

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

**Лабораторна робота №2**

**з дисципліни**

**«Бази даних та засоби управління»**

**Назва: “Засоби оптимізації роботи СУБД PostgreSQL”**

Виконав студент групи: КВ-12

ПІБ: Сілін І.Д.

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Київ 2023**

*Метою роботи* є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

| *№ варіанта* | *Види індексів* | *Умови для тригера* |
| --- | --- | --- |
| *21* | *Btree, Hash* | *before delete, update* |

*Варіант:* **Система управління персоналом компанії.**

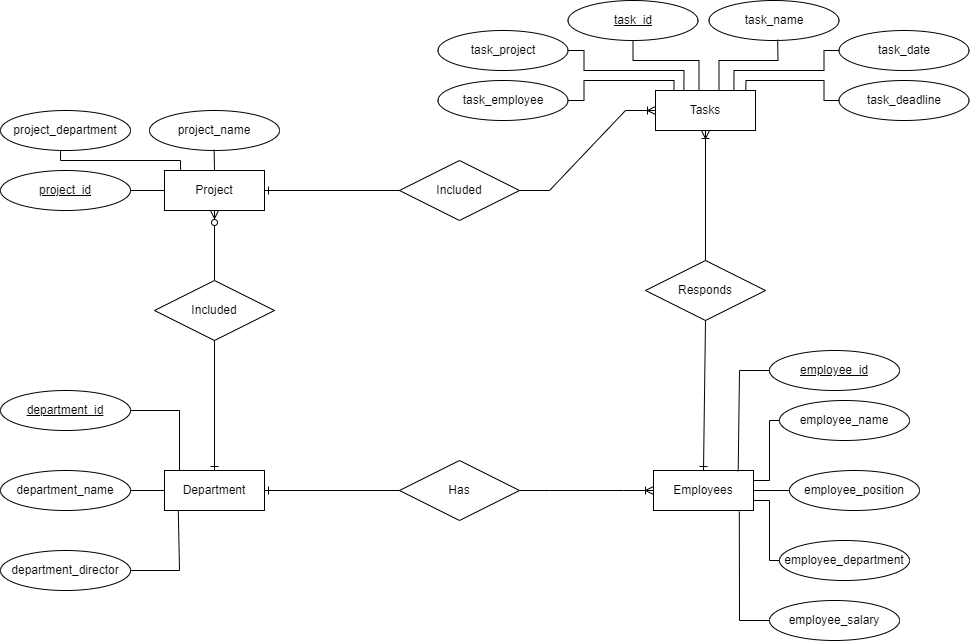
*Репозиторій:* [*https://github.com/canco32/CPMS-ORM*](https://github.com/canco32/CPMS-ORM)

*Контакт у Телеграм:* [*@IicancoI*](https://t.me/IicancoI)

*Мова розробки: PHP 8.1.0*

*Бібліотеки: Bootstrap, Faker, jQuery, Propel.*

Розробка моделі «сутність-зв’язок» предметної галузі бази даних «Система управління персоналом компанії».



*Рис. 1. ER-діаграма побудована за нотацією* «Crow’s foot».

**Сутності з описом та призначенням**

Модель предметної галузі «сутність-зв’язок» бази даних «Система управління персоналом компанії» включає у себе 4 сутності, які містять в собі від 3 до 6 атрибутів:

1. Department (department\_id, department\_name, department\_director).
2. Employees (employee\_id, employee\_name, employee\_position, employee\_department, employee\_salary).
3. Tasks (task\_id, task\_name, task\_project, task\_employee, task\_date, task\_deadline).
4. Project (project\_id, project\_name, project\_department).

Сутність «Department» описує відділ, до якого належать співробітники компанії. «Department» містить атрибути, які описують унікальний номер відділу, назву відділу, а також його керівника.

Сутність «Employees» описує співробітників компанії. «Employees» містить атрибути, які описують унікальний номер співробітника, ім’я співробітника, його посаду, номер відділу, а також його заробітну плату.

Сутність «Tasks» описує завдання, що отримали співробітники компанії. «Tasks» містить атрибути, які описують унікальний номер завдання, унікальний номер співробітника, унікальний номер проекту, до якого належить завдання, назву задачі, а також початок та дедлайн завдання.

Сутність «Project» описує проекти, що існують у компанії. «Project» містить атрибути, які описують унікальний номер завдання, унікальний номер проекту, назву проекту, а також відділ, який займається цим проектом.

**Зв’язки між сутностями**

*Сутність «Department» та «Employees»:*

* Кожен співробітник компанії може належати лише до одного відділу. Тим часом, як відділ може мати 1-го та більше співробітників. Тому зв’язок між сутностями «Department» та «Employees» є **1:N**.

*Сутність «Employees» та* «Tasks»*:*

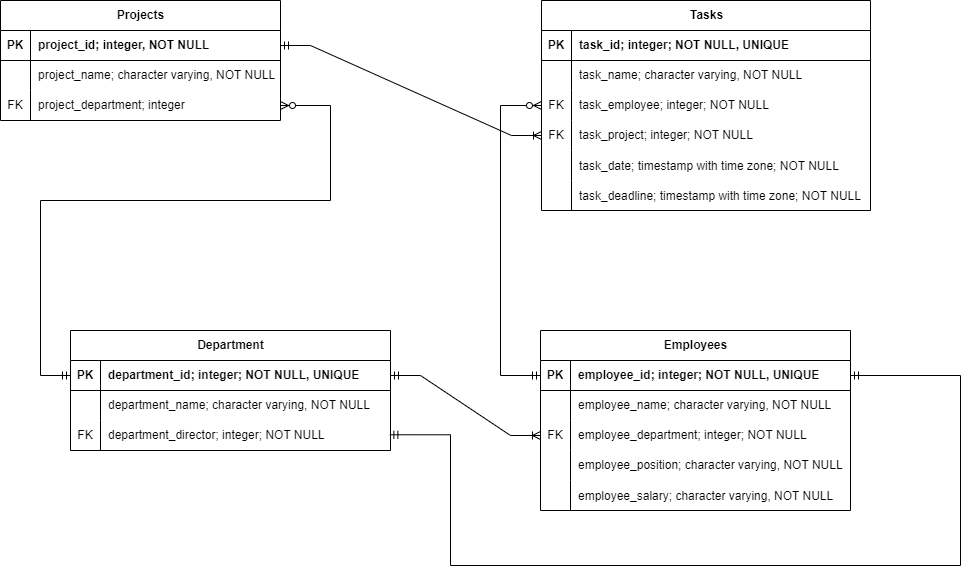
* Зазвичай у компаніях на кожного співробітника виділяється щонайменше одне, а частіше декілька завдань. Тому кожен співробітник може належати до декількох завдань, але кожне завдання належить лише одному відповідному співробітнику. Тому зв’язок між сутностями «Employees» та «Tasks» є **1:N**.

*Сутність «Tasks» та «Project»:*

* Кожне завдання може належати одному проекту. Кількість завдань, що можуть бути у проекті необмежена. Звідси, кожному завданню належить лише один проект, а кожному проекту безліч завдань. Тому зв’язок між сутностями «Tasks» та «Project»є **1:N**.

*Сутність «Project» та «Department»:*

* Кожному проекту компанії належить лише один підпорядкований відділ, тим часом як відділу може належати багато проектів або жодного. Тому сутність між «Project» та «Department»є **1:N**.



*Рис. 2. Логічна модель бази даних.*

**Опис проекту**

Проект побудований на серверні мові програмування PHP та включає у себе реалізацію побудови бази даних для задачі керування компанією, а саме керуванням персоналом та їх задачами та проектами.

Для запуску проекту знадобиться віртуальний локальний сервер PostgreSQL, а також локальний сервер MAMP або XAMPP. Також необхідне підключення до інтернету, адже у програмі використовуються підключення до CDN серверів для надання необхідних бібліотек.

У проекті використовується бібліотека jQuery для підключення стилів за допомогою Bootstrap. Цей пакет бібліотек необхідний для придання проекту приємного зовнішнього вигляду. Також використовується PHP бібліотека Faker з українською локалізацією. Призначення бібліотеки полягає у створенні випадкових даних для заповнення правдоподібним вмістом таблиці бази даних.

Програма включає у себе 4 основні файли з розширенням \*.php, а саме: «index.php», «model.php», «view.php, «controller.php». А також директорію «vendor» та файли «composer.json» і «composer.lock», які призначені для використання у проекті бібліотеки генерації випадкових даних Faker.

**Детальніше про програмні модулі управління програмою:**

*Програмний модуль Model:* У цьому модулі ми використовуємо підключення до бази даних за допомогою об’єктно-реляційного відображення для виконання операцій додавання, оновлення та вилучення записів. ORM бібліотека, яка використовується у цьому модулі має назву Propel.

*Програмний модуль View:* У цьому модулі виконуються основні дії над таблицею, адже всі дії пов’язані з вводом та виводом таблиці. Маємо функцію, що виводить список усіх таблиць, функції перегляду даних усієї таблиці або певної кількості елементів. Також є функція генерації та обробки форм для заповнення, змінення або видалення таблиць.

*Програмний модуль Controller:* Цей модуль організовує зв’язок між класами інших модулів, а також створює та призначає дії для органів керування програмою.

**Опис функціоналу програми. Зв’язок з базою даних**

Зв’язок з базою даних у програмі реалізовано двома способами:

1. За допомогою ORM Propel
2. Традиційним методом SQL підключення.

Для виконання операцій вставки, вилучення та оновлення даних за умовою використовується ORM бібліотека Propel. Клас DataBaseActions призначений для виконання SQL запитів, що реалізовані за допомогою об’єктно-реляційної проекції.

Конфігураційний файл, що відповідає за налаштування підключення до PostgreSQL:

**<?php**

$serviceContainer = \Propel\Runtime\Propel::getServiceContainer();

$serviceContainer->checkVersion(2);

$serviceContainer->setAdapterClass('default', 'pgsql');

$manager = **new** \Propel\Runtime\Connection\ConnectionManagerSingle('default');

$manager->setConfiguration(array (

'dsn' => 'pgsql:host=localhost;port=5432;dbname=postgres',

'user' => 'postgres',

'password' => '1',

'settings' =>

array (

'charset' => 'utf8',

'queries' =>

array (

),

),

'classname' => '\\Propel\\Runtime\\Connection\\ConnectionWrapper',

'model\_paths' =>

array (

0 => 'src',

1 => 'vendor',

),

));

$serviceContainer->setConnectionManager($manager);

$serviceContainer->setDefaultDatasource('default');

require\_once \_\_DIR\_\_ . '\./loadDatabase.php';

Файл для підключення Логеру помилок та ініціалізації ORM index.php:

**<?php**

spl\_autoload\_register(**function** ($class) {

require\_once ('generated-classes/' . $class . '.php');

});

require\_once ('vendor/autoload.php');

require\_once ('generated-conf/config.php');

**use** Propel\Runtime\Propel;

**use** Monolog\Logger;

**use** Monolog\Handler\StreamHandler;

$defaultLogger = **new** Logger('defaultLogger');

$defaultLogger->pushHandler(**new** StreamHandler('/var/log/propel.log', Logger::WARNING));

$defaultLogger->pushHandler(**new** \Monolog\Handler\StreamHandler('php://output', Logger::DEBUG));

$serviceContainer->setLogger('defaultLogger', $defaultLogger);

require\_once ('controller.php');

**?>**

Для виконання операцій генерації випадкових даних, каскадного очищення, видалення таблиці та операціям з індексами використовується традиційне підключення, яке встановлюється за допомогою PDO та належить класу PostgreSQLConnection:

**public** **function** \_\_construct() {

try {

$dsn = "pgsql:host=**{$this->host}**;port=**{$this->port}**;dbname=**{$this->database}**";

$this->connection = **new** PDO($dsn, $this->username, $this->password);

$this->connection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

} catch (PDOException $e) {

echo "Помилка підключення: " . $e->getMessage();

}

}

**Опис функціоналу програми. Управління даними. Вилучення, Додавання та Редагування даних**

Для кожного елементу таблиці наявні кнопки управління - *“Редагувати”* та *“Видалити”*. Користувач може змінювати зміст таблиці цими кнопками з урахуванням зовнішнього ключа.

Також маємо 5 кнопок над таблицею:

* *“Усі таблиці”* - повернення на головний екран.
* *“Додати запис”* - додати новий запис до таблиці.
* *“Видалити таблицю”* - видалення таблиці з урахуванням зовнішніх ключів.
* *“Очистити каскадно”* - каскадне очищення поточної таблиці разом з пов’язаними таблицями.
* *“Випадкові поля”* - додавання рандомізованих полів у кількості до 100.000 засобами SQL та бібліотеки Faker.

Усі дії над записами таблиці так само виконуються за допомогою відповідних GET-параметрів, а саме: $\_GET['edit\_table'] та $\_GET['delete'] для вилучення запису з таблиці; $\_GET['add\_note'] для додавання запису до таблиці; $\_GET['edit\_table'] та $\_GET['edit'] для додавання запису до таблиці.

**Опис функціоналу програми. Операції вставки, видалення та оновлення за допомогою ORM.**

Операції оновлення, видалення та вставки даних реалізовані за допомогою об’єктно-реляційного відображення у класі DataBaseActions. Перед використанням необхідно ініціалізувати усі таблиці, що є у проекті:

**public** **function** \_\_construct() {

$this->Employees = **new** Employees();

$this->Department = **new** Department();

$this->Tasks = **new** Tasks();

$this->Project = **new** Project();

}

Після ініціалізації можемо виконати операцію вставки запису до таблиці:

**public** **function** dataNoteInsert($table\_name, $data) {

switch($table\_name) {

case "employees":

try {

$this->Employees->setEmployeeName($data['EmployeeName']);

$this->Employees->setEmployeePosition($data['EmployeePosition']);

$this->Employees->setEmployeeDepartment($data['EmployeeDepartment']);

$this->Employees->setEmployeeSalary($data['EmployeeSalary']);

$this->Employees->save();

echo "Запис успішно додано.";

} catch (Exception $e) {

echo "Помилка при додаванні запису: " . $e->getMessage();

}

break;

case "tasks":

try {

$this->Tasks->setTaskName($data['TaskName']);

$this->Tasks->setTaskProject($data['TaskProject']);

$this->Tasks->setTaskEmployee($data['TaskEmployee']);

$this->Tasks->setTaskDate($data['TaskDate']);

$this->Tasks->setTaskDeadline($data['TaskDeadline']);

$this->Tasks->save();

echo "Запис успішно додано.";

} catch (Exception $e) {

echo "Помилка при додаванні запису: " . $e->getMessage();

}

break;

case "department":

try {

$this->Department->setDepartmentName($data['DepartmentName']);

$this->Department->save();

echo "Запис успішно додано.";

} catch (Exception $e) {

echo "Помилка при додаванні запису: " . $e->getMessage();

}

break;

case "project":

try {

$this->Project->setProjectName($data['ProjectName']);

$this->Project->setProjectDepartment($data['ProjectDepartment']);

$this->Project->save();

echo "Запис успішно додано.";

} catch (Exception $e) {

echo "Помилка при додаванні запису: " . $e->getMessage();

}

break;

default:

echo "Сталася невідома помилка, спробуйте ще раз!";

break;

}

}

Для кожної таблиці виконується вставка за допомогою методів класів, які генеруються для кожної таблиці автоматично за допомогою бібліотеки Propel.

Видалення запису з таблиці є динамічним і не потребує конструкції switch:

**public** **function** dataNoteDelete($table\_name, $id) {

try {

$modelClass = "**\\**Base**\\{$table\_name}**Query";

$record = $modelClass::create()->findPk($id);

if ($record !== **null**) {

$record->delete();

echo "Запис успішно видалено.";

} else {

echo "Запис з вказаним ID не знайдено.";

}

} catch (Exception $e) {

echo "Помилка при видаленні запису: " . $e->getMessage();

}

}

Оновлення вмісту запису таблиці:

**public** **function** dataNoteUpdate($table\_name, $id, $data) {

try {

$modelClass = "**\\**Base**\\{$table\_name}**Query";

$record = $modelClass::create()->findPk($id);

if ($record !== **null**) {

foreach ($data as $column => $value) {

$setterMethod = 'set' . ucfirst($column);

if (method\_exists($record, $setterMethod)) {

$record->$setterMethod(reset($value));

}

}

$record->save();

echo "Запис успішно оновлений.";

} else {

echo "Запис з вказаним ID не знайдено.";

}

} catch (Exception $e) {

echo "Помилка при оновленні запису: " . $e->getMessage();

}

}

Для виконання дій, представлених у функціях оновлення та видалення даних використовуються додаткові функції, які також реалізовані за допомогою методів об’єктно-реляційної проекції бібліотеки Propel.

Функція, що призначена для отримання усіх назв наявних у базі таблиць:

**public** **function** getAllTables() {

$databaseMap = Propel\Runtime\Propel::getDatabaseMap();

$tables = $databaseMap->getTables();

return $tables;

}

Функція, що призначена для отримання кількості записів у таблиці:

**public** **function** countRows($table\_name) {

$queryClass = "**\\**Base**\\{$table\_name}**Query";

$rowCount = $queryClass::create()->count();

return $rowCount;

}

Також за допомогою ORM у проекті можна отримати вміст таблиць та окремих записів наступними функціями:

**public** **function** getNecessaryTable($table\_name, $note\_id = **null**, $primary\_key = **null**) {

$tableClass = "**\\**Base**\\{$table\_name}**Query";

$query = $tableClass::create();

if ($note\_id !== **null** && $primary\_key !== **null**) {

$query->filterBy($primary\_key, $note\_id);

}

$table\_content = $query->find();

return $table\_content->toArray();

}

**public** **function** getRecordById($table\_name, $note\_id) {

$queryClass = "**\\{$table\_name}**Query";

$record = $queryClass::create()->findPk($note\_id);

return $record;

}

**public** **function** getColumnsContentById($table\_name, $note\_id) {

$record = $this->getRecordById($table\_name, $note\_id);

$columns\_content = [];

if ($record !== **null**) {

foreach ($record->toArray() as $column\_name => $column\_value) {

$columns\_content[] = [

'column\_name' => $column\_name,

'column\_value' => $column\_value,

];

}

}

return $columns\_content;

}

**Опис функціоналу програми. Індекси.**

У програмі використовуються індекси типу Btree та Hash. Ці індекси реалізовані у класі DataBaseActionsPDO та вже є підготовленими. Функція BtreeRequest представляє створення Btree індексу, а також його виконання для двох випадків:

**public** **function** BtreeRequest($num) {

$pdo = $this->dbConnection->getConnection();

$result = "";

switch($num) {

case 1:

*# Btree - ТОП-5 найбільших заробітних плат*

try {

$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_employee\_salary\_btree ON employees (employee\_salary DESC);";

$pdo->exec($sqlCreateIndex);

$sqlTopSalaries = "SELECT employee\_id, employee\_name, employee\_salary

FROM employees

ORDER BY employee\_salary DESC

LIMIT 5;";

$stmt = $pdo->query($sqlTopSalaries);

$result\_request = $stmt->fetchAll(PDO::FETCH\_ASSOC);

$i = 1;

foreach ($result\_request as $row) {

$result .= "#**{$i}** - (№**{$row['employee\_id']}**) **{$row['employee\_name']}**, **{$row['employee\_salary']}** <br>";

$i++;

}

} catch (PDOException $e) {

$result = "<br>Помилка (# Btree - ТОП-5 найбільших заробітніх плат): " . $e->getMessage();

}

break;

case 2:

*# Btree - ТОП-5 департаментів з найбільшою кількістю співробітників*

try {

$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_employee\_department\_btree ON employees (employee\_department);";

$pdo->exec($sqlCreateIndex);

$sqlTopDepartments = "SELECT employee\_department, COUNT(\*) as employee\_count

FROM employees

GROUP BY employee\_department

ORDER BY employee\_count DESC

LIMIT 5;";

$stmt = $pdo->query($sqlTopDepartments);

$result\_request = $stmt->fetchAll(PDO::FETCH\_ASSOC);

$i = 1;

foreach ($result\_request as $row) {

$result .= "#**{$i}** - Департамент №**{$row['employee\_department']}** - кількість співробітників: **{$row['employee\_count']}** <br>";

$i++;

}

} catch (PDOException $e) {

$result = "<br>Помилка (ТОП-5 департаментів): " . $e->getMessage();

}

break;

default:

$result = "<br>Помилка виконання Btree запиту";

break;

}

return $result;

}

Функція HashRequest представляє у собі створення Hash індексу та виконання двох підготовлених запитів, так само як це виконано у BtreeRequest:

**public** **function** HashRequest($num) {

$pdo = $this->dbConnection->getConnection();

switch($num) {

case 1:

*# Hash - Пошук кількості усіх співробітників з зарплатньою $450*

try {

$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_salary\_hash ON employees USING HASH (employee\_salary);";

$pdo->exec($sqlCreateIndex);

$targetSalary = '$450';

$sqlEmployeeCount = "SELECT COUNT(\*) as employee\_count

FROM employees

WHERE employee\_salary = :targetSalary;";

$stmt = $pdo->prepare($sqlEmployeeCount);

$stmt->bindParam(':targetSalary', $targetSalary, PDO::PARAM\_INT);

$stmt->execute();

$result = "Співробітників - ". $stmt->fetch(PDO::FETCH\_ASSOC)['employee\_count'] . "<br>";

} catch (PDOException $e) {

$result = "<br>Помилка (Пошук кількості співробітників із зарплатою **$450**): " . $e->getMessage();

}

break;

case 2:

*# Hash - Пошук кількості усіх співробітників з департамента №412*

try {

$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_department\_hash ON employees USING HASH (employee\_department);";

$pdo->exec($sqlCreateIndex);

$targetDepartment = 412;

$sqlEmployeeCount = "SELECT COUNT(\*) as employee\_count

FROM employees

WHERE employee\_department = :targetDepartment;";

$stmt = $pdo->prepare($sqlEmployeeCount);

$stmt->bindParam(':targetDepartment', $targetDepartment, PDO::PARAM\_INT);

$stmt->execute();

$result = "Співробітників - ". $stmt->fetch(PDO::FETCH\_ASSOC)['employee\_count'] . "<br>";

} catch (PDOException $e) {

$result = "<br>Помилка (Пошук кількості співробітників з департамента №412): " . $e->getMessage();

}

break;

default:

$result = "<br>Помилка виконання Hash запиту";

break;

}

return $result;

}

Виклик цих функцій здійснюється в головному меню за допомогою функції модуля View indexRequests:

**public** **function** indexRequests() {

echo "<h2>Індексні запити</h2><br>";

$measureTime = **function** ($callback) {

$start = microtime(**true**);

$callback();

$end = microtime(**true**);

return ($end - $start) \* 1000;

};

echo "# Btree - ТОП-5 найбільших заробітніх плат:<br>";

$time1 = $measureTime(**function** () {

echo $this->dbDefault->BtreeRequest(1);

});

echo "<b>Час виконання запиту: **{$time1}** мс</b><br><br>";

echo "# Btree - ТОП-5 департаментів з найбільшою кількістю співробітників:<br>";

$time2 = $measureTime(**function** () {

echo $this->dbDefault->BtreeRequest(2);

});

echo "<b>Час виконання запиту: **{$time2}** мс</b><br><br>";

echo "# Hash - Пошук кількості усіх співробітників з зарплатньою **$450**:<br>";

$time3 = $measureTime(**function** () {

echo $this->dbDefault->HashRequest(1);

});

echo "<b>Час виконання запиту: **{$time3}** мс</b><br><br>";

echo "# Hash - Пошук кількості усіх співробітників з департамента №412:<br>";

$time4 = $measureTime(**function** () {

echo $this->dbDefault->HashRequest(2);

});

echo "<b>Час виконання запиту: **{$time4}** мс</b><br><br>";

}

Ця функція викликає покроково усі індексні запити, а також робить їх швидкісні заміри.

Btree індекси швидко оптимізують запити з діапазоном і точним порівнянням, впорядковуючи ключі. Хеш-індекси ефективні для точного збігу, але не підходять для діапазонів чи сортування.

**Опис функціоналу програми. Тригери.**

У проекті використовується тригер на оновлення та перед видаленням запису з таблиці employees. Для реалізації тригеру розроблена SQL функція, яка реагує на тригер before\_delete\_update\_employees.

SQL код, який створює тригерну функцію:

*-- FUNCTION: public.before\_delete\_update\_employees()*

*-- DROP FUNCTION IF EXISTS public.before\_delete\_update\_employees();*

**CREATE** **OR** **REPLACE** **FUNCTION** public.before\_delete\_update\_employees()

**RETURNS** **TRIGGER**

**LANGUAGE** 'plpgsql'

COST 100

VOLATILE **NOT** LEAKPROOF

**AS** $BODY$

**BEGIN**

**IF** TG\_OP = 'DELETE' **THEN**

RAISE NOTICE 'Note with id = % successfully deleted', **OLD**.employee\_id;

ELSIF TG\_OP = 'UPDATE' **THEN**

RAISE NOTICE 'Note with id = % successfully updated. Old value: %, New value: %', **OLD**.employee\_id, **OLD**, **NEW**;

**END** **IF**;

**RETURN** **NEW**;

**END**;

$BODY$;

**ALTER** **FUNCTION** public.before\_delete\_update\_employees()

OWNER **TO** postgres;

Код, який створює тригер для таблиці employees:

**CREATE** **OR** **REPLACE** **TRIGGER** before\_delete\_update\_employees

**BEFORE** **DELETE** **OR** **UPDATE**

**ON** public.employees

**FOR** EACH **ROW**

**EXECUTE** **FUNCTION** public.before\_delete\_update\_employees();

За допомогою тригера можна побачити старе значення та нове, а також ідентифікаційний номер запису, з яким відбуваються дії.

На жаль, побачити результат виконання тригерної функції можна лише у випадку використання ключового слова RAISE EXCEPTION, що спричинить помилку і не дасть виконати запит, але виведе текст, що вказано у функції. Ключове слово RAISE NOTICE має виводити у вікно “Messages”повідомлення про номер запису та зміну, що відбулася. Але ця дія не відбувається навіть після переключення в режим тестування.

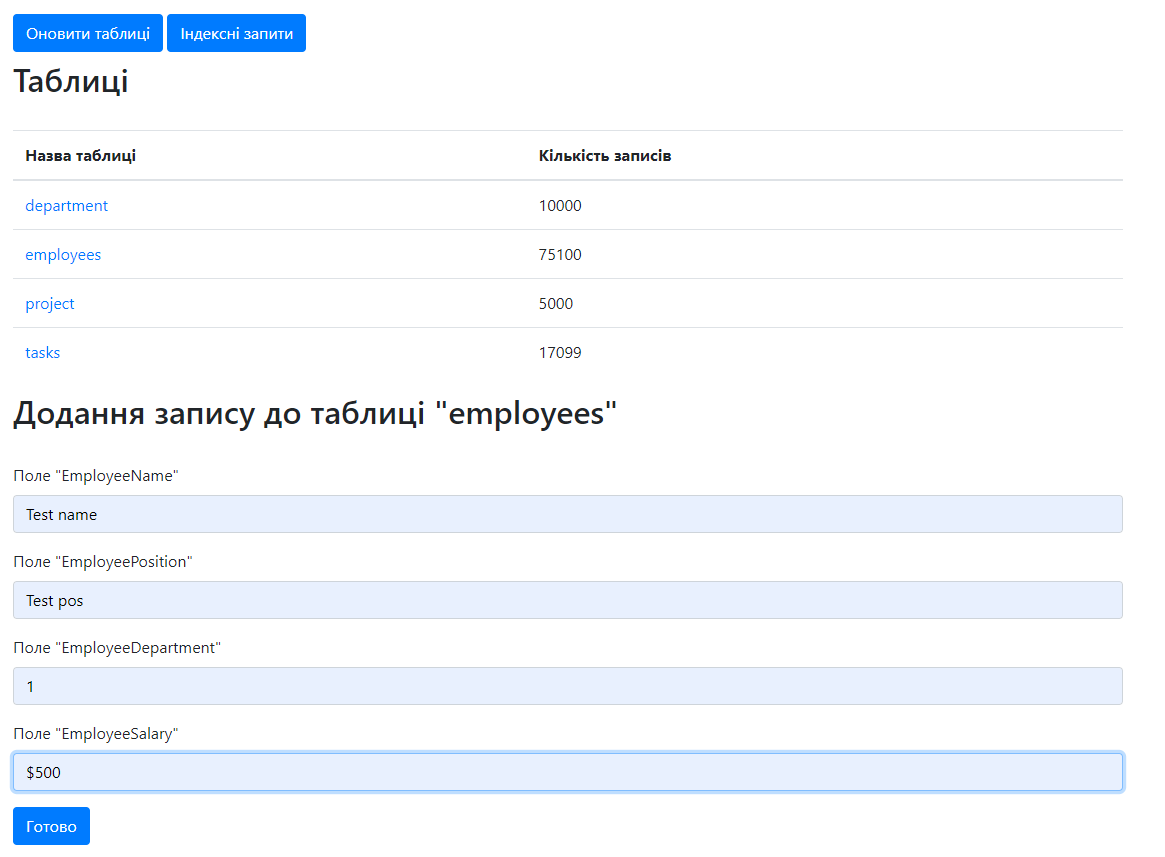
**Ізоляції транзакцій**

При READ COMMITTED може виникнути "Dirty Read" - читання незафіксованих даних, коли одна транзакція змінює дані, інша транзакція читає їх до фіксації. REPEATABLE READ усуває цей феномен, використовуючи блокування на рівні рядка. Однак, REPEATABLE READ може призвести до "Non-Repeatable Read", коли одна транзакція читає рядок, інша його модифікує, і перша транзакція читає знову, отримуючи інші дані. SERIALIZABLE вирішує цю проблему, використовуючи строгі блокування та відокремленість транзакцій. Однак вона може призвести до "Phantom Read", коли транзакція читає набір рядків, інша вставляє новий рядок, і перша транзакція читає знову, побачивши новий рядок.

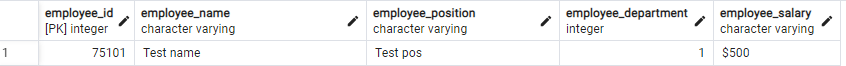
**Ілюстрації роботи програми**

****

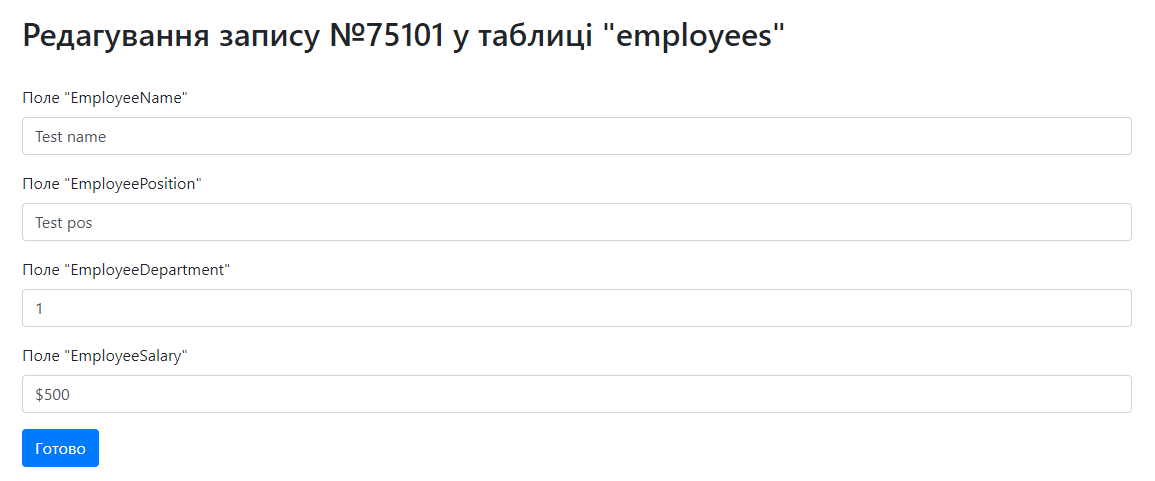
*Рис. 3. Головне меню програми з клікабельними назвами таблиць та керуючими кнопками “Оновити таблиці” та “Індексні запити”.*

**

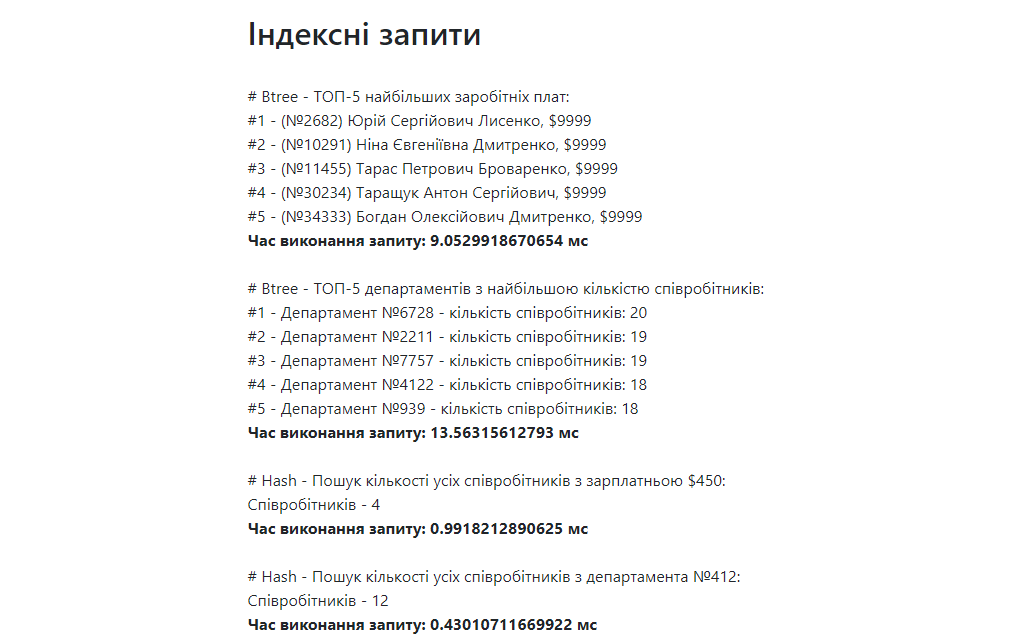
*Рис. 4. Форма додавання запису до таблиці за допомогою ORM Propel.*

**

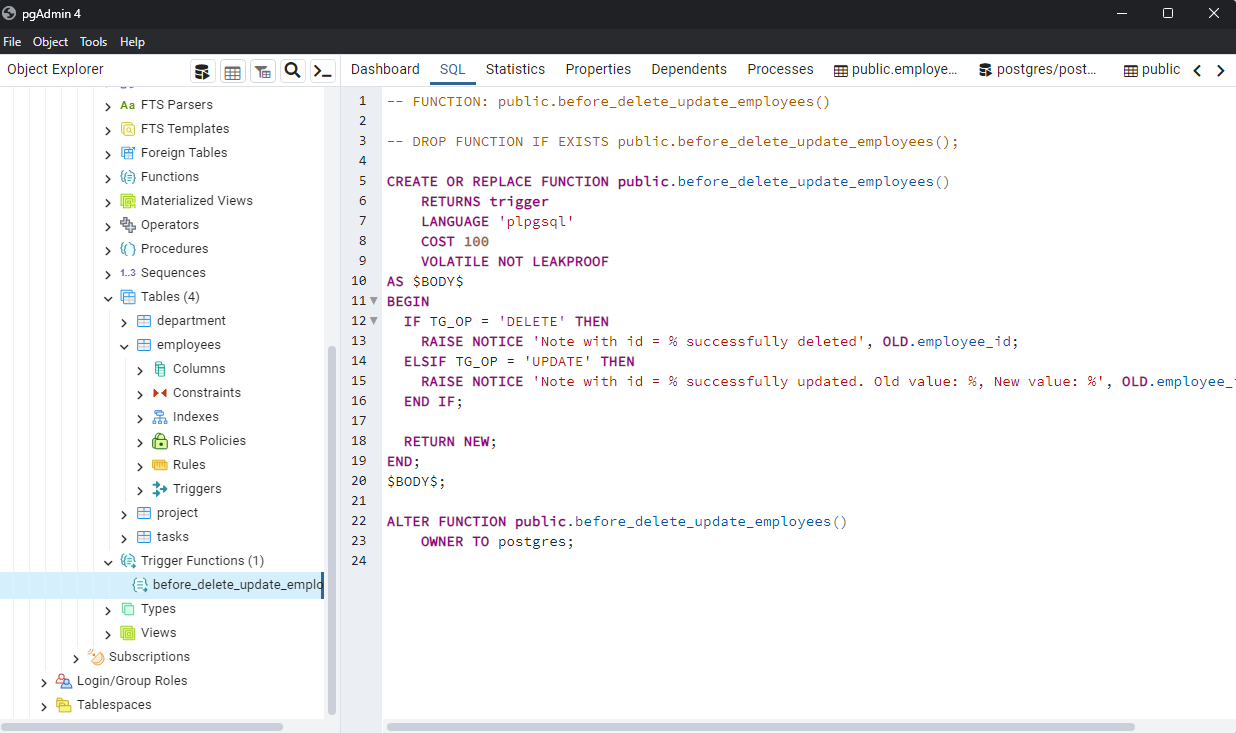
*Рис. 5. Успіх додавання запису до таблиці employees.*

**

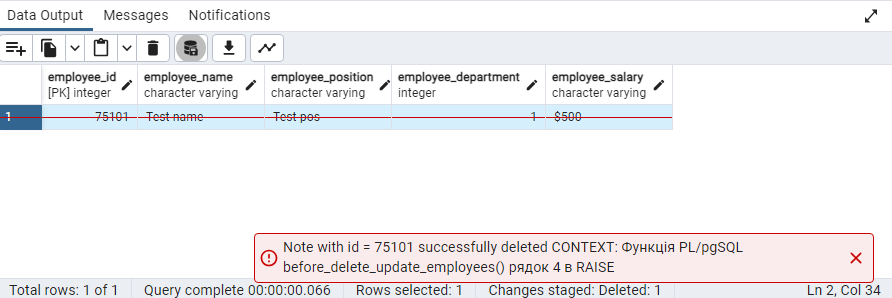
*Рис. 6. Форма редагування запису №75101.*

**

*Рис. 7. Виконання підготовлених індексних запитів.*

**

*Рис. 8. Тригерна функція.*

****

*Рис. 9. Відредагована функція для відображення повідомлення (Змінено NOTICE на EXCEPTION для демонстрації).*

**Програмний код. Model**

**<?php**

**class PostgreSQLConnection {**

**private $host = "localhost";**

**private $port = 5432;**

**private $database = "postgres";**

**private $username = "postgres";**

**private $password = "1";**

**private $connection;**

**public function \_\_construct() {**

**try {**

**$dsn = "pgsql:host={$this->host};port={$this->port};dbname={$this->database}";**

**$this->connection = new PDO($dsn, $this->username, $this->password);**

**$this->connection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);**

**} catch (PDOException $e) {**

**echo "Помилка підключення: " . $e->getMessage();**

**}**

**}**

**public function disconnect() {**

**$this->connection = null;**

**}**

**public function getConnection() {**

**return $this->connection;**

**}**

**}**

**class DataBaseActions {**

**private $Employees;**

**private $Department;**

**private $Tasks;**

**private $Project;**

**public function \_\_construct() {**

**$this->Employees = new Employees();**

**$this->Department = new Department();**

**$this->Tasks = new Tasks();**

**$this->Project = new Project();**

**}**

**public function dataNoteInsert($table\_name, $data) {**

**switch($table\_name) {**

**case "employees":**

**try {**

**$this->Employees->setEmployeeName($data['EmployeeName']);**

**$this->Employees->setEmployeePosition($data['EmployeePosition']);**

**$this->Employees->setEmployeeDepartment($data['EmployeeDepartment']);**

**$this->Employees->setEmployeeSalary($data['EmployeeSalary']);**

**$this->Employees->save();**

**echo "Запис успішно додано.";**

**} catch (Exception $e) {**

**echo "Помилка при додаванні запису: " . $e->getMessage();**

**}**

**break;**

**case "tasks":**

**try {**

**$this->Tasks->setTaskName($data['TaskName']);**

**$this->Tasks->setTaskProject($data['TaskProject']);**

**$this->Tasks->setTaskEmployee($data['TaskEmployee']);**

**$this->Tasks->setTaskDate($data['TaskDate']);**

**$this->Tasks->setTaskDeadline($data['TaskDeadline']);**

**$this->Tasks->save();**

**echo "Запис успішно додано.";**

**} catch (Exception $e) {**

**echo "Помилка при додаванні запису: " . $e->getMessage();**

**}**

**break;**

**case "department":**

**try {**

**$this->Department->setDepartmentName($data['DepartmentName']);**

**$this->Department->save();**

**echo "Запис успішно додано.";**

**} catch (Exception $e) {**

**echo "Помилка при додаванні запису: " . $e->getMessage();**

**}**

**break;**

**case "project":**

**try {**

**$this->Project->setProjectName($data['ProjectName']);**

**$this->Project->setProjectDepartment($data['ProjectDepartment']);**

**$this->Project->save();**

**echo "Запис успішно додано.";**

**} catch (Exception $e) {**

**echo "Помилка при додаванні запису: " . $e->getMessage();**

**}**

**break;**

**default:**

**echo "Сталася невідома помилка, спробуйте ще раз!";**

**break;**

**}**

**}**

**public function dataNoteDelete($table\_name, $id) {**

**try {**

**$modelClass = "\\Base\\{$table\_name}Query";**

**$record = $modelClass::create()->findPk($id);**

**if ($record !== null) {**

**$record->delete();**

**echo "Запис успішно видалено.";**

**} else {**

**echo "Запис з вказаним ID не знайдено.";**

**}**

**} catch (Exception $e) {**

**echo "Помилка при видаленні запису: " . $e->getMessage();**

**}**

**}**

**public function dataNoteUpdate($table\_name, $id, $data) {**

**try {**

**$modelClass = "\\Base\\{$table\_name}Query";**

**$record = $modelClass::create()->findPk($id);**

**if ($record !== null) {**

**foreach ($data as $column => $value) {**

**$setterMethod = 'set' . ucfirst($column);**

**if (method\_exists($record, $setterMethod)) {**

**$record->$setterMethod(reset($value));**

**}**

**}**

**$record->save();**

**echo "Запис успішно оновлений.";**

**} else {**

**echo "Запис з вказаним ID не знайдена.";**

**}**

**} catch (Exception $e) {**

**echo "Помилка при оновленні запису: " . $e->getMessage();**

**}**

**}**

**public function getAllTables() {**

**$databaseMap = Propel\Runtime\Propel::getDatabaseMap();**

**$tables = $databaseMap->getTables();**

**return $tables;**

**}**

**public function countRows($table\_name) {**

**$queryClass = "\\Base\\{$table\_name}Query";**

**$rowCount = $queryClass::create()->count();**

**return $rowCount;**

**}**

**public function getColumnsNames($table\_name, $limit = null) {**

**$tableMapClass = "\\Map\\{$table\_name}TableMap";**

**$tableMap = $tableMapClass::getTableMap();**

**$columns\_names = [];**

**foreach ($tableMap->getFieldNames() as $column) {**

**$columns\_names[] = [**

**'column\_name' => $column,**

**'data\_type' => $tableMap->getColumn($column)->getType()**

**];**

**}**

**if ($limit !== null) {**

**$columns\_names = array\_slice($columns\_names, 0, $limit);**

**}**

**return $columns\_names;**

**}**

**public function getNecessaryTable($table\_name, $note\_id = null, $primary\_key = null) {**

**$tableClass = "\\Base\\{$table\_name}Query";**

**$query = $tableClass::create();**

**if ($note\_id !== null && $primary\_key !== null) {**

**$query->filterBy($primary\_key, $note\_id);**

**}**

**$table\_content = $query->find();**

**return $table\_content->toArray();**

**}**

**public function getRecordById($table\_name, $note\_id) {**

**$queryClass = "\\{$table\_name}Query";**

**$record = $queryClass::create()->findPk($note\_id);**

**return $record;**

**}**

**public function getColumnsContentById($table\_name, $note\_id) {**

**$record = $this->getRecordById($table\_name, $note\_id);**

**$columns\_content = [];**

**if ($record !== null) {**

**foreach ($record->toArray() as $column\_name => $column\_value) {**

**$columns\_content[] = [**

**'column\_name' => $column\_name,**

**'column\_value' => $column\_value,**

**];**

**}**

**}**

**return $columns\_content;**

**}**

**}**

**class DataBaseActionsPDO {**

**private $dbConnection;**

**public function \_\_construct(PostgreSQLConnection $dbConnection) {**

**$this->dbConnection = $dbConnection;**

**}**

**public function randomFillTable($pdo, $table\_name, $form\_data) {**

**$faker = new Faker\Generator();**

**$faker->addProvider(new Faker\Provider\uk\_UA\Person($faker));**

**$faker->addProvider(new Faker\Provider\uk\_UA\Company($faker));**

**$limit = $form\_data['fillTable'];**

**$startTime = microtime(true);**

**$columns\_data = $this->getColumnsNames($pdo, $table\_name);**

**$primary\_key\_name = $columns\_data[0]['column\_name'];**

**$sql = "INSERT INTO \"$table\_name\" (";**

**foreach ($columns\_data as $column) {**

**$column\_name = $column['column\_name'];**

**if ($column\_name !== $primary\_key\_name) {**

**$sql .= "$column\_name, ";**

**}**

**}**

**$sql = rtrim($sql, ", ") . ") VALUES ";**

**for ($i = 0; $i < $limit; $i++) {**

**$sql .= "(";**

**foreach ($columns\_data as $column) {**

**$column\_name = $column['column\_name'];**

**if($column\_name !== $primary\_key\_name) {**

**switch($column\_name) {**

**case "employee\_department":**

**$sql .= "(SELECT department\_id FROM \"department\" ORDER BY RANDOM() LIMIT 1), ";**

**break;**

**case "department\_name":**

**$fake\_department = $faker->company();**

**$sql .= "('$fake\_department'),";**

**break;**

**case "employee\_name":**

**$fake\_name = $faker->name();**

**if (strpos($fake\_name, "'") !== false) {**

**$fake\_name = str\_replace("'", "`", $fake\_name);**

**}**

**$sql .= "('$fake\_name'),";**

**break;**

**case "employee\_position":**

**$fake\_position = $faker->jobTitle();**

**$sql .= "('$fake\_position'),";**

**break;**

**case "task\_employee":**

**$sql .= "(SELECT employee\_id FROM \"employees\" ORDER BY RANDOM() LIMIT 1), ";**

**break;**

**case "task\_project":**

**$sql .= "(SELECT project\_id FROM \"project\" ORDER BY RANDOM() LIMIT 1), ";**

**break;**

**case "project\_department":**

**$sql .= "(SELECT department\_id FROM \"department\" ORDER BY RANDOM() LIMIT 1), ";**

**break;**

**case "task\_date":**

**$sql .= "**

**(SELECT**

**date\_trunc('day', CURRENT\_DATE - INTERVAL '1 month') +**

**random() \* (CURRENT\_DATE - date\_trunc('day', CURRENT\_DATE - INTERVAL '1 month')) AS random\_date**

**),**

**";**

**break;**

**case "task\_deadline":**

**$sql .= "**

**(SELECT**

**CURRENT\_DATE +**

**random() \* INTERVAL '6 months' AS random\_date**

**)**

**";**

**break;**

**case "employee\_salary":**

**$sql .= "('$' || floor(random() \* (10000 - 300 + 1) + 300)::int), ";**

**break;**

**default:**

**$sql .= $this->generateRandomSqlValue($column) . ", ";**

**break;**

**}**

**}**

**}**

**$sql = rtrim($sql, ", ") . "), ";**

**}**

**$sql = rtrim($sql, ", ") . ";";**

**$stmt = $pdo->prepare($sql);**

**$stmt->execute();**

**$endTime = microtime(true);**

**$executionTime = ($endTime - $startTime) \* 1000;**

**echo "Запит виконано за " . number\_format($executionTime, 2, '.', ' ') . " мс";**

**}**

**private function generateRandomSqlValue($column) {**

**switch ($column['data\_type']) {**

**case 'integer':**

**return 'floor(random() \* 100)';**

**case 'character varying':**

**return "(SELECT left(md5(random()::text), 10))";**

**default:**

**return 'NULL';**

**}**

**}**

**private function getColumnsNames(PDO $pdo, $table\_name, $limit = NULL) {**

**if($limit == NULL) {**

**$sql = "SELECT column\_name, data\_type FROM information\_schema.columns WHERE table\_name = '$table\_name' ORDER BY ordinal\_position";**

**} else {**

**$sql = "SELECT column\_name, data\_type FROM information\_schema.columns WHERE table\_name = '$table\_name' ORDER BY ordinal\_position LIMIT $limit";**

**}**

**$stmt = $pdo->prepare($sql);**

**$stmt->execute();**

**$columns\_names = $stmt->fetchAll(PDO::FETCH\_ASSOC);**

**return $columns\_names;**

**}**

**public function dataTableDelete($table\_name) {**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Видалення таблиці \"$table\_sanitized\"</h2><br>";**

**try {**

**$pdo = $this->dbConnection->getConnection();**

**if ($this->checkForeignKey($table\_sanitized)) {**

**$result .= "<p>Таблиця містить зовнішній ключ. Видаліть зовнішній ключ перед видаленням таблиці.</p>";**

**} else {**

**$sql = $this->generateDeleteTableSQL($table\_sanitized);**

**$pdo->exec($sql);**

**$result .= "<p>Таблиця \"$table\_sanitized\" успішно видалено.</p>";**

**}**

**} catch (PDOException $e) {**

**$result .= "<br>Помилка виконання запиту: " . $e->getMessage();**

**}**

**return $result;**

**}**

**public function cascadeTruncate($table\_name) {**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Каскадне очищення таблиці \"$table\_sanitized\"</h2><br>";**

**try {**

**$pdo = $this->dbConnection->getConnection();**

**$sql = $this->generateTruncateTableSQL($table\_sanitized);**

**$pdo->exec($sql);**

**$result .= "<p>Таблиця \"$table\_sanitized\" та усі залежні від неї таблиці каскадно очищені.</p>";**

**} catch (PDOException $e) {**

**$result .= "<br>Помилка каскадного видалення: " . $e->getMessage();**

**}**

**return $result;**

**}**

**public function BtreeRequest($num) {**

**$pdo = $this->dbConnection->getConnection();**

**$result = "";**

**switch($num) {**

**case 1:**

***# Btree - ТОП-5 найбільших заробітніх плат***

**try {**

**$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_employee\_salary\_btree ON employees (employee\_salary DESC);";**

**$pdo->exec($sqlCreateIndex);**

**$sqlTopSalaries = "SELECT employee\_id, employee\_name, employee\_salary**

**FROM employees**

**ORDER BY employee\_salary DESC**

**LIMIT 5;";**

**$stmt = $pdo->query($sqlTopSalaries);**

**$result\_request = $stmt->fetchAll(PDO::FETCH\_ASSOC);**

**$i = 1;**

**foreach ($result\_request as $row) {**

**$result .= "#{$i} - (№{$row['employee\_id']}) {$row['employee\_name']}, {$row['employee\_salary']} <br>";**

**$i++;**

**}**

**} catch (PDOException $e) {**

**$result = "<br>Помилка (# Btree - ТОП-5 найбільших заробітніх плат): " . $e->getMessage();**

**}**

**break;**

**case 2:**

***# Btree - ТОП-5 департаментів з найбільшою кількістю співробітників***

**try {**

**$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_employee\_department\_btree ON employees (employee\_department);";**

**$pdo->exec($sqlCreateIndex);**

**$sqlTopDepartments = "SELECT employee\_department, COUNT(\*) as employee\_count**

**FROM employees**

**GROUP BY employee\_department**

**ORDER BY employee\_count DESC**

**LIMIT 5;";**

**$stmt = $pdo->query($sqlTopDepartments);**

**$result\_request = $stmt->fetchAll(PDO::FETCH\_ASSOC);**

**$i = 1;**

**foreach ($result\_request as $row) {**

**$result .= "#{$i} - Департамент №{$row['employee\_department']} - кількість співробітників: {$row['employee\_count']} <br>";**

**$i++;**

**}**

**} catch (PDOException $e) {**

**$result = "<br>Помилка (ТОП-5 департаментів): " . $e->getMessage();**

**}**

**break;**

**default:**

**$result = "<br>Помилка виконання Btree запиту";**

**break;**

**}**

**return $result;**

**}**

**public function HashRequest($num) {**

**$pdo = $this->dbConnection->getConnection();**

**switch($num) {**

**case 1:**

***# Hash - Пошук кількості усіх співробітників з зарплатньою $450***

**try {**

**$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_salary\_hash ON employees USING HASH (employee\_salary);";**

**$pdo->exec($sqlCreateIndex);**

**$targetSalary = '$450';**

**$sqlEmployeeCount = "SELECT COUNT(\*) as employee\_count**

**FROM employees**

**WHERE employee\_salary = :targetSalary;";**

**$stmt = $pdo->prepare($sqlEmployeeCount);**

**$stmt->bindParam(':targetSalary', $targetSalary, PDO::PARAM\_INT);**

**$stmt->execute();**

**$result = "Співробітників - ". $stmt->fetch(PDO::FETCH\_ASSOC)['employee\_count'] . "<br>";**

**} catch (PDOException $e) {**

**$result = "<br>Помилка (Пошук кількості співробітників із зарплатою $450): " . $e->getMessage();**

**}**

**break;**

**case 2:**

***# Hash - Пошук кількості усіх співробітників з департамента №412***

**try {**

**$sqlCreateIndex = "CREATE INDEX IF NOT EXISTS idx\_department\_hash ON employees USING HASH (employee\_department);";**

**$pdo->exec($sqlCreateIndex);**

**$targetDepartment = 412;**

**$sqlEmployeeCount = "SELECT COUNT(\*) as employee\_count**

**FROM employees**

**WHERE employee\_department = :targetDepartment;";**

**$stmt = $pdo->prepare($sqlEmployeeCount);**

**$stmt->bindParam(':targetDepartment', $targetDepartment, PDO::PARAM\_INT);**

**$stmt->execute();**

**$result = "Співробітників - ". $stmt->fetch(PDO::FETCH\_ASSOC)['employee\_count'] . "<br>";**

**} catch (PDOException $e) {**

**$result = "<br>Помилка (Пошук кількості співробітників з департамента №412): " . $e->getMessage();**

**}**

**break;**

**default:**

**$result = "<br>Помилка виконання Hash запиту";**

**break;**

**}**

**return $result;**

**}**

**private function checkForeignKey($table\_name) {**

**$pdo = $this->dbConnection->getConnection();**

**$sql = "SELECT COUNT(\*) FROM information\_schema.table\_constraints**

**WHERE constraint\_type = 'FOREIGN KEY' AND table\_name = :table\_name";**

**$stmt = $pdo->prepare($sql);**

**$stmt->bindParam(':table\_name', $table\_name, PDO::PARAM\_STR);**

**$stmt->execute();**

**$count = $stmt->fetchColumn();**

**return $count > 0;**

**}**

**private function generateDeleteTableSQL($table\_name) {**

**return "DROP TABLE IF EXISTS \"$table\_name\"";**

**}**

**private function generateTruncateTableSQL($table\_name) {**

**return "TRUNCATE \"$table\_name\" RESTART IDENTITY CASCADE";**

**}**

**}**

**?>**

**Програмний код. View**

**<?php**

**class Viewer {**

**private $dbConnection;**

**private $dbPropel;**

**private $dbDefault;**

**public function \_\_construct(PostgreSQLConnection $dbConnection) {**

**$this->dbConnection = $dbConnection;**

**$this->dbPropel = new DataBaseActions($dbConnection);**

**$this->dbDefault = new DataBaseActionsPDO($dbConnection);**

**}**

**public function displayTables($table\_name = NULL) {**

**$result = "";**

**if ($table\_name == NULL) {**

**$result = "**

**<h2>Таблиці</h2><br>**

**<table class=\"table\">**

**<thead>**

**<tr>**

**<th>Назва таблиці</th>**

**<th>Кількість записів</th>**

**</tr>**

**</thead>**

**<tbody>**

**";**

**$tables = $this->dbPropel->getAllTables();**

**foreach ($tables as $table) {**

**$result .= "<tr>";**

**$result .= "<td><a href=\"index.php?view\_table={$table->getName()}\">{$table->getName()}</a></td>";**

**$result .= "<td>{$this->dbPropel->countRows($table->getName())}</td>";**

**$result .= "</tr>";**

**}**

**$result .= " </tbody>**

**</table>**

**";**

**} else {**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>$table\_sanitized</h2><br>";**

**$columns\_data = $this->dbPropel->getColumnsNames($table\_sanitized);**

**$table\_data = $this->dbPropel->getNecessaryTable($table\_sanitized);**

**$first\_column\_saved = false;**

**$data\_id = "";**

**$result .= "**

**<table class=\"table\">**

**<thead>**

**<tr>**

**";**

**foreach ($columns\_data as $column) {**

**if (!$first\_column\_saved) {**

**$data\_id = $column['column\_name'];**

**}**

**$result .= "<th>{$column['column\_name']}</th>";**

**$first\_column\_saved = true;**

**}**

**$result .= "<th style=\"text-align: center;\" colspan=\"2\">Дії</th>";**

**$result .= " </tr>**

**</thead>**

**<tbody>**

**";**

**foreach ($table\_data as $row) {**

**$result .= "<tr>";**

**foreach ($row as $cell) {**

**$result .= "<td>{$cell}</td>";**

**}**

**$result .= "<td><a href=\"index.php?edit\_table=$table\_sanitized&edit=$row[$data\_id]\">Змінити</a></td>";**

**$result .= "<td><a href=\"index.php?edit\_table=$table\_sanitized&delete=$row[$data\_id]\">Видалити</a></td>";**

**$result .= "</tr>";**

**}**

**$result .= " </tbody>**

**</table>**

**";**

**}**

**return $result;**

**}**

**public function addNote($table\_name) {**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Додання запису до таблиці \"$table\_sanitized\"</h2><br>";**

**try {**

**$pdo = $this->dbConnection->getConnection();**

**$columns\_data = $this->dbPropel->getColumnsNames($table\_sanitized);**

**$result .= "<form method=\"post\">";**

**$id = 0;**

**foreach($columns\_data as $input) {**

**if($id != 0) {**

**if (stripos($input['column\_name'], 'date') !== false || stripos($input['column\_name'], 'deadline') !== false) {**

**$result.= "**

**<div class=\"form-group\">**

**<label for=\"input$id\">Поле \"{$input['column\_name']}\"</label>**

**<input name=\"{$input['column\_name']}\" type=\"date\" class=\"form-control\" placeholder=\"Введіть значення для поля '{$input['column\_name']}'\">**

**</div>**

**";**

**} else {**

**$result.= "**

**<div class=\"form-group\">**

**<label for=\"input$id\">Поле \"{$input['column\_name']}\"</label>**

**<input name=\"{$input['column\_name']}\" type=\"text\" class=\"form-control\" placeholder=\"Введіть значення для поля '{$input['column\_name']}'\">**

**</div>**

**";**

**}**

**}**

**$id++;**

**}**

**$result.= "**

**<button type=\"submit\" class=\"btn btn-primary\">Готово</button>**

**</form>**

**";**

**if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {**

**$this->dbPropel->dataNoteInsert($table\_sanitized, $\_POST);**

**}**

**} catch (PDOException $e) {**

**echo "<br>Помилка виконання запиту: " . $e->getMessage();**

**}**

**return $result;**

**}**

**public function editTableNote($table\_name, $note\_id) {**

**$note\_id = trim(filter\_var($note\_id, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Редагування запису №$note\_id у таблиці \"$table\_sanitized\"</h2><br>";**

**try {**

**$columns\_data = $this->dbPropel->getColumnsNames($table\_sanitized);**

**$columns\_content = $this->dbPropel->getColumnsContentById($table\_sanitized, $note\_id);**

**$result .= "<form method=\"post\">";**

**foreach ($columns\_data as $input) {**

**$column\_name = $input['column\_name'];**

**$foundColumns = array\_filter($columns\_content, function ($column) use ($column\_name) {**

**return $column['column\_name'] === $column\_name;**

**});**

**foreach ($foundColumns as $index => $foundColumn) {**

**$value = isset($foundColumn['column\_value']) ? $foundColumn['column\_value'] : '';**

**if (strpos($column\_name, 'Id') === false) {**

**$result .= "**

**<div class=\"form-group\">**

**<label for=\"{$column\_name}\">Поле \"{$column\_name}\"</label>**

**<input name=\"{$column\_name}[$index]\" type=\"text\" value=\"$value\" class=\"form-control\" placeholder=\"Введіть значення для поля '{$column\_name}'\">**

**</div>**

**";**

**}**

**}**

**}**

**$result .= "**

**<button type=\"submit\" class=\"btn btn-primary\">Готово</button>**

**</form>";**

**if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {**

**$this->dbPropel->dataNoteUpdate($table\_sanitized, $note\_id, $\_POST);**

**}**

**} catch (PDOException $e) {**

**echo "<br>Помилка виконання запиту: " . $e->getMessage();**

**}**

**return $result;**

**}**

**public function deleteTableNote($table\_name, $note\_id) {**

**$note\_id = trim(filter\_var($note\_id, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Видалення запису №$note\_id з таблиці \"$table\_sanitized\"</h2><br>";**

**$this->dbPropel->dataNoteDelete($table\_sanitized, $note\_id);**

**return $result;**

**}**

**public function dataTableFill($table\_name) {**

**$table\_sanitized = trim(filter\_var($table\_name, FILTER\_SANITIZE\_SPECIAL\_CHARS));**

**$result = "<h2>Додання випадкових полів до таблиці \"$table\_sanitized\"</h2><br>";**

**$result .= "**

**<form method=\"post\">**

**<div class=\"form-group\">**

**<label for=\"input\">Введіть кількість нових випадкових полей для таблиці</label>**

**<input name=\"fillTable\" type=\"number\" min=\"1\" max=\"100000\" class=\"form-control\" placeholder=\"Введіть ціле значення від 0 до 100000 \">**

**</div>**

**<button type=\"submit\" class=\"btn btn-primary\">Створити</button>**

**</form>**

**";**

**if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {**

**$pdo = $this->dbConnection->getConnection();**

**$this->dbDefault->randomFillTable($pdo, $table\_sanitized, $\_POST);**

**}**

**return $result;**

**}**

**public function indexRequests() {**

**echo "<h2>Індексні запити</h2><br>";**

**$measureTime = function ($callback) {**

**$start = microtime(true);**

**$callback();**

**$end = microtime(true);**

**return ($end - $start) \* 1000;**

**};**

**echo "# Btree - ТОП-5 найбільших заробітніх плат:<br>";**

**$time1 = $measureTime(function () {**

**echo $this->dbDefault->BtreeRequest(1);**

**});**

**echo "<b>Час виконання запиту: {$time1} мс</b><br><br>";**

**echo "# Btree - ТОП-5 департаментів з найбільшою кількістю співробітників:<br>";**

**$time2 = $measureTime(function () {**

**echo $this->dbDefault->BtreeRequest(2);**

**});**

**echo "<b>Час виконання запиту: {$time2} мс</b><br><br>";**

**echo "# Hash - Пошук кількості усіх співробітників з зарплатньою $450:<br>";**

**$time3 = $measureTime(function () {**

**echo $this->dbDefault->HashRequest(1);**

**});**

**echo "<b>Час виконання запиту: {$time3} мс</b><br><br>";**

**echo "# Hash - Пошук кількості усіх співробітників з департамента №412:<br>";**

**$time4 = $measureTime(function () {**

**echo $this->dbDefault->HashRequest(2);**

**});**

**echo "<b>Час виконання запиту: {$time4} мс</b><br><br>";**

**}**

**}**

**?>**

**Програмний код. Controller**

**<?php**

**require\_once('model.php');**

**require\_once('view.php');**

**$connection = new PostgreSQLConnection();**

**$viewer = new Viewer($connection);**

**$database = new DataBaseActions();**

**$databasePDO = new DataBaseActionsPDO($connection);**

**?>**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">**

**<title>CPMS ORM, Silin KV-12</title>**

**</head>**

**<body>**

**<div class="container mt-4">**

**<?php if(isset($\_GET['view\_table'])): ?>**

**<a class="btn btn-primary mb-2" href="index.php">Усі таблиці</a>**

**<a class="btn btn-primary mb-2" href="index.php?add\_note=<?=$\_GET['view\_table']?>">Додати запис</a>**

**<a class="btn btn-primary mb-2" href="index.php?delete\_table=<?=$\_GET['view\_table']?>">Видалити таблицю</a>**

**<a class="btn btn-primary mb-2" href="index.php?truncate\_table=<?=$\_GET['view\_table']?>">Очистити каскадно</a>**

**<a class="btn btn-primary mb-2" href="index.php?fill\_table=<?=$\_GET['view\_table']?>">Випадкові поля</a>**

**<?= $viewer->displayTables($\_GET['view\_table']); ?>**

**<?php endif; ?>**

**<?php if(!isset($\_GET['view\_table'])): ?>**

**<a class="btn btn-primary mb-2" href="index.php">Оновити таблиці</a>**

**<a class="btn btn-primary mb-2" href="index.php?index-requests">Індексні запити</a>**

**<?= $viewer->displayTables(); ?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['add\_note'])): ?>**

**<?= $viewer->addNote($\_GET['add\_note']); ?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['edit\_table']) && isset($\_GET['edit'])): ?>**

**<?= $viewer->editTableNote($\_GET['edit\_table'], $\_GET['edit']); ?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['edit\_table']) && isset($\_GET['delete'])): ?>**

**<?= $viewer->deleteTableNote($\_GET['edit\_table'], $\_GET['delete']); ?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['delete\_table'])): ?>**

**<?= $databasePDO->dataTableDelete($\_GET['delete\_table']);?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['truncate\_table'])): ?>**

**<?= $databasePDO->cascadeTruncate($\_GET['truncate\_table']);?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['fill\_table'])): ?>**

**<?= $viewer->dataTableFill($\_GET['fill\_table']);?>**

**<?php endif; ?>**

**<?php if(isset($\_GET['index-requests'])): ?>**

**<?= $viewer->indexRequests(); ?>**

**<?php endif; ?>**

**</div>**

**<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>**

**<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.2/dist/umd/popper.min.js"></script>**

**<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>**

**<?php $connection->disconnect(); ?>**

**</body>**

**</html>**