ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Бази даних для оптимізації роботи з великими обсягами інформації»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем ОМЕЛЬЧЕНКО
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-43
Артем ОМЕЛЬЧЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Іван ШАХМАТОВ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність Інформаційні системи та технлогії
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____Катерина НЕСТЕРЕНКО
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Омельченко Артем Вадимович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Бази даних для оптимізації роботи з великими обсягами інформації
керівник кваліфікаційної роботи Іван ШАХМАТОВ
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» березня 2025 р. № 56
2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи:
1.Науково-технічна література з теми бакалаврської роботи.
2.Принципи функціонування серверних СУБД при масштабних навантаженнях
3.Основні принципи та методи аналізу і оптимізації баз даних

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1.Теоретичні засади оптимізації баз даних при роботі з великими обсягами інформації.
2.Аналіз об'єкта дослідження та оцінка ефективності функціонування бази даних.
3.Практична реалізація та оцінка ефективності оптимізації бази даних.

5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	30.02.25-05.03.25	
2	Аналіз великих баз даних	05.03.25-15.03.25	
3	Вступ	15.03.25-17.03.25	
4	Написання першого розділу	17.03.25-25.03.25	
5	Написання другого розділу	25.03.25-12.04.25	
6	Написання третього розділу	12.04.25-30.04.25	
7	Висновки	30.04.25-05.05.25	
8	Оформлення роботи	05.05.25-15.05.25	

Здобувач(ка) вищої освіти _____ Артем ОМЕЛЬЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Іван ШАХМАТОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра : 50 стор., 29 рис., 9 табл., 20 джерел.

Мета роботи – дослідити особливості зберігання і обробки великих обсягів даних у сучасних системах управління базами даних, а також дослідити та реалізувати підходи до їх оптимізації з метою підвищення продуктивності, масштабованості та швидкодії.

Об'єкт дослідження – процеси зберігання, обробки та оптимізації структур баз даних у системах управління базами даних, при роботі з великими обсягами даних

Предмет дослідження – методи аналізу, моделювання та оптимізації структур баз даних, спрямовані на забезпечення ефективної роботи з великими обсягами даних у середовищі сучасних СУБД .

Короткий зміст роботи: *У роботі досліджено особливості побудови та оптимізації баз даних для роботи з великими обсягами інформації, також розглянуто проблеми продуктивності при обробці великих даних, зокрема на прикладі бази IMDb.*

У практичній частині реалізовано структуру бази даних, виконано завантаження даних, проведена ключова оптимізація. Проведено порівняльне тестування до та після оптимізації.

КЛЮЧОВІ СЛОВА: БАЗА ДАНИХ, ОПТИМІЗАЦІЯ, ВЕЛИКІ ОБСЯГИ ДАНИХ, ІНДЕКСАЦІЯ, ПРОДУКТИВНІСТЬ, СУБД, ІНФОРМАЦІЯ, SQL.

ABSTRACT

Text part of the master's qualification work: 50 pages, 29 pictures, 9 table, 20 sources.

The purpose of the study - is to investigate the features of storing and processing large amounts of data in modern database management systems, as well as to investigate and implement approaches to their optimization in order to increase productivity, scalability and speed.

Object of research - the processes of storing, processing and optimizing database structures in database management systems when working with large amounts of data

Subject of research - methods of analysis, modeling and optimization of database structures aimed at ensuring efficient work with large amounts of data in the environment of modern DBMSs .

Summary of the work: The paper investigates the peculiarities of building and optimizing databases for working with large amounts of information, and also considers performance problems in processing big data, in particular, using the example of the IMDb database.

In the practical part, the database structure is implemented, data is loaded, and key optimization is performed. Comparative testing before and after optimization is carried out.

KEYWORDS: DATABASE, OPTIMIZATION, LARGE AMOUNTS OF DATA, INDEXING, PERFORMANCE, SUBD, INFORMATION, SQL.

ЗМІСТ

ВСТУП

Актуальність теми: У сучасному інформаційному суспільстві обсяги даних зростають в геометричній прогресії, що зумовлено широким впровадженням цифрових технологій у всіх сферах діяльності.Особливо це актуально для України, яка активно впроваджує цифрові технології в різних сферах: економіці, медицині, державному управлінні, освіті тощо. У зв'язку з цим значно зросли вимоги до ефективності систем зберігання та обробки даних. Центральне місце в цих процесах займають бази даних, які забезпечують структуроване зберігання, доступ, оновлення та аналіз інформації.

Однак зі збільшенням обсягів даних традиційні підходи до проектування та використання баз даних стикаються з низкою обмежень. Проблеми повільної обробки запитів, перевантаження системи, нераціонального використання пам'яті та обчислювальних ресурсів характерні для погано оптимізованих баз даних. Це особливо актуально для організацій, де обсяги інформації можуть досягати терабайт і петабайт. В таких умовах оптимізація структури бази даних стає критичним фактором забезпечення стабільної, швидкої та масштабованої роботи інформаційної системи.

Вивчення та впровадження ефективних методів аналізу та оптимізації структури баз даних дозволяє значно скоротити час виконання запитів, підвищити продуктивність системи, покращити індексування, зменшити дублювання даних та скоротити витрати обчислювальних ресурсів. Актуальність дослідження підтверджується також тим, що ринок інформаційних технологій потребує фахівців, які вміють не тільки створювати бази даних, але й оптимізувати їх для роботи з великими інформаційними потоками.

Таким чином, дослідження в галузі аналізу та оптимізації структур баз даних для великих обсягів даних є актуальними та практично значущими. Воно відповідає викликам сучасної цифрової трансформації та має безпосереднє застосування для розробки продуктивних та надійних інформаційних систем.

Мета дослідження: підвищити ефективність роботи з великими обсягами даних шляхом оптимізації структури бази даних з урахуванням вимог до швидкодії та стабільності.

Завдання дослідження:

— Вивчити теоретичні основи побудови та оптимізації структур баз даних.

— Проаналізувати особливості роботи з великими обсягами даних та їх вплив на структуру БД.

— Дослідити чинники, котрі впливають на продуктивність баз даних, та визначити найбільш критичні з них.

— Спроектувати базу даних з урахуванням виявлених вимог та обмежень.

— Реалізувати проектну модель у середовищі СУБД.

— Провести тестування продуктивності та зафіксувати результати.

— Виявити вузькі місця у роботі бази даних та запропонувати шляхи їх усунення.

— Здійснити оптимізацію структури бази даних і провести повторне тестування.

—Зробити висновки щодо ефективності запропонованих змін.

Об'єкт дослідження: процес проєктування та функціонування структур баз даних в умовах роботи з великими обсягами інформації. Цей процес охоплює методи організації даних, забезпечення їх цілісності, доступності, а також впливає на продуктивність, масштабованість та ефективність інформаційних систем.

Предмет дослідження: методи та засоби аналізу й оптимізації структур баз даних, що безпосередньо впливають на ефективність зберігання та обробки великих обсягів даних.

Методи дослідження:

—Аналіз наукової літератури та публікацій з даної теми для вивчення сучасних підходів до побудови та оптимізації структур баз даних в умовах роботи з великими обсягами інформації.

—Порівняльний аналіз для оцінки ефективності різних структур баз даних за критеріями швидкодії, навантаження та масштабованості.

—Моделювання для розробки логічної та фізичної структури бази даних, що відповідає вимогам продуктивності та надійності.

—Експериментальний метод для тестування продуктивності бази даних до та після оптимізації на основі створеного тестового навантаження.

—Системний аналіз для оцінки результатів оптимізації та визначення можливих шляхів подальшого вдосконалення структури бази даних.

Очікувані результати:

—Очікується підвищення швидкодії запитів завдяки впровадженню методів індексації та партиціювання даних.

—Зниження часу відповіді на запити шляхом коригування структури таблиць та налаштувань індексів.

—Порівняння результатів до та після оптимізації для визначення покращень у швидкості виконання операцій.

Практична значимість:

Практичне значення цієї кваліфікаційної роботи полягає в тому, що знання та методи, розглянуті в ході її виконання, можна застосовувати на підприємствах, що працюють з великими обсягами даних. Оптимізація структур баз даних дозволить відчутно підвищити ефективність обробки та зберігання інформації, зменшити час відповіді на запити, забезпечити масштабованість та стабільність систем у разі збільшення обсягів даних. Розроблені підходи можуть бути впроваджені в існуючі інформаційні системи для підвищення їх продуктивності та надійності, особливо в таких сферах, як фінансові установи, великі торгові платформи, системи управління даними, де обробляються великі обсяги даних.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ ОПТИМІЗАЦІЇ БАЗ ДАНИХ ПРИ РОБОТІ З ВЕЛИКИМИ ОБСЯГАМИ ІНФОРМАЦІЇ

1.1 Огляд наукових джерел за напрямом дослідження

Оптимізація структур баз даних в умовах обробки великих обсягів даних є однією з провідних тем сучасних досліджень в сфері інформаційних технологій. З розвитком цифрових систем стрімко зростають обсяги даних, що підлягають зберіганню та обробці, що висуває нові вимоги до побудови та організації структур баз даних. Ефективність зберігання, доступу та обробки інформації значною мірою залежить від того, як побудована внутрішня схема бази даних.

Одним із ключових напрямів досліджень є підвищення продуктивності бази даних за рахунок структурної реорганізації: зміна методів зберігання, створення індексів, використання фрагментації таблиць, кешування агрегованих значень тощо. Провідні дослідники відзначають, що при роботі з великими даними традиційні методи, такі як повна нормалізація, втрачають свою ефективність і часто замінюються денормалізованими або гібридними структурами, особливо в аналітичних системах.

Не менш важливим питанням є індексування. Дослідники пропонують використовувати не тільки класичні дерева B+, але й нові підходи, такі як адаптивні або навчальні індекси, які створюються динамічно під час виконання запиту. Це може підвищити продуктивність без необхідності втручання адміністратора бази даних.

Останні дослідження також підкреслюють важливість оптимізації в хмарних або розподілених середовищах, де структура бази даних має бути спроектована таким чином, щоб забезпечувати швидкий доступ до даних, балансування навантаження і відмовостійкість. У цьому контексті оптимізація структури включає реплікацію, шардінг, створення багаторівневих сховищ та інші підходи до масштабування.

Незважаючи на наявність великої кількості досліджень, проблема універсальної та ефективної оптимізації структури баз даних для великих обсягів даних залишається актуальною та потребує подальшого теоретичного опрацювання й практичної перевірки на практиці.

1.2 Аналіз різних методів та концепцій оптимізації баз даних

У процесі оптимізації баз даних існує безліч методів, кожен з яких має свої переваги й недоліки. Одним із ключових підходів є нормалізація — це розділення таблиць на менші логічні одиниці з метою усунення надмірного дублювання даних та забезпечення їхньої цілісності. Такий підхід дозволяє краще структурувати інформацію, однак ускладнює запити, оскільки збільшується кількість об’єднань (JOIN), що в свою чергу може негативно позначатися на швидкодії.

Іншим підходом, який часто використовується у протилежних ситуаціях, є денормалізація. Тут ідеться про навмисне об’єднання таблиць для пришвидшення доступу до даних, особливо при виконанні складних вибірок (SELECT). Це зменшує потребу в частих об'єднаннях, що позитивно впливає на продуктивність, проте призводить до дублювання інформації, що ускладнює її оновлення та контроль цілісності.

Ще одним критично важливим методом є індексування, коли на певні стовпці створюються індекси, що значно пришвидшують операції пошуку. Це особливо помітно при великій кількості рядків у таблицях. Але індекси мають свою ціну — вони уповільнюють операції вставки та оновлення, і займають додаткове місце у сховищі.

Коли обсяги даних зростають настільки, що одна база даних вже не справляється, вдаються до шардингу — тобто поділу бази на менші фрагменти, які розміщуються на окремих серверах. Це дозволяє масштабувати систему горизонтально й зберігати стабільність роботи навіть при величезних обсягах даних. Проте реалізація такого підходу вимагає складної логіки обробки запитів та особливої уваги до транзакційності.

Для підвищення швидкості доступу до часто використовуваної інформації використовують кешування — зберігання результатів запитів в оперативній пам’яті або спеціалізованих кеш-сервісах, таких як Redis чи Memcached. Це зменшує навантаження на базу, однак може спричинити проблеми з актуальністю даних, а також потребує додаткової логіки синхронізації.

Оптимізувати роботу з базою даних також можна на рівні самих запитів. Так звана оптимізація SQL-запитів передбачає їхній аналіз і переписування з метою зменшення споживання ресурсів. Це дозволяє покращити продуктивність без зміни структури БД, але вимагає високого рівня експертизи й глибокого розуміння принципів роботи SQL.

Коли таблиці містять величезні обсяги даних, корисною може стати парцеляція — розбиття таблиці на частини за певною ознакою, наприклад, за датою. Це дозволяє прискорити пошук, однак такий підхід не завжди підтримується СУБД, а управління подібною структурою може бути доволі складним.

Для прискорення доступу до агрегованих або складних результатів запитів застосовують матеріалізовані подання — спеціальні об’єкти в базі даних, які зберігають проміжні результати і періодично оновлюються. Це значно підвищує швидкодію при читанні даних, але існує ризик, що інформація буде застарілою, якщо оновлення не відбувається своєчасно.

Важливим елементом є також архівування історичних даних, тобто винесення неактивної або застарілої інформації в окремі сховища чи таблиці. Це дозволяє зменшити навантаження на активну базу даних, але ускладнює доступ до таких даних і потребує розробки додаткових механізмів їх обробки.

У випадках, коли класичні реляційні СУБД не справляються з поставленими завданнями, доцільним може бути використання спеціалізованих систем управління базами даних, таких як NoSQL-рішення. Вони краще підходять для зберігання неструктурованих або великих обсягів інформації, однак мають свої обмеження — зокрема, відсутність стандартного SQL та необхідність кастомної логіки взаємодії.

Нарешті, невід’ємною частиною підтримки продуктивності є профілювання та моніторинг, який дозволяє виявити «вузькі місця» в системі — наприклад, за допомогою таких інструментів, як EXPLAIN, pg_stat_statements або SQL Profiler. Вони не усувають проблеми безпосередньо, але допомагають точно визначити, де саме потрібні зміни — у структурі таблиць чи в запитах.

У підсумку, ефективна оптимізація бази даних — це поєднання різних підходів, що потребує комплексного аналізу потреб системи, обсягів даних, навантаження і специфіки використання. Кожен із методів має як переваги, так і обмеження, тож їх застосування вимагає зваженого підходу.

1.3 Особливості роботи з великими обсягами даних

У сучасну епоху цифрової трансформації ми живемо в умовах постійного зростання обсягів даних. Інформація надходить до нас звідусіль — від сенсорів і смарт-пристроїв, що фіксують усе навколо в режимі реального часу, до веб-сервісів, додатків, платформ електронної комерції та соціальних мереж, які щосекунди генерують неймовірні масиви цифрового контенту. Усе це перетворює великі обсяги даних не лише на виклик для зберігання й обробки, а й на цінний ресурс, здатний стати драйвером розвитку бізнесу, науки та державного управління.

Робота з такими масивами інформації має свої характерні особливості, і перш за все — це швидкість, із якою дані потрібно не лише зібрати, а й проаналізувати. Часом затримка навіть у кілька секунд може виявитися критичною, особливо у випадках, коли йдеться про фінансові операції, інтелектуальні системи ухвалення рішень або моніторинг складних технологічних процесів. Саме тому системи, які обробляють дані, повинні бути здатні діяти блискавично — реагувати на потоки даних майже миттєво.

Другою важливою особливістю є масштаб — великі обсяги даних вимагають відповідних ресурсів. Традиційні підходи до зберігання й обробки часто виявляються недостатніми. Потрібні нові архітектури — розподілені

файлові системи, паралельні обчислення, масштабовані бази даних, що забезпечують не лише збереження гігантських масивів, а й ефективний доступ до них у будь-який момент.

Не менш серйозним викликом є різноманітність джерел та форматів даних. У сучасних інформаційних системах можна зустріти структуровані дані з класичних реляційних баз, напівструктуровані дані у форматі XML чи JSON, а також зовсім неструктуровану інформацію — текстові документи, зображення, відео чи аудіо. Поєднання всіх цих типів даних у єдиному аналітичному середовищі — завдання складне, що вимагає спеціалізованих інструментів та ретельно продуманої стратегії інтеграції.

Ще один аспект, який часто недооцінюють, але який має колосальне значення, — це якість даних. У великих масивах навіть незначна помилка або шум можуть призвести до суттєвих викривлень результатів аналізу. Тому надзвичайно важливо вміти проводити очистку, валідацію та нормалізацію даних на етапі підготовки, щоб результати, отримані в процесі обробки, були достовірними й надійними.

Нарешті, безпека та конфіденційність — це ключові питання при роботі з великими обсягами інформації, особливо якщо дані зберігаються й обробляються в розподіленому середовищі, наприклад, у хмарних інфраструктурах. З одного боку, потрібно забезпечити доступ до даних для авторизованих користувачів, а з іншого — гарантувати, що сторонні особи не матимуть змоги отримати доступ до конфіденційної інформації. Це потребує впровадження сучасних методів шифрування, багаторівневої автентифікації, а також постійного моніторингу безпеки системи.

Таким чином, робота з великими обсягами даних — це багатогранний процес, що охоплює питання швидкості, масштабу, різноманітності, якості та безпеки. Лише комплексний підхід, що враховує всі ці аспекти, дозволить ефективно використовувати потенціал великих даних у різних сферах людської діяльності.

Незважаючи на те, що робота з великими обсягами даних може бути складною, вона також створює багато можливостей для розвитку науки та практичних застосувань. Одна з найкращих переваг полягає в тому, що ви можете дізнатися про нові речі, які не могли б знайти за допомогою звичайних методів. Уважно вивчаючи дані та виявляючи закономірності, компанії можуть ефективніше реагувати на зміни, приймати обґрунтовані рішення та прогнозувати, наскільки добре вони будуть працювати.

У бізнесі великі дані допомагають краще зрозуміти, чого хочуть клієнти, і надавати їм кращий сервіс. Це основа для кастомізації послуг, автоматизації процесів та оптимізації ресурсів. Зокрема, компанії можуть аналізувати великі масиви транзакцій для виявлення підозрілих операцій або формувати індивідуальні рекомендації для онлайн-покупців, що, у свою чергу, сприяє підвищенню рівня задоволеності клієнтів та зростанню обсягів продажів.

У сфері охорони здоров'я їх використовують для прогнозування епідемій, аналізу ефективності лікування та управління лікарськими ресурсами. У наукових дослідженнях - для створення моделей складних процесів, аналізу експериментальних даних і перевірки гіпотез.

Великі дані допомагають розробляти нові способи планування міст (розумні міста), покращувати логістичні системи, підвищувати ефективність державного управління тощо. Все це робить їх потужним інструментом для інновацій, що дозволяє перетворювати сирі дані на цінні знання та практичні рішення.

1.4 Методи та інструменти аналізу продуктивності баз даних

Продуктивність бази даних є критично важливим параметром, який визначає здатність системи ефективно обробляти запити користувачів, надавати швидкий доступ до даних і забезпечувати безперебійну роботу додатків. При роботі з великими обсягами даних питання продуктивності набуває особливої актуальності, оскільки навіть невеликі затримки можуть стати критичними.

Для повноцінного й об'єктивного аналізу продуктивності бази даних важливо враховувати низку ключових показників, які відображають, наскільки ефективно система справляється з поточним навантаженням, зазвичай оцінюють наступні показники:

— час відповіді на запит - час, необхідний для обробки одного або декількох SQL-запитів.

— пропускна здатність - кількість запитів, які система може обробити за одиницю часу.

— завантаження процесора та оперативної пам'яті - важливий показник ефективності використання ресурсів.

— I/O активність (дискові операції) - читання/запис даних на/з диска.

— кількість блокувань і конфліктів - особливо важливо для систем з високим рівнем паралельності.

— кешування - використання кешу для оптимізації частих запитів.

Методи аналізу продуктивності баз даних охоплюють різні підходи та інструменти, що дозволяють оцінити ефективність роботи системи, виявити вузькі місця та оптимізувати обробку запитів і використання ресурсів. Залежно від поставлених цілей і технічних особливостей середовища, застосовуються як базові, так і поглиблені методи дослідження, що забезпечують комплексне розуміння стану продуктивності. Серед найпоширеніших методів можна виділити такі:

а)Моніторинг SQL-запитів

1) аналіз часу виконання запитів (Query Execution Time).

2) виявлення повільних або часто виконуваних запитів.

3) побудова планів виконання запитів (Execution Plans).

б)Аналіз індексації

1)оцінка наявності, типів та ефективності індексів.

2)виявлення надлишкових або неефективних індексів.

3)визначення індексів, які не використовуються.

в)Оцінка структури таблиць

1)Аналіз розміру таблиць та кількості рядків.

2)Виявлення надмірної денормалізації чи дублювання даних.

3)Перевірка типів даних і констрейнтів.

г)Виявлення вузьких місць (bottlenecks)

1)Ідентифікація проблемних місць у CPU, пам’яті, дисковому I/O або мережі.

2)Вимірювання навантаження на систему при пікових запитах.

д)Аналіз транзакцій

1)Перевірка рівнів ізоляції та блокувань.

2)Виявлення deadlock'ів або довготривалих транзакцій.

Для ефективного аналізу продуктивності баз даних застосовуються як вбудовані, так і зовнішні інструменти, що дозволяють відстежувати навантаження, знаходити повільні запити та контролювати стан інфраструктури.

У складі самих СУБД є потужні засоби моніторингу. Наприклад, у SQL Server використовується SQL Profiler та Activity Monitor у середовищі SSMS для перегляду активності запитів і навантаження на сервер. У PostgreSQL корисним є розширення pg_stat_statements, яке збирає статистику виконання запитів, а також команда EXPLAIN ANALYZE, що детально показує, як виконується запит. Для MySQL застосовують SHOW PROFILE для розбору запитів і Slow Query Log для фіксації повільних запитів.

Зовнішні інструменти, як-от Prometheus і Grafana, забезпечують збір і візуалізацію метрик у реальному часі, а Zabbix і Nagios слідкують за загальним станом серверів та інфраструктури.

Для тестування продуктивності застосовують Apache JMeter та sysbench, які моделюють навантаження, а BenchmarkSQL імітує типові бізнес-сценарії, щоб оцінити поведінку системи під реальними умовами.

Загалом ці інструменти дозволяють своєчасно виявляти проблеми, оцінювати ефективність запитів та забезпечувати стабільну роботу бази даних.

У підсумку, систематичний аналіз продуктивності бази даних не лише допомагає підтримувати її стабільну роботу, але й є запорукою швидкого реагування на зростаючі навантаження та бізнес-вимоги. Використання комплексного підходу до оцінки продуктивності дозволяє не лише виявляти та

усувати поточні проблеми, а й запобігати потенційним збоям у майбутньому, забезпечуючи тим самим високу доступність, надійність та масштабованість.

РОЗДІЛ 2 АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ФУНКЦІОНУВАННЯ БАЗИ ДАНИХ

2.1. Характеристика об'єкта дослідження

Об'єктом дослідження є реляційна база даних на основі відкритого набору даних IMDb (Internet Movie Database), який містить велику кількість структурованих даних про кіно: художні, документальні фільми, серіали, телешоу, а також інформацію про акторів, режисерів, продюсерів, рейтинги, жанри, дати прем'єр, типи контенту тощо.

IMDb - один з найповніших і найвідоміших джерел розважального контенту, а його регулярно оновлюваний відкритий набір даних IMDb Datasets містить мільйони записів, що робить його чудовим прикладом пошуку у великих обсягах даних. Цей набір даних публікується у форматі TSV (Tab-Separated Values), що означає, що значення в рядках розділені табуляцією. Кожен файл у цьому наборі відображає певний зріз інформації, що стосується різних складових кіноіндустрії.

Основним джерелом загальної інформації про фільми, серіали та інші відеороботи є таблиця title.basics.tsv. У ній зберігаються такі відомості, як унікальний ідентифікатор назви, формат, основна та оригінальна назви, ознака дорослого контенту, рік виходу, тривалість у хвилинах і до трьох жанрів.

Таблиця title.ratings.tsv зосереджена на рейтингах – вона містить середню оцінку, яку проєкт отримав від глядачів, а також загальну кількість голосів, що дозволяє оцінити популярність чи сприйняття тієї чи іншої стрічки.

Для зберігання даних про людей, які брали участь у створенні фільмів, використовується таблиця name.basics.tsv, вона містить персональний ідентифікатор, ім’я, роки життя, основні професії та список найвідоміших проєктів, у яких особа брала участь.

Більш детальну інформацію про конкретну участь людей у проєктах можна знайти в таблиці title.principals.tsv. Тут зазначено, хто саме виконував ключові ролі

у виробництві – актори, режисери, продюсери тощо, а також, за можливості, вказуються конкретні ролі чи персонажі.

Таблиця title.akas.tsv фіксує альтернативні назви проєктів – ті, що використовуються в різних країнах чи мовних середовищах. Вона зберігає локалізовані назви, мову, регіон, типи назв (наприклад, телевізійна версія, робоча назва, фестивальна тощо), а також вказує, чи є ця назва оригінальною.

Інформацію про режисерів та сценаристів зібрано у таблиці title.crew.tsv. У ній для кожного проєкту зазначаються відповідні ідентифікатори осіб, що виконували ці важливі функції, дозволяючи простежити творчий внесок окремих спеціалістів у конкретні роботи.

Ті проєкти, що мають структуру серіалу з окремими епізодами, представлені в таблиці title.episode.tsv. Вона містить дані про кожен епізод, включаючи його належність до певного сезону та серіалу загалом, дозволяючи відновити логічну послідовність і структуру серіалу.

У рамках дипломної роботи було здійснено процес імпорту та структурування и даних IMDb у локальну реляційну базу даних. З огляду на обсяг даних, у роботі зосереджено увагу на найбільш типових та взаємопов'язаних таблицях, які дозволяють відтворити типову схему бази даних для кіноархіву або інформаційного сервісу.

Метою створення такої бази даних є подальший аналіз її логічної та фізичної структури, виявлення потенційних вузьких місць, дослідження факторів, що впливають на продуктивність, а також тестування методів оптимізації.

Слід зазначити, що ці дослідження не передбачають роботу з реальним виробничим сервером або високонавантаженим середовищем. Всі експерименти проводяться в умовах емуляції, з використанням локального сховища, з акцентом на структурному аналізі та моделюванні поведінки бази даних при обробці великих обсягів інформації.

Таким чином, набір даних IMDb виступає репрезентативною моделлю великої інформаційної системи, що дозволяє дослідити та продемонструвати

основні принципи побудови, аналізу та оптимізації структур баз даних для великих обсягів даних.

2.2 Аналіз структури створеної бази даних

В ході виконання було створено реляційну базу даних на основі відкритого датасету IMDb з метою аналізу та подальшої оптимізації його структури для роботи з великими обсягами даних. Розробка здійснювалася в середовищі SQL Server Management Studio (SSMS) з використанням MS SQL Server як платформи для зберігання та обробки даних.

Структура бази даних була спроектована за принципами реляційної моделі з урахуванням майбутнього навантаження на систему. Основна увага приділялася забезпеченню логічної цілісності, узгодженості даних, ефективності виконання запитів та масштабованості.

Структура бази даних передбачає використання первинних та зовнішніх ключів для забезпечення зв'язків між таблицями та підтримки цілісності посилань.

Логічна модель бази даних створена згідно структури даних IMDb і призначена для відображення сутностей предметної області та зв'язків між ними. Основна увага була приділена коректному визначенню зв'язків «один-до-багатьох» (наприклад, в одному фільмі багато акторів) та «багато-до-багатьох» (наприклад, одна людина може зніматися в багатьох фільмах, а в одному фільмі може бути кілька учасників з різними ролями). Реалізації зв'язків «багато-до-багатьох» реалізовані через проміжні таблиці, зокрема title_principals, яка об'єднує інформацію про назви фільмів, людей та їхні ролі в проектах.

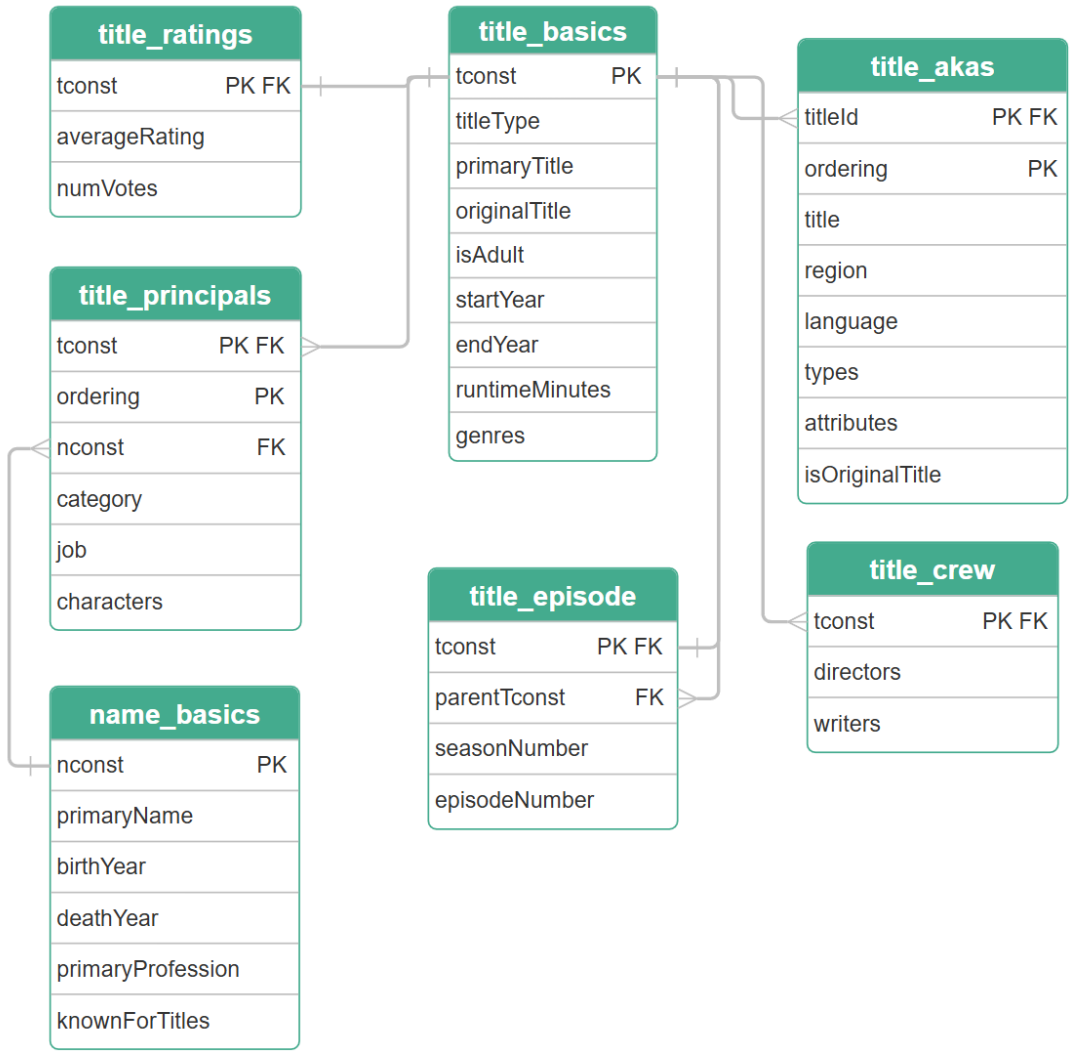


Рис. 2.1 Логічна модель

Фізична модель реалізована в середовищі Microsoft SQL Server за допомогою SQL Server Management Studio. При розробці фізичної структури особливу увагу було приділено продуктивності та масштабованості бази даних. Всі таблиці були створені з урахуванням вимог до типів даних та зв'язків між об'єктами.

Були реалізовані наступні фізичні оптимізації:

- Створено первинні та зовнішні ключі для забезпечення швидкого доступу до даних та підтримки зв'язків між таблицями.
- Оптимальні типи даних: для кожного стовпця обрано найефективніший тип даних (наприклад, VARCHAR, INT, FLOAT), що зменшує обсяг даних, які зберігаються, та пришвидшує операції читання та запису.

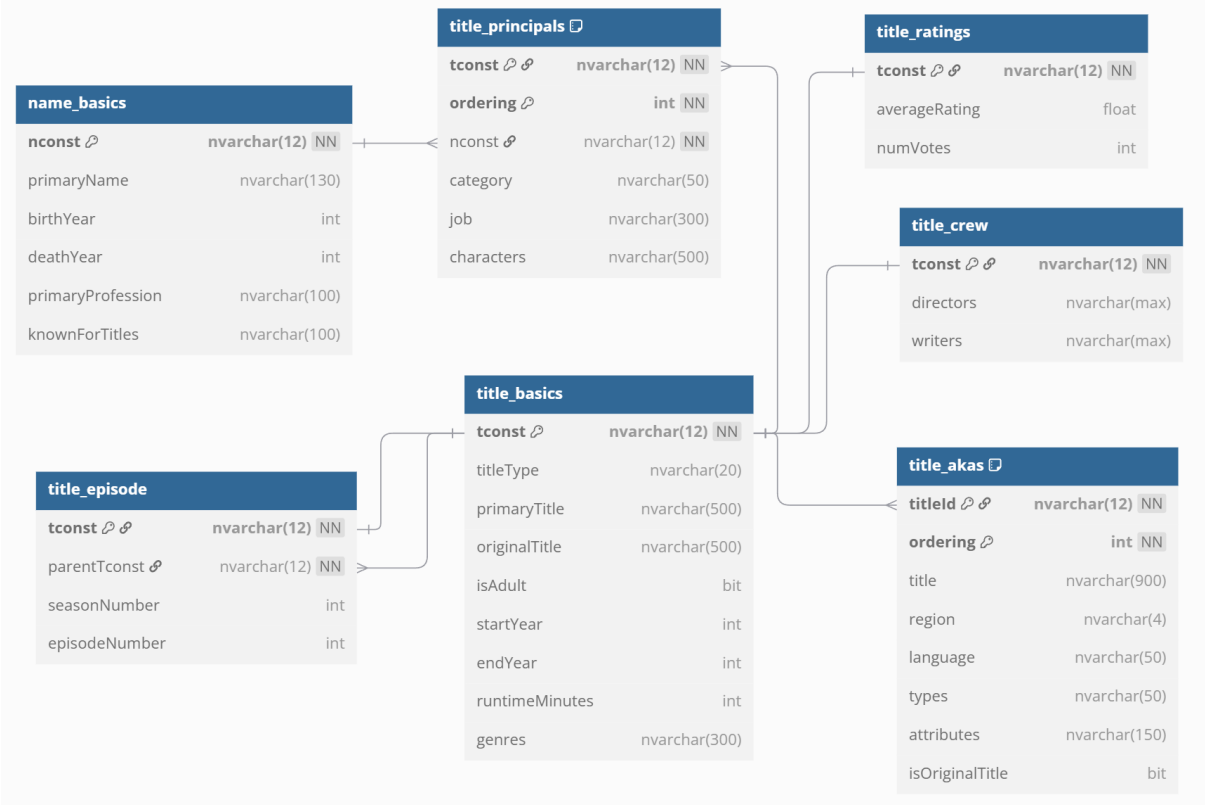


Рис. 2.2 Фізична модель

Для кращого уявлення про обсяг і складність цієї бази даних доцільно навести кількісну характеристику основних таблиць, що її формують. Усього в ній наявні сім ключових таблиць, причому кількість записів у кожній з них становить:

—таблиця title_principals налічує 92 343 976 записів

—таблиця title_akas налічує 52 035 728 записів

—таблиця name_basics налічує 14 376 758 записів

—таблиця title_basics налічує 11 630 899 записів

—таблиця title_crew налічує 11 630 712 записів

—таблиця title_episode налічує 8 949 408 записів

—таблиця title_ratings налічує 1 565 445 записів

```sql
-- Створення тимчасової таблиці для збереження результатів підрахунку
CREATE TABLE #table_counts (
    table_name NVARCHAR(128),
    record_count INT
);
DECLARE @table_name NVARCHAR(128)
DECLARE @sql NVARCHAR(MAX)
DECLARE @record_count INT

-- Створення динамічного SQL для підрахунку записів у кожній таблиці
DECLARE table_cursor CURSOR FOR
SELECT table_name
FROM information_schema.tables
WHERE table_type = 'BASE TABLE' AND table_catalog = 'IMDb'

OPEN table_cursor
FETCH NEXT FROM table_cursor INTO @table_name

-- Вставка кількості записів для кожної таблиці в тимчасову таблицю
WHILE @@FETCH_STATUS = 0
BEGIN
    SET @sql = 'SELECT @record_count = COUNT(*) FROM ' + @table_name
    EXEC sp_executesql @sql, N'@record_count INT OUTPUT', @record_count OUTPUT

    INSERT INTO #table_counts (table_name, record_count)
    VALUES (@table_name, @record_count)

    FETCH NEXT FROM table_cursor INTO @table_name
END

CLOSE table_cursor
DEALLOCATE table_cursor

SELECT * FROM #table_counts
order by record_count desc
;
```

100 %

Results | Messages

	table_name	record_count
1	title_principals	92343976
2	title_akas	52035728
3	name_basics	14376758
4	title_basics	11630899
5	title_crew	11630712
6	title_episode	8949408
7	title_ratings	1565445

Рис. 2.3 Підрахунок кількості записів в таблицях

Загальний обсяг бази даних становить приблизно 24 ГБ, з яких близько 19,5 ГБ припадає на самі дані, а ще приблизно 4,5 ГБ займають журнали транзакцій. Такий масштаб даних дає змогу моделювати сценарії роботи з великими обсягами інформації й досліджувати ефективність різних стратегій оптимізації.

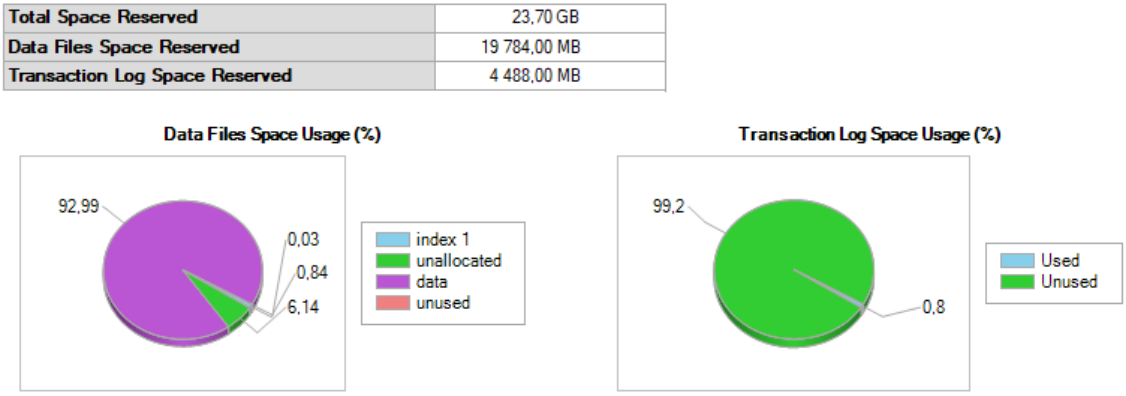


Рис. 2.4 Загальний обсяг бази даних

2.3 Аналіз продуктивності

Аналіз продуктивності створеної бази даних проводилась на основі аналізу структури таблиць, обсягів даних, часу виконання основних запитів, а також оцінки ефективності використання наявних індексів. Оскільки дослідження проводиться в середовищі MS SQL Server без підключення зовнішнього користувача, тести виконувалися в локальному середовищі за допомогою SQL Server Management Studio.

Для ефективного аналізу продуктивності запитів у SQL Server використовують кілька ключових інструментів, що дозволяють детально оцінити план виконання та витрати ресурсів:

a)Live Query Statistics і Actual Execution Plan

Ці два типи планів дозволяють оцінити стратегію, яку SQL Server буде використовувати для виконання конкретного запиту:

1)Live Query Statistics - дозволяє побачити, як система планує виконати запит до того, як він буде фактично виконаний. Це допомагає розпізнати проблеми з застосуванням індексів або надто громіздких з'єднань таблиць.

2)Actual Execution Plan - формується після виконання запиту і показує фактичний час і ресурси, витрачені на кожну операцію.

Ці плани буде використано для аналізу складних запитів, що охоплюють декілька таблиць (наприклад, title_principles, title_basics, name_basics) з фільтрацією, сортуванням та об'єднанням.

б)SET STATISTICS TIME і SET STATISTICS IO

Ці команди надають інформацію про ресурси, що були використані під час виконання запиту:

1)SET STATISTICS TIME - повертає час, витрачений на компіляцію і виконання запиту в мілісекундах, що дозволяє провести оцінку продуктивності на рівні процесора.

2)SET STATISTICS IO - показує число логічних і фізичних сторінок, зчитаних з диска, і відображає навантаження на підсистему вводу/виводу. Це має особливе значення при аналізі запитів до великих таблиць, де невірна побудова індексів може призвести до значного зростання операцій вводу-виводу.

Отримані значення були використані для зіставлення різних реалізацій запитів та виявлення неефективних конструкцій.

в)Database Tuning Advisor (DTA)

Цей інструмент аналізує конкретне робоче навантаження і надає автоматичні пропозиції щодо покращення ефективності роботи бази даних. Зокрема, він може запропонувати створення нових індексів, перегляд існуючих індексів та оцінку їхнього впливу на продуктивність, створення матеріалізованих подань для оптимізації частих запитів, а також актуалізацію статистичних даних, що використовуються для побудови планів виконання.

Після попереднього аналізу структури бази даних та оцінки ефективності виконання SQL-запитів було виявлено кілька проблем, що впливають на продуктивність системи. Кожна з таблиць має певні вузькі місця, які потребують оптимізації або реорганізації.

Детальний опис виявлених недоліків:

1)Таблиця title_principals

а)Запити, що фільтрують дані за полем nconst, мають малу ефективність виконання через відсутність відповідного індексу

б)Поле category, яке часто використовується у фільтрах, для визначення ролі, не оптимізоване для пошуку, необхідно розглянути створення індексу

2)Таблиця title_akas

а)Відсутність індексу на полі language ускладнює пошук фільмів або серіалів за мовою

б)Відсутність індексу на полі region, що ускладнює географічну фільтрацію назв.

3)Таблиця name_basics

а)Поле knownForTitles зберігає список ідентифікаторів у вигляді ідентифікаторів розділених комами, що ускладнює ефективний пошук.

б)Запити, що фільтрують за полем birthYear і deathYear, виконуються повільно через відсутність відповідних індексів

4)Таблиця title_basics

а)Поле genres зберігає перелік жанрів розділених комою, що обмежує можливості ефективного фільтрування за окремими жанрами.

б)Відсутність індексу на полі startYear, що впливає на швидкість побудови списків фільмів за роком випуску.

в)Відсутність індексу на полі titleType, яке часто використовується для розділення серіалів,фільмів, короткометражок.

5)Таблиця title_crew

а)Поля directors та writers містять списки ідентифікаторів розділених комами, такий підхід ускладнює пошук.

6)Таблиця title_episode

а)Відсутність індексу на полі parentTconst, яке вказує на батьківський серіал.

7)Таблиця title_ratings

а)Часті запити до популярних фільмів та серіалів потребують додаткових ресурсів.

б)Відсутність індексу на полях averageRating і numVotes,які дуже часто використовуються для фільтрації даних.

2.4 Висновки за результатами дослідження структури та продуктивності БД

У ході дослідження бази даних на основі великого відкритого набору IMDb було проаналізовано структуру та продуктивність інформаційної системи, що працює з великими обсягами даних. Загальний обсяг даних становить близько 24 ГБ та налічує понад 180 мільйонів записів, розподілених між 7 взаємопов'язаними таблицями, що дозволяє моделювати реальні сценарії обробки запитів та виявляти типові проблеми масштабування та продуктивності. Незважаючи на відсутність активного клієнтського навантаження, обсяг даних і структура бази даних забезпечує потрібні умови для діагностики вузьких місць та оцінки ефективності роботи запитів.

На основі проведеної діагностики були ідентифіковані низка характерних проблем, які суттєво знижують ефективність обробки запитів у базі даних. До них належать:

1)Відсутність індексації по ключових полях, які часто використовуються у фільтрації та приєднаннях таблиць . Внаслідок цього більшість запитів змушені використовувати менш ефективні методи для сканування таблиць, що значно підвищує час виконання та ресурсне навантаження на систему.

2)Зберігання складних і повторюваних значень у текстовому вигляді,що ускладнює процес обробки даних. Оскільки ці значення представлені у вигляді списків, розділених комами, це не лише унеможливлює пряме використання стандартних механізмів фільтрації та індексації, пошуку та підтримки цілісності даних. Такий підхід перешкоджає масштабуванню запитів до цих полів.

3)Відсутність спеціалізованих рішень для обробки частих запитів, особливо коли йдеться про популярні записи, до яких звертаються найчастіше. Такі запити часто повторюють однакову логіку фільтрації або агрегації, однак через відсутність попередньо підготовлених результатів щоразу запускається повноцінне опрацювання, що спричиняє зайве навантаження на ресурси.

Усі зазначені проблеми є типовими для систем, що працюють з великими обсягами даних і свідчать про важливість застосування цілеспрямованих заходів структурного та запитного аудиту великих БД ще на етапі проектування. Ці результати стануть базою для подальших етапів оптимізації, які будуть представлені в наступному розділі. Зокрема, планується застосування таких підходів, як створення додаткових індексів, нормалізація даних, а також створення матеріалізованих представлень для підвищення продуктивності.

Таким чином, проведене дослідження підтвердило, що навіть створена у локальному середовищі база даних здатна виявити типові проблеми промислових інформаційних систем, а аналіз її структури та продуктивності є важливим кроком до розуміння реальних викликів при роботі з великими даними. Це підкреслює актуальність теми дипломної роботи та доцільність впровадження оптимізаційних заходів на наступному етапі роботи.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ОПТИМІЗАЦІЇ БАЗИ ДАНИХ

3.1 Побудова структури бази даних та первинне завантаження даних

Перш ніж розпочати імпорт даних, необхідно було створити повноцінне середовище для їх зберігання та обробки. Для цього в SQL Server була створена нова база даних під назвою IMDb, яка в подальшому буде використовуватися як основне сховище інформації.

Після створення бази даних розпочався процес проектування таблиць відповідно до структури файлів IMDb. Важливим кроком було не лише відтворення таблиць, а й вибір правильних типів даних, визначення обмежень, ключів та зв'язків між таблицями.

Як приклад, розглянемо створення таблиці title_akas, яка містить альтернативні назви фільмів або серіалів у різних регіонах. Перед створенням було виконано аналіз типів даних у джерелі. Після аналізу було вирішено використовувати тип NVARCHAR для текстових значень (для підтримки багатомовності), INT для числового поля ordering та BIT для логічного поля isOriginalTitle. Для таблиці також було створено зовнішній ключ і вказані внутрішні ключі для забезпечення цілісності даних.

```
CREATE TABLE title_akas (
    titleId NVARCHAR(12) NOT NULL,
    ordering INT NOT NULL,
    title NVARCHAR(900),
    region NVARCHAR(4),
    language NVARCHAR(50),
    types NVARCHAR(50),
    attributes NVARCHAR(100),
    isOriginalTitle BIT,
    PRIMARY KEY (titleId, ordering),
    FOREIGN KEY (titleId) REFERENCES title_basics(tconst)
);
```

Рис. 3.1 Приклад створення таблиці title_akas

Такий підхід до створення таблиць було застосовано до кожної сутності в базі даних IMDb. Це забезпечило чітку структуру, яка відповідала вимогам до відображення реальних даних.

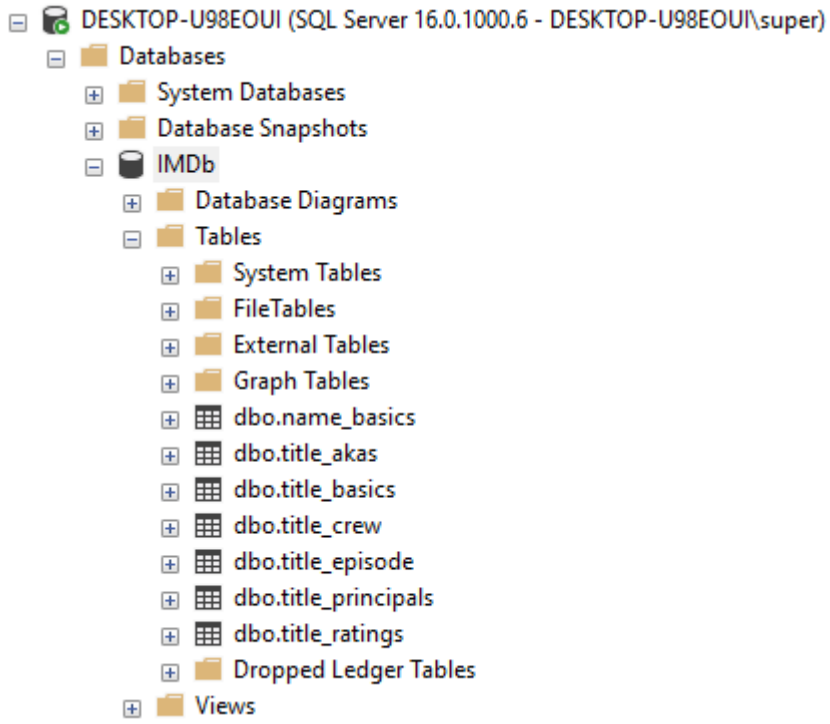


Рис. 3.2 Вигляд БД і всіх створених таблиць

Так як початкові дані були у форматі файлів з розширенням .tsv, тому для імпорту даних з файлів у базу даних я використовував вбудований в SQL Server Management Studio функціонал Import Flat File.

Оскільки інструмент Import Flat File в SQL Server погано підтримує роботу з tsv-файлами і невірно імпортує NULL значення. Я обрав стратегію імпортувати більшість даних у вигляді текстових значень з типом nvarchar.

Щоб спростити процес імпорту та зменшити кількість помилок, були створені тимчасові таблиці, в які спочатку завантажувалися дані. Це дозволило перевірити та обробити дані перед тим, як вставити їх в остаточну структуру таблиці.

Тимчасові таблиці були схожі за структурою на основні таблиці, але без зовнішніх ключів і з мінімальними перевірками, щоб забезпечити швидший процес імпорту.

Це дозволило значно полегшити та прискорити імпорт, оскільки не виникало проблем із некоректною обробкою роздільників у tsv файлах.

Нижче наведено приклад створення тимчасової таблиці title_akas_temp через функціонал Import Flat File:

1)Відкриваємо інструмент Import Flat File для нашої бази даних.

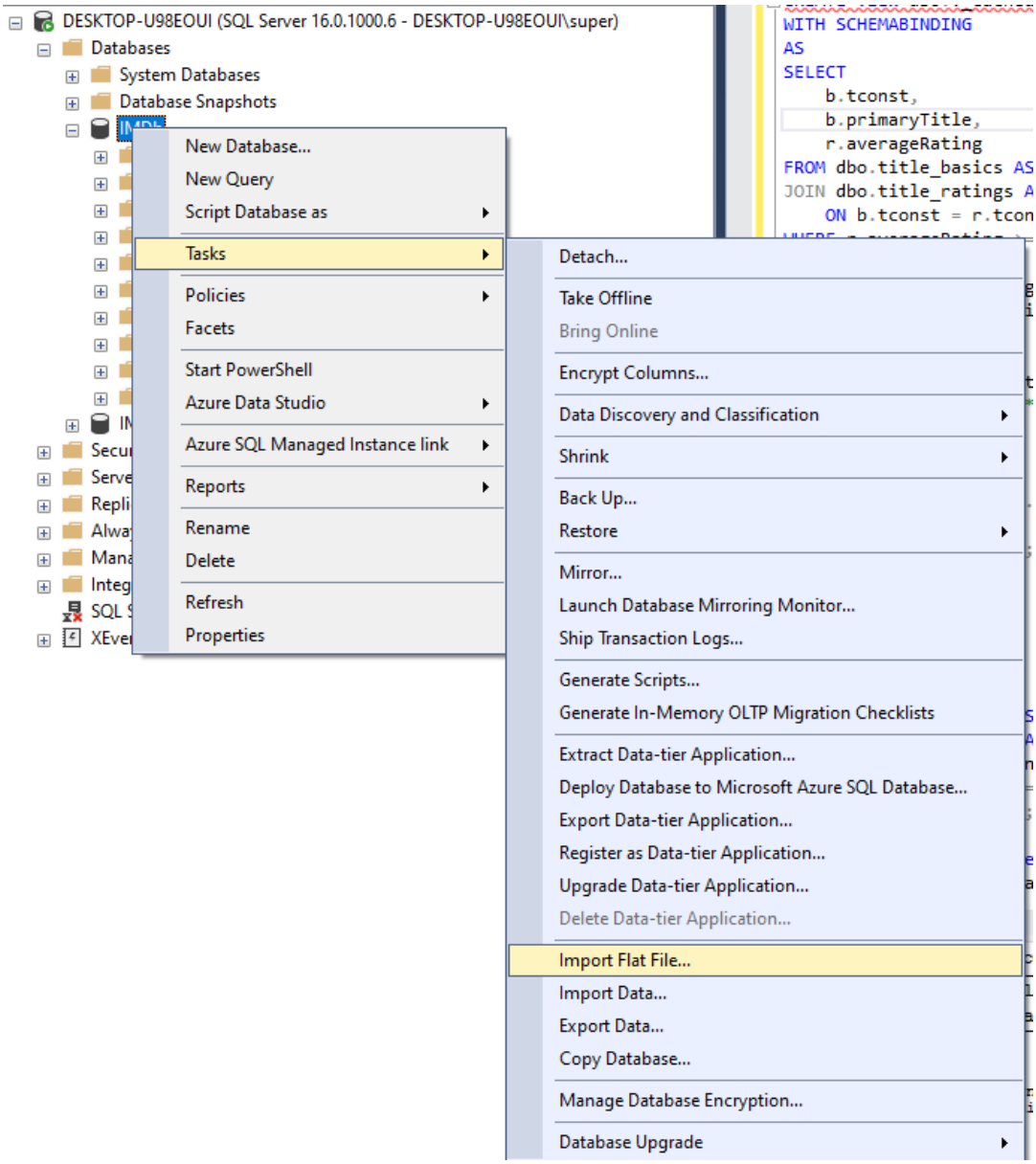


Рис. 3.3 Відкриття Import Flat File

2)Вказується шлях до tsv-файлу з даними, а також назва таблиці, яка буде створена на основі цього файлу в цьому випадку title_akas_temp.

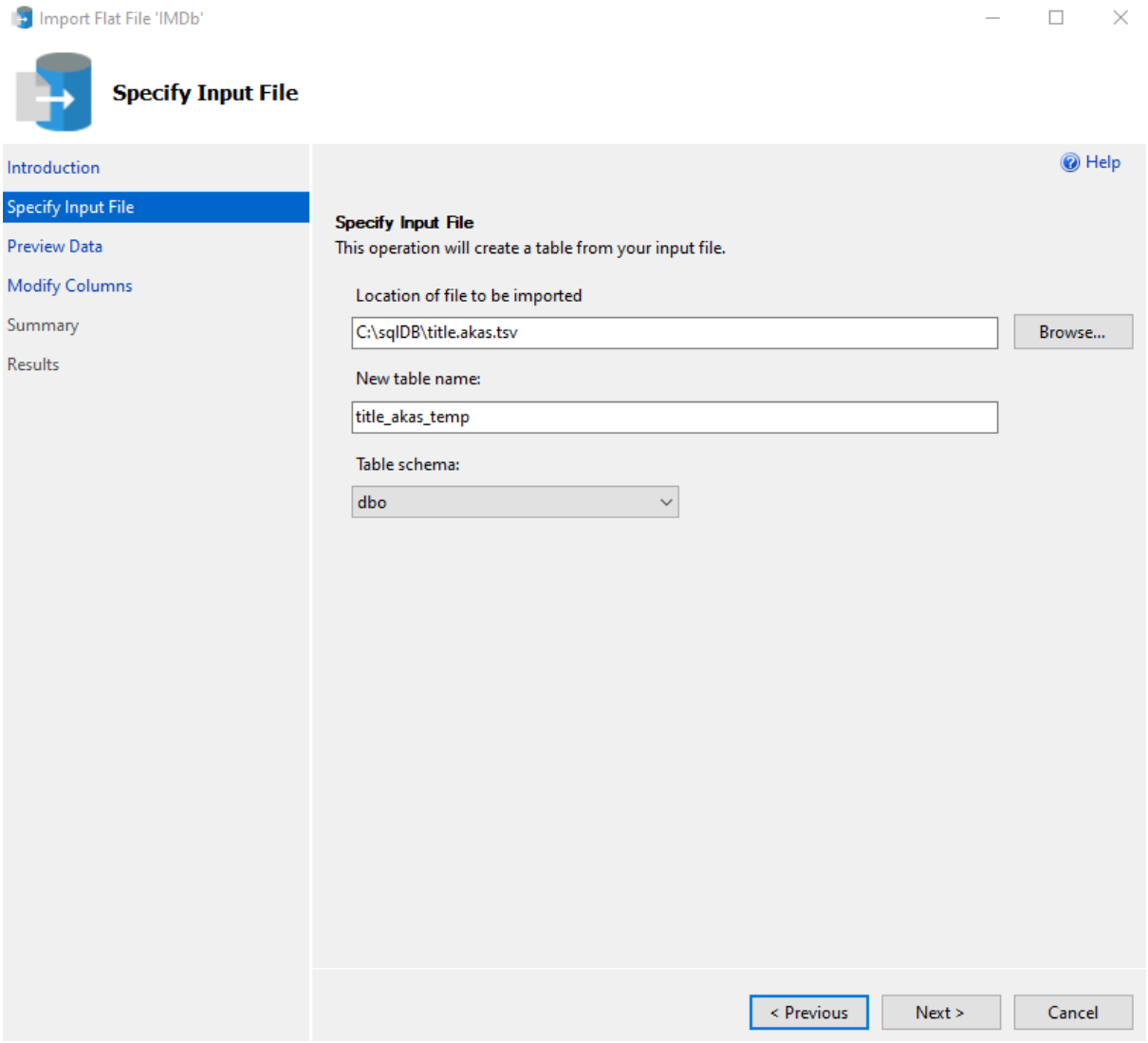


Рис. 3.4 Вказування шляху до файлу

3) На цьому кроці користувачу відкривається можливість попереднього перегляду вмісту файлу, а саме перші 50 рядків. Такий перегляд виконує важливу роль на етапі підготовки до імпорту, оскільки дозволяє швидко оцінити загальну структуру даних, перевірити відповідність колонок очікуваному формату, а також виявити потенційні проблеми, наприклад наявність пропущених значень, помилкових символів або зміщення роздільників. Візуальна перевірка сприяє впевненості в тому, що дані будуть імпортовані коректно, без втрат чи спотворень.

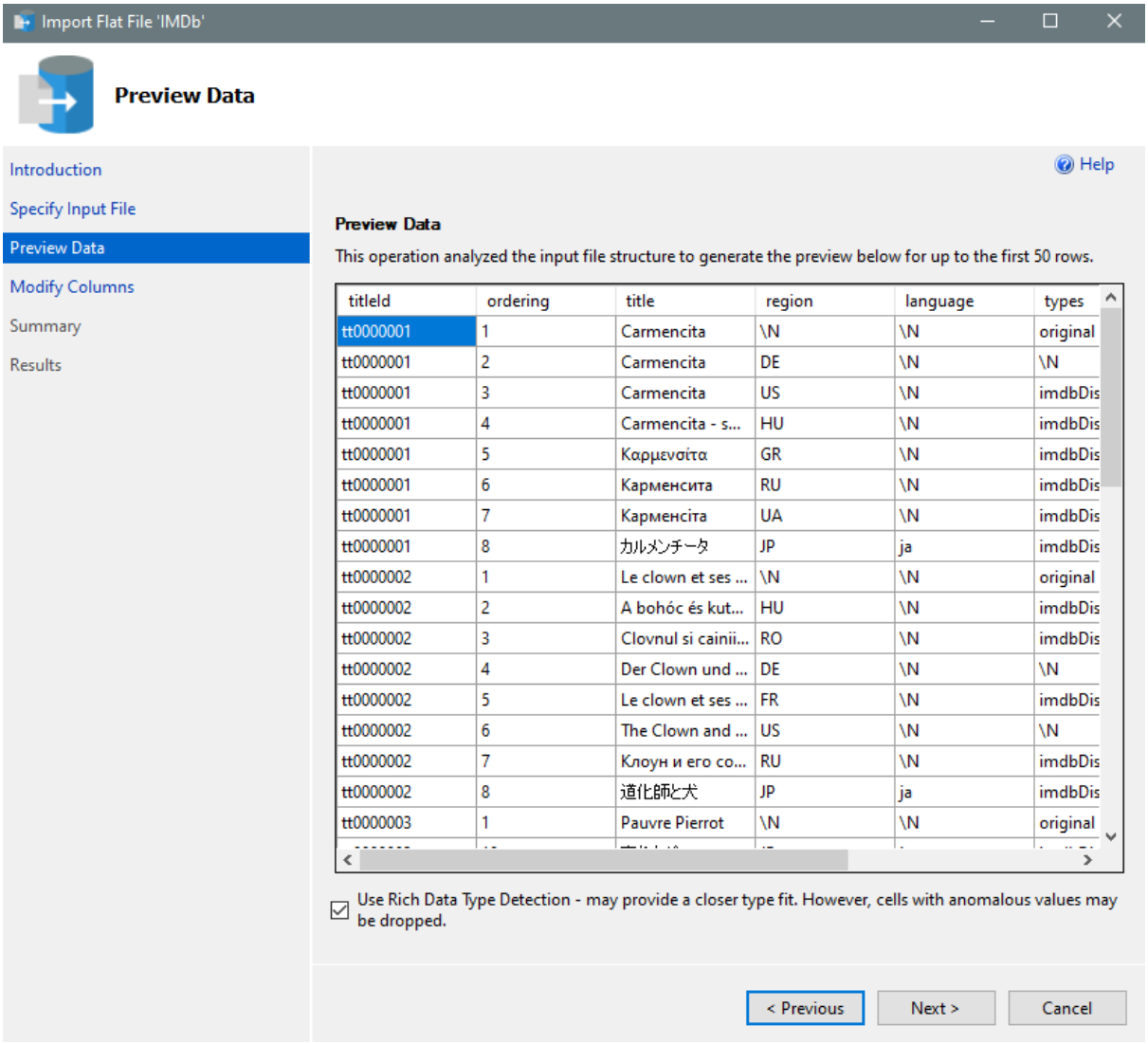


Рис. 3.5 Попередній перегляд даних

4)Залежно від частини вмісту файлу генерується структура таблиці.Тут я змінював типи даних відповідно до таблиці в яку буде відбуватися вставка даних і там де можливиі значення NULL обов’язково вказував nvarchar тип даних.

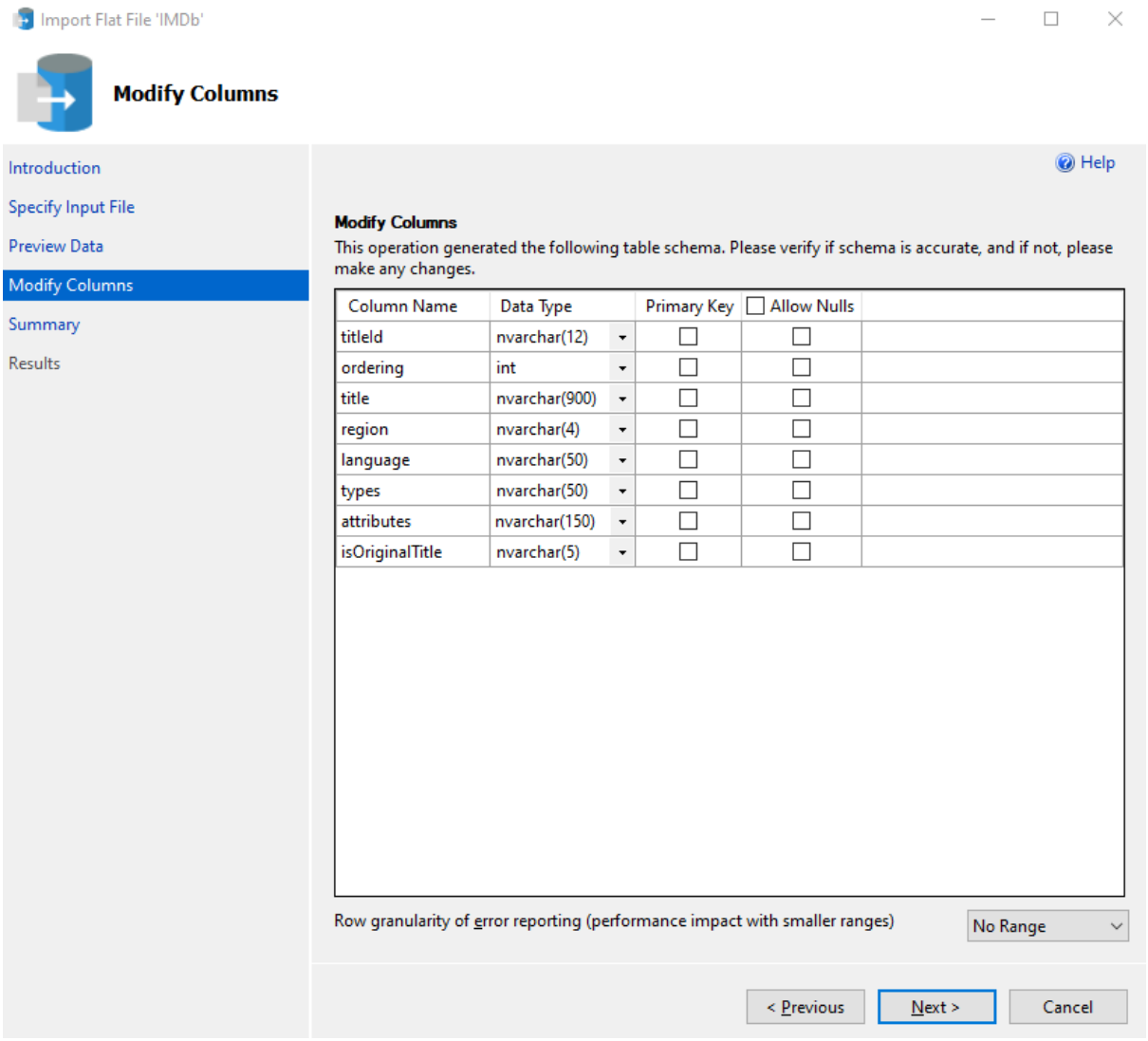


Рис. 3.6 Структура данних створюваної таблиці

5) На цьому етапі користувачеві надається узагальнена інформація, яка дозволяє ще раз переконатися у правильності всіх ключових параметрів перед початком імпорту. Зокрема, відображається назва сервера, до якого здійснюється підключення, обрана база даних, ім’я таблиці, яка буде створена або оновлена в процесі імпорту, а також повний шлях до зовнішнього файлу, що слугуватиме джерелом даних.

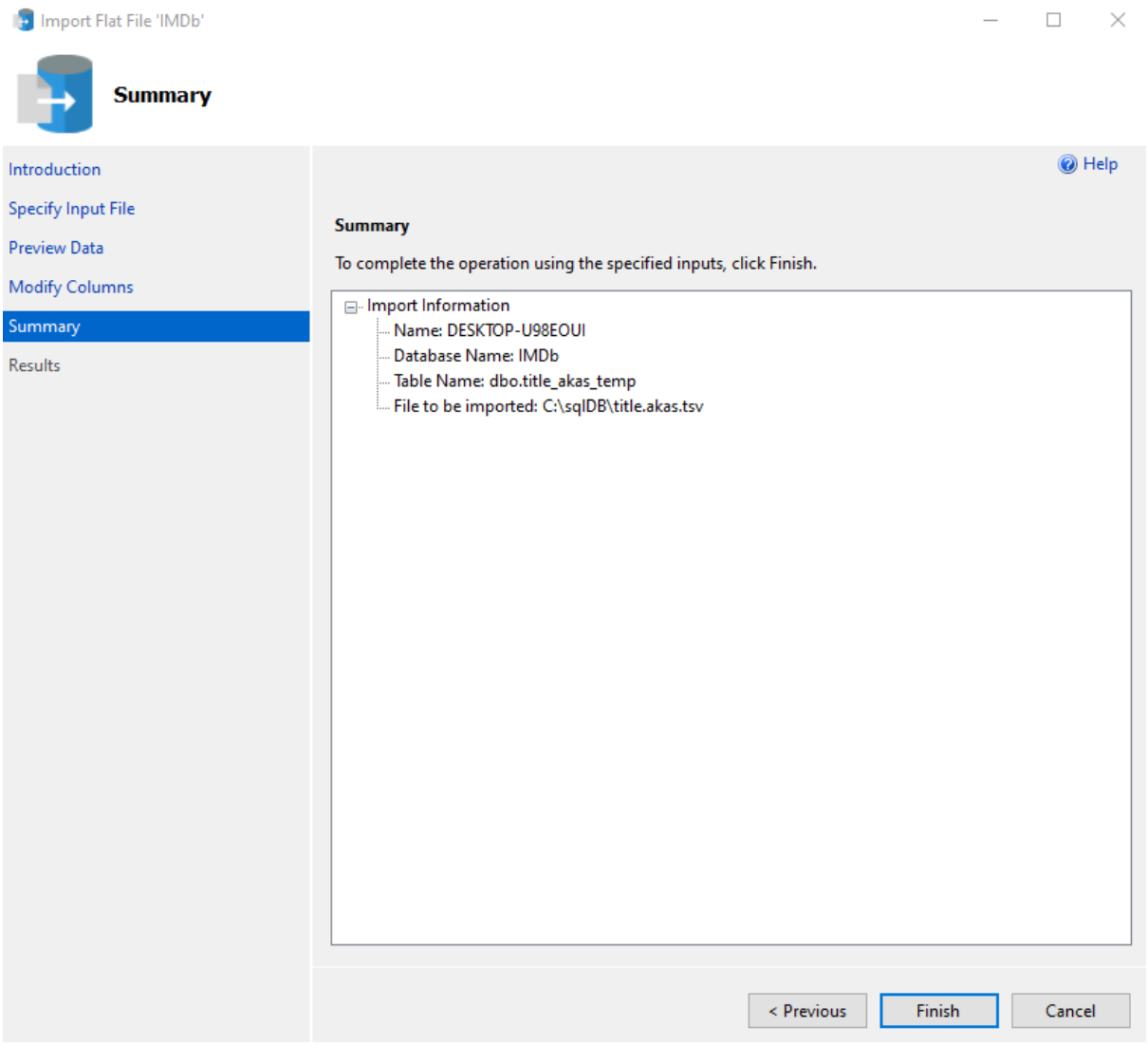


Рис. 3.7 Підсумок імпорту

6) Завершальним етапом роботи з даними є безпосередньо процес їх імпорту до бази. На цьому етапі здійснюється остаточне перенесення інформації у відповідні таблиці. У разі виникнення помилок або конфліктів у структурі чи форматі даних, система негайно повідомить про це, згенерувавши відповідне повідомлення про помилку. Це дозволяє оперативно ідентифікувати проблему та внести необхідні корективи до даних або налаштувань імпорту.

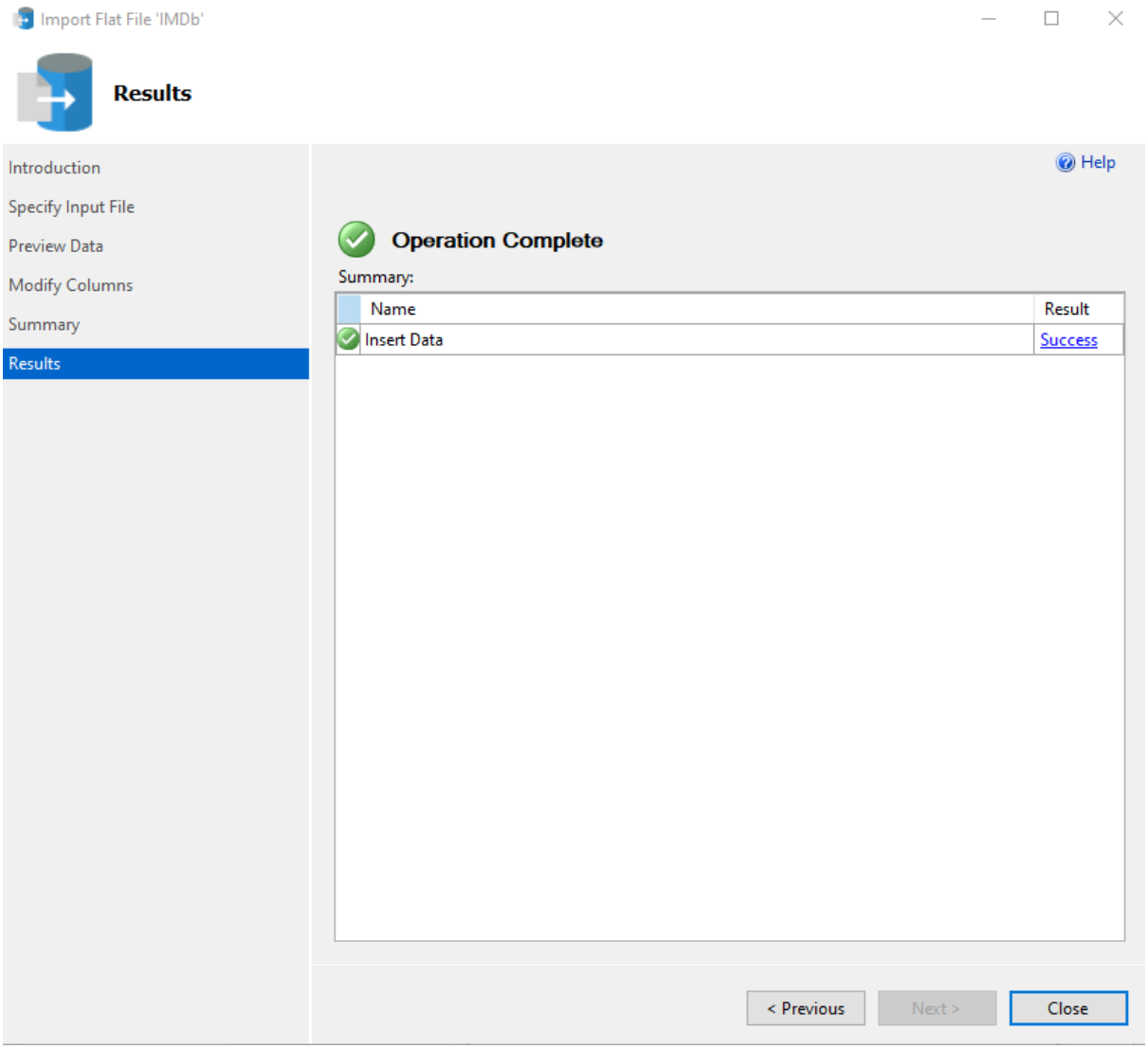


Рис. 3.8 Успішна операція імпорту файла

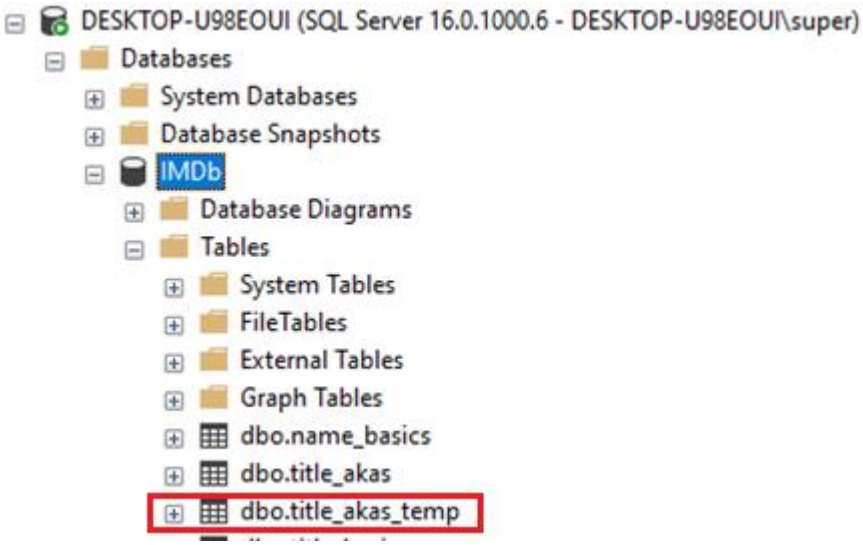


Рис. 3.9 Створена тимчасова таблиця

Після того, як дані в тимчасових таблицях були перевірені, вони були перенесені в основні таблиці за допомогою SQL-запитів INSERT INTO. Цей етап

був важливим, оскільки він забезпечив правильний та ефективний процес перенесення даних, необхідний для збереження цілісності бази даних та уникнення будь-яких можливих помилок.

Для полів, де значення “\N” означало відсутність даних, було використано механізм NULLIF. Це дозволило автоматично замінити ці значення на NULL, забезпечивши коректне відображення відсутніх даних у таблицях. Такий підхід дозволив уникнути появи некоректних значень, які могли б вплинути на точність аналізу та подальшу обробку даних.

Крім того, після перенесення даних до основних таблиць була проведена додаткова перевірка, щоб переконатися, що значення були вставлені правильно. Цей крок є критично важливим, оскільки дозволяє виявити будь-які помилки, пропущені значення або невідповідності в типі даних, які могли виникнути під час попередніх етапів обробки. Така ретельна перевірка забезпечує надійність і точність даних, що зберігаються в базі даних.

Такий підхід до імпорту даних значно підвищує якість обробки інформації, зберігаючи її цілісність і точність та уникаючи дублювання записів. Водночас він забезпечує зручний процес перенесення великих обсягів даних з тимчасових таблиць в основні таблиці, що дозволяє проводити подальшу обробку та аналіз без зайвих затримок і помилок.

```
INSERT INTO title_akas (
    titleId, ordering, title, region, language, types, attributes, isOriginalTitle
)
SELECT
    titleId,
    TRY_CAST(ordering AS INT),
    NULLIF(title, '\N'),
    NULLIF(region, '\N'),
    NULLIF(language, '\N'),
    NULLIF(types, '\N'),
    NULLIF(attributes, '\N'),
    CASE
        WHEN isOriginalTitle = '1' THEN 1
        WHEN isOriginalTitle = '0' THEN 0
        ELSE NULL
    END
FROM title_akas_temp

drop table title_akas_temp
```

Рис. 3.10 Приклад імпорту даних в таблицю title_akas з тимчасової таблиці

3.2 Реалізація підходів до оптимізації продуктивності

Після аналізу виявлених структурних недоліків і проблем з продуктивністю, описаних у попередньому розділі, наступним етапом дослідження стала реалізація оптимізаційних заходів, спрямованих на покращення роботи бази даних. Метою цього етапу є зниження навантаження на систему, прискорення виконання запитів, а також створення умов для масштабування та подальшого розвитку інформаційної структури.

Оптимізація здійснюється як на рівні структури бази даних ,шляхом індексації, нормалізації , так і через створення допоміжних представлень. У межах цього підрозділу буде детально описано прийняті рішення, їх обґрунтування та практичне впровадження з використанням засобів SQL Server.

Оскільки статистика використання бази даних та реальне навантаження були відсутні, оптимізація виконувалась на основі логічного аналізу структури таблиць та ймовірних сценаріїв використання даних. Це включало прогнозування найчастіших запитів потенційних користувачів системи, аналіз типів зв'язків між таблицями та розуміння особливостей джерела даних.

1)Таблиця title_principals

Після проведеного аналізу запитів до таблиці title_principals було встановлено, що найбільш поширеними критеріями для фільтрації та приєднання до інших таблиць є поля category та nconst. Таке використання є цілком логічним і характерним, адже у більшості випадків користувачі шукають усі фільми або проєкти, в яких брала участь конкретна особа, що ідентифікується за унікальним ідентифікатором nconst. Крім того, фільтрація за полем category дозволяє відокремлювати різні ролі учасників, наприклад актори, режисери, сценаристи, що дуже важливо для точного і гнучкого вибору інформації. Таким чином, саме ці поля визначають ключові сценарії використання таблиці і є критичними для оптимізації запитів, пов’язаних з пошуком інформації про учасників кінопроєктів.

Для підвищення продуктивності було виконано:

а)Створено індекс idx_title_principals_category на полі category з включенням поля nconst.

б)Створено індекс idx_title_principals_nconst на полі nconst

```
CREATE NONCLUSTERED INDEX idx_title_principals_category
ON [dbo].[title_principals] ([category]) INCLUDE ([nconst])

CREATE NONCLUSTERED INDEX idx_title_principals_nconst
ON [dbo].[title_principals] ([nconst])
```

Рис. 3.11 Створені індекси до таблиці title_principals

Таблиця 3.1

Порівняння продуктивності запитів до таблиці title_principals

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Пошук усіх акторів (category = 'actor')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	1078170	177793	83,51%
		Процесорний час(мс)	18654	4453	76,13%
		Загальний час(мс)	60068	58556	2,52%
2	Пошук фільмів за участі конкретної особи(nconst)	Кількість сканувань	13	1	92,31%
		Логічні зчитування	1121066	3860	99,66%
		Процесорний час(мс)	9420	16	99,83%
		Загальний час(мс)	960	107	88,85%

2)Таблиця title_akas

Після аналізу запитів до таблиці title_akas встановлено, що найбільш часто для фільтрації та приєднання до інших таблиць використовуються поля language та region. Таке використання є типовим, адже більшість запитів спрямовані на пошук фільмів, доступних певною мовою або у визначеному регіоні. Це дозволяє точно відбирати локалізовані версії проєктів, що є важливим аспектом для користувачів із різних мовних та географічних груп. Таким чином, ці поля відіграють ключову роль у забезпеченні гнучкості та релевантності пошуку в базі даних.

Для підвищення продуктивності було виконано:

а)Створено індекс idx_title_akas_language на полі language з включенням поля title.

б)Створено індекс idx_title_akas_region на полі region з включенням поля title.

```
CREATE NONCLUSTERED INDEX idx_title_akas_language
ON title_akas (language) INCLUDE([title])

CREATE NONCLUSTERED INDEX idx_title_akas_region
ON title_akas (region) INCLUDE([title]);
```

Рис. 3.12 Створені індекси до таблиці title_akas

Таблиця 3.2

Порівняння продуктивності запитів до таблиці title_akas

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Пошук усіх фільмів за мовою (language = 'es')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	590923	46375	92,15%
		Процесорний час(мс)	8686	1890	78,24%
		Загальний час(мс)	17563	16967	3,39%

Продовження таблиці 3.2

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
2	Пошук усіх фільмів для конкретного регіону (region = 'JP')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	590878	42774	92,76%
		Процесорний час(мс)	8264	2016	75,61%
		Загальний час(мс)	18011	17625	2,14%

3)Таблиця name_basics

Після аналізу запитів до таблиці name_basics було виявлено, що найбільше використовуються поля birthYear та deathYear для фільтрації даних. Це типово для запитів, що шукають людей за роками народження або смерті. Окрім того, поле knownForTitles містить список ідентифікаторів фільмів, розділених комами, що ускладнює виконання пошуку та фільтрації зв’язків між особами та відповідними проєктами. Такий формат даних створює додаткові труднощі при обробці запитів і знижує ефективність роботи бази даних.

Для підвищення продуктивності було виконано:

а)Створено індекс idx_name_basics_birthYear на полі birthYear з включенням полів primaryName, deathYear та PrimaryProfession.

б)Створено індекс idx_name_basics_deathYear на полі deathYear з включенням полів primaryName, birthYear та PrimaryProfession.

```
CREATE NONCLUSTERED INDEX idx_name_basics_birthYear
ON name_basics ([birthYear]) include ([primaryName],[deathYear],[PrimaryProfession])
CREATE NONCLUSTERED INDEX idx_name_basics_deathYear
ON name_basics ([deathYear]) include ([primaryName],[birthYear],[PrimaryProfession])
```

Рис. 3.13 Створені індекси до таблиці name_basics

в)Проведено нормалізацію поля knownForTitles.А саме:

— було створено нову таблицю known_for_titles, яка зберігає відношення між nconst та tconst

—розділено значення з поля knownForTitles за допомогою STRING_SPLIT() і перенесено в створену таблицю known_for_titles

—поле knownForTitles було видалено з таблиці name_basics.

```
CREATE TABLE known_for_titles (
    nconst NVARCHAR(12) NOT NULL,
    tconst NVARCHAR(12) NOT NULL,
    FOREIGN KEY (nconst) REFERENCES name_basics(nconst),
    FOREIGN KEY (tconst) REFERENCES title_basics(tconst)
);

INSERT INTO known_for_titles (nconst, tconst)
SELECT
    nb.nconst,
    s.value AS tconst
FROM name_basics nb
CROSS APPLY STRING_SPLIT(nb.knownForTitles, ',') s
WHERE nb.knownForTitles IS NOT NULL

ALTER TABLE name_basics
DROP COLUMN knownForTitles;
```

Рис. 3.14 Нормалізація поля knownForTitles

г)Створено індекс idx_known_for_titles_nconst_tconst для таблиці known_for_titles на полі nconst з включенням поля tconst

```
CREATE NONCLUSTERED INDEX idx_known_for_titles_nconst_tconst
ON [dbo].[known_for_titles] ([nconst]) INCLUDE ([tconst])
```

Рис. 3.15 Створені індекси до таблиці known_for_titles

4)Таблиця title_basics

Після аналізу запитів до таблиці title_basics було виявлено, що найчастіше використовуються поля startYear та titleType для фільтрації даних. Це типово для запитів,що шукають фільми за роком випуску та типом. Також поле genres зберігало список жанрів через кому, що ускладнювало фільтрацію та пошук за конкретними жанрами.

Для підвищення продуктивності було виконано:

а) Створено індекс idx_title_basics_startYear на полі startYear з включенням поля primaryTitle. Це дозволяє більш ефективно виконувати запити, які фільтрують записи по роках.

Таблиця 3.3

Порівняння продуктивності запитів до таблиці name_basics

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Пошук усіх осіб за роком народження (birthYear = 1980)	Кількість сканувань	13	1	92,31%
		Логічні зчитування	242524	139	99,94%
		Процесорний час (мс)	985	5	99,49%
		Загальний час (мс)	854	86	89,93%
2	Пошук усіх осіб за роком смерті (deathYear = 2000)	Кількість сканувань	13	1	92,31%
		Логічні зