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

**Лабораторна робота № 2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав:

студентк ІІI курсу

групи КВ-22

Савельєв О.С

Перевірив:

Павловский В. І.

Посилання на гітхаб [AlexandrCC/LR2\_3-DB (github.com)](https://github.com/AlexandrCC/LR2_3-DB)

Київ – 2024

**Мета:** здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

**Завдання за варіантом:**

|  |  |  |
| --- | --- | --- |
| *22* | *Hash, BRIN* | *after delete, insert* |

**Виконання роботи**

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

Рисунок 1 – Логічна модель

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Відповідні класи ORM утворюють зв’язки що можна зобразити чином показаним на рисунку 2.

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Рисунок 2 – Класи ORM

В міру того що для кожного класу свої окремі функції приклад буде приведений на різних класах

Запит на вставку має наступний вигляд:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Запит на оновлення:

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Запит на читання:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Запит на видалення

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Код зміненого модуля Model:

#include <iostream>

#include <string>

#include <optional>

#include <orm/db.hpp>

#include <orm/tiny/model.hpp>

using namespace Orm; // Or your chosen namespace as per TinyORM setup

class Category : public Model<Category> {

public:

int Category\_ID;

std::string Category\_Name;

static constexpr auto tableName = "Categories";

// Columns mapping (if TinyORM requires manual mapping)

TINYORM\_COLUMNS(Category\_ID, Category\_Name);

// Relationships: 1 Category has many Medicines

// The foreign key is on the Medicines table.

auto medicines() {

return hasMany<Medicine>("Category\_id");

}

};

class Disease : public Model<Disease> {

public:

int Disease\_id;

std::string Disease\_name;

static constexpr auto tableName = "Diseases";

TINYORM\_COLUMNS(Disease\_id, Disease\_name);

// Relationships:

// Many-to-many to Medicine through Medicine\_Treats\_Diseases

auto treatedByMedicines() {

return belongsToMany<Medicine>("Medicine\_Treats\_Diseases", "Disease\_id", "Medicine\_id");

}

// Many-to-many to Medicine through Medicine\_Contradicts\_Diseases

auto contradictedByMedicines() {

return belongsToMany<Medicine>("Medicine\_Contradicts\_Diseases", "Disease\_id", "Medicine\_id");

}

};

class Manufacturer : public Model<Manufacturer> {

public:

int Manufacturer\_id;

std::string Manufacturer\_name;

std::string Location\_of\_facilities; // Remember to handle the column name with spaces appropriately

static constexpr auto tableName = "Manufactures";

TINYORM\_COLUMNS(Manufacturer\_id, Manufacturer\_name, Location\_of\_facilities);

// Relationships:

// 1 Manufacturer has many Medicines

auto medicines() {

return hasMany<Medicine>("Manufacturer\_id");

}

// Many-to-many with Medicine through man\_med\_tab

auto medicinesThroughManMedTab() {

return belongsToMany<Medicine>("man\_med\_tab", "Manufacturer\_id", "Medicine\_id");

}

};

class Specs : public Model<Specs> {

public:

int specs\_id;

std::string Lisence\_name;

bool Term\_of\_issue;

std::string Medicine\_form;

static constexpr auto tableName = "Specs";

TINYORM\_COLUMNS(specs\_id, Lisence\_name, Term\_of\_issue, Medicine\_form);

// 1 Specs belongs to 1 Medicine (or 1:1)

// If Specs\_id is unique in Medicines, we can define:

auto medicine() {

return hasOne<Medicine>("Specs\_id");

}

};

class Medicine : public Model<Medicine> {

public:

int Medicine\_id;

int Manufacturer\_id;

int Specs\_id;

int Category\_id;

int Treats\_id;

int Contradicts\_id;

std::string Medicine\_name;

static constexpr auto tableName = "Medicines";

TINYORM\_COLUMNS(Medicine\_id, Manufacturer\_id, Specs\_id, Category\_id, Treats\_id, Contradicts\_id, Medicine\_name);

// Relationships:

auto manufacturer() {

return belongsTo<Manufacturer>("Manufacturer\_id");

}

auto specs() {

return belongsTo<Specs>("Specs\_id");

}

auto category() {

return belongsTo<Category>("Category\_id");

}

// Many-to-many relationships via pivot tables:

auto treatsDiseases() {

return belongsToMany<Disease>("Medicine\_Treats\_Diseases", "Medicine\_id", "Disease\_id");

}

auto contradictsDiseases() {

return belongsToMany<Disease>("Medicine\_Contradicts\_Diseases", "Medicine\_id", "Disease\_id");

}

};

// Pivot models if needed explicitly (TinyORM might not require explicit pivot models):

class MedicineTreatsDiseases : public Model<MedicineTreatsDiseases> {

public:

int tab\_id;

int Medicine\_id;

int Disease\_id;

static constexpr auto tableName = "Medicine\_Treats\_Diseases";

TINYORM\_COLUMNS(tab\_id, Medicine\_id, Disease\_id);

};

class MedicineContradictsDiseases : public Model<MedicineContradictsDiseases> {

public:

int tab\_id;

int Medicine\_id;

int Disease\_id;

static constexpr auto tableName = "Medicine\_Contradicts\_Diseases";

TINYORM\_COLUMNS(tab\_id, Medicine\_id, Disease\_id);

};

class ManMedTab : public Model<ManMedTab> {

public:

int tab\_id;

int Medicine\_id;

int Manufacturer\_id;

static constexpr auto tableName = "man\_med\_tab";

TINYORM\_COLUMNS(tab\_id, Medicine\_id, Manufacturer\_id);

};

class Model {

public:

Model() {

// Ensure that DB::connect(...) or equivalent TinyORM initialization

// has been done at application start. Not shown here.

}

//-------------------------

// Medicine CRUD

//-------------------------

void createMedicine(int manufacturer\_id, int specs\_id, int category\_id,

int treats\_id, int contradicts\_id, const std::string& medicine\_name)

{

Medicine medicine;

medicine.Manufacturer\_id = manufacturer\_id;

medicine.Specs\_id = specs\_id;

medicine.Category\_id = category\_id;

medicine.Treats\_id = treats\_id;

medicine.Contradicts\_id = contradicts\_id;

medicine.Medicine\_name = medicine\_name;

medicine.save();

std::cout << "Medicine created successfully." << std::endl;

}

void readMedicine(int medicine\_id) {

auto medicine = Medicine::find(medicine\_id);

if (!medicine) {

std::cout << "No medicine found with ID: " << medicine\_id << std::endl;

return;

}

std::cout << "Medicine ID: " << medicine->Medicine\_id << "\n"

<< "Manufacturer ID: " << medicine->Manufacturer\_id << "\n"

<< "Specs ID: " << medicine->Specs\_id << "\n"

<< "Category ID: " << medicine->Category\_id << "\n"

<< "Treats ID: " << medicine->Treats\_id << "\n"

<< "Contradicts ID: " << medicine->Contradicts\_id << "\n"

<< "Medicine Name: " << medicine->Medicine\_name << "\n";

}

void updateMedicine(int medicine\_id, const std::string& column\_name, const std::string& new\_value) {

auto medicine = Medicine::find(medicine\_id);

if (!medicine) {

std::cerr << "No medicine found with ID: " << medicine\_id << std::endl;

return;

}

if (column\_name == "Medicine\_name") {

medicine->Medicine\_name = new\_value;

}

else if (column\_name == "Manufacturer\_id") {

medicine->Manufacturer\_id = std::stoi(new\_value);

}

else if (column\_name == "Specs\_id") {

medicine->Specs\_id = std::stoi(new\_value);

}

else if (column\_name == "Category\_id") {

medicine->Category\_id = std::stoi(new\_value);

}

else if (column\_name == "Treats\_id") {

medicine->Treats\_id = std::stoi(new\_value);

}

else if (column\_name == "Contradicts\_id") {

medicine->Contradicts\_id = std::stoi(new\_value);

}

else {

std::cerr << "Unknown column: " << column\_name << std::endl;

return;

}

medicine->save();

std::cout << "Medicine updated successfully." << std::endl;

}

void deleteMedicine(int medicine\_id) {

auto medicine = Medicine::find(medicine\_id);

if (!medicine) {

std::cerr << "No medicine found with ID: " << medicine\_id << std::endl;

return;

}

medicine->remove();

std::cout << "Medicine deleted successfully." << std::endl;

}

//-------------------------

// Category CRUD

//-------------------------

void createCategory(int category\_id, const std::string& category\_name) {

Category category;

category.Category\_ID = category\_id;

category.Category\_Name = category\_name;

category.save();

std::cout << "Category created successfully." << std::endl;

}

void readCategory(int category\_id) {

auto category = Category::find(category\_id);

if (!category) {

std::cout << "No category found with ID: " << category\_id << std::endl;

return;

}

std::cout << "Category ID: " << category->Category\_ID << "\n"

<< "Category Name: " << category->Category\_Name << "\n";

}

void updateCategory(int category\_id, const std::string& new\_name) {

auto category = Category::find(category\_id);

if (!category) {

std::cerr << "No category found with ID: " << category\_id << std::endl;

return;

}

category->Category\_Name = new\_name;

category->save();

std::cout << "Category updated successfully." << std::endl;

}

void deleteCategory(int category\_id) {

auto category = Category::find(category\_id);

if (!category) {

std::cerr << "No category found with ID: " << category\_id << std::endl;

return;

}

category->remove();

std::cout << "Category deleted successfully." << std::endl;

}

//-------------------------

// Disease CRUD

//-------------------------

void createDisease(int disease\_id, const std::string& disease\_name) {

Disease disease;

disease.Disease\_id = disease\_id;

disease.Disease\_name = disease\_name;

disease.save();

std::cout << "Disease created successfully." << std::endl;

}

void readDisease(int disease\_id) {

auto disease = Disease::find(disease\_id);

if (!disease) {

std::cout << "No disease found with ID: " << disease\_id << std::endl;

return;

}

std::cout << "Disease ID: " << disease->Disease\_id << "\n"

<< "Disease Name: " << disease->Disease\_name << "\n";

}

void updateDisease(int disease\_id, const std::string& new\_name) {

auto disease = Disease::find(disease\_id);

if (!disease) {

std::cerr << "No disease found with ID: " << disease\_id << std::endl;

return;

}

disease->Disease\_name = new\_name;

disease->save();

std::cout << "Disease updated successfully." << std::endl;

}

void deleteDisease(int disease\_id) {

auto disease = Disease::find(disease\_id);

if (!disease) {

std::cerr << "No disease found with ID: " << disease\_id << std::endl;

return;

}

disease->remove();

std::cout << "Disease deleted successfully." << std::endl;

}

//-------------------------

// Manufacturer CRUD

//-------------------------

void createManufacturer(int manufacturer\_id, const std::string& manufacturer\_name, const std::string& location) {

Manufacturer manufacturer;

manufacturer.Manufacturer\_id = manufacturer\_id;

manufacturer.Manufacturer\_name = manufacturer\_name;

manufacturer.Location\_of\_facilities = location;

manufacturer.save();

std::cout << "Manufacturer created successfully." << std::endl;

}

void readManufacturer(int manufacturer\_id) {

auto manufacturer = Manufacturer::find(manufacturer\_id);

if (!manufacturer) {

std::cout << "No manufacturer found with ID: " << manufacturer\_id << std::endl;

return;

}

std::cout << "Manufacturer ID: " << manufacturer->Manufacturer\_id << "\n"

<< "Manufacturer Name: " << manufacturer->Manufacturer\_name << "\n"

<< "Location of Facilities: " << manufacturer->Location\_of\_facilities << "\n";

}

void updateManufacturer(int manufacturer\_id, const std::string& column\_name, const std::string& new\_value) {

auto manufacturer = Manufacturer::find(manufacturer\_id);

if (!manufacturer) {

std::cerr << "No manufacturer found with ID: " << manufacturer\_id << std::endl;

return;

}

if (column\_name == "Manufacturer\_name") {

manufacturer->Manufacturer\_name = new\_value;

}

else if (column\_name == "Location\_of\_facilities") {

manufacturer->Location\_of\_facilities = new\_value;

}

else {

std::cerr << "Unknown column: " << column\_name << std::endl;

return;

}

manufacturer->save();

std::cout << "Manufacturer updated successfully." << std::endl;

}

void deleteManufacturer(int manufacturer\_id) {

auto manufacturer = Manufacturer::find(manufacturer\_id);

if (!manufacturer) {

std::cerr << "No manufacturer found with ID: " << manufacturer\_id << std::endl;

return;

}

manufacturer->remove();

std::cout << "Manufacturer deleted successfully." << std::endl;

}

//-------------------------

// Specs CRUD

//-------------------------

void createSpec(int specs\_id, const std::string& license\_name, bool term\_of\_issue, const std::string& medicine\_form) {

Specs specs;

specs.specs\_id = specs\_id;

specs.Lisence\_name = license\_name;

specs.Term\_of\_issue = term\_of\_issue;

specs.Medicine\_form = medicine\_form;

specs.save();

std::cout << "Spec created successfully." << std::endl;

}

void readSpec(int specs\_id) {

auto specs = Specs::find(specs\_id);

if (!specs) {

std::cout << "No specs found with ID: " << specs\_id << std::endl;

return;

}

std::cout << "Specs ID: " << specs->specs\_id << "\n"

<< "License Name: " << specs->Lisence\_name << "\n"

<< "Term of Issue: " << (specs->Term\_of\_issue ? "True" : "False") << "\n"

<< "Medicine Form: " << specs->Medicine\_form << "\n";

}

void updateSpec(int specs\_id, const std::string& column\_name, const std::string& new\_value) {

auto specs = Specs::find(specs\_id);

if (!specs) {

std::cerr << "No specs found with ID: " << specs\_id << std::endl;

return;

}

if (column\_name == "Lisence\_name") {

specs->Lisence\_name = new\_value;

}

else if (column\_name == "Term\_of\_issue") {

// Convert string to bool if necessary (e.g. "true"/"false")

bool val = (new\_value == "true" || new\_value == "1");

specs->Term\_of\_issue = val;

}

else if (column\_name == "Medicine\_form") {

specs->Medicine\_form = new\_value;

}

else {

std::cerr << "Unknown column: " << column\_name << std::endl;

return;

}

specs->save();

std::cout << "Specs updated successfully." << std::endl;

}

void deleteSpec(int specs\_id) {

auto specs = Specs::find(specs\_id);

if (!specs) {

std::cerr << "No specs found with ID: " << specs\_id << std::endl;

return;

}

specs->remove();

std::cout << "Spec deleted successfully." << std::endl;

}

};

Створення індексів:

Изображение выглядит как текст, программное обеспечение, Шрифт, Значок на компьютере

Автоматически созданное описание

Приклади запитів із виведенням результуючих даних, що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

Приклад 1: Просте фільтрування

EXPLAIN ANALYZESELECT \* FROM "Medicines"WHERE "Medicine\_id" = 12345;

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

**Пояснення:** Хеш-індекс за **medicine\_id** повинен прискорити цей запит, швидко знаходячи запис, а не скануючи всю таблиц

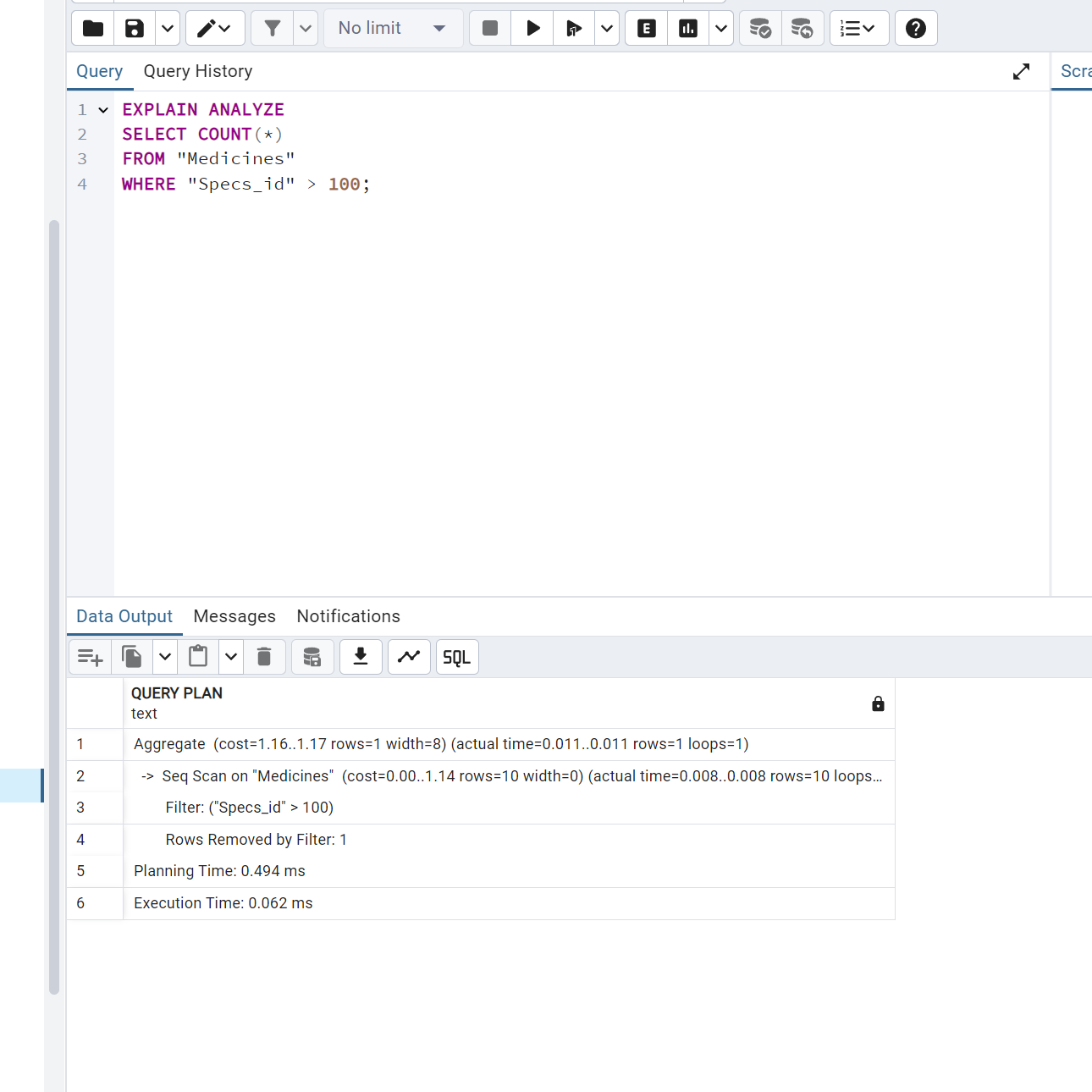
Приклад 2: Агрегатні функції

EXPLAIN ANALYZE

SELECT COUNT(\*)

FROM "Medicines"

WHERE "Specs\_id" > 100;



**Пояснення:** BRIN-індекс ефективний для великих таблиць з природно впорядкованими даними. Він повинен покращити виконання цього запиту, скануючи менше блоків.

Приклад 3: Групування та сортування

EXPLAIN ANALYZESELECT COUNT(\*)FROM "Medicines"WHERE "Specs\_id" > 100;

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**Пояснення:** Індекси можуть не істотно покращити групування та сортування, якщо використовувана колонка не індексована або якщо розмір таблиці малий

Приклад 4: Операції з'єднання

EXPLAIN ANALYZE

SELECT M."Medicine\_name", F."Manufacturer\_name", F."Location of facilities"

FROM "Medicines" M

JOIN "Manufactures" F

ON M."Manufacturer\_id" = F."Manufacturer\_id"

WHERE M."Category\_id" = 1;

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

**Пояснення:** Індекси на умовах з'єднання можуть значно покращити продуктивність, особливо для великих наборів даних

Коли не варто використовувати індексацію:

* **Малі таблиці:** Індексація може не бути корисною для малих таблиць, де повне сканування таблиці швидше.
* **Часті записи:** Висока активність вставки, оновлення або видалення може зробити індексацію менш ефективною через навантаження на підтримку індексу.
* **Колонки з низькою кардинальністю:** Колонки з малою кількістю унікальних значень (наприклад, стать) є поганими кандидатами для індексації.

**Розробка тригерів**

Код тригера:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Даний тригер викликає повідомлення про видалення або додавання в табличку ліків

Прикріплення тригеру до відповідної таблиці:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Тестування тригеру:

* Вставка запису про ліки

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Отримано відповідне повідомлення про вставку від тригера

* Видалення запису про ліки

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание Отримано тримано відповідне повідомлення тригеру.

**Використання рівнів ізоляції**

1. READ COMMITTED

На цьому рівні кожна команда в транзакції бачить лише дані, зафіксовані до початку команди.

Вікно 1:

BEGIN;

UPDATE "Medicines" SET "Medicine\_name" = 'NewName' WHERE "Medicine\_id" 1 Не фіксуємо зміни

Вікно 2:

BEGIN;

SELECT "Medicine\_name" FROM "Medicines" WHERE "Medicine\_id" = 1;

Вивід буде оригінальною назвою, не 'NewName'

COMMIT;

Вікно 1:

COMMIT;

Вікно 2:

SELECT "Medicine\_name" FROM "Medicines" WHERE "Medicine\_id" = 1;

Тепер вивід буде 'NewName'

COMMIT;

Зміна, зроблена у Вікні 1, не буде видима у Вікні 2, доки вона не зафіксована. Це демонструє ізоляцію READ COMMITTED.

2. REPEATABLE READ

Цей рівень гарантує, що будь-які дані, прочитані під час транзакції, не будуть змінені іншими транзакціями до завершення першої транзакції.

Вікно 1:

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN;

SELECT "Medicine\_name" FROM "Medicines" WHERE "Medicine\_id" = 1;

Не фіксуємо

Вікно 2:

BEGIN;

UPDATE "Medicines" SET "Medicine\_name" = 'AnotherName' WHERE "Medicine\_id" = 1;

COMMIT;

Вікно 1:

SELECT "Medicine\_name" FROM "Medicines" WHERE "Medicine\_id" = 1;

Вивід все ще буде оригінальним ім'ям, а не 'AnotherName'

COMMIT;

Незважаючи на зміну у Вікні 2, Вікно 1 продовжує бачити оригінальні дані до завершення своєї транзакції.

3. SERIALIZABLE

Це найстрогіший рівень, який повністю ізолює транзакцію від будь-якої іншої паралельної транзакції.

Вікно 1:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

DELETE FROM "Medicines" WHERE "Medicine\_id" = 1;

Не фіксуємо

Вікно 2:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

INSERT INTO "Medicines" ("Medicine\_id", "Manufacturer\_id", "Specs\_id", "Category\_id", "Treats\_id", "Contradicts\_id", "Medicine\_name")

VALUES (1, 10, 20, 30, 40, 50, 'NewMedicine');

Ця операція буде заблокована або не вдасться через незафіксоване видалення у Вікні 1

COMMIT;

Вікно 1:

COMMIT;

INSERT у Вікні 2 буде заблокованим або відхилено через конфлікт із незафіксованим DELETE у Вікні 1.

**Висновки:**

- READ COMMITTED: Для випадків, де важлива продуктивність із допустимою можливістю змін даних між запитами в одній транзакції.

- REPEATABLE READ: Для сценаріїв, коли необхідна стабільність даних під час всієї транзакції.

- SERIALIZABLE: Найвищий рівень ізоляції для сценаріїв, де потрібна повна узгодженість, навіть якщо це зменшує конкурентоспроможність.