Міністерство освіти і науки Укрвїни

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка
До кваліфікаційної роботи
магістра

на тему: «Аналіз механізмів роботи з базами даних при використанні ORM Entity Framework та запитів SQL»
за освітньою програмою інформаційні технології
зі спеціальності: 121 Інженерія програмного забезпечення.

Виконав: студент групи ПЗ2121М: /Дар'я НАЗАРЕНКО/

Керівник: /Олександр ІВАНОВ/

Нормоконтролер: /Світлана ВОЛКОВА/

Консультанти:
Економічний розділ /Микола ГНЕНИЙ/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Дніпро – 2022 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка
До кваліфікаційної роботи
магістра

на тему: «Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL»
за освітньою програмою інформаційні технології
зі спеціальності: 121 Інженерія програмного забезпечення.

Виконав: студент групи ПЗ2121М: /Дар'я НАЗАРЕНКО/

Керівник: /Олександр ІВАНОВ/

Нормоконтролер: /Світлана ВОЛКОВА/

Консультанти:
Економічний розділ /Микола ГНЕНИЙ/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note
To Master's Thesis
master

on the topic: « Comparison of database queries using ORM Entity Framework and SQL »
according to educational curriculum information technologies
in the Speciality: 121 software engineering.

Done by the student of the group ПЗ2121М: /Daria NAZARENKO/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Svitlana VOLKOVA/

Supervisors:
Economic part /Mykola GNENNIYN/

Dnipro – 2022
Міністерство освіти і науки України

Український державний університет науки і технологій
Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інформаційні технології
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
___ Вадим ГОРЯЧКІН
15 грудня 2022 р.

ЗАВДАННЯ
На кваліфікаційну роботу ОС Магістр
студенту Назаренко Дар’ї Сергіївни

1. Тема дипломної роботи: Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL.
   Керівник роботи: Іванов Олександр Петрович
   Затверджені наказом від 1 листопада 2021 року.

2. Термін подання студентом закінченої роботи 15 грудня 2022 року

3. Вихідні дані до дипломної роботи: по закінченню дослідження очікуються висновки щодо доцільності використання ORM та SQL. Графік залежності часової тенденції використання ORM чи SQL.

4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Аналіз мови запитів SQL та ORM Entity Framework;
   4.2. Обґрунтування експериментального методу дослідження використання ORM;
   4.3. Проектування й розробка інструментального забезпечення для дослідження ефективності використання ORM;
   4.4. Дослідження ефективності використання ORM та SQL;
   4.5. Економічна частина;
   4.6. Загальні висновки.

5. Перелік демонстраційного матеріалу:
   5.1. Знімки екрану з виконанням деяких запитів;
   5.2. Наглядна демонстрація працюючої програми;
   5.3. Демонстрація відстеження запитів за допомогою SQL Server Profiler;

5.4. Вихідний графік актуальності використання ORM.

6. Консультанти:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Технічне завдання і розрахунки	Іванов О. П.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва розділу дипломної роботи	Термін виконання	Примітка
1	Вступ	5.06.2022	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	5.06.2022	
3	Аналіз сучасного стану програмно-апаратного забезпечення	5.06.2022	
4	Постановка задачі, технічне завдання	5.06.2022	
5	Розробка інструментальних засобів дослідження	1.11.2022	
6	Виконання досліджень	1.11.2022	
7	Оформлення тез доповідей	10.12.2022	
8	Оформлення пояснювальної записки	10.12.2022	

9	Розробка демонстраційних матеріалів	10.12.2022	

Керівник дипломної роботи Іванов Олександр Петрович

Завдання прийняв до виконання Назаренко Дар’я Сергіївна

РЕФЕРАТ

Об’єктом цього дослідження є використання ORM та SQL для запитів до бази даних.

Предметом дослідження є вплив використання ORM та SQL на час виконання та обробки запитів до бази даних.

Метою роботи є визначення механізмів роботи та оптимізації запитів з використанням ORM Entity Framework. Дослідити витрати часу на виконання запитів з використанням ORM та SQL.

Методи дослідження: порівняння часових характеристик виконання запитів з використанням ORM та SQL.

Результати та їх новизна: дослідження робить внесок у визначення практичної доцільності використання ORM та SQL. Результати дослідження дозволяють зробити висновки щодо можливостей та доцільность використання ORM.

Пояснювальна записка складається зі вступу, 5 розділів, висновків, бібліографічного списку та 2 додатків:

У вступі описується сутність розробки, її актуальність;

У першому розділі висвітлено аналіз сучасного стану дослідження проблеми за науковими літературними джерелами також проаналізовано програмні аналоги для різних мов програмування. Складається з 4 сторінок;

У другому розділі надано обґрунтування експериментального методу дослідження. Складається з 3 сторінок;

У третьому розділі представлене проектування й розробка інструментального забезпечення для дослідження. Складається з 4 сторінок;

У четвертому розділі описано виконані дослідження. Складається з 9 сторінок;

У п‘ятому розділі розкриті питання охорони та безпеки праці.
Складається з 15 сторінок;

Додатки містять технічне завдання, робочий проект, керівництво по роботі з програмою. Бібліографічний список. Ключові слова: ORM, SQL, запит, часові показники.

Зміст

Перелік умовних познак, термінів, скорочень

— ORM - технологія програмування, яка пов'язує бази даних із концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».
— SQL - це мова програмування для роботи з наборами фактів і зв’язками між ними.
— VS – Visual Studio.
— EF – Entity Framework.
— Ado .NET - технологія, що надає доступ та керування даними, що зберігаються в базі даних або інших джерелах, що базуються на платформі .NET Framework.
— CRUD – Create, Read, Update, Delete операції.
— API - Application Programming Interface - інтерфейс програмування додатків, програмний інтерфейс програми.
— SSMS – SQL Server Management Studio.
— T-SQL – різновид мови SQL, що використовується в SQL Server.

Вступ

Актуальність дослідження «Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL» спричинено великою популярністю серед веб-розробників на різних мовах програмування використання ORM.

Предметом дослідження обрано саме на ORM Entity Framework для .net, адже .net є однією з найпопулярніших веб-платформою.

Мета і завдання дослідження спрямоване на виявлення механізму роботи ORM Entity Framework, а саме конвертацію в запит мовою SQL та його оптимальність за часом та ресурсоємністтю.

Методом дослідження обране порівняння часу, витраченого на опрацювання запиту ORM Entity Framework та запиту, написаного мовою SQL. Також, за допомогою програми SQL Profiler, існує можливість перегляду SQL запитів у які конвертується запит, написаний за допомогою ORM Entity Framewor.

Наукова цінність даного дослідження полягає в необхідності розробникам використовувати та обробляти дані з бази даних, отже є необхідність у визначені кращого шляху обробки даних.

Під час виробничої практики було складено технічне завдання для виконання дослідження, налаштований локальний комп’ютер для вдалого виконання необхідних запитів.

Для практичної складової було знайдено тестову базу даних «Northwind» з офіційного сайту Microsoft та виконано декілька маніпуляцій с даними для відстеження часу, необхідного для виконання цих запитів.

1. Аналіз мови запитів SQL та ORM Entity Framework.

1.1. Використання мови запитів SQL та її проблема.

Коли є необхідність керування та зберігання великої кількості даних, використовуються бази даних, оскільки вони гарантують швидку та легку цілісність даних та отримання інформації. Керування програмами має включати надійну базу даних, щоб відповідати вашим уподобанням і вимогам проекту.

Однак безпосередньо працювати з базами даних із додатків може бути складно через відмінності в структурі даних. Виявлення тонкощів взаємовідносин сутностей також створює проблеми. Саме тому потрібен інструмент, який виступає в якості інтерфейсу між рівнем даних і основною програмою. І два найпоширеніші доступні інструменти – SQL та ORM.

Мова структурованих запитів (SQL) — це мова, розроблена спеціально для керування великими обсягами даних, що зберігаються в системі керування реляційною базою даних SQL. Цей тип мови широко відомий як декларативна мова програмування.

Основна мета цієї мови програмування полягає в тому, щоб дозволити адміністратору мати повний контроль над інформацією, до якої необхідно отримати доступ із сервера бази даних. Цей тип мови можна використовувати для створення системи, здатної обробляти дані на різних етапах.

Коли запит SQL виконуються, результати заносяться в журнал. Ці журнали потім використовуються для відстеження того, що робить адміністратор. Програма, яка створює файли журналів, називається механізмом запитів або програмою мови запитів (QL).

Ці програми можна зберігати у файлі бази даних або на жорсткому диску. Адміністратор може переглянути результати однієї з цих програм, виконавши команду, наприклад «EXEC SQL SERVER».

Запит — це процедура, яка повідомляє системі, що робити — наприклад, як зібрати необхідні дані для виконання бажаної функції. Результати запиту можна відобразити різними способами. Зазвичай вони зберігаються у файлі журналу, який створюється під час виконання запиту.

1.2. Огляд програмних аналогів

Порівняння різних ORM для деяких мов програмування наведено в табл. 1. У табл. 1 використано такі умовні позначення: «+» означає, що фу-нкціональна можливість наявна, «+/–» – наявна частково, «–» – відсутня.

Табл.1 порівняння функціональних можливостей різних ORM

Функція	**Entity Framework**	**NHibernate**	**JPA**
Мова програмування	.NET (C#, VB)	.NET (C#, VB), Java	Java
Генерація XML файлу відображення SQL	Автоматична	Тільки з використання сторонніх бібліотек	Тільки з використання сторонніх бібліотек
Двонаправлені відносини	+	+	+
Виконування транзакцій	Автоматично	Автоматично	Автоматично
Оптимізація	Lazy loading, кешування	Lazy loading, кешування	Lazy loading, кешування
Методи запитів	LINQ, Entity SQL, SQL	HQL, Criteria API, SQL	Jpql, SQL
Простота використання	+	+	+/-
Адаптованість таблиць до моделей	+	+	+/-
Залежність від SQL	+	+	+/-

Підтримка кешування	+	+	-
Відображення	Автоматичне відображення .net моделей та таблиці бази даних завдяки вбудованому механізму у Microsoft Visual Studio IDE	Відображає Java POJO та таблиці бази даних	Використовує анотації для відображення POJO та таблиць
Підтримка СУБД	MS SQL Server, MySqL, комерційна для інших СУБД	MS SQL Server, MySqL, Oracle, PostgreSQL	MS SQL Server, MySqL, Oracle, PostgreSQL

1.3. У табл. 2 описано переваги та недоліки наведених ORM.

