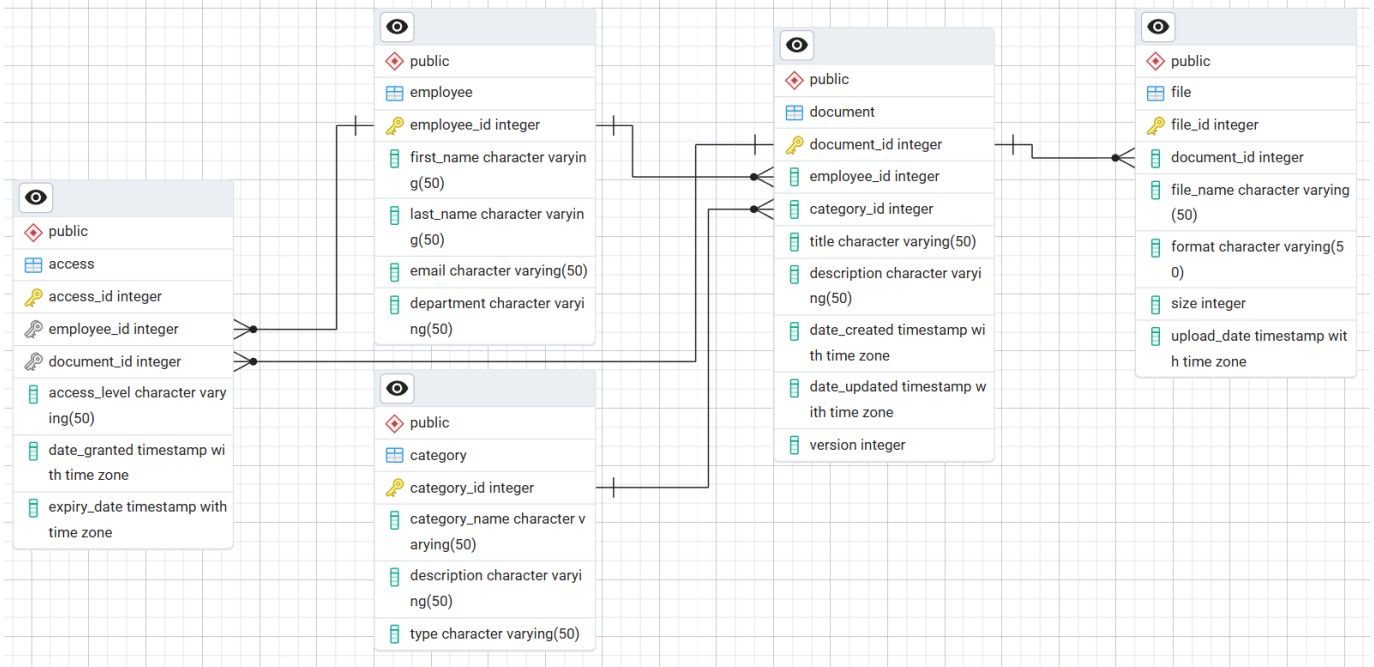

Рис. 3 - Схема бази даних у pgAdmin 4

Середовище та компоненти розробки

Для розробки використовувалась мова програмування C++, середовище розробки Visual Studio 2022. Стороння бібліотека для доступу до PostgreSQL – SOCI. Зі стандартних бібліотек C++ використовувалась iostream для керування базовими операціями введення та виведення. Також застосовувалась ctime для роботи з датою і часом, перетворення рядків та вимірювання часу виконання запитів, і iomanip для форматування консольного виводу.

Шаблон проектування

MVC (Модель-подання-контролер) – це шаблон проектування, який розділяє логіку програми на три основні, взаємопов'язані компоненти:

1. Model (Модель) – це клас, що відображає логіку роботи з даними, обробляє всі операції з даними, такі як додавання, оновлення, вилучення.
2. View (Представлення) – це клас, через який користувач взаємодіє з програмою. У даному випадку, консольний інтерфейс, який відображає дані для користувача та зчитує їх з екрану.
3. Controller (Контролер) – це клас, який відповідає за зв’язок між користувачем і системою. Він приймає введені користувачем дані та обробляє їх. В залежності від результатів, викликає відповідні дії з Model або View.

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

Структура програми та її опис

З файлу main.cpp відбувається ініціалізація програми. Тут створюється об'єкт View (для виведення повідомлень про запуск/помилки) та об'єкт Controller, якому передається рядок підключення до бази даних. Після цього викликається головний цикл програми app.main_run(), передаючи управління контролеру.

У файлах Model.h та Model.cpp описаний клас Model. Він відповідає за управління підключенням до бази даних PostgreSQL (використовуючи бібліотеку SOCI) та виконанням усіх низькорівневих запитів до неї. Це включає додавання, оновлення, видалення, вибірку, генерацію та пошук даних у таблицях.

У файлах Controller.h та Controller.cpp реалізовано клас Controller, який є "мозком" програми. Він містить екземпляри Model та View. Контролер обробляє запити користувача (отримані через View), викликає відповідні методи Model для маніпуляцій з даними та наказує View відобразити оновлені меню або результати.

У файлах View.h та View.cpp описаний клас View. Цей компонент відповідає виключно за взаємодію з користувачем через консоль: відображення меню, виведення повідомлень, отримання вводу від користувача та форматування результатів.

Отже, структура програми повністю відповідає патерну MVC, де Model керує даними, View — представленням, а Controller — логікою взаємодії між ними.

Подання стуруктури програми зображено на рис.4.

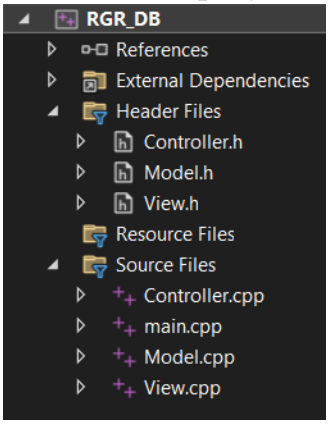

Рис. 4 – Структура програми

Структура меню програми

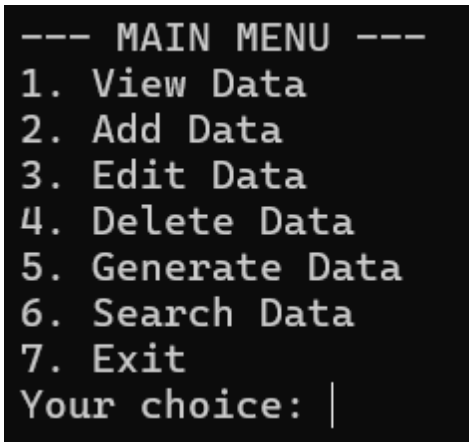

Рис. 5 – Головне меню програми

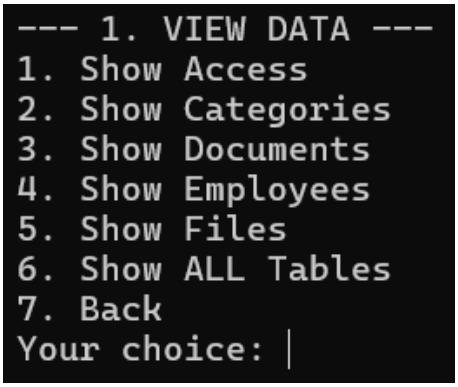

Рис. 6 – Підменю "View Data" (Перегляд даних)

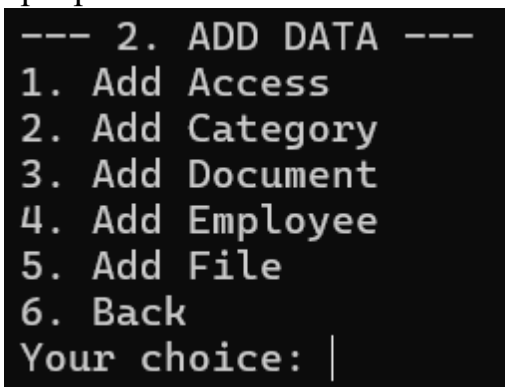

Рис. 7 – Підменю "Add Data" (Додавання даних)

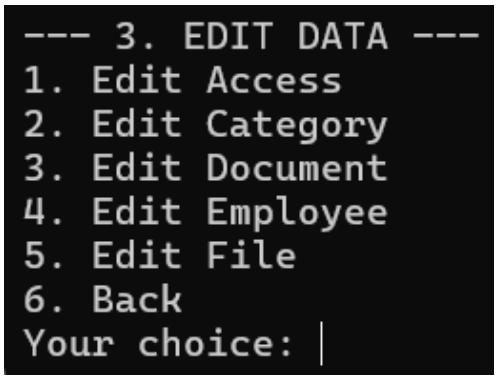

Рис. 8 – Підменю "Edit Data" (Редагування даних)

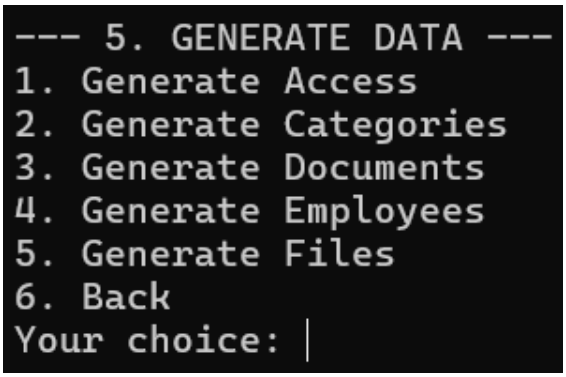

Рис. 9 – Підменю "Generate Data" (Генерація даних)

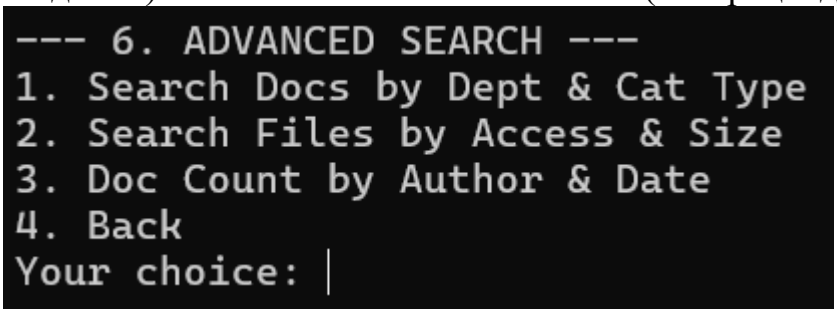

Рис. 10 – Підменю "Search Data" (Пошук даних)

Опис функціональності пунктів меню користувача

Пункт 1 «View Data»:

Відповідає за виклик підменю, яке дозволяє виводити дані з окремих таблиць (Access, Category, Employee і т.д.) або з усіх таблиць одразу для перегляду.

Пункт 2 «Add Data»:

Відповідає за виклик підменю для додавання нових записів у відповідні таблиці бази даних. Перед додаванням виконується перевірка наявності батьківських записів (якщо додаються дані у дочірню таблицю).

Пункт 3 «Edit Data»:

Відповідає за виклик підменю для оновлення (зміни) існуючих даних у таблицях за їх унікальним ідентифікатором (ID).

Пункт 4 «Delete Data»:

Відповідає за видалення даних з вказаної таблиці за унікальним ідентифікатором. Цей пункт перехоплює помилки та не дозволяє видалити запис, якщо на нього посилаються дочірні таблиці.

Пункт 5 «Generate Data»:

Відповідає за виклик підменю для пакетної генерації псевдовипадкових даних. Генерація відбувається безпосередньо в PostgreSQL за допомогою SQL-запитів, а користувач вказує бажану кількість записів.

Пункт 6 «Search Data»:

Відповідає за виклик підменю для виконання складних пошукових SQL-запитів, які об'єднують декілька таблиць, фільтрують та групують дані. Користувач вводить параметри пошуку (шаблони LIKE, діапазони дат/чисел), а програма виводить результат та час виконання запиту.

Пункт 7 «Exit»:

Відповідає за коректне завершення роботи програми та вихід з головного циклу.

Аналіз реалізації та логіки програмних модулів

Нижче проводиться короткий аналіз коду кожного модуля:

Модуль main.cpp

Ініціалізує ключові компоненти (Controller та View) і запускає головний цикл програми. Його найважливіша функція — обробка критичних помилок запуску (наприклад, збій підключення до БД).

```
#include "Controller.h"
#include "View.h"

int main() {
```

```
    View mainView;
    try {
        Controller    app("dbname=postgres    user=postgres    password=1234567890
host=localhost port=5432");
        app.main_run();
    }
    catch (const std::exception& e) {
        std::cerr << "Application failed to start or encountered a critical error:
" << e.what() << std::endl;
        mainView.showMessage("Exiting due to critical error...");
        return 1;
    }
    catch (...) {
        std::cerr << "Unknown critical error occurred." << std::endl;
        mainView.showMessage("Exiting due to unknown critical error...");
        return 2;
    }

    mainView.showMessage("Exiting...");
    return 0;
}
```

Модуль View

View.h

Декларує інтерфейс класу View. Тут оголошено enum class Status, який використовується для передачі стандартизованих кодів стану від Controller до View. Також тут оголошено всі публічні методи, які Controller може викликати для взаємодії з користувачем (показати меню, отримати ввід, вивести помилку).

```
#pragma once
#include <iostream>
using std::string;
// Перелік кодів стану
enum class Status {
    OK,
    INPUT_SUCCESS,
    UPDATE_SUCCESS,
    DELETE_FAILED_FK, // Помилка видалення через зовнішній ключ
    ADD_FAILED_FK,    // Помилка додавання через зовнішній ключ
    //... (інші подібні)
};
class View {
public:
    // Методи виводу
    void showMessage(const string& message);
    void displayStatus(Status s, const string& detail = "");
    void displayError(const string& error_message);
    // Методи вводу
    string getInput(const string& prompt);
    int getIntInput(const string& prompt);
    // Відображення меню
    void displayMainMenu();
    void displayShowMenu();
    //... (інші меню)
};
```

View.cpp

Містить реалізацію логіки методів, оголошених у View.h.

Метод getIntInput: Реалізує зчитування цілого числа. Цикл while(true) гарантує, що функція не завершиться, доки не буде введено коректне число.

std::cin.fail() виявляє помилкове введення (наприклад, текст), std::cin.clear() скидає прапорець помилки, а std::cin.ignore() очищує буфер вводу.

```
int View::getIntInput(const string& prompt) {
    int value;
    while (true) {
        std::cout << prompt;
        std::cin >> value;
        if (std::cin.fail()) { // Якщо ввід не є числом
            std::cin.clear();// Скидаємо прапорець помилки
            std::cin.ignore(1024, '\n'); // Очищуємо буфер вводу
            showMessage("ERROR: Please enter a valid integer.");
        }
        else {
            std::cin.ignore(1024, '\n'); // Очищуємо буфер від '\n'
            return value;
        }
    }
}
```

Метод displayStatus: Реалізує логіку перетворення enum Status на зрозуміле користувачу повідомлення за допомогою оператора switch.

```
void View::displayStatus(Status s, const string& detail) {
    switch (s) {
    case Status::OK:                        break;
    case Status::INPUT_SUCCESS:              showMessage("Data added successfully.");
break;
    //...
    default:                                      showMessage("An unspecified status
occurred."); break;
    }
}
```

Модуль Controller

Controller.h

Декларує інтерфейс класу Controller. Включає Model.h та View.h, так як містить їх екземпляри (model та view) як приватні члени. Тут оголошено публічний метод main_run() та набір приватних методів-обробників (handle...), які реалізують логіку для кожного пункту меню.

```
#pragma once
#include "Model.h"
#include "View.h"

class Controller {
public:
    Controller(const char* db_connection_string);
    void main_run();

private:
    Model model; // Екземпляр Model
    View view;   // Екземпляр View

    // Методи-обробники
    void handleShowData();
    void handleAddData();
    void handleUpdateData();
    void handleDeleteData();
    void handleGenerateData();
    void handleSearchData();

    // Спеціалізовані обробники
    void handleAddEmployee();
```

```
    void handleSearch1();
    // ... (інші обробники)
};
```

Controller.cpp

Містить реалізацію логіки методів, оголошених у Controller.h. Ініціалізує приватні члени model та view. Model ініціалізується до view.

```
Controller::Controller(const char* db_connection_string)
    :model(db_connection_string), view() // Ініціалізація Моделі та View
{}
```

Метод main_run: Містить while(true) головний цикл програми. try...catch блок всередині циклу перехоплює *некритичні* помилки, дозволяючи програмі повідомити користувача (view.displayError) і продовжити роботу.

```
void Controller::main_run() {
    bool running = true;
    while (running) {
        try {
            int choice = getMainMenuChoice(); // Отримання вибору від View
            switch (choice) {
            case 1: handleShowData(); break;
            case 2: handleAddData(); break;
            //...
            case 7: running = false; break;
            default: view.displayStatus(Status::INVALID_CHOICE);
            }
        }
        catch (const std::exception& e) {
            // Перехоплення помилок з Model
            view.displayError(e.what()); // Показ помилки і продовження циклу
        }
    }
}
```

Метод-обробники (на прикладі handleAddEmployee): Controller отримує дані від View і передає їх у Model.

```
void Controller::handleAddEmployee() {
    // Отримання даних від View
    std::string first = view.getInput("Enter first name: ");
    std::string last = view.getInput("Enter last name: ");
    std::string email = view.getInput("Enter email: ");
    std::string dep = view.getInput("Enter department: ");

    // Передача даних в Model для обробки
    model.addEmployee(first, last, email, dep);
}
```

Модуль Model

Model.h

Декларує інтерфейс класу Model. Включає бібліотеку soci/soci.h для роботи з БД. Оголошені всі C++ структури (Employee, Category тощо), що відображають таблиці БД. Клас Model містить приватний soci::session sql (об'єкт з'єднання) та публічні методи для всіх операцій з даними.

```
#pragma once
#include <soci/soci.h>
#include <ctime>

using std::string;

// Структури, що відображають таблиці БД
struct Employee { int id; string first, last, email, dept; };
```

```
struct Category { int id; string name, desc, type; };
struct Document { int id, emp_id, cat_id; string title, desc; std::tm created,
updated; int ver; };
//... (інші структури)

class Model {
private:
    soci::session sql; // Приватний об'єкт сесії SOCI
    long long getLastId(const string& table, const string& id_col);
    void checkDbConnection();
    std::tm stringToTm(const string& date_str);

public:
    Model(const string& conn_string);
    ~Model();

    // Методи виведення таблиці
    void printAllEmployees();
    //...

    // Методи додавання
    void addEmployee(const string& first, const string& last, const string& email,
const string& dep);
    //...

    // Метод оновлення
    void updateEmployee(int id, const string& first, const string& last, const
string& email, const string& dep);
    //...

    // Методи видалення
    void deleteData(const string& table_name, const string& id_col, int id);
    //...

    // Методи генерації
    void generateEmployees(int count);
    //...

    // Методи пошуку
    double searchDocsByDeptAndCat(const string& department, const string&
category_type);
    //...
};
```

Model.cpp

Містить реалізацію логіки методів, оголошених у Model.h

Метод-додавання (на прикладі addDocument): Перш ніж виконати INSERT, вона робить два SELECT запити, щоб переконатися, що employee_id та category_id існують. Якщо ні, вона кидає помилку std::runtime_error з чітким описом, запобігаючи помилці бази даних.

```
void Model::addDocument(int emp_id, int cat_id, const string& title, const string&
desc, int version) {
    checkDbConnection();
    //...
    int exists = 0;
    // Перевірка існування Employee ID
    sql << "SELECT 1 FROM public.employee WHERE employee_id = :id",
soci::use(emp_id), soci::into(exists);
    if (!exists)
        //помилка, якщо ID не знайдено
        throw std::runtime_error("ERROR: Employee ID=" + std::to_string(emp_id) + "
not found.");
```

```
    exists = 0;
    // Перевірка існування Category ID
    sql << "SELECT 1 FROM public.category WHERE category_id = :id",
soci::use(cat_id), soci::into(exists);
    if (!exists)
        throw std::runtime_error("ERROR: Category ID=" + std::to_string(cat_id) + "
not found.");

    // Якщо обидва FK валідні, виконуємо INSERT
    sql << "INSERT INTO public.document ...";
}
```

Метод-видалення (deleteData): Одразу намагається видалити запис у try блоці. Якщо база даних повертає помилку, catch блок аналізує текст помилки і виводить у зрозумілий текст користувачу.

```
void Model::deleteData(const string& table_name, const string& id_col, int id) {
    checkDbConnection();
    try {
        // Спроба видалити
        sql << "DELETE FROM public.\"" + table_name + "\" WHERE \"" + id_col + "\"
= :id", soci::use(id);
    }
    catch (const std::exception& e) { // Помилка якщо не вийшло видалити
        string msg = e.what();
        // Аналіз помилки
        if (msg.find("violates foreign key constraint") != string::npos) {
            throw std::runtime_error("ERROR: Cannot delete record ID=" + ... + "
it's referenced by other tables.");
        }
        else { throw std::runtime_error("ERROR during deletion: " + msg); }
    }
}
```

Метод-пошук (searchDocsByDeptAndCat): Використання JOIN для об'єднання даних з кількох таблиць (employee, document, category) в один запит; А також вимірює продуктивність. Вони фіксують час до запиту (clock_t start) і після (clock_t end), а потім повертають різницю в мілісекундах, щоб показати, як швидко виконався пошук.

```
double Model::searchDocsByDeptAndCat(const string& department, const string&
category_type) {
    checkDbConnection();
    //...
    clock_t start = clock(); // Час початку
    soci::rowset<soci::row> rs = (sql.prepare <<
        R"( SELECT e.first_name, e.last_name, d.title, c.category_name FROM
public.employee e
            JOIN public.document d ON e.employee_id = d.employee_id
            JOIN public.category c ON d.category_id = c.category_id
            WHERE e.department LIKE :dept AND c.type LIKE :cat_type )",
        soci::use(dept_pattern, "dept"), soci::use(cat_pattern, "cat_type"));
    //... (вивід результатів)
    clock_t end = clock(); //Час завершення

    // Розрахунок та повернення часу в мілісекундах
    double ms_double = (double)(end - start) * 1000.0 / CLOCKS_PER_SEC;
    return ms_double;
}
```

Тестування

Для перевірки коректності роботи програми було проведено тестування, що охоплюють основний функціонал: перегляд, додавання, видалення, генерацію та пошук даних.

Перевірка виведення всіх існуючих таблиць. Було обрано в меню пункт "1. View Data" -> "6. Show ALL Tables" для перегляду вмісту всіх таблиць. Результат зображений на рисунку нижче:

```
--- 1. ACCESS ---
ID     | EmpID    | DocID    | Access Level    | Granted      | Expires
-------------------------------------------------------------------------------
301    | 1        | 101      | editing         | 2023-10-01   | 2033-10-01
302    | 2        | 101      | reading         | 2023-10-20   | 2024-10-20

--- 2. CATEGORIES ---
ID     | Name                          | Description                                 | Type
-----------------------------------------------------------------------------------------------
10     | Technical specifications      | Descriptions of products and solutions   | Internal
11     | User guides                   | Instructions and guides for end users    | Public

--- 3. DOCUMENTS ---
-------------------------------------------------------------------------------
ID:   101       | Title: API specification for the authorization module | V: 2
  Desc: Description of API methods and endpoints v2.1
  AuthorID: 1  | CatID: 10
  Created: 2023-10-01   | Updated: 2023-10-15
-------------------------------------------------------------------------------
ID:   102       | Title: 2FA setup instructions | V: 1
  Desc: Guide to increasing account security
  AuthorID: 2  | CatID: 11
  Created: 2023-11-05   | Updated: 2023-11-05

--- 4. EMPLOYEES ---
ID     | Name                          | Email                              | Department
-----------------------------------------------------------------------------------------------------------
1      | Ivan Petrenko                 | ivan.petrenko@example.com          | Development Department
2      | Elena Kovalenko               | olena.kovalenko@example.com        | Analytics Department

--- 5. FILES ---
ID     | Name                             | Format     | Size(KB)   | DocID    | Uploaded
-------------------------------------------------------------------------------------------
201    | auth_api_v2.1                    | PDF        | 1024       | 101      | 2023-10-01
202    | 2fa_guide                        | DOCX       | 512        | 102      | 2023-11-05
```

Рис. 11. Результат виведення всіх таблиць бази даних.

Перевірка додавання нового запису (на прикладі Employee): Було обрано пункт "2. Add Data" -> "4. Add Employee" та введено коректні дані. Результати зображено на рисунках нижче:

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 2

--- 2. ADD DATA ---
1. Add Access
2. Add Category
3. Add Document
4. Add Employee
5. Add File
6. Back
Your choice: 4
Enter first name: Andri
Enter last name: Shevchenko
Enter email: andri.sh@gmail.com
Enter department: HR Department
Data added successfully.
```

Рис. 12. Повідомлення про успішне додавання нового запису Employee.

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 1

--- 1. VIEW DATA ---
1. Show Access
2. Show Categories
3. Show Documents
4. Show Employees
5. Show Files
6. Show ALL Tables
7. Back
Your choice: 4

--- 4. EMPLOYEES ---
ID     | Name                          | Email                          | Department
-------------------------------------------------------------------------------------------------------
1      | Ivan Petrenko                 | ivan.petrenko@example.com      | Development Department
2      | Elena Kovalenko               | olena.kovalenko@example.com    | Analytics Department
3      | Andri Shevchenko              | andri.sh@gmail.com             | HR Department
```

Рис. 13. Перегляд таблиці Employees в програмі

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (50)	department character varying (50)
1	1	Ivan	Petrenko	ivan.petrenko@example.com	Development Department
2	2	Elena	Kovalenko	olena.kovalenko@example.com	Analytics Department
3	3	Andri	Shevchenko	andri.sh@gmail.com	HR Department

Рис. 14. Перегляд таблиці Employees у середовищі pgAdmin

Перевірка обробки помилки при додаванні (Порушення Foreign Key): Була обрано "2. Add Data" -> "3. Add Document". Для тестування обробки помилки, було введено ID автора (employee_id), якого гарантовано не існує в батьківській таблиці Employee. Було обрано ID 9999. Результат зображений на рисунку нижче:

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 2

--- 2. ADD DATA ---
1. Add Access
2. Add Category
3. Add Document
4. Add Employee
5. Add File
6. Back
Your choice: 3
Enter Author's Employee ID: 9999
Enter Category ID: 10
Enter title: Test Document
Enter description: Teste with non-existent author
Enter version: 1

!!! ERROR !!!
ERROR: Employee ID=9999 not found.
```

Рис. 15. Результат помилки при додаванні: "ERROR: Employee ID=9999 not found."

Перевірка обробки помилки при видаленні (порушення Foreign Key): Була здійснена спроба видалити батьківський запис, який мав пов'язані дочірні записи. Для цього було обрано пункт "4. Delete Data" та введено дані для видалення Employee з ID 1 ("Ivan Petrenko"). Результат зображений на рисунку нижче:

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 4

--- 4. DELETE DATA ---
Enter table name (e.g., employee): employee
Enter ID column name (e.g., employee_id): employee_id
Enter ID to delete: 1

!!! ERROR !!!
ERROR during deletion: Cannot execute query. Fatal error.
```

Рис. 16. Результат помилки при видаленні

Перевірка генерації 100 000 записів для таблиці Employee: Було обрано пункт "5. Generate Data" -> "4. Generate Employees" та введено "100000". Результати зображено на рисункках нижче:

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 5

--- 5. GENERATE DATA ---
1. Generate Access
2. Generate Categories
3. Generate Documents
4. Generate Employees
5. Generate Files
6. Back
Your choice: 4
Number to generate: 100000
100000 records generated successfully.
```

Рис. 17. Повідомлення про успішну генерацію 100 000 записів.

Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page No: 1

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (50)	department character varying (50)
982	982	FirstName_586	LastName_829	user982@example.com	Analytics
983	983	FirstName_180	LastName_932	user983@example.com	HR
984	984	FirstName_71	LastName_276	user984@example.com	Development
985	985	FirstName_560	LastName_928	user985@example.com	Analytics
986	986	FirstName_655	LastName_829	user986@example.com	Analytics
987	987	FirstName_208	LastName_103	user987@example.com	Analytics
988	988	FirstName_851	LastName_63	user988@example.com	Analytics
989	989	FirstName_769	LastName_247	user989@example.com	Development
990	990	FirstName_732	LastName_691	user990@example.com	Analytics
991	991	FirstName_892	LastName_740	user991@example.com	HR
992	992	FirstName_820	LastName_904	user992@example.com	Support
993	993	FirstName_607	LastName_358	user993@example.com	HR
994	994	FirstName_243	LastName_630	user994@example.com	HR
995	995	FirstName_892	LastName_936	user995@example.com	Support
996	996	FirstName_667	LastName_678	user996@example.com	Support
997	997	FirstName_201	LastName_156	user997@example.com	Development
998	998	FirstName_951	LastName_549	user998@example.com	HR
999	999	FirstName_678	LastName_478	user999@example.com	HR
1000	1000	FirstName_397	LastName_804	user1000@example.com	Support

Рис. 18. Перегляд таблиці Employees у середовищі pgAdmin

Перевірка пошуку №1 (Документи за департаментом та типом): Було обрано пункт в меню "6. Search Data" -> "1. Search Docs by Dept & Cat Type" та введено параметри пошуку. Результат зображений на рисунку нижче:

```
--- MAIN MENU ---
1. View Data
2. Add Data
3. Edit Data
4. Delete Data
5. Generate Data
6. Search Data
7. Exit
Your choice: 6

--- 6. ADVANCED SEARCH ---
1. Search Docs by Dept & Cat Type
2. Search Files by Access & Size
3. Doc Count by Author & Date
4. Back
Your choice: 1
Enter department (like): Development
Enter category type (like): Internal

--- Search Results (Docs by Dept & Category Type) ---
Author                    | Document Title                                    | Category
------------------------------------------------------------------------------------------------------------------------
Ivan Petrenko             | API specification for the authorization module | Technical specifications

[i] Query execution time: 7 ms
```

Рис. 19. Результати пошуку №1 та виведений час виконання запиту

Перевірка пошуку №2 (Файли за розміром та доступом): Було обрано пункт в меню"6. Search Data" -> "2. Search Files by Access & Size" та введено діапазон розміру і рівень доступу. Результат зображений на рисунку нижче:

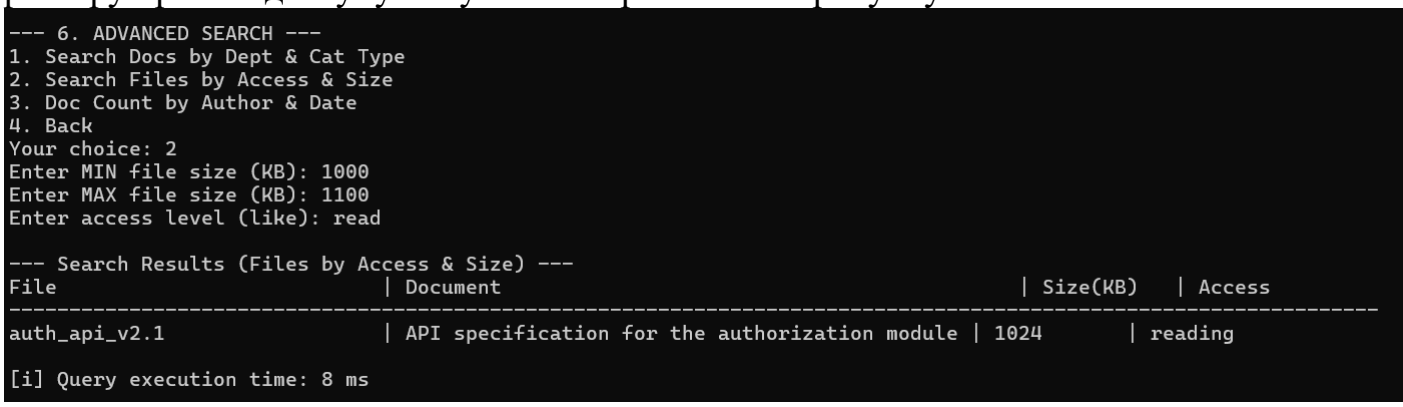

Рис. 20. Результати пошуку №2 та виведений час виконання запиту

Перевірка пошуку №3 (Кількість документів за автором та датою): Було обрано пункт "6. Search Data" -> "3. Doc Count by Author & Date" та введено діапазон дат. Результат зображений на рисунку нижче:

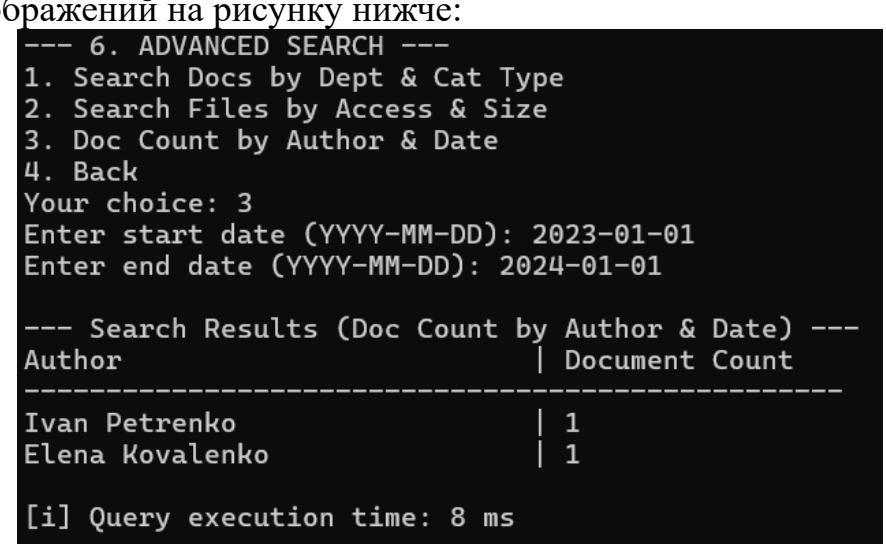

Рис. 21. Результати пошуку №3 та виведений час виконання запиту

Повний код програми

main.cpp:

```
#include "Controller.h"
#include "View.h"

int main() {
    View mainView;
    try {
        Controller app("dbname=postgres user=postgres password=1234567890
host=localhost port=5432");
        app.main_run();
    }
    catch (const std::exception& e) {
        std::cerr << "Application failed to start or encountered a critical error:
" << e.what() << std::endl;
        mainView.showMessage("Exiting due to critical error...");
        return 1;
    }
```

```
    catch (...) {
        std::cerr << "Unknown critical error occurred." << std::endl;
        mainView.showMessage("Exiting due to unknown critical error...");
        return 2;
    }

    mainView.showMessage("Exiting...");
    return 0;
}
```

View.h:

```
#pragma once
#include <iostream>

using std::string;

enum class Status {
    OK,
    INPUT_SUCCESS,
    UPDATE_SUCCESS,
    DELETE_SUCCESS,
    GENERATE_SUCCESS,
    INVALID_CHOICE,
    INVALID_INPUT_GENERAL,
    INVALID_INPUT_NUMBER,
    INVALID_INPUT_DATE,
    DELETE_FAILED_FK,
    DELETE_FAILED_GENERAL,
    ADD_FAILED_FK,
    ADD_FAILED_GENERAL,
    UPDATE_FAILED_GENERAL,
    GENERATE_FAILED_DEPENDENCY,
    DB_CONNECTION_ERROR,
    DB_QUERY_ERROR,
    UNKNOWN_ERROR
};

class View {
public:
    void showMessage(const string& message);
    string getInput(const string& prompt);
    int getIntInput(const string& prompt);

    void displayMainMenu();
    void displayShowMenu();
    void displayAddMenu();
    void displayUpdateMenu();
    void displayGenerateMenu();
    void displaySearchMenu();
    void displaySectionTitle(const string& title);

    void displayStatus(Status s, const string& detail = "");
    void showExecutionTime(double time_ms);
    void displayError(const string& error_message);
};
```

View.cpp:

```
#include "View.h"
#include "Model.h"
#include <iostream>

void View::showMessage(const string& message) {
```

```
    std::cout << message << std::endl;
}

string View::getInput(const string& prompt) {
    string input;
    std::cout << prompt;
    std::getline(std::cin >> std::ws, input);
    return input;
}

int View::getIntInput(const string& prompt) {
    int value;
    while (true) {
        std::cout << prompt;
        std::cin >> value;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(1024, '\n');
            showMessage("ERROR: Please enter a valid integer.");
        }
        else {
            std::cin.ignore(1024, '\n');
            return value;
        }
    }
}

void View::displayMainMenu() {
    showMessage("\n--- MAIN MENU ---");
    showMessage("1. View Data");
    showMessage("2. Add Data");
    showMessage("3. Edit Data");
    showMessage("4. Delete Data");
    showMessage("5. Generate Data");
    showMessage("6. Search Data");
    showMessage("7. Exit");
}

void View::displayShowMenu() {
    showMessage("\n--- 1. VIEW DATA ---");
    showMessage("1. Show Access");
    showMessage("2. Show Categories");
    showMessage("3. Show Documents");
    showMessage("4. Show Employees");
    showMessage("5. Show Files");
    showMessage("6. Show ALL Tables");
    showMessage("7. Back");
}

void View::displayAddMenu() {
    showMessage("\n--- 2. ADD DATA ---");
    showMessage("1. Add Access");
    showMessage("2. Add Category");
    showMessage("3. Add Document");
    showMessage("4. Add Employee");
    showMessage("5. Add File");
    showMessage("6. Back");
}

void View::displayUpdateMenu() {
    showMessage("\n--- 3. EDIT DATA ---");
    showMessage("1. Edit Access");
    showMessage("2. Edit Category");
```

```
    showMessage("3. Edit Document");
    showMessage("4. Edit Employee");
    showMessage("5. Edit File");
    showMessage("6. Back");
}

void View::displayGenerateMenu() {
    showMessage("\n--- 5. GENERATE DATA ---");
    showMessage("1. Generate Access");
    showMessage("2. Generate Categories");
    showMessage("3. Generate Documents");
    showMessage("4. Generate Employees");
    showMessage("5. Generate Files");
    showMessage("6. Back");
}
void View::displaySearchMenu() {
    showMessage("\n--- 6. ADVANCED SEARCH ---");
    showMessage("1. Search Docs by Dept & Cat Type");
    showMessage("2. Search Files by Access & Size");
    showMessage("3. Doc Count by Author & Date");
    showMessage("4. Back");
}
void View::displaySectionTitle(const string& title) {
    showMessage("\n--- " + title + " ---");
}


void View::displayStatus(Status s, const string& detail) {
    switch (s) {
    case Status::OK:                        break;
    case Status::INPUT_SUCCESS:             showMessage("Data added successfully.");
break;
    case Status::UPDATE_SUCCESS:        showMessage("Record updated successfully.");
break;
    case Status::DELETE_SUCCESS:        showMessage("Record deleted successfully.");
break;
    case Status::GENERATE_SUCCESS:          showMessage(detail + " records generated
successfully."); break;
    case Status::INVALID_CHOICE:            showMessage("Invalid choice. Please try
again."); break;
    case Status::INVALID_INPUT_GENERAL:   showMessage("Invalid input: " + detail);
break;
    case Status::INVALID_INPUT_NUMBER:     showMessage("Invalid number entered.");
break;
    case Status::INVALID_INPUT_DATE:     showMessage("Invalid date format. Use YYYY-
MM-DD."); break;
    case Status::DELETE_FAILED_FK:         showMessage("ERROR: Cannot delete record,
it is referenced by other tables."); break;
    case Status::DELETE_FAILED_GENERAL:   showMessage("ERROR: Deletion failed. " +
detail); break;
    case Status::ADD_FAILED_FK:             showMessage("ERROR: Cannot add record.
Referenced ID not found. " + detail); break;
    case Status::ADD_FAILED_GENERAL:      showMessage("ERROR: Failed to add record.
" + detail); break;
    case Status::UPDATE_FAILED_GENERAL:    showMessage("ERROR: Failed to update
record. " + detail); break;
    case Status::GENERATE_FAILED_DEPENDENCY:  showMessage("ERROR: Cannot generate
data. Generate prerequisite data first (e.g., Employees, Categories)."); break;
    case Status::DB_CONNECTION_ERROR:       showMessage("CRITICAL ERROR: Database
connection failed. " + detail); break;
    case Status::DB_QUERY_ERROR:          showMessage("DATABASE ERROR: " + detail);
break;
```

```
    case Status::UNKNOWN_ERROR:             showMessage("An unknown critical error
occurred."); break;
    default:                                    showMessage("An unspecified status
occurred."); break;
    }
}

void View::showExecutionTime(double time_ms) {
    std::cout << "\n[i] Query execution time: " << time_ms << " ms" << std::endl;
}

void View::displayError(const string& error_message) {
    showMessage("\n!!! ERROR !!!");
    showMessage(error_message);
}
```

Controller.h:

```
#pragma once
#include "Model.h"
#include "View.h"

class Controller {
public:
    Controller(const char* db_connection_string);
    void main_run();

private:
    Model model;
    View view;

    int getMainMenuChoice();
    int getShowMenuChoice();
    int getAddMenuChoice();
    int getUpdateMenuChoice();
    int getGenerateMenuChoice();
    int getSearchMenuChoice();

    void handleShowData();
    void handleAddData();
    void handleUpdateData();
    void handleDeleteData();
    void handleGenerateData();
    void handleSearchData();

    void handleAddEmployee();
    void handleAddCategory();
    void handleAddDocument();
    void handleAddFile();
    void handleAddAccess();

    void handleUpdateEmployee();
    void handleUpdateCategory();
    void handleUpdateDocument();
    void handleUpdateFile();
    void handleUpdateAccess();


    void handleGenerateEmployees();
    void handleGenerateCategories();
    void handleGenerateDocuments();
    void handleGenerateFiles();
    void handleGenerateAccess();
```

```
    void handleSearch1();
    void handleSearch2();
    void handleSearch3();

    int getGenerationCount();
};
```

Controller.cpp:

```
#include "Controller.h"
#include <iostream>

Controller::Controller(const char* db_connection_string)
    : model(db_connection_string), view()
{
}

void Controller::main_run() {
    bool running = true;
    while (running) {
        try {
            int choice = getMainMenuChoice();
            switch (choice) {
            case 1: handleShowData(); break;
            case 2: handleAddData(); break;
            case 3: handleUpdateData(); break;
            case 4: handleDeleteData(); break;
            case 5: handleGenerateData(); break;
            case 6: handleSearchData(); break;
            case 7: running = false; break;
            default: view.displayStatus(Status::INVALID_CHOICE);
            }
        }
        catch (const std::exception& e) {
            view.displayError(e.what());
        }
        catch (...) {
            view.displayStatus(Status::UNKNOWN_ERROR);
            running = false;
        }
    }
}

int     Controller::getMainMenuChoice()     {     view.displayMainMenu();     return
view.getIntInput("Your choice: "); }
int     Controller::getShowMenuChoice()     {     view.displayShowMenu();     return
view.getIntInput("Your choice: "); }
int     Controller::getAddMenuChoice()      {     view.displayAddMenu();      return
view.getIntInput("Your choice: "); }
int    Controller::getUpdateMenuChoice()    {    view.displayUpdateMenu();    return
view.getIntInput("Your choice: "); }
int   Controller::getGenerateMenuChoice()   {   view.displayGenerateMenu();   return
view.getIntInput("Your choice: "); }
int    Controller::getSearchMenuChoice()    {    view.displaySearchMenu();    return
view.getIntInput("Your choice: "); }

void Controller::handleShowData() {
    bool running = true;
    while (running) {
        int choice = getShowMenuChoice();
        switch (choice) {
        case 1:
```

```
            view.displaySectionTitle("1. ACCESS");
            model.printAllAccess();
            break;
        case 2:
            view.displaySectionTitle("2. CATEGORIES");
            model.printAllCategories();
            break;
        case 3:
            view.displaySectionTitle("3. DOCUMENTS");
            model.printAllDocuments();
            break;
        case 4:
            view.displaySectionTitle("4. EMPLOYEES");
            model.printAllEmployees();
            break;
        case 5:
            view.displaySectionTitle("5. FILES");
            model.printAllFiles();
            break;
        case 6:
            view.displaySectionTitle("1. ACCESS");
            model.printAllAccess();
            view.displaySectionTitle("2. CATEGORIES");
            model.printAllCategories();
            view.displaySectionTitle("3. DOCUMENTS");
            model.printAllDocuments();
            view.displaySectionTitle("4. EMPLOYEES");
            model.printAllEmployees();
            view.displaySectionTitle("5. FILES");
            model.printAllFiles();
            break;
        case 7:
            running = false;
            break;
        default:
            view.displayStatus(Status::INVALID_CHOICE);
        }
    }
}

void Controller::handleAddData() {
    bool running = true;
    while (running) {
        int choice = getAddMenuChoice();
        bool added = false;
        switch (choice) {
        case 1: handleAddAccess(); added = true; break;
        case 2: handleAddCategory(); added = true; break;
        case 3: handleAddDocument(); added = true; break;
        case 4: handleAddEmployee(); added = true; break;
        case 5: handleAddFile(); added = true; break;
        case 6: running = false; break;
        default: view.displayStatus(Status::INVALID_CHOICE);
        }
        if (added && running) {
            view.displayStatus(Status::INPUT_SUCCESS);
        }
    }
}

void Controller::handleUpdateData() {
    bool running = true;
    while (running) {
```

```
        int choice = getUpdateMenuChoice();
        bool updated = false;
        switch (choice) {
        case 1: handleUpdateAccess(); updated = true; break;
        case 2: handleUpdateCategory(); updated = true; break;
        case 3: handleUpdateDocument(); updated = true; break;
        case 4: handleUpdateEmployee(); updated = true; break;
        case 5: handleUpdateFile(); updated = true; break;
        case 6: running = false; break;
        default: view.displayStatus(Status::INVALID_CHOICE);
        }
        if (updated && running) {
            view.displayStatus(Status::UPDATE_SUCCESS);
        }
    }
}

void Controller::handleDeleteData() {
    view.displaySectionTitle("4. DELETE DATA");
    std::string table = view.getInput("Enter table name (e.g., employee): ");
    std::string db_table = (table == "category") ? "category" : table;
    std::string id_col = view.getInput("Enter ID column name (e.g., employee_id):
");
    int id = view.getIntInput("Enter ID to delete: ");
    model.deleteData(db_table, id_col, id);
    view.displayStatus(Status::DELETE_SUCCESS);
}

int Controller::getGenerationCount() {
    int n = view.getIntInput("Number to generate: ");
    if (n <= 0) {
        view.displayStatus(Status::INVALID_INPUT_GENERAL, "Please enter a positive
number.");
        return 0;
    }
    return n;
}

void Controller::handleGenerateData() {
    bool running = true;
    while (running) {
        int choice = getGenerateMenuChoice();
        switch (choice) {
        case 1: handleGenerateAccess(); break;
        case 2: handleGenerateCategories(); break;
        case 3: handleGenerateDocuments(); break;
        case 4: handleGenerateEmployees(); break;
        case 5: handleGenerateFiles(); break;
        case 6: running = false; break;
        default: view.displayStatus(Status::INVALID_CHOICE);
        }
    }
}

void Controller::handleSearchData() {
    bool running = true;
    while (running) {
        int choice = getSearchMenuChoice();
        switch (choice) {
        case 1: handleSearch1(); break;
        case 2: handleSearch2(); break;
        case 3: handleSearch3(); break;
        case 4: running = false; break;
```

```
        default: view.displayStatus(Status::INVALID_CHOICE);
        }
    }
}

void Controller::handleAddEmployee() {
    std::string first = view.getInput("Enter first name: ");
    std::string last = view.getInput("Enter last name: ");
    std::string email = view.getInput("Enter email: ");
    std::string dep = view.getInput("Enter department: ");
    model.addEmployee(first, last, email, dep);
}
void Controller::handleAddCategory() {
    std::string name = view.getInput("Enter category name: ");
    std::string desc = view.getInput("Enter description: ");
    std::string type = view.getInput("Enter type: ");
    model.addCategory(name, desc, type);
}
void Controller::handleAddDocument() {
    int emp_id = view.getIntInput("Enter Author's Employee ID: ");
    int cat_id = view.getIntInput("Enter Category ID: ");
    std::string title = view.getInput("Enter title: ");
    std::string desc = view.getInput("Enter description: ");
    int ver = view.getIntInput("Enter version: ");
    model.addDocument(emp_id, cat_id, title, desc, ver);
}
void Controller::handleAddFile() {
    int doc_id = view.getIntInput("Enter Document ID: ");
    std::string name = view.getInput("Enter file name: ");
    std::string format = view.getInput("Enter format: ");
    int size = view.getIntInput("Enter size (KB): ");
    model.addFile(doc_id, name, format, size);
}
void Controller::handleAddAccess() {
    int emp_id = view.getIntInput("Enter Employee ID: ");
    int doc_id = view.getIntInput("Enter Document ID: ");
    std::string level = view.getInput("Enter access level: ");
    std::string expiry = view.getInput("Enter expiry date (YYYY-MM-DD): ");
    model.addAccess(emp_id, doc_id, level, expiry);
}
void Controller::handleUpdateEmployee() {
    int id = view.getIntInput("Enter Employee ID to update: ");
    std::string first = view.getInput("Enter new first name: ");
    std::string last = view.getInput("Enter new last name: ");
    std::string email = view.getInput("Enter new email: ");
    std::string dep = view.getInput("Enter new department: ");
    model.updateEmployee(id, first, last, email, dep);
}

void Controller::handleUpdateCategory() {
    int id = view.getIntInput("Enter Category ID to update: ");
    std::string name = view.getInput("Enter new name: ");
    std::string desc = view.getInput("Enter new description: ");
    std::string type = view.getInput("Enter new type: ");
    model.updateCategory(id, name, desc, type);
}

void Controller::handleUpdateDocument() {
    int id = view.getIntInput("Enter Document ID to update: ");
    std::string title = view.getInput("Enter new title: ");
    std::string desc = view.getInput("Enter new description: ");
    int ver = view.getIntInput("Enter new version: ");
    model.updateDocument(id, title, desc, ver);
```

```
}

void Controller::handleUpdateFile() {
    int id = view.getIntInput("Enter File ID to update: ");
    std::string name = view.getInput("Enter new file name: ");
    std::string format = view.getInput("Enter new format: ");
    int size = view.getIntInput("Enter new size (KB): ");
    model.updateFile(id, name, format, size);
}

void Controller::handleUpdateAccess() {
    int id = view.getIntInput("Enter Access ID to update: ");
    std::string level = view.getInput("Enter new access level: ");
    std::string expiry = view.getInput("Enter new expiry date (YYYY-MM-DD): ");
    model.updateAccess(id, level, expiry);
}

void Controller::handleGenerateEmployees() {
    int n = getGenerationCount(); if (n == 0) return;
    model.generateEmployees(n);
    std::ostringstream oss; oss << n;
    view.displayStatus(Status::GENERATE_SUCCESS, oss.str());
}
void Controller::handleGenerateCategories() {
    int n = getGenerationCount(); if (n == 0) return;
    model.generateCategories(n);
    std::ostringstream oss; oss << n;
    view.displayStatus(Status::GENERATE_SUCCESS, oss.str());
}
void Controller::handleGenerateDocuments() {
    int n = getGenerationCount(); if (n == 0) return;
    model.generateDocuments(n);
    std::ostringstream oss; oss << n;
    view.displayStatus(Status::GENERATE_SUCCESS, oss.str());
}
void Controller::handleGenerateFiles() {
    int n = getGenerationCount(); if (n == 0) return;
    model.generateFiles(n);
    std::ostringstream oss; oss << n;
    view.displayStatus(Status::GENERATE_SUCCESS, oss.str());
}
void Controller::handleGenerateAccess() {
    int n = getGenerationCount(); if (n == 0) return;
    model.generateAccess(n);
    std::ostringstream oss; oss << n;
    view.displayStatus(Status::GENERATE_SUCCESS, oss.str());
}

void Controller::handleSearch1() {
    std::string dept = view.getInput("Enter department (like): ");
    std::string type = view.getInput("Enter category type (like): ");
    double time = model.searchDocsByDeptAndCat(dept, type);
    view.showExecutionTime(time);
}
void Controller::handleSearch2() {
    int min_s = view.getIntInput("Enter MIN file size (KB): ");
    int max_s = view.getIntInput("Enter MAX file size (KB): ");
    if (min_s > max_s) { view.displayStatus(Status::INVALID_INPUT_GENERAL, "Min size
cannot be greater than max size."); return; }
    std::string level = view.getInput("Enter access level (like): ");
    double time = model.searchFilesByAccessAndSize(level, min_s, max_s);
    view.showExecutionTime(time);
}
```

```
void Controller::handleSearch3() {
    std::string start = view.getInput("Enter start date (YYYY-MM-DD): ");
    std::string end = view.getInput("Enter end date (YYYY-MM-DD): ");
    double time = model.searchDocCountByEmployeeAndDate(start, end);
    view.showExecutionTime(time);
}
```

Model.h:

```
#pragma once
#include <soci/soci.h>
#include <ctime>

using std::string;

struct Employee { int id; string first, last, email, dept; };
struct Category { int id; string name, desc, type; };
struct Document { int id, emp_id, cat_id; string title, desc; std::tm created,
updated; int ver; };
struct File { int id, doc_id; string name, format; int size; std::tm uploaded; };
struct Access { int id, emp_id, doc_id; string level; std::tm granted, expiry; };

struct SearchResult1 { string emp_name, emp_last, doc_title, cat_name; };
struct SearchResult2 { string file_name, format; int size; string doc_title, level;
};
struct SearchResult3 { string first, last; long long count; };

class Model {
private:
    soci::session sql;
    long long getLastId(const string& table, const string& id_col);
    void checkDbConnection();
    std::tm stringToTm(const string& date_str);

public:
    Model(const string& conn_string);
    ~Model();

    void printAllEmployees();
    void printAllCategories();
    void printAllDocuments();
    void printAllFiles();
    void printAllAccess();

    void addEmployee(const string& first, const string& last, const string& email,
const string& dep);
    void addCategory(const string& name, const string& desc, const string& type);
    void addDocument(int emp_id, int cat_id, const string& title, const string&
desc, int version);
    void addFile(int doc_id, const string& name, const string& format, int size);
    void addAccess(int emp_id, int doc_id, const string& level, const string&
expiry_str);

    void updateEmployee(int id, const string& first, const string& last, const
string& email, const string& dep);
    void updateCategory(int id, const string& name, const string& desc, const string&
type);
    void updateDocument(int id, const string& title, const string& desc, int
version);
    void updateFile(int id, const string& name, const string& format, int size);
    void updateAccess(int id, const string& level, const string& expiry_str);

    void deleteData(const string& table_name, const string& id_col, int id);
```

```
    void generateEmployees(int count);
    void generateCategories(int count);
    void generateDocuments(int count);
    void generateFiles(int count);
    void generateAccess(int count);

    double searchDocsByDeptAndCat(const string& department, const string& category_type);
    double searchFilesByAccessAndSize(const string& access_level, int min_size, int max_size);
    double searchDocCountByEmployeeAndDate(const string& date_start_str, const string& date_end_str);
};
```

Model.cpp:

```
#include "Model.h"
#include <soci/postgresql/soci-postgresql.h>
#include <soci/rowset.h>
#include <iostream>
#include <ctime>
#include <iomanip>

namespace soci
{
     template<> struct type_conversion<Employee> {
        typedef values base_type;
        static void from_base(values const& v, indicator, Employee& e) {
            e.id = v.get<int>("employee_id"); e.first = v.get<string>("first_name");
            e.last = v.get<string>("last_name"); e.email = v.get<string>("email");
            e.dept = v.get<string>("department");
        }
    };
    template<> struct type_conversion<Category> {
        typedef values base_type;
        static void from_base(values const& v, indicator, Category& c) {
            c.id = v.get<int>("category_id"); c.name = v.get<string>("category_name");
            c.desc = v.get<string>("description"); c.type = v.get<string>("type");
        }
    };
    template<> struct type_conversion<Document> {
        typedef values base_type;
        static void from_base(values const& v, indicator, Document& d) {
            d.id = v.get<int>("document_id"); d.emp_id = v.get<int>("employee_id");
            d.cat_id = v.get<int>("category_id"); d.title = v.get<string>("title");
            d.desc = v.get<string>("description"); d.created = v.get<std::tm>("date_created");
            d.updated = v.get<std::tm>("date_updated"); d.ver = v.get<int>("version");
        }
    };
    template<> struct type_conversion<File> {
        typedef values base_type;
        static void from_base(values const& v, indicator, File& f) {
            f.id = v.get<int>("file_id"); f.doc_id = v.get<int>("document_id");
            f.name = v.get<string>("file_name"); f.format = v.get<string>("format");
            f.size = v.get<int>("size"); f.uploaded = v.get<std::tm>("upload_date");
        }
```

```
    };
    template<> struct type_conversion<Access> {
        typedef values base_type;
        static void from_base(values const& v, indicator, Access& a) {
            a.id = v.get<int>("access_id"); a.emp_id = v.get<int>("employee_id");
            a.doc_id       =       v.get<int>("document_id");       a.level       =
v.get<string>("access_level");
            a.granted     =     v.get<std::tm>("date_granted");     a.expiry     =
v.get<std::tm>("expiry_date");
        }
    };
}

static std::string truncate(std::string str, size_t width) {
    if (str.length() > width) {
        return str.substr(0, width - 3) + "...";
    }
    return str;
}


Model::Model(const string& conn_string) {
    try {
        sql.open(soci::postgresql, conn_string);
        checkDbConnection();
    }
    catch (const std::exception& e) {
        throw std::runtime_error("Database connection failed: " + string(e.what()));
    }
}

Model::~Model() {
    if (sql.is_connected()) { sql.close(); }
}

void Model::checkDbConnection() {
    if   (!sql.is_connected())   {   throw   std::runtime_error("Database   connection
lost."); }
}

long long Model::getLastId(const string& table, const string& id_col) {
    long long last_id = 0;
    sql << "SELECT COALESCE(MAX(\"" + id_col + "\"), 0) FROM public.\"" + table +
"\"", soci::into(last_id);
    return last_id;
}

std::tm Model::stringToTm(const string& date_str) {
    std::tm t = {};
    if (date_str.length() != 10 || date_str[4] != '-' || date_str[7] != '-') {
        throw std::runtime_error("ERROR: Invalid date format. Use YYYY-MM-DD.");
    }
    try {
        t.tm_year = std::stoi(date_str.substr(0, 4));
        t.tm_mon = std::stoi(date_str.substr(5, 2));
        t.tm_mday = std::stoi(date_str.substr(8, 2));
    }
    catch (...) {
        throw std::runtime_error("ERROR: Invalid date components. Use YYYY-MM-DD.");
    }
    t.tm_year -= 1900;
    t.tm_mon -= 1;
    return t;
```

```
}

string tmToStringHelper(std::tm t) {
    char buffer[11];
    if (std::strftime(buffer, sizeof(buffer), "%Y-%m-%d", &t)) {
        return string(buffer);
    }
    return "invalid date";
}

void Model::printAllEmployees() {
    checkDbConnection();
    soci::rowset<Employee> rs = (sql.prepare << "SELECT * FROM public.employee ORDER
BY employee_id");
    int count = 0;

    std::cout << std::left
        << std::setw(6) << "ID" << " | "
        << std::setw(30) << "Name" << " | "
        << std::setw(35) << "Email" << " | "
        << std::setw(30) << "Department" << "\n";

    std::cout << string(109, '-') << "\n";

    for (const auto& e : rs) {
        std::cout << std::left
            << std::setw(6) << e.id << " | "
            << std::setw(30) << truncate(e.first + " " + e.last, 30) << " | "
            << std::setw(35) << truncate(e.email, 35) << " | "
            << std::setw(30) << truncate(e.dept, 30) << "\n";
        count++;
    }
    if (count == 0) std::cout << "No employees found.\n";
}
void Model::printAllCategories() {
    checkDbConnection();
    soci::rowset<Category> rs = (sql.prepare << "SELECT * FROM public.category ORDER
BY category_id");
    int count = 0;

    std::cout << std::left
        << std::setw(6) << "ID" << " | "
        << std::setw(30) << "Name" << " | "
        << std::setw(40) << "Description" << " | "
        << std::setw(15) << "Type" << "\n";
    std::cout << string(100, '-') << "\n";

    for (const auto& c : rs) {
        std::cout << std::left
            << std::setw(6) << c.id << " | "
            << std::setw(30) << truncate(c.name, 30) << " | "
            << std::setw(40) << truncate(c.desc, 40) << " | "
            << std::setw(15) << truncate(c.type, 15) << "\n";
        count++;
    }
    if (count == 0) std::cout << "No categories found.\n";
}

void Model::printAllDocuments() {
    checkDbConnection();
    soci::rowset<Document> rs = (sql.prepare << "SELECT * FROM public.document ORDER
BY document_id");
    int count = 0;
```

```
    for (const auto& d : rs) {
        std::cout << "-------------------------------------------------------------
--------------------\n"
            << std::left
            << std::setw(6) << "ID: " << std::setw(10) << d.id
            << "| Title: " << truncate(d.title, 50)
            << " | V: " << d.ver << "\n"
            << "  Desc: " << truncate(d.desc, 65) << "\n"
            << "  " << std::setw(13) << ("AuthorID: " + std::to_string(d.emp_id))
            << "| " << std::setw(10) << ("CatID: " + std::to_string(d.cat_id)) <<
"\n"
            << "  " << std::setw(22) << ("Created: " + tmToStringHelper(d.created))
            << "| Updated: " << tmToStringHelper(d.updated) << "\n";
        count++;
    }
    if (count == 0) std::cout << "No documents found.\n";
}

void Model::printAllFiles() {
    checkDbConnection();
    soci::rowset<File> rs = (sql.prepare << "SELECT * FROM public.file ORDER BY
file_id");
    int count = 0;

    std::cout << std::left
        << std::setw(6) << "ID" << " | "
        << std::setw(35) << "Name" << " | "
        << std::setw(10) << "Format" << " | "
        << std::setw(10) << "Size(KB)" << " | "
        << std::setw(8) << "DocID" << " | "
        << std::setw(12) << "Uploaded" << "\n";
    std::cout << string(95, '-') << "\n";

    for (const auto& f : rs) {
        std::cout << std::left
            << std::setw(6) << f.id << " | "
            << std::setw(35) << truncate(f.name, 35) << " | "
            << std::setw(10) << truncate(f.format, 10) << " | "
            << std::setw(10) << f.size << " | "
            << std::setw(8) << f.doc_id << " | "
            << std::setw(12) << tmToStringHelper(f.uploaded) << "\n";
        count++;
    }
    if (count == 0) std::cout << "No files found.\n";
}

void Model::printAllAccess() {
    checkDbConnection();
    soci::rowset<Access> rs = (sql.prepare << "SELECT * FROM public.access ORDER BY
access_id");
    int count = 0;

    std::cout << std::left
        << std::setw(6) << "ID" << " | "
        << std::setw(8) << "EmpID" << " | "
        << std::setw(8) << "DocID" << " | "
        << std::setw(15) << "Access Level" << " | "
        << std::setw(12) << "Granted" << " | "
        << std::setw(12) << "Expires" << "\n";
    std::cout << string(84, '-') << "\n";

    for (const auto& a : rs) {
```

```
        std::cout << std::left
            << std::setw(6) << a.id << " | "
            << std::setw(8) << a.emp_id << " | "
            << std::setw(8) << a.doc_id << " | "
            << std::setw(15) << truncate(a.level, 15) << " | "
            << std::setw(12) << tmToStringHelper(a.granted) << " | "
            << std::setw(12) << tmToStringHelper(a.expiry) << "\n";
        count++;
    }
    if (count == 0) std::cout << "No access found.\n";
}

void Model::addEmployee(const string& first, const string& last, const string&
email, const string& dep) {
    checkDbConnection();
    long long new_id = getLastId("employee", "employee_id") + 1;
    sql << "INSERT INTO public.employee (employee_id, first_name, last_name, email,
department) VALUES (:id, :f, :l, :e, :d)",
        soci::use(new_id), soci::use(first), soci::use(last), soci::use(email),
soci::use(dep);
}

void Model::addCategory(const string& name, const string& desc, const string& type)
{
    checkDbConnection();
    long long new_id = getLastId("category", "category_id") + 1;
    sql << "INSERT INTO public.category (category_id, category_name, description,
type) VALUES (:id, :n, :d, :t)",
        soci::use(new_id), soci::use(name), soci::use(desc), soci::use(type);
}

void Model::addDocument(int emp_id, int cat_id, const string& title, const string&
desc, int version) {
    checkDbConnection();
    long long new_id = getLastId("document", "document_id") + 1;
    std::tm now = {}; time_t t = time(0); localtime_s(&now, &t);

    int exists = 0;
    sql << "SELECT 1 FROM public.employee WHERE employee_id = :id",
soci::use(emp_id), soci::into(exists);
    if (!exists) throw std::runtime_error("ERROR: Employee ID=" +
std::to_string(emp_id) + " not found.");
    exists = 0;
    sql << "SELECT 1 FROM public.category WHERE category_id = :id",
soci::use(cat_id), soci::into(exists);
    if (!exists) throw std::runtime_error("ERROR: Category ID=" +
std::to_string(cat_id) + " not found.");

    sql << "INSERT INTO public.document (document_id, employee_id, category_id,
title, description, date_created, date_updated, version) "
        "VALUES (:id, :eid, :cid, :ti, :d, :cr, :up, :v)",
        soci::use(new_id), soci::use(emp_id), soci::use(cat_id), soci::use(title),
soci::use(desc),
        soci::use(now), soci::use(now), soci::use(version);
}

void Model::addFile(int doc_id, const string& name, const string& format, int size)
{
    checkDbConnection();
    long long new_id = getLastId("file", "file_id") + 1;
    std::tm now = {}; time_t t = time(0); localtime_s(&now, &t);

    int exists = 0;
```

```
    sql << "SELECT 1 FROM public.document WHERE document_id = :id",
soci::use(doc_id), soci::into(exists);
    if (!exists) throw std::runtime_error("ERROR: Document ID=" +
std::to_string(doc_id) + " not found.");

    sql << "INSERT INTO public.file (file_id, document_id, file_name, format, size,
upload_date) "
        "VALUES (:id, :did, :n, :f, :s, :up)",
        soci::use(new_id), soci::use(doc_id), soci::use(name), soci::use(format),
soci::use(size), soci::use(now);
}

void Model::addAccess(int emp_id, int doc_id, const string& level, const string&
expiry_str) {
    checkDbConnection();
    long long new_id = getLastId("access", "access_id") + 1;
    std::tm now = {}; time_t t = time(0); localtime_s(&now, &t);
    std::tm expiry_tm = stringToTm(expiry_str);

    int emp_exists = 0, doc_exists = 0;
    sql << "SELECT 1 FROM public.employee WHERE employee_id = :id",
soci::use(emp_id), soci::into(emp_exists);
    if (!emp_exists) throw std::runtime_error("ERROR: Employee ID=" +
std::to_string(emp_id) + " not found.");
    sql << "SELECT 1 FROM public.document WHERE document_id = :id",
soci::use(doc_id), soci::into(doc_exists);
    if (!doc_exists) throw std::runtime_error("ERROR: Document ID=" +
std::to_string(doc_id) + " not found.");

    sql << "INSERT INTO public.access (access_id, employee_id, document_id,
access_level, date_granted, expiry_date) "
        "VALUES (:id, :eid, :did, :l, :gr, :ex)",
        soci::use(new_id), soci::use(emp_id), soci::use(doc_id), soci::use(level),
soci::use(now), soci::use(expiry_tm);
}

void Model::updateEmployee(int id, const string& first, const string& last, const
string& email, const string& dep) {
    checkDbConnection();
    sql << "UPDATE public.employee SET first_name = :f, last_name = :l, email = :e,
department = :d WHERE employee_id = :id",
        soci::use(first), soci::use(last), soci::use(email), soci::use(dep),
soci::use(id);
}

void Model::updateCategory(int id, const string& name, const string& desc, const
string& type) {
    checkDbConnection();
    sql << "UPDATE public.category SET category_name = :n, description = :d, type =
:t WHERE category_id = :id",
        soci::use(name), soci::use(desc), soci::use(type), soci::use(id);
}

void Model::updateDocument(int id, const string& title, const string& desc, int
version) {
    checkDbConnection();
    std::tm now = {}; time_t t = time(0); localtime_s(&now, &t);

    sql << "UPDATE public.document SET title = :ti, description = :d, version = :v,
date_updated = :up WHERE document_id = :id",
        soci::use(title), soci::use(desc), soci::use(version), soci::use(now),
soci::use(id);
}
```

```
void Model::updateFile(int id, const string& name, const string& format, int size)
{
    checkDbConnection();
    sql << "UPDATE public.file SET file_name = :n, format = :f, size = :s WHERE
file_id = :id",
        soci::use(name), soci::use(format), soci::use(size), soci::use(id);
}

void Model::updateAccess(int id, const string& level, const string& expiry_str) {
    checkDbConnection();
    std::tm expiry_tm = stringToTm(expiry_str);

    sql << "UPDATE public.access SET access_level = :l, expiry_date = :ex WHERE
access_id = :id",
        soci::use(level), soci::use(expiry_tm), soci::use(id);
}

void Model::deleteData(const string& table_name, const string& id_col, int id) {
    checkDbConnection();
    try {
        sql << "DELETE FROM public.\"" + table_name + "\" WHERE \"" + id_col + "\"
= :id", soci::use(id);
    }
    catch (const std::exception& e) {
        string msg = e.what();
        if (msg.find("violates foreign key constraint") != string::npos) {
            throw   std::runtime_error("ERROR:   Cannot   delete   record   ID="   +
std::to_string(id) + " from '" + table_name + "', it's referenced by other tables.");
        }
        else { throw std::runtime_error("ERROR during deletion: " + msg); }
    }
}

void Model::generateEmployees(int count) {
    checkDbConnection();
    sql << R"( INSERT INTO public.employee (employee_id, first_name, last_name,
email,  department)  SELECT  n  +  (SELECT  COALESCE(MAX(employee_id),  0)  FROM
public.employee), 'FirstName_' || (RANDOM() * 1000)::int, 'LastName_' || (RANDOM()
*  1000)::int,  'user'  ||  (n  +  (SELECT  COALESCE(MAX(employee_id),  0)  FROM
public.employee))  ||  '@example.com',  (ARRAY['Development',  'Analytics',  'HR',
'Support'])[ (RANDOM() * 3 + 1)::int ] FROM generate_series(1, :count) AS s(n) ON
CONFLICT (employee_id) DO NOTHING )", soci::use(count);
}
void Model::generateCategories(int count) {
    checkDbConnection();
    sql << R"( INSERT INTO public.category (category_id, category_name, description,
type)  SELECT  n  +  (SELECT  COALESCE(MAX(category_id),  0)  FROM  public.category),
'Category  '  ||  (RANDOM()  *  1000)::int,  'Description  '  ||  n,  (ARRAY['Internal',
'Public', 'Confidential'])[ (RANDOM() * 2 + 1)::int ] FROM generate_series(1, :count)
AS s(n) ON CONFLICT (category_id) DO NOTHING )", soci::use(count);
}
void Model::generateDocuments(int count) {
    checkDbConnection();
    long long emp_c = 0, cat_c = 0; sql << "SELECT COUNT(*) FROM public.employee",
soci::into(emp_c); sql << "SELECT COUNT(*) FROM public.category", soci::into(cat_c);
    if (emp_c == 0 || cat_c == 0) throw std::runtime_error("ERROR: Generate employees
and categories first.");
    sql << R"( INSERT INTO public.document (document_id, employee_id, category_id,
title,  description,  date_created,  date_updated,  version)  SELECT  n  +  (SELECT
COALESCE(MAX(document_id),  0)  FROM  public.document),  (SELECT  employee_id  FROM
public.employee ORDER BY RANDOM() LIMIT 1), (SELECT category_id FROM public.category
ORDER BY RANDOM() LIMIT 1), 'Document Title ' || n, 'Generated description', NOW()
```

```
- (RANDOM() * '365 days'::interval), NOW() - (RANDOM() * '30 days'::interval), (RANDOM() * 5 + 1)::int FROM generate_series(1, :count) AS s(n) ON CONFLICT (document_id) DO NOTHING )", soci::use(count);
}
void Model::generateFiles(int count) {
    checkDbConnection(); long long doc_c = 0; sql << "SELECT COUNT(*) FROM public.document", soci::into(doc_c);
    if (doc_c == 0) throw std::runtime_error("ERROR: Generate documents first.");
    sql << R"( INSERT INTO public.file (file_id, document_id, file_name, format, size, upload_date) SELECT n + (SELECT COALESCE(MAX(file_id), 0) FROM public.file), (SELECT document_id FROM public.document ORDER BY RANDOM() LIMIT 1), 'file_' || n || (ARRAY['.pdf', '.docx', '.txt', '.zip'])[ (RANDOM() * 3 + 1)::int ], (ARRAY['PDF', 'DOCX', 'TXT', 'ZIP'])[ (RANDOM() * 3 + 1)::int ], (RANDOM() * 10000 + 100)::int, NOW() - (RANDOM() * '90 days'::interval) FROM generate_series(1, :count) AS s(n) ON CONFLICT (file_id) DO NOTHING )", soci::use(count);
}
void Model::generateAccess(int count) {
    checkDbConnection(); long long emp_c = 0, doc_c = 0; sql << "SELECT COUNT(*) FROM public.employee", soci::into(emp_c); sql << "SELECT COUNT(*) FROM public.document", soci::into(doc_c);
    if (emp_c == 0 || doc_c == 0) throw std::runtime_error("ERROR: Generate employees and documents first.");
    sql << R"( INSERT INTO public.access (access_id, employee_id, document_id, access_level, date_granted, expiry_date) SELECT n + (SELECT COALESCE(MAX(access_id), 0) FROM public.access), (SELECT employee_id FROM public.employee ORDER BY RANDOM() LIMIT 1), (SELECT document_id FROM public.document ORDER BY RANDOM() LIMIT 1), (ARRAY['read', 'edit', 'delete'])[ (RANDOM() * 2 + 1)::int ], NOW() - (RANDOM() * '10 days'::interval), NOW() + (RANDOM() * '365 days'::interval) FROM generate_series(1, :count) AS s(n) ON CONFLICT (access_id) DO NOTHING )", soci::use(count);
}

double Model::searchDocsByDeptAndCat(const string& department, const string& category_type) {
    checkDbConnection();
    string dept_pattern = "%" + department + "%"; string cat_pattern = "%" + category_type + "%";
    int count = 0;
    clock_t start = clock();

    soci::rowset<soci::row> rs = (sql.prepare <<
        R"( SELECT e.first_name, e.last_name, d.title, c.category_name FROM public.employee e
            JOIN public.document d ON e.employee_id = d.employee_id JOIN public.category c ON d.category_id = c.category_id
            WHERE e.department LIKE :dept AND c.type LIKE :cat_type )",
        soci::use(dept_pattern, "dept"), soci::use(cat_pattern, "cat_type"));

    std::cout << "\n--- Search Results (Docs by Dept & Category Type) ---\n";
    std::cout << std::left
        << std::setw(30) << "Author" << " | "
        << std::setw(50) << "Document Title" << " | "
        << std::setw(25) << "Category" << "\n";
    std::cout << string(111, '-') << "\n";

    for (const auto& r : rs) {
        std::cout << std::left
            << std::setw(30) << truncate(r.get<string>(0) + " " + r.get<string>(1), 30) << " | "
            << std::setw(40) << truncate(r.get<string>(2), 50) << " | "
            << std::setw(25) << truncate(r.get<string>(3), 25) << "\n";
        count++;
    }
```

```
    clock_t end = clock();
    if (count == 0) std::cout << "No results found.\n";
    double ms_double = (double)(end - start) * 1000.0 / CLOCKS_PER_SEC;
    return ms_double;
}

double Model::searchFilesByAccessAndSize(const string& access_level, int min_size,
int max_size) {
    checkDbConnection();
    string level_pattern = "%" + access_level + "%";
    int count = 0;
    clock_t start = clock();

    soci::rowset<soci::row> rs = (sql.prepare <<
        R"( SELECT f.file_name, f.format, f.size, d.title, a.access_level FROM
public.file f
            JOIN public.document d ON f.document_id = d.document_id JOIN
public.access a ON d.document_id = a.document_id
            WHERE a.access_level LIKE :level AND f.size BETWEEN :min_s AND :max_s
ORDER BY f.size DESC )",
        soci::use(level_pattern, "level"), soci::use(min_size, "min_s"),
soci::use(max_size, "max_s"));

    std::cout << "\n--- Search Results (Files by Access & Size) ---\n";
    std::cout << std::left
        << std::setw(30) << "File" << " | "
        << std::setw(50) << "Document" << " | "
        << std::setw(10) << "Size(KB)" << " | "
        << std::setw(15) << "Access" << "\n";
    std::cout << string(114, '-') << "\n";

    for (const auto& r : rs) {
        std::cout << std::left
            << std::setw(30) << truncate(r.get<string>(0), 30) << " | "
            << std::setw(30) << truncate(r.get<string>(3), 50) << " | "
            << std::setw(10) << r.get<int>(2) << " | "
            << std::setw(15) << truncate(r.get<string>(4), 15) << "\n";
        count++;
    }

    clock_t end = clock();
    if (count == 0) std::cout << "No results found.\n";
    double ms_double = (double)(end - start) * 1000.0 / CLOCKS_PER_SEC;
    return ms_double;
}

double Model::searchDocCountByEmployeeAndDate(const string& date_start_str, const
string& date_end_str) {
    checkDbConnection();
    std::tm date_start = stringToTm(date_start_str);
    std::tm date_end = stringToTm(date_end_str);
    int count = 0;
    clock_t start = clock();
    soci::rowset<soci::row> rs = (sql.prepare <<
        R"( SELECT e.first_name, e.last_name, COUNT(d.document_id) as doc_count FROM
public.employee e
            JOIN public.document d ON e.employee_id = d.employee_id WHERE
d.date_created BETWEEN :start AND :end
            GROUP BY e.employee_id, e.first_name, e.last_name ORDER BY doc_count
DESC )",
        soci::use(date_start, "start"), soci::use(date_end, "end"));
    std::cout << "\n--- Search Results (Doc Count by Author & Date) ---\n";
```

```
    std::cout << std::left
        << std::setw(40) << "Author" << " | "
        << std::setw(15) << "Document Count" << "\n";
    std::cout << string(50, '-') << "\n";
    for (const auto& r : rs) {
        std::cout << std::left
            << std::setw(40) << truncate(r.get<string>(0) + " " + r.get<string>(1),
30) << " | "
            << std::setw(15) << r.get<long long>(2) << "\n";
        count++;
    }
    clock_t end = clock();
    if (count == 0) std::cout << "No results found.\n";
    double ms_double = (double)(end - start) * 1000.0 / CLOCKS_PER_SEC;
    return ms_double;
}
```

Контакти:

Репозиторій GIT: https://github.com/Pixe1Pa1/Databases_course.git

Telegram: @justz_egor