Табл.2 переваги та недоліки різних ORM

Назва програми	Переваги	Недоліки
Entity Framework	Дозволяє створювати моделі за різними способами: code first, database first, Model first. Простий синтаксис LINQ. Багато коду є автозгенерованим, що зменшує кількість коду в цілому. Підтримує асинхронні запити до бази даних.	Адаптована лише для платформи .net та має добру сумісність з СУБД MS SQL Server, MySqL. Для великих запитів може втрачати свою продуктивність. Генерація запитів відбувається автоматично і не може бути контрольованою.
NHibernate	Має гнучкі та дуже потужні можливості відображення. Він підтримує кеш другого рівня, який можна використовувати	Збільшений час запуску за рахунок підготовки метаданих. Без попередньої роботи з ORM складна в використанні.

	серед кількох ISessionFactory Дуже вдала реалізація Unit Of Work. Популярний у корпоративному середовищі. Добре інтегрований у всі СУБД.	Написання XML-відображення може бути дуже виснажливим, особливо для великих баз даних із сотнями й тисячами таблиць, представлень і збережених процедур. NHibernate сильно відстає щодо документації, оскільки його процес застарів, а ресурси для навчання розкидані по всьому Інтернету.
JPA	Простий у використанні. Використовує анотації для визначення зіставлення між об'єктами Java та реляційними базами даних, тоді як традиційні ORM здебільшого використовують файли конфігурації XML. JPA є уніфікованим API, офіційно визначеним Java.	Використовується лише для роботи з Java. Дуже нова технологія і знадобиться час на тестування та оновлення недоліків, перш ніж вона стане стабільною. JPA це скоріше специфікація, ніж продукт.

1.4. Огляд літератури

Сьогодні існує велика кількість різноманітних ORM для різних платформ, але більшість має схожий принцип роботи. Однією з найпопулярніших ORM систем для платформи .Net є Entity Framework. Вивчаючи офіційну документацію даного ресурсу [1] стає зрозуміло, що Entity Framework має відкритий вихідний код, який підтримується Microsoft.

Серед наведених переваг для використання саме цієї ORM системи є: легкість керування, рівень абстракції, легкість використання через схожість запитів до LINQ,

дизайнер для створення бази даних, Code First метод написання сутностей завдяки коду, висока інтеграція з продуктами компанії Microsoft.

Тим не менше, рано чи пізно будь який програміст затикається з проблемою повільної роботи додатка через використання Entity Framework, саме на це і спрямоване дослідження. Розробники цієї ORM не приховують даної проблеми і пропонують різні рішення для прискорення роботи, або пояснюють чому відбувається сповільнення та якими шляхами можна покращити ситуацію. Наприклад: використання механізму кешування, використання прямої(повної) загрузки замість «лінивої» загрузки та інш.

Також, виходячи з написаного [2], стає зрозумілим, що робота Entity Framework не є безкорисливою, увесь механізм потребує додаткового навантаження на систему через першу компіляцію моделей та встановлення усіх зв'язків.

1.5. Постановка задачі

— Розробити програму для дослідження доцільності використання ORM та SQL, яка реалізує такі можливості для дослідження:

— часова ефективність використання ORM Entity Framework та SQL;

— оптимізації запитів з використанням ORM Entity Framework;

— рекомендації щодо доцільності використання ORM Entity Framework.

Висновки до розділу 1

У цьому розділі було розглянуто аналоги, виконано їх аналіз для подальшого дослідження. Завдяки цьому при проведенні дослідження були враховані усі недоліки та скарги на схожі продукти.

У ході огляду літератури було розглянуто офіційну документацію однієї з найпопулярніших ORM систем. Отримано важливі теоретичні відомості для подальшого розвитку теми доцільності використання ORM систем.

Аналіз аналогів та огляд літератури щодо різних ORM систем показав, що ORM є зручною для використання програмістами, бо це спрощує написання коду, але усі системи сповільнюють виконання запитів та власне роботу усього додатка.

Дослідження є актуальним через нерозуміння багатьох програмістів роботи ORM Entity Framework, її механізму конвертації запитів до мови SQL та шляхів поліпшення швидкодії та оптимізації.

2. Обґрунтування експериментального методу дослідження використання ORM

Актуальність дослідження використання ORM чи SQL завжди була і буде, адже коли йдеться про керування та зберігання даних, найчастіше використовуються бази даних, оскільки вони гарантують швидку та легку цілісність даних і пошук інформації. Керування програмами має включати надійну базу даних, яка відповідає вимогам проекту.

Однак безпосередня робота з базами даних із додатків може бути складною через відмінності в структурі даних. Вираження тонкощів відносин між сутностями також створює проблеми. Ось чому був створений інструмент, який діє як інтерфейс між рівнем даних і основною програмою. А два найпоширеніші доступні інструменти — це SQL і ORM.

2.1. Часова ефективність використання ORM чи SQL

Основним методом дослідження було обрано саме порівняння витраченого часу на обробку запитів з використанням ORM чи SQL. Це найефективніший метод дослідження даної теми, адже при розробці ПЗ най значущим критерієм є саме час виконання певної дії.

Таким чином, для знаходження часового показника t – слід скористатися формулою: t1 – t2, де t1 - початок виконання запиту і t2- кінець виконання запиту.

2.2. Легкість підтримки і написання коду

Наступним методом дослідження є складність написання коду, кількісна різниця строк коду та легкість підтримки - статистика знання ORM та SQL серед розробників.

3. Проектування й розробка інструментального забезпечення для дослідження ефективності використання ORM

3.1. Формалізація задачі

Формалізація задачі на рівні зовнішнього користувача наведена у діаграмі використання. Користувач представлений у вигляді актора, що взаємодіє із системою за допомогою варіантів використання. Діаграма варіантів використання зображені на рис. 3.1.

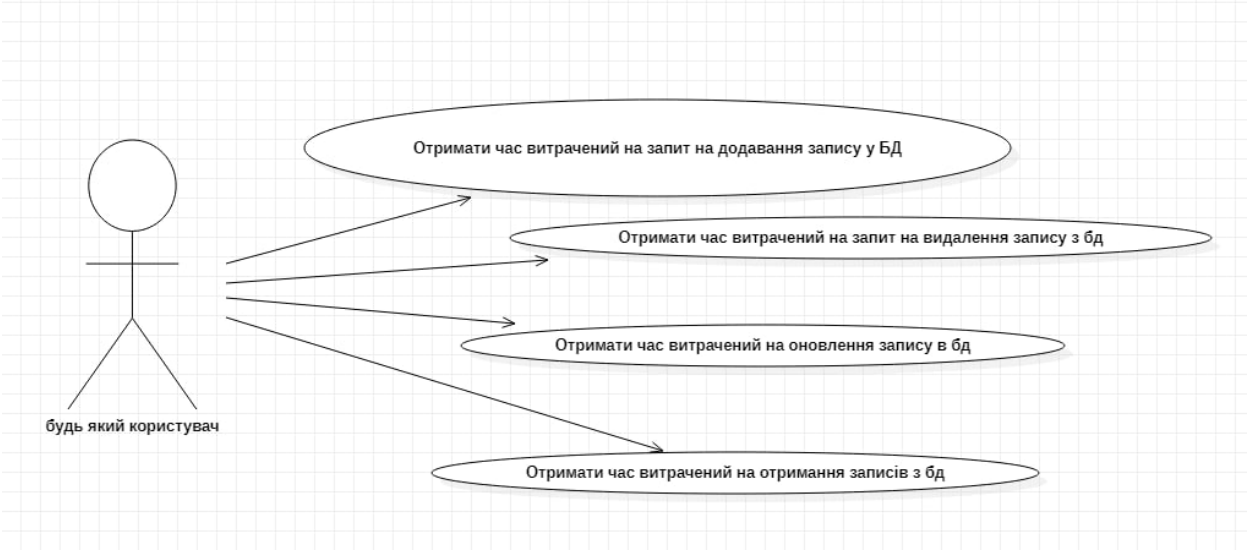


Рис. 3.1 Діаграма варіантів використання

3.2. Базова Архітектура системи

Для розробки системи було обрано архітектурну модель web API - «програмний інтерфейс додатка». Обрана парадигма програмування – об’єктно-орієнтована.

Було обрано саме REST API. Архітектурний стиль REST являє собою набір функцій, таких як POST, PUT, DELETE, GET та ін., які клієнти можуть використовувати для доступу до серверних даних. Клієнти і сервери обмінюються даними за протоколом HTTP. Головною особливістю REST API є те, що на сервері не зберігається жодних даних про клієнта, тим самим в одного сервера може бути декілька клієнтів. Клієнтські запити до сервера – URL-адреси, а відповіді від сервера

приходять у вигляді звичайних даних, без графічного відображення веб сторінки. Архітектура REST API наведена на рис. 3.2.

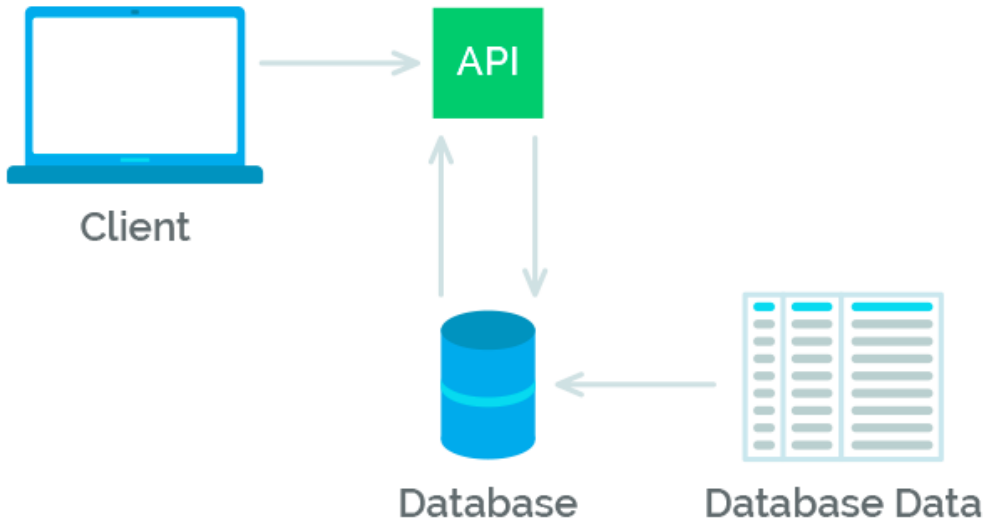


Рис. 3.2 Архітектура REST API

3.3. Внутрішнє програмування

3.3.1. Вибір мови програмування

Мовою програмування було обрано мову програмування C# через достатню кількість знань та досвіду роботи з нею.

3.3.2. Технологічна платформа

Під час розробки системи використовувалися такі технології:

— .net 6 – популярна платформа для створення веб додатків, в нашому випадку - web API.

— ORM Entity Framework – одна з найпопулярніших ORM для платформи .net, яка має добру сумісність адже обидва продукти розроблені компанією Microsoft.

— Swagger – набір інструментів для розробників API, що дозволяє виконувати запити до сервера з мінімальним користувацьким інтерфейсом.

— SQL - декларативна мова програмування для взаємодії користувача з базами даних.

3.3.3. Ієрархія та взаємодія класів системи наведена на рис. 3.3.

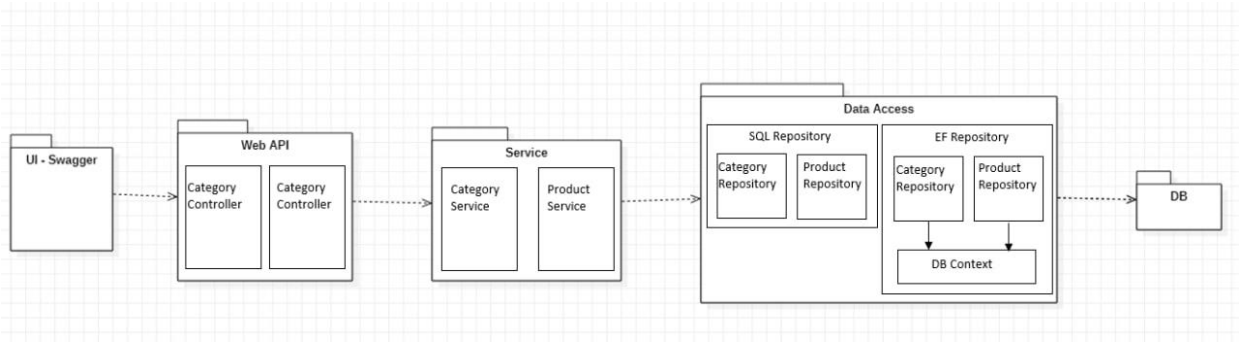


Рис. 3.3 Ієрархія взаємодії класів

3.3.4. Використані принципи програмування

Для проектування внутрішньої структури використовувалися такі принципи програмування:

— Принцип єдиного обов’язку – кожен клас має лише один обов’язок та відповідальний лише за одну дію.

— Відкритості/закритості – усі зміни в коді відбувалися шляхом додавання нового функціоналу, а не зміну існуючого.

— Підстановки Барари Лісков – об’єкти в програмі можуть бути замінені їх нащадками.

— Інверсії залежності – залежності в системі будуються на основі абстракцій, які не повинні залежати від деталей, а навпаки, деталі повинні залежати від абстракцій.

— Ін'єкція залежності – це реалізаця принципу інверсії залежності.

3.3.5. Використані шаблони проектування

— Repository – це клас, визначений для сутності, з усіма операціями, можливими над цією конкретною сутністю. Наприклад, репозиторій для

Product матиме основні операції CRUD та будь-які інші можливі операції, пов’язані з ним. Шаблон Repository є одним з найпопулярніших на даний момент. Він дотримується SOLID принципів та простий у використанні. Цей шаблон має дві мети: по-перше, це абстракція рівня даних, а по друге це спосіб централізації об’єктів домену.

3.4. Інтерфейс користувача

Як було сказано вище, розроблявся додаток web API, що дає змогу працювати з будь-яким клієнтом з використанням HTTP протоколу.

Дане дослідження було спрямоване перш за все на вимір часу роботи з базою даних, тому замість користувацького інтерфейсу є Swagger, який дозволяє робити запити на сервер і має стандартний користувацький інтерфейс – рис. 3.4.

Swagger дозволяє описати структуру API, щоб машини могли їх читати. Здатність API описувати власну структуру є корінням Swagger. Прочитавши структуру нашого API Swagger автоматично створює красиву та інтерактивну документацію.

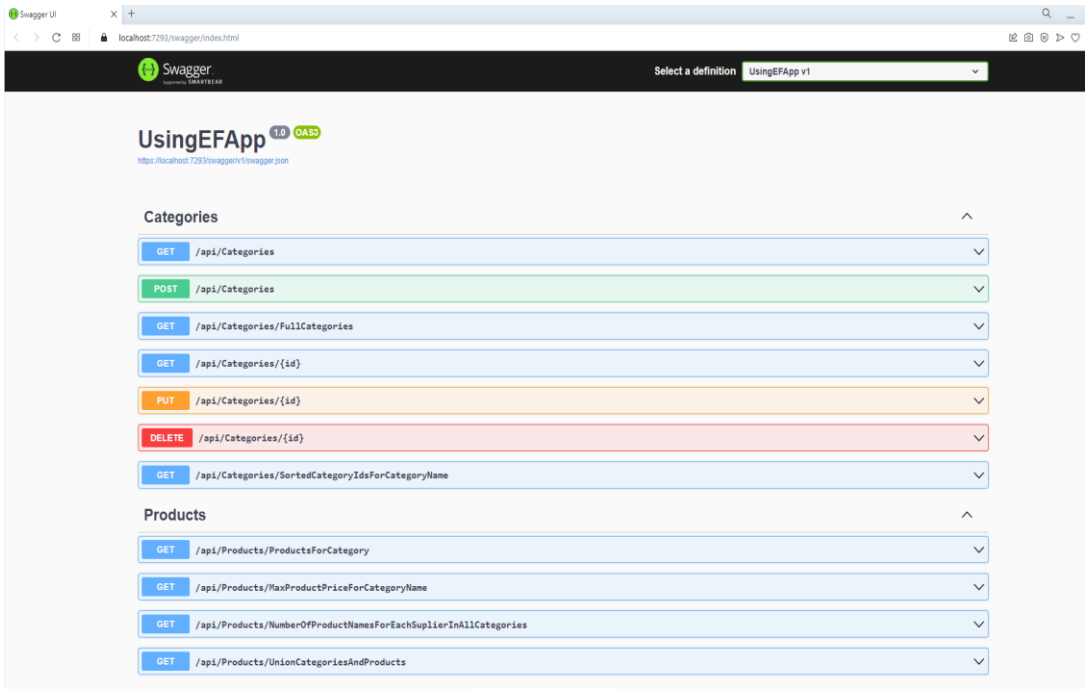


Рис. 3.4

3.5. Тестування та налагодження програми

3.5.1. Аналіз методів тестування та відлагодження

Під час розробки програми найбільш трудомістким є етап відлагодження програми, який полягає у виявленні наявних у програмі помилок. Мета відлагодження полягає саме у виявленні та усуненні причин помилок.

Для ефективного тестування рекомендується використовувати методи білої та чорної скриньки.

3.5.2. Тестування системи методом припущення про помилку

Було проведено тестування сервісів, а отже на виході ми завжди очікуємо результат з виміру часу, а саме TimeMeasuringResponse { EfTime ; SQLTime }.

Якщо база порожня – чи отримаємо ми час – ні, буде викликана SQL помилка;

Якщо база повна – чи отримаємо ми час – так;

Якщо при оновленні не було знайдено потрібного елемента – чи отримаємо ми час – так;

Якщо при видаленні не було знайдено потрібного елемента – чи отримаємо ми час – ні, буде викликана SQL помилка.

Висновки до розділу 3

Виконано проектування архітектури інструментального забезпечення для дослідження часової ефективності використання SQL та ORM. У ході проектування архітектури було передбачено можливість модернізації та розширення системи за рахунок розбиття на слабопов’язані модулі. Відповідно до SOLID-принципів, кожен клас системи має тільки один обов’язок, розроблені програмні інтерфейси відкриті для розширення, але закриті для змін. Це дозволить повторно використовувати розроблені класи та модулі, розширювати функціонал.

Завдяки використанню загальновідомим шаблонам проектування, будь-якому розробнику буде легко зрозуміти принципи функціонування системи та виконувати її підтримку.

У розділі наведено опис інтерфейсу користувача. Для інтерфейсу використовувався Swagger, який знаходиться у вільному доступі;
увесь інтерфейс системи має зрозумілий вигляд та поділений на сторінки, завдяки чому користувач з легкістю зможе знайти та змінити необхідні йому параметри та виконати дослідження.

Виконане тестування розробленого інструментального забезпечення для аналізу часової ефективності запитів за допомогою методу припущення про похибку.

Система готова до впровадження та використання.

4. Дослідження ефективності використання SQL та ORM

4.1. Підготовка до експерименту

Для проведення експерименту було заздалегідь обрано мову програмування та середовище виконання, а саме: мова програмування C#, середовище виконання VisualStudio. Для роботи з базою даних було обрано SQL Server та SQL Server Management Studio, також було обрано саме EntityFramework як дослідницьку ORM. Найголовнішим обґрунтуванням такого вибору є один розробник – Microsoft.

4.1.1. Опис використаного програмно-апаратного середовища

VisualStudio – це інтегроване середовище розробки (IDE), яке використовується для розробки консолі, додатків графічного інтерфейсу користувача, додатків Windows Forms або WPF, веб-додатків, веб-сайтів і веб-сервісів тощо. До того ж VisualStudio має досить розгорнуту безкоштовну версію свого продукту – Community edition.

SQL Server – це система керування реляційною базою даних (RDBMS), яка підтримує широкий спектр програм обробки транзакцій, бізнес-аналітики та аналітики в корпоративних IT-середовищах. SQL Server використовує Transact-SQL (T-SQL), реалізацією SQL від Microsoft, яка додає набір власних розширень програмування до стандартної мови. Основним компонентом Microsoft SQL Server є SQL Server Database Engine, який контролює зберігання, обробку та безпеку даних. Він включає реляційний механізм, який обробляє команди та запити, і механізм зберігання, який керує файлами бази даних, таблицями, сторінками, індексами, буферами даних і транзакціями. Збережені процедури, тригери, подання та інші об’єкти бази даних також створюються та виконуються за допомогою Database Engine.

SQL Server Management Studio – це розширене середовище розробки, яке дає

нам змогу налаштовувати, керувати й адмініструвати механізми баз даних SQL Server. SSMS дуже популярний і широко використовується розробниками та адміністраторами баз даних.

EntityFramework - основний засіб Microsoft для взаємодії між програмами .NET і реляційними базами даних. Entity Framework — це Object Relational Mapper (ORM), який є типом інструменту, який спрощує зіставлення між об'єктами вашого програмного забезпечення та таблицями та стовпцями реляційної бази даних. Entity Framework (EF) — це платформа ORM з відкритим кодом для ADO.NET, яка є частиною .NET Framework. ORM піклується про створення з'єднань з базою даних і виконання команд, а також отримання результатів запитів і автоматичну матеріалізацію цих результатів як об'єктів вашої програми. ORM також допомагає відстежувати зміни в цих об'єктах, і, коли буде отримано вказівку, він також зберігатиме ці зміни назад у базі даних.

4.1.2. Опис підходу для визначення часу обробки запитів

Оскільки найголовніша мета – порівняти часову ефективність запитів з використанням SQL та ORM EntityFramework, було обрано наступний підхід: проектувався запит до бази даних спочатку на мові SQL з використанням різних операторів, далі цей самий запит формулювався з використанням ORM EntityFramework. Було створено два репозиторії: SQL та ORM, та один сервіс, який відповідальний за вимір часу. Робилося два запити до бази даних: перший – з використанням ORM, другий – з використанням SQL та вимірювався час кожного запиту. Таким чином, для знаходження часового показника t – використовувалася формула: t1 – t2, де t1 - початок виконання запиту і t2- кінець виконання запиту.

4.1.3. Вплив системи кешування ORM та особливості EntityFramework

У наведеному нижче описі наведено загальний огляд процесу, який проходить кожен запит.

Запит LINQ обробляється Entity Framework Core для створення представлення, готового до обробки постачальником бази даних. Результат кешується, тому цю обробку не потрібно виконувати кожного разу, коли виконується запит. Результат передається постачальнику бази даних. Постачальник бази даних визначає, які частини запиту можна оцінити в базі даних. Ці частини запиту перекладаються на SQL. Запит надсилається до бази даних і повертається набір результатів (результати – це значення з бази даних, а не екземпляри сутності). Для кожного елемента в наборі результатів: якщо запит є запитом відстеження, EF перевіряє, чи дані представляють сутність, яка вже є в системі відстеження змін для екземпляра контексту. Якщо так, повертається наявна сутність. Якщо ні, створюється нова сутність, налаштовується відстеження змін і повертається нова сутність. Якщо запит не відстежується, завжди створюється та повертається нова сутність.

Поведінка відстеження контролює, чи Entity Framework Core зберігатиме інформацію про екземпляр сутності у своєму відстежувачі змін. Якщо сутність відстежується, будь-які зміни, виявлені в сутності, будуть збережені в базі даних під час SaveChanges(). EF Core також виправить навігаційні властивості між сутностями в результатах запиту відстеження та сутностями, які знаходяться в системі відстеження змін.

За замовчуванням запити, які повертають типи сутностей, відстежуються. Це означає, що ви можете вносити зміни в ці екземпляри сутності та зберігати ці зміни за допомогою SaveChanges(). Коли результати повертаються в запиті на відстеження, EF Core перевірить, чи сутність уже знаходиться в контексті. Якщо EF Core знаходить існуючу сутність, повертається той самий екземпляр. EF Core не перезаписує поточні та початкові значення властивостей сутності в записі зі значеннями бази даних. Якщо сутність не знайдено в контексті, EF Core створить новий екземпляр сутності та приєднає його до контексту. Результати запиту не містять жодної сутності, доданої до контексту, але ще не збереженої в базі даних.

Жодні запити відстеження не є корисними, якщо результати використовуються в сценарії лише для читання. Вони виконуються швидше, оскільки немає необхідності налаштовувати інформацію про відстеження змін. Якщо вам не потрібно оновлювати сутності, отримані з бази даних, тоді слід використовувати запит без відстеження. Ви можете змінити окремий запит на невідстежуваний. Жоден запит відстеження також не дасть вам результатів на основі того, що є в базі даних, ігноруючи будь-які локальні зміни чи додані сутності.

4.2. Проведення експерименту

Оскільки виконання запитів з використанням EF Core має 2 шляхи:

- Cold query – виконання запиту у самий перший раз, коли виконується багато роботи щоб загрузити та перевірити дані;
- Warm query – виконання запиту наступні рази, з урахуванням механізму кешування.

Було обрано наступний шлях експерименту: замір часу першого запиту, наступного і запиту з використанням SQL.

Для проведення експерименту було використано підхід DatabaseFirst. Для проведення експерименту також було розроблені такі запити:

- 1 - вибірка усіх категорій;
- 2 – вибірка усіх категорій із списком продуктів до кожної;
- 3 – вибірка 1 категорії за вказаним параметром;
- 4 – додавання нової категорії;
- 5 – оновлення вибраної категорії;
- 6 – видалення вказаної категорії;
- 7 - перелічує кількість продуктів для кожної категорії;
- 8 - перераховує кількість categoriesId у кожній назві категорії, відсортованих за спадінням;

— 9 – знайти максимальну вартість продукту у кожній категорії;

— 10 – в таблиці продуктів знайти число продуктів від кожного постачальника для всіх категорій;

— 11 – об’єднати вибірки всіх категорій та продуктів.

Запити до бази даних розроблені різні: легкі, та більш складні з використанням різних операторів, задля порівняння важкості написання на мові

Нижче наведена демонстрація деяких запитів і їх результати рис. 3.5 – 3.8

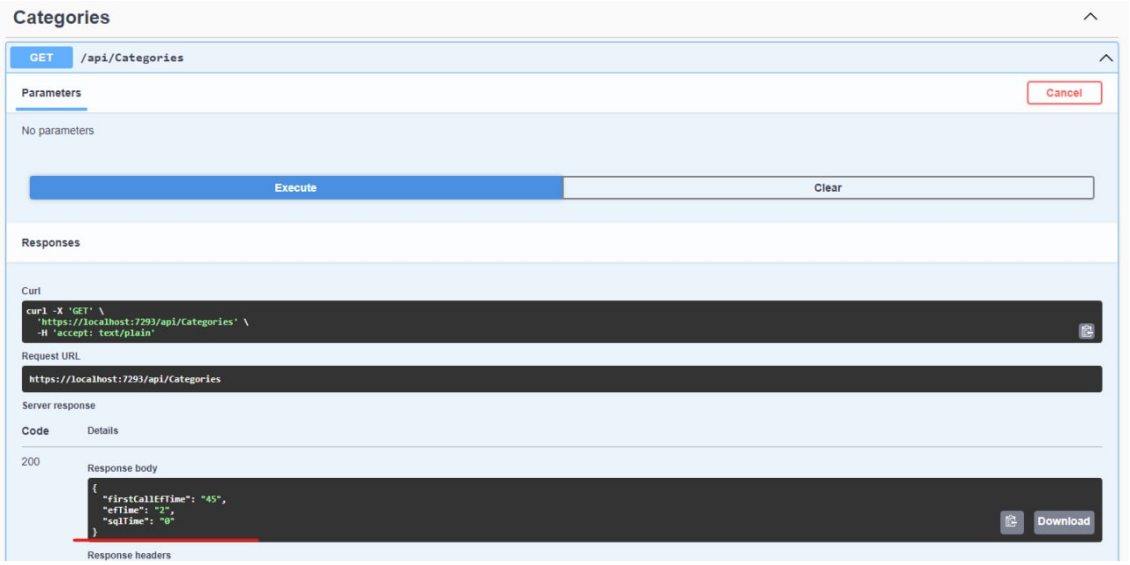


Рис. 3.5 знімок екрана swagger, а саме виконання запиту № 1

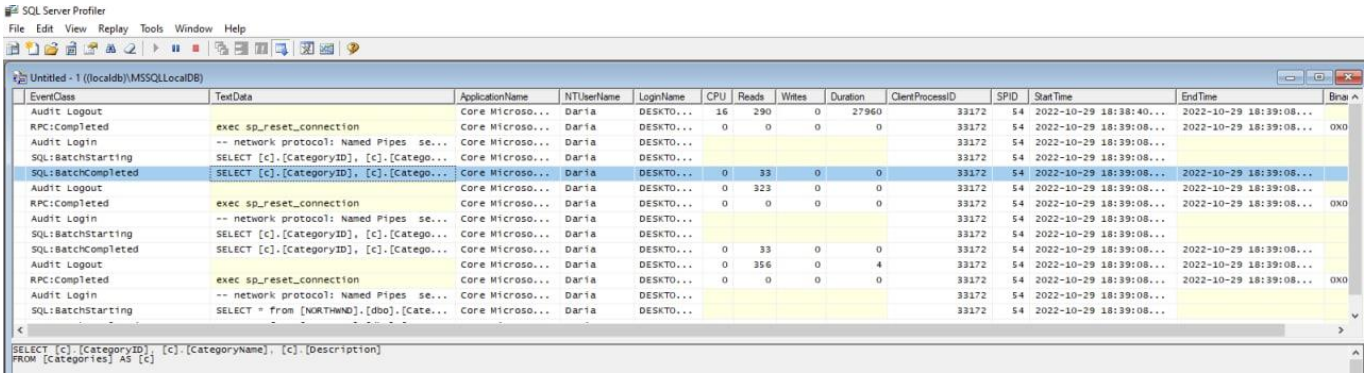


Рис. 3.6 знімок екрана SQL Server Profiler, а саме виконання запиту № 1

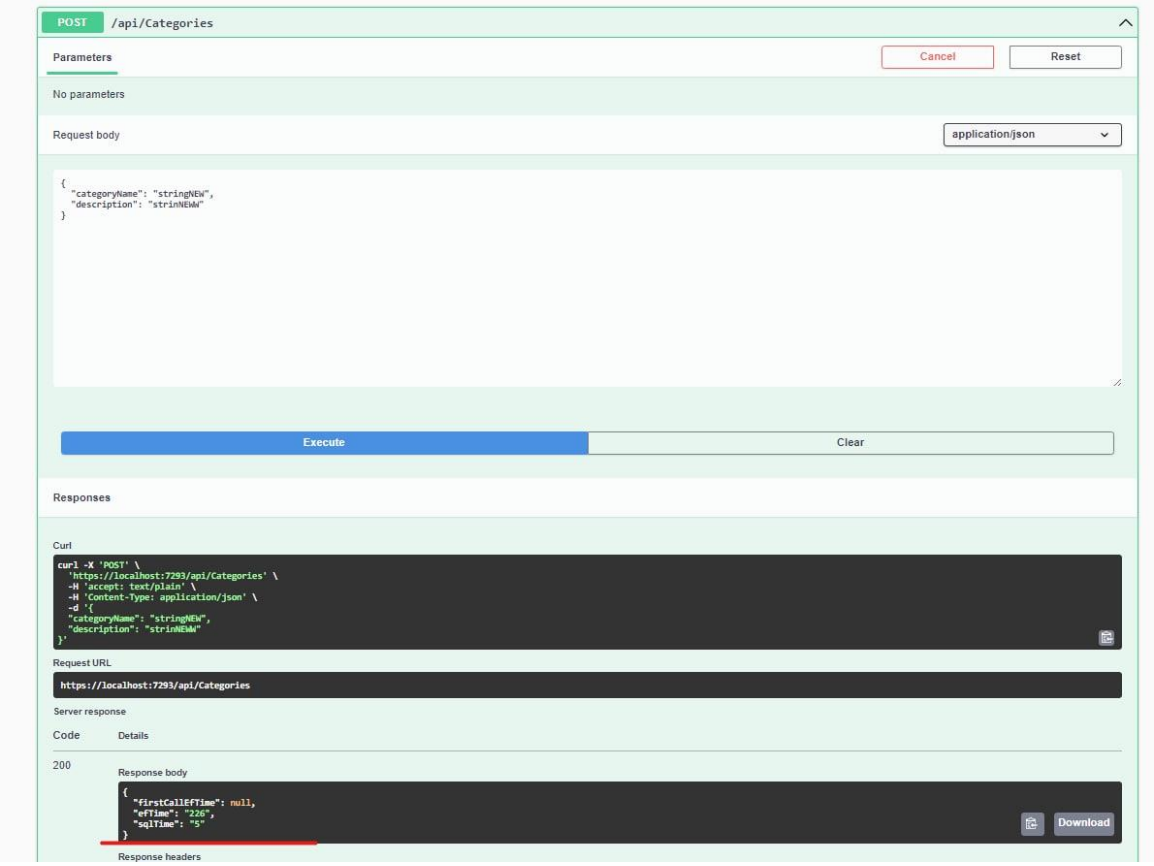


Рис. 3.7 знімок екрана Swagger, а саме виконання запиту № 4

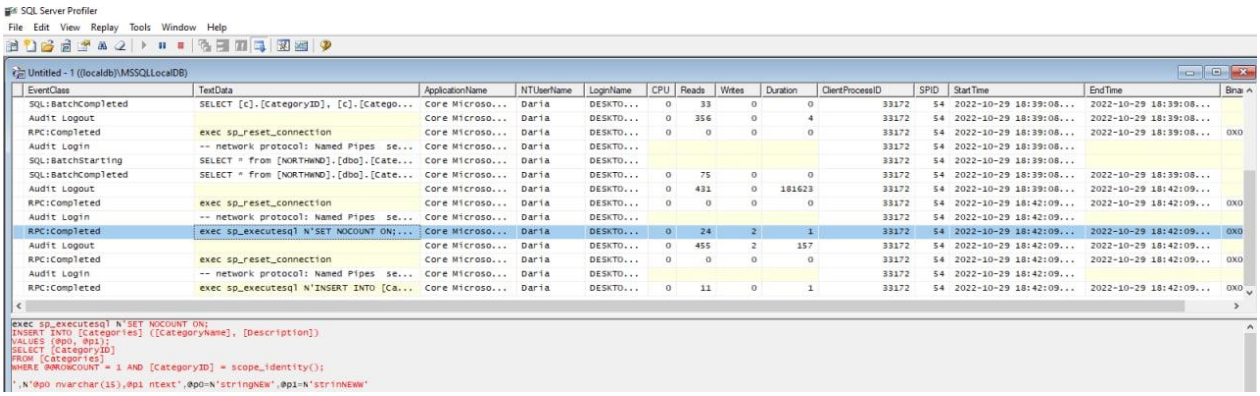


Рис. 3.8 знімок екрана SQL Server Profiler, а саме виконання запиту № 4

4.3.Результати експерименту

Результати експерименту наведено у таблиці таб. 1 з трьома колонками.

Перша – виконання запиту з використанням EF Core перший раз;

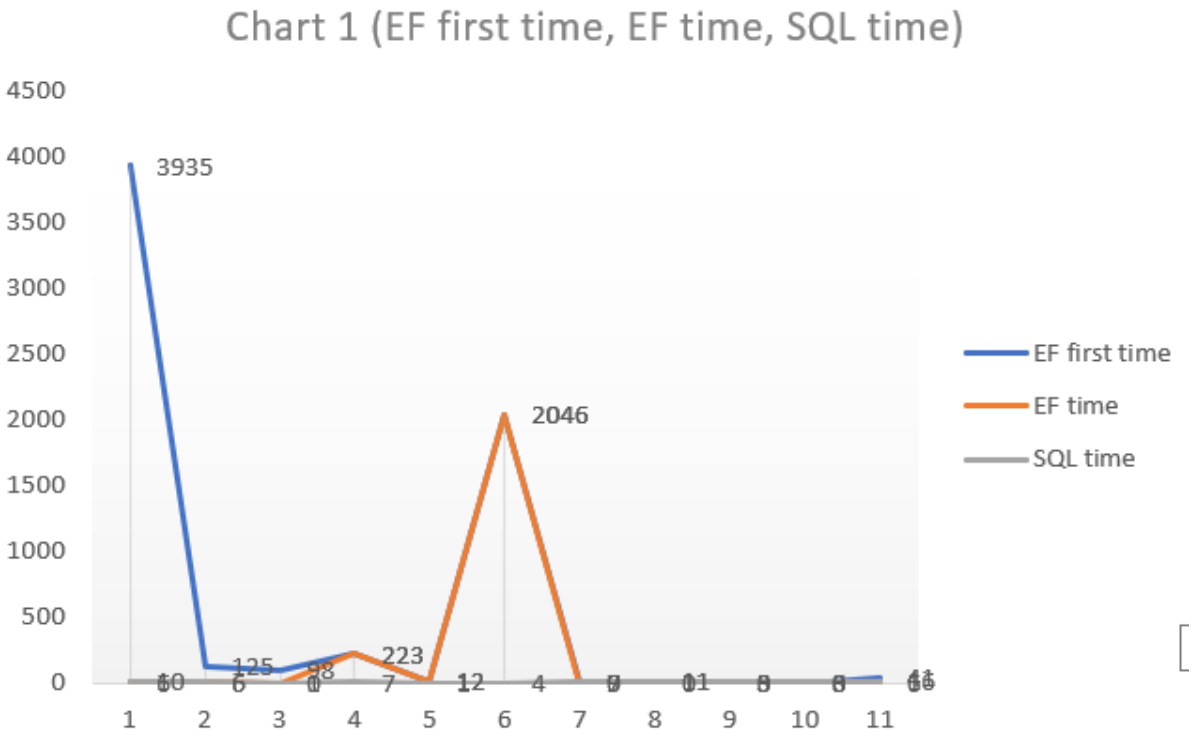
Друга - виконання запиту з використанням EF Core наступний раз;

Третя – виконання запиту з використанням SQL. Перший експеримент виконувався з постійно працюючим сервером.

Таб. 1

Query #	EF first time	EF time	SQL
1	3935	10	6
2	125	6	6
3	98	0	1
4 add		223	7
5 update		12	1
6 delete		2046	4
7	9	0	7
8	11	0	11
9	3	0	5
10	3	0	6
11	41	6	16

На діаграмі 1 наведено наглядне порівняння таблиці 1.



Діагр. 1

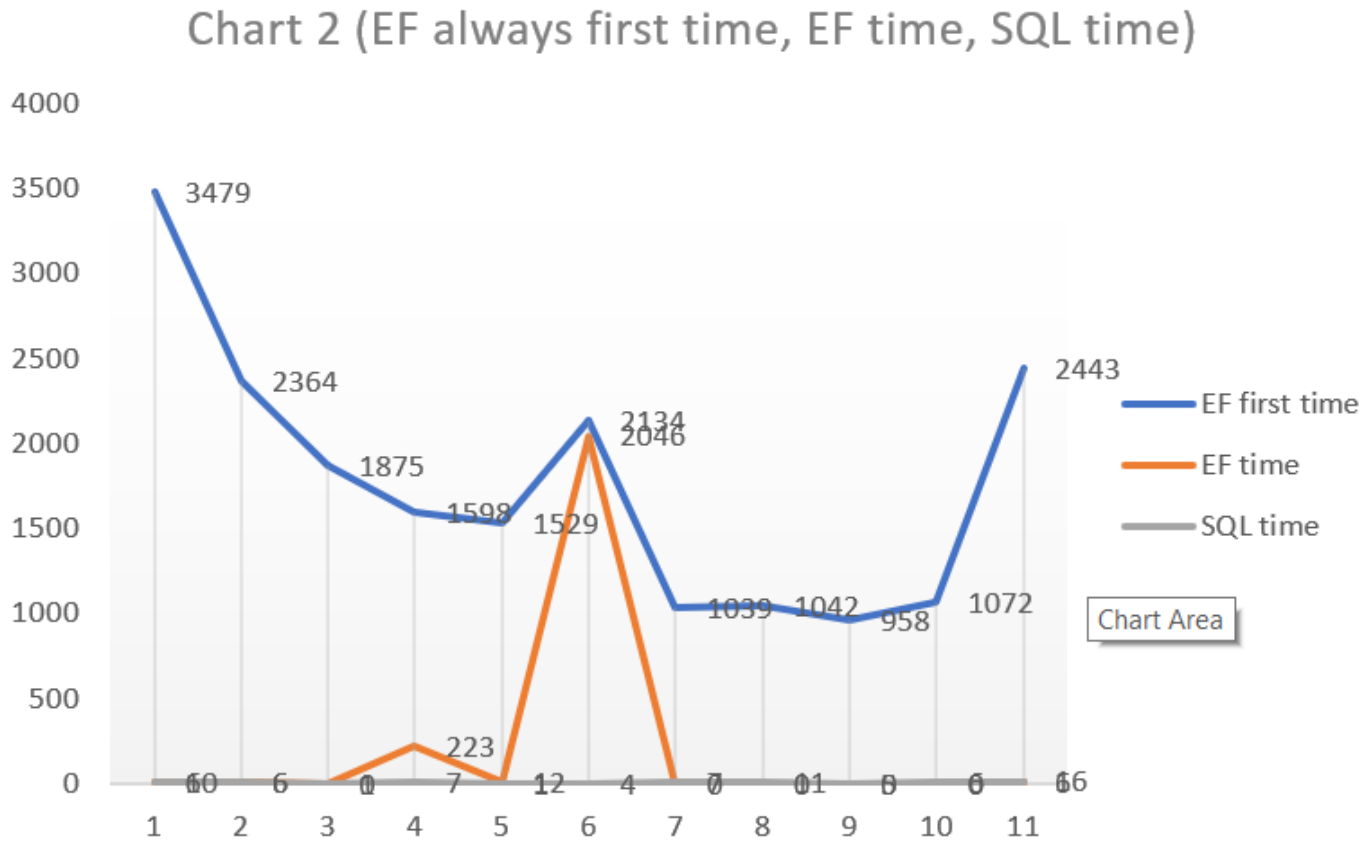
У таблиці таб. 2 наведено часові результати досліду якщо система не працює весь час.

Таб. 2

Query #	EF first time	SQL
1	3479	21
2	2364	10

3	1875	18
4 add	1598	11
5 update	1529	7
6 delete	2134	13
7	1039	20
8	1042	20
9	958	24
10	1072	11
11	2443	16

На діаграмі 2 наведено наглядне порівняння роботи запитів з використанням EF при постійно увімкненому сервері та ні і запитів з використанням SQL.



Діагр. 2

Нижче наведено конвертація EF запиту до мови SQL та власне сам запит з використанням SQL таб. 3.

Таб. 3

Query #	EF query	EF converted query	SQL query
1	_context.Categories.ToList()	SELECT [c].[CategoryID], [c].[CategoryName], [c].[Description] FROM [Categories] AS [c]	SELECT * from [NORTHWND].[dbo].[Categories]
2	_context.Categories.Include(c => c.Products).ToList()	SELECT [c].[CategoryID], [c].[CategoryName], [c].[Description], [p].[ProductID], [p].[CategoryID], [p].[Discontinued], [p].[ProductName], [p].[QuantityPerUnit], [p].[ReorderLevel], [p].[SupplierID], [p].[UnitPrice], [p].[UnitsInStock], [p].[UnitsOnOrder] FROM [Categories] AS [c] LEFT JOIN [Products] AS [p] ON [c].[CategoryID] = [p].[CategoryID] ORDER BY [c].[CategoryID]	SELECT * FROM [Categories] AS [c] LEFT JOIN [Products] AS [p] ON [c].[CategoryID] = [p].[CategoryID] ORDER BY [c].[CategoryID]
3	_context.Categories.Find(id)	exec sp_executesql N'SELECT TOP(1) [c].[CategoryID], [c].[CategoryName], [c].[Description] FROM [Categories] AS [c] WHERE [c].[CategoryID] = @__p_0',N'@__p_0 int',@__p_0=1	"SELECT * FROM [Categories] WHERE [CategoryID] = " + id
4	_context.Categories.Add(category); _context.SaveChanges();	exec sp_executesql N'SET NOCOUNT ON; INSERT INTO [Categories] ([CategoryName], [Description]) VALUES (@p0, @p1); SELECT [CategoryID] FROM [Categories] WHERE @@ROWCOUNT = 1 AND [CategoryID] = scope_identity(); ',N'@p0 nvarchar(15),@p1 ntext',@p0=N'Dashka1199',@p1 =N'Created from App using EF'	exec sp_executesql N'INSERT INTO [Categories] VALUES (@CategoryName, @Description, null)',N'@CategoryName nvarchar(6),@Description nvarchar(9)',@CategoryName=N'string',@Description=N'stringSQl'

5	_context.Entry(category).State = EntityState.Modified; _context.SaveChanges();	exec sp_executesql N'SET NOCOUNT ON; UPDATE [Categories] SET [CategoryName] = @p0, [Description] = @p1 WHERE [CategoryID] = @p2; SELECT @@ROWCOUNT; ',N'@p2 int,@p0 nvarchar(15),@p1 ntext',@p2=14,@p0=N'Category DSNAX',@p1=N'EDITED'	"UPDATE [Categories] SET [CategoryName] = @CatName, [Description] = @Desc WHERE [CategoryID] = " + id

6	var category = _context.Categories.Find(id);
	
	_context.Categories.Remove(category);
	_context.SaveChanges();
		exec sp_executesql N'SELECT TOP(1) [c].[CategoryID], [c].[CategoryName], [c].[Description]
		FROM [Categories] AS [c]
		WHERE [c].[CategoryID] = @__p_0',N'@__p_0 int',@__p_0=15
		
		exec sp_executesql N'SET NOCOUNT ON;
		DELETE FROM [Categories]
		WHERE [CategoryID] = @p0;
		SELECT @@ROWCOUNT;
		
		',N'@p0 int',@p0=15
			$"DELETE FROM [Categories] WHERE [CategoryID] = {id}"

7	_context.Products.Include(p=> p.Category).GroupBy(p => p.Category.CategoryName).Select(g => new { name = g.Key, count = g.Count() })	SELECT [c].[CategoryName] AS [name], COUNT(*) AS [count] FROM [Products] AS [p] LEFT JOIN [Categories] AS [c] ON [p].[CategoryID] = [c].[CategoryID] GROUP BY [c].[CategoryName]	SELECT cat.CategoryName, COUNT(prod.ProductID) AS NumberOfProducts FROM [NORTHWND].[dbo].[Products] as prod LEFT JOIN [NORTHWND].[dbo].[Categories] as cat ON prod.CategoryID = cat.CategoryID GROUP BY cat.CategoryName
8	_context.Categories.GroupBy(c => c.CategoryName).Select(g => new { name = g.Key, count = g.Count() }).OrderByDescending(s => s.count);	SELECT [c].[CategoryName] AS [name], COUNT(*) AS [count] FROM [Categories] AS [c] GROUP BY [c].[CategoryName] ORDER BY COUNT(*) DESC	SELECT Count(CategoryID), CategoryName FROM [NORTHWND].[dbo].[Categories] GROUP BY CategoryName ORDER BY COUNT(CategoryID) DESC
9	_context.Products.Include(p => p.Category).GroupBy(p => p.Category.CategoryName).Sel	SELECT [c].[CategoryName] AS [name], MAX([p].[UnitPrice]) AS [count] FROM [Products] AS [p]	SELECT cat.CategoryName, MAX(prod.UnitPrice) FROM [NORTHWND].[dbo].[Products] as

	ect(g => new { name = g.Key, count = g.Max(p => p.UnitPrice) });	LEFT JOIN [Categories] AS [c] ON [p].[CategoryID] = [c].[CategoryID] GROUP BY [c].[CategoryName]	prod LEFT JOIN [NORTHWND].[dbo].[Categories] as cat ON prod.CategoryID = cat.CategoryID GROUP BY cat.CategoryName
10	_context.Products.Include(p => p.Category).Include(p => p.Supplier).GroupBy(p => new { p.Category.CategoryName, p.Supplier.ContactName }).Select(g => new { name = g.Key.CategoryName, name2 = g.Key.ContactName, count = g.Count() });	SELECT [c].[CategoryName] AS [name], [s].[ContactName] AS [name2], COUNT(*) AS [count] FROM [Products] AS [p] LEFT JOIN [Categories] AS [c] ON [p].[CategoryID] = [c].[CategoryID] LEFT JOIN [Suppliers] AS [s] ON [p].[SupplierID] = [s].[SupplierID] GROUP BY [c].[CategoryName], [s].[ContactName]"	SELECT cat.CategoryName, supp.ContactName, Count(*) as ProductsCount FROM [NORTHWND].[dbo].[Products] as prod LEFT JOIN [NORTHWND].[dbo].[Categories] as cat ON prod.CategoryID = cat.CategoryID LEFT JOIN [NORTHWND].[dbo].[Suppliers] as supp ON prod.SupplierID = supp.SupplierID GROUP BY supp.ContactName, cat.CategoryName ORDER BY COUNT(*)
11	var query1 = _context.Products.Select(p=>p.ProductName).ToList(); var query2 = _context.Categories.Where(c => c.CategoryName != "Beverages").Select(c=>c.CategoryName).ToList(); var query3 = query1.Concat(query2);	SELECT [p].[ProductName] FROM [Products] AS [p] SELECT [c].[CategoryName] FROM [Categories] AS [c] WHERE [c].[CategoryName] <> N'Beverages'	SELECT CategoryName FROM [NORTHWND].[dbo].[Categories] as cat WHERE cat.CategoryName <> 'Beverages' UNION ALL SELECT ProductName FROM [NORTHWND].[dbo].Products

Висновки до розділу 4

У розділі описано процес досліду та результати аналізу часової ефективності запитів з використанням SQL та ORM Entity Framework.

Після наведених експериментів було виявлено такі закономірності:

— Найперший запит до бази даних з використанням ORM Entity Framework виконувався найдовше – у середньому у 220 разів довше, ніж запит з використанням SQL. Це зумовлене тим, що система повинна встановити

залежності між моделлю та базою. Звісно час виконання найпершого запиту може займати навіть довше, це залежить від розміру контексту даних та самого запиту.

— Наступні запити до бази, якщо вибірка застосовується вперше – то час, витрачений на обробку запиту з використанням ORM Entity Framework, довше в середньому на 55 %. Адже витрачається час на перетворення запиту LINQ на запит SQL та вже потім вмконується вибірка.

— При повторному виконанні запитів Entity Framework час витрачений на обробку вибірки суттєво зменшується, у середньому на 30%. Що у деяких випадках навіть менше, ніж запити написані на SQL. Це зумовлене тим, що вибірка уже трансформована у SQL та закешована.

— Запити SQL завжди показують відносно стабільний результат 3-30 мс.

— Деякі запити з використанням Entity Framework розбивалися на 2 запити, що збільшувало навантаженість на систему і час виконання. У той час, як з використанням SQL можна було об’єднати ці запити в 1.

Існують також деякі недоліки та незручності використання SQL:

— Необхідно кожен раз відкривати та закривати з’єднання до бази даних.

— Після отримання вибірки необхідно власноруч пройтись в циклі та занести вибірку у список чи масив, що займає додатково час та ускладнює написання коду.

— Необхідно враховувати також можливості власного ПК такі як: кількість ядер, процесор. Звичайно, якщо виконувати дослід на на більш потужному комп’ютері – то часові результати будуть кращими. Але слід розуміти, що цілковито катина не зміниться і завжди саме різниця у часі виконання буде майже однаковою.

Загальні висновки

У роботі було виконано дослідження часової ефективності запитів до бази даних з використанням ORM Entity Framework та SQL.

У ході розробки проекту використовувалися новітні технології. Завдяки дотриманню принципів об’єктно-орієнтованого дизайну та широкому використанню архітектурних шаблонів проектування, систему буде легко супроводжувати та вдосконалювати.

Найголовнішою метою даного дослідження було сформулювати висновок коли доцільно використовувати ORM Entity Framework, а коли SQL.

Звісно часова ефективність SQL була доведена, але й у ході розробки було доведено складнішу обробку отриманих даних з бази та написання самого запиту. Натомість Entity Framework не потребує ніякої особливої обробки отриманих даних з бази, адже вони вже у результаті перетворюються на об’єкти.

Додатково, з висновків до розділу 4, де описувалися саме кількісні характеристики, можемо сміло сказати, що найдовше обробляється саме найперший запит до бази, коли потрібно встановити усі залежності, далі запити з використанням Entity Framework та SQL мають різницю не більшу, ніж 50 мс. Така часова ефективність у Entity Framework досягнена завдяки механізму відстеження та кешування.

Із проведених досліджень можна сказати , що коли ми хочемо досягти більшого контролю над командами та операціями SQL за допомогою необроблених запитів SQL, тоді ADO.NET буде чудовим вибором для початку роботи. У той час якщо ми хочемо розробляти програму набагато швидше з чіткою можливістю обслуговування коду, Entity Framework буде кращим вибором. Крім того, ми можемо використовувати обидва підходи в одній програмі, як-от Entity Framework для операцій, пов’язаних із CRUD, і ADO.NET для отримання записів із бази даних для цілей звітування та операцій SQL, пов’язаних із масовими даними.

Було проведено опитування серед працівників крупної ІТ компанії України, яка працює з замовниками з усього світу та розроблює крупні проекти у різних сферах:

медицина, розумний дім, оподаткування, страхування, харчова промисловість, інтернет магазини. Через те, що проекти написані різними мовами проектування, було поставлене лише одне запитання «Які засоби використовуються для зв’язку з базою даних?». Також була ціль відслідкувати тенденцію впровадження чи вилучення використання ORM. Результати даного опитування наведені у додатку В.

ЕКОНОМІЧНА ЧАСТИНА ВИЗНАЧЕННЯ ВИТРАТ НА ПРОЕКТУВАННЯ ПРОГРАМИ

Техніко-економічне обґрунтування – це комплект розрахункових та аналітичних документів, які включають у себе основні технічні, організаційні, розрахункові та інші показники, що дозволяють оцінити доцільність та економічну стійкість проекту.

Метою такого обґрунтування є доказ того, що обрані технології, затверджений процес виробництва та організація роботи є оптимальною. Також важливо аргументувати, що даний проект не буде збитком на економічному рівні.

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях.

$$E = a \cdot S^b \cdot EAF, \qquad (5.1)$$

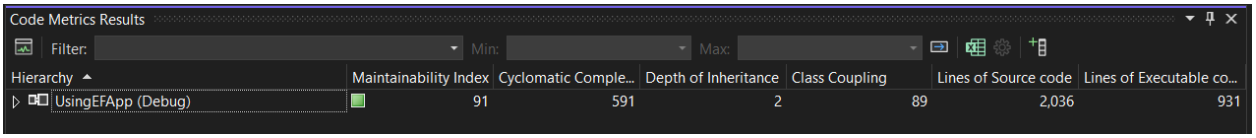
де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, a = 2,4; b = 1,05

Розмір програмного коду було підраховано за допомогою інструменту вбудованного у Visual Studio 2022 Community – Code Metrics Results.



Отже розмір програмного коду становить 2,036 рядки:

$$E = 2{,}4 \cdot 2{,}036^{1{,}05} \cdot 1 = 4{,}704$$

Згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 4,704 людино-місяці.

Нижче наведені розрахунки вартості розробки «Засіб порівняння запитів до

бази даних». Основними витратами прийнято:

- основна заробітна плата;
- відрахування на соціальні потреби (податки);
- накладні витрати;
- витрати на персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного забезпечення і визначається виходячи з кількості розробників, часу виконання розробки (годин), а також заробітної плати в розрахунку на одну годину. Розрахунок заробітної платні проводиться по формі табл. 5.1.

Таблиця 5.1 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	17000 [2]	1	4,7	79900

Описаний в проекті програмний продукт був розроблений одним програмістом в період з 1.06.22 до 5.11.22, що складає 22 робочих тиждені. Витрати робочого часу було прийнято за 40 годин у тиждень. Погодинна ставка досить кваліфікованого інженера–програміста складає 200,75 грн/год. Таким чином, пораховано витрачено робочого часу:

$$t_{\text{розробки}} = N_{\text{чол}} \times N_{\text{тиж}} \times N_{\text{год}}, \quad (5.2)$$

де $N_{\text{чол}}$ – кількість виконавців, чол;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, год;

$$t_{\text{розробки}} = 1 \cdot 22 \cdot 40 = 880 \text{ чол/год.}$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5.3)$$

де $t_{\text{розробки}}$ – витрати праці у чол/год;

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. Коефіцієнт кваліфікації розробника (k) - ступінь підготовленості виконавця до дорученої йому роботі (він визначається залежність від стажу праці та становить:

- для працюючих до 2 років- 0,75;
- від 2 до 3 років 1,0;
- від 3 до 5 років - 1,1-1,2;
- від 5 до 7 років - 1,3-1,4;
- понад 7 років - 1,5-1,6.

В даному випадку KKB приймається 0,75. ОЗП складає:

$$\text{ОЗП} = 880 \cdot 200{,}75 \cdot 0{,}75 = 132495 \text{ грн.}$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати (22% [4]):

$$C_{\text{соц}} = \frac{\text{ОЗП} \cdot 22\%}{100\%} \quad (5.4)$$

$$C_{\text{соц}} = \frac{132495 \cdot 22\%}{100\%} = 29149 \text{грн.}$$

Отримані результати за (5.3) та (5.4) підсумовуються. Вони складають 156344 грн. та визначають основні прямі витрати.

Накладн витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються у відсотках (30 – 40%) від суми прямих витрат:

$$С_{накл} = \frac{(ОЗП + С_{соц}) \cdot 35\%}{100\%}; \quad (5.5)$$

$$С_{накл} = \frac{(132495 + 29149) \cdot 35\%}{100\%} = 56575 \text{ грн.}$$

Протягом усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на персональний комп’ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп’ютеру. В експлуатаційні витрати входять:

- вартість витратних матеріалів;
- витрати на ремонт;
- заробітна плата ремонтника;
- оренда приміщення;
- додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;
- амортизаційні витрати на персональний комп'ютер і програмне забезпечення;
- витрати на електроенергію ($С_{ел}$),

які визначаються за формулою:

$$С_{ел} = P \cdot B \cdot T_{розр}, \quad (5.6)$$

де Р – потужність комп’ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/годин для непобутових споживачів,

складає 4,7 грн;

Tрозр – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{\text{ел}} = 0{,}45 \cdot 4{,}7 \cdot 880 = 1861 \text{ грн.}$$

Витрати на витратні матеріали ($\text{С}_{\text{вм}}$) протягом всього терміну експлуатації приблизно 10% від вартості комп'ютеру. Вартість робочої станції приймається 20 000 грн. [6], термін експлуатації – 5 років. Отже, можна визначити ці витрати за період створення програмного засобу:

$$\text{С}_{\text{вм}} = \text{В}_{\text{ком}} \cdot \frac{N_{\text{Д}}}{N_{\text{експ}} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (5.7)$$

де $B_{КОМ}$ – вартість персонального комп'ютеру;

$N_{\text{Д}}$ – кількість днів розробки програмного продукту;

$N_{\text{експ}}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

Витрати на витратні матеріали визначаються так:

$$C_{\text{вм}} = 20000 \cdot \frac{110}{5 \cdot 365} \cdot \frac{10}{100} = 120 \text{ грн.}$$

Заробітна плата ремонтника ($C_{\text{рем}}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 10000 грн. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$C_{\text{рем}} = \frac{C_{\text{рем}}'}{N_{КОМ}} \cdot T_{\text{міс}}, \quad (5.8)$$

де $C_{\text{рем}}'$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{мес}$ – час розробки програмного продукта, міс.

Заробітна плата ремонтника ($C_{\text{рем}}$) буде складати:

$$C_{\text{рем}} = \frac{10000}{50} \cdot 4{,}7 = 940 \text{ грн.}$$

За статистикою витрати на комплектуючи вироби (C_{KOM}) для ремонту персонального комп'ютера складаєм 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{KOM} = C_{BM} = 120\text{грн.} \quad (5.9)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$\text{АКП} = \text{В}_{KOM} \cdot \frac{N_{\text{Д}}}{N_{\text{експ}} \cdot 365}; \quad (5.10)$$

$$\text{АКП} = 20000 \cdot \frac{4{,}7}{3 \cdot 12} = 2611$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для Windows 5 років та Visual Studio за 2 рік то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп'ютера використовувалася операційна система Windows 10, для написання програмного забезпечення - програмне середовище Visual Studio 2019 Community.

$$\text{АКП}_w = 13449 \cdot \frac{4{,}7}{5 \cdot 12} = 1053{,}5$$

$$\text{АКП}_w = 0 \cdot \frac{4{,}7}{2 \cdot 12} = 0$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2. Додаткові витрати ($C_{\text{дод}}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера-програміст, тобто 8500 гривень на місяць.

Оренду приміщень для однієї людини приймемо рівною 3500 гривень на місяць [5]. Тобто за весь період розробки – 16450 грн. Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (5.11)$$

$$C_{\text{експ}} = 665.3 + 120 + 940 + 2611 + 1053{,}2 + 16450 + 8500 = 30339{,}50 \text{ грн}$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, грн	Джерело придбання	Амортизаційні відрахування, грн
Windows 10	13449	https://www.moyo.ua/ua/po_microsoft_windows_pro_for_workstations_10_64bit_eng_intl_1pk_oem_dvd_hzv-00055_/499273.html	1053,2
Visual Studio 2019 Community	0	https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes	0
Всього:	13449		1053,2

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	665

Вартість витратних матеріалів	120
Витрати на ремонт	940
Амортизація персонального комп’ютера	2611
Амортизація програмного забезпечення	1053
Оренда приміщення	16450
Додаткові витрати	8500
Всього	30339

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \qquad (5.12)$$

$$C_{\text{розробки}} = 132495 + 29149 + 56575 + 30339 = 248558\text{грн.}$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	132495
Відрахування на соціальні потреби	29149
Накладні витрати	56575
Експлуатаційні витрати	30339
Всього	248558

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку сучасного програмного забезпечення для порівняння запитів до бази даних. За результатами розрахунків, приблизна вартість розробки складає 248558 грн.

Список матеріалів

1. Формули 5.1 – 5.12 http://repository.enu.kz/bitstream/handle/data/12881/METODIKA-TRUDOZATRAT.pdf
2. https://jobs.dou.ua/salaries/#period=jun2021&city=Dnipro&title=Junior%20Software%20Engineer&language=C%23%2F.NET&spec=&exp1=0&exp2=1
3. http://mpe.kmxu.gov.ua/minugol/control/publish/article?art_id=245583544
4. https://services.dtkt.ua/catalogues/indexes/13
5. https://besplatka.ua/ru/dnepropetrovsk/nedvizhimost/arenda-komercheskoy-nedvizhimosti/arenda-ofisov
6. https://rozetka.com.ua/dell_n3001vn3500ua03_2201_ubu/p290435868/

Використані джерела:

1. https://entityframework.net/what-is-ef
2. https://entityframework.net/why-first-query-slow
3. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/downloading-sample-databases

ДОДАТОК А
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор українського державного
університету науки і технологій
Б. Є. Боднар
2.10.22

Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.00001-01-ЛЗ

Завідувач кафедри КІТ
В. Горячкін
28.10.22
Керівник розробки
О. П. Іванов
25.10.22
Виконавець
Д. С. Назаренко
23.10.22
Нормоконтролер
С. А. Волкова
27.10.22

2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО
1116130.00001-01

Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL

Технічне завдання

Аркушів 8

2022

1. Вступ

ORM (Object-Relational Mapping) – технологія, яка зв’язує бази даних з концепціями об’єктно орієнтованого програмування та дозволяє писати запити до бази даних використовуючи парадигму об’єктно орієнтованого програмування на зручній, для користувача, мові програмування.

Буде проведено дослідження доцільності використання ORM у порівнянні з SQL запитами, а саме: створення таблиці, додавання рядків у таблицю, проста вибірка, складна вибірка, видалення рядків з таблиці, видалення таблиці.

Актуальністю даного дослідження є попит на використання ORM у сучасному програмуванні, а також суперечні твердження щодо погіршення швидкодії та оптимальності виконання програми.

Дослідження буде корисне розробникам додатків з великою кількістю маніпуляцій даними.

2. Підстави для розробки

Підставою для розробки є написання магістерської роботи на замовлення кафедри КТС українського державного університету науки і технологій.

Тема розробки: проведення дослідження щодо доцільності використання ORM у порівнянні з SQL.

3. Призначення розробки

Функціональне призначення:

Створення нової таблиці за допомогою ORM Entity Framework;

Додавання нового рядка в таблицю за допомогою ORM Entity Framework;

Видалення рядка з таблиці за допомогою ORM Entity Framework;

Редагування рядка в таблиці за допомогою ORM Entity Framework;

Вибірка за допомогою ORM Entity Framework;

Створення нової таблиці за допомогою SQL;

Додавання нового рядка в таблицю за допомогою SQL;

Видалення рядка з таблиці за допомогою SQL;

Редагування рядка в таблиці за допомогою SQL;

Вибірка за допомогою SQL.

Експлуатаційне призначення:

Створення програми викликано необхідністю у проведенні дослідження «Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL» для визначення швидкодії та оптимальності роботи механізму ORM Entity Framework.

4. Вимоги до програми

Вимоги до функціональних характеристик

Обробка тестової бази даних взятої з офіційного сайту Microsoft [3]. Створення власної таблиці з 100 рядками.

Користувацький інтерфейс відсутній, адже програма виконуватиметься лише для зібрання інформації для дослідження.

Формат вхідних даних – SQL чи ORM Entity Framework запит до бази даних.

Формат вихідних даних – час, за який обробився запит.

Вимоги до надійності

Надійність системи визначатиме безпосередній потік запитів до бази даних, у разі невиконання запиту повинно відкритися вікно з описом помилки.

Умови експлуатації

Експлуатація передбачає підтримку програми та збереження бази даних на локальному комп’ютері.

Вимоги до складу і параметрів технічних засобів

Програмний продукт розрахований для функціонування на ПК, що має такі характеристики:

Маніпулюючий пристрій миша;

Клавіатура;

Процесор від Core I3;

Внутрішня пам’ять від 8гб.

Вимоги до інформаційної та програмної сумісності

Для використання додатка необхідній комп’ютер з Windows 7 і новіше. VisualStudio 2019, Microsoft SQL Server Management Studio 18, SQL Server Profiler 18, Microsoft SQL Server 2017. Мова програмування C# та SQL.

Вимоги до маркування і упаковки

Продукт не є комерційним та не виробляється з метою розповсюдження. Метою продукту є письмове дослідження.

5. Вимоги до програмної документації

До складу програмної документації входить:

Специфікація;

Окремі пункти тексту програми;

Результати дослідження.

6. Стадії та етапи розробки

№	Зміст роботи	Строки виконання
1	Постановка задачі	1.09.2021-24.12.2021
2	Програмування і відладка програми	1.01.2022-1.10.2022
3	Збір необхідної інформації для статті	1.01.2022-1.10.2022
4	Презентація дослідження	25.12.2022

ДОДАТОК Б

Порівняння роботи запитів до бази даних з використанням ORM Entity Framework та SQL

Назаренко Дар’я Сергіївна, Іванов Олександр Петрович,Український державний університет науки і технологій

Кожен зі світу розробки хоч раз чув термін як «Object-Relational-Mapper», або скорочено «ORM». Гарна частина в ORM це звісно те, що її використання спрощує написання коду для програміста. Однак є багато незрозумілого в самому принципі дії та у тому, як використання ORM впливатиме на роботу усього додатку, як погіршить чи покращить перфоманс.

Мета і завдання дослідження спрямоване на виявлення механізму роботи ORM Entity Framework, а саме конвертацію в запит мовою SQL та його оптимальність за часом та ресурсоємністтю.

Наукова цінність даного дослідження полягає в необхідності розробникам використовувати та обробляти дані з бази даних, отже є необхідність у визначені кращого шляху обробки даних.

З огляду на вищесказане під час дослідження були поставлені наступні задачі:

— розробити додаток для порівняння часової ефективності запитів;
— знайти тестову базу даних для проведення експериментів;
— розробити запити різної складності;
— проаналізувати роботу ORM та SQL.

Отже, можна виділити наступні етапи проведення дослідження:

— формулювання задачі та постановка проблеми для якої визначаються цілі;
— аналіз об’єктів дослідження, порівняння програмних аналогів та оцінка обраної технології;
— розробка середовища для проведення експериментів.

Було розроблено додаток, який виконує один і той самий запит до БД за допомогою ORM та SQL і на виході отримуємо час, за який обробився даний запит. За допомогою SQL SERVER PROFILER була змога подивитися та оцінити конвертацію запитів з використанням ORM на мову запитів SQL, що дало змогу оцінити оптимальність роботи обраної ORM.

Таким чином, у кінці дослідження було сформульовано висновки та оцінку роботи ORM, додатково було запропоновано оптимальне рішення коли і як доцільно використовувати ORM.

І, як доказ того, що ORM широко використовується на великих Ентерпрайз проектах, було проведено опитування серед працівників крупної ІТ компанії України, яка працює з замовниками з усього світу та розроблює крупні проекти у різних сферах: медицина, розумний дім, оподаткування, страхування, харчова промисловість, інтернет магазини. Через те, що проекти написані різними мовами програмування, було поставлене лише одне запитання «Які засоби використовуються для зв’язку з базою даних?». Також була ціль відслідкувати тенденцію впровадження чи вилучення використання ORM.

ДОДАТОК В

Назаренко Дар’я Сергіївна

Студентка гр. ПЗ2121М

Оцінка тенденції використання ORM для запитів до бази даних

Анотація. Досліджено тенденцію використання ORM в одній з найкрупніших IT компанії України методом соціального опитування.

Вступ. Кожен зі світу розробки хоч раз чув термін як «Object-Relational-Mapper», або скорочено «ORM». Гарна частина в ORM це звісно те, що її використання спрощує написання коду для програміста.

У цій статті коротко розберемо що таке ORM найважливіші плюси та мінуси використання та результати соціального опитування використання ORM на Ентерпрайз проектах.

Почнемо з самого визначення що таке Object-Relational-Mapper. Це ідея можливості писати запити до бази даних використовуючи об’єктно-орієнтовану парадигму будь-якої мови програмування. Іншими словами, це спосіб взаємодіяти з базою даних використовуючи обрану мову замість SQL. Отже, коли йдеться про ORM – зазвичай мається на увазі бібліотека, яка дозволяє взаємодіяти з базою.

Наразі існує неймовірна кількість ORM, деякі все ж таки залишаються більш популярними через кращу інтеграцію з обраною мовою програмування, нижче наведено кілька найбільш популярних бібліотек для різних мов:

1. NHibernate – одна з перших популярних ORM, написана для Microsoft .net та Java. Має відкритий код і є абсолютно безкоштовною. Перший випуск був у 2007 році, а останній у 2018.
2. Entity Framework – вважається найпопулярнішою ORM для Microsoft .net адже випускаються однією компанією і має найкращу інтеграцію. Перший випуск був у 2008 році і фреймворк постійно оновлюється з виходом нової версії .net.
3. Prisma – це сучасний фреймоворк, створений для мов програмування: Node.js і TypeScript. Перший випуск був у 2019 році і досі ведеться активна підтримка і оновлення цієї бібліотеки.

Які переваги використання ORM?

1. Все одно можна писати тією мовою, яка вже використовується;
2. Завдяки ORM абстрагується система бази даних, тим самим вводиться додатковий рівень абстракції;
3. Оскільки ORM – це бібліотека, вона має багато функцій «з коробки» такі як: підтримка транзакцій, міграцій, потоки та інші вже влаштовані функції;
4. Багато запитів працюватимуть швидше, адже новітні ORM підтримують механізм кешування.

Які мінуси використання ORM?

1. Якщо мова SQL не викликає труднощів – то запити можуть бути написані більш ефективно;
2. Робота з бібліотекою потребує додаткових знань;
3. Розробнику важливо розуміти, що відбувається всередині ORM, оскільки невдале використання бібліотеки може сильно вплинути на продуктивність роботи програми.

І, як доказ того, що ORM широко використовується на великих Ентерпрайз проектах, було проведено опитування серед працівників крупної IT компанії України, яка працює з замовниками з усього світу та розроблює крупні проекти у різних сферах: медицина, розумний дім, оподаткування, страхування, харчова промисловість, інтернет магазини. Через те, що проекти написані різними мовами програмування, було поставлене лише одне запитання «Які засоби використовуються для зв’язку з базою даних?». Також була ціль відслідкувати тенденцію впровадження чи вилучення використання ORM.

Для зручності дослідження було створено 5 категорій проектів:

Перша – проект написаний в період 2000 – 2005 рр;

Друга – проект написаний в період 2006 – 2010 рр;

Третя – проект написаний в період 2011 – 2015 рр;

Четверта – проект написаний в період 2016 – 2020 рр;

П’ята – проект написаний в період 2020 – 2022 рр.

Також, кожен проект матиме групу активної розробки чи підтримки вже існуючого функціоналу.

Було проведено опитування 20 працівників, працюючих, на різних проектах, результати наведені нижче у таблиці:

Група за роками	Стадія проекту	Використання ORM	Використання SQL
Четверта	active	+	+
Четверта	active	+	+
Четверта	active	+	+
Друга	support	-	+
Друга	support	+	+
П’ята	active	+	+
Четверта	active	+	+
Четверта	active	-	+
Четверта	active	+	+
П’ята	active	-	+
Перша	support	-	+
Третя	active	+	+
Четверта	active	+	+
Третя	support	+	+
Третя	support	+	+
Четверта	support	+	+
Четверта	support	+	+
П’ята	active	+	+
Перша	support	-	+
Третя	active	+	+

З наведеної таблиці бачимо, що 15 з 20 проектів використовують ORM для зв’язку з базою із цих 15, 8 знаходиться у стадії підтримки. 5 проектів не використовують ORM, з них 3 знаходяться у стадії підтримки.

Якщо оцінювати сама тенденцію впровадження ORM – то можна побачити, що проекти першої групи (до 2005 р.) – використовують лише SQL. Починаючи з Другої групи, проекти вже почали використовувати ORM, але все ж таки є деякі, їх меншість, які відмовляються від ORM.

З наведеного дослідження можемо зробити висновок, що використання ORM є досить популярним, але все ж таки від SQL повністю не відмовляються, найпопулярнішим є саме комбінований спосіб написання запитів до бази даних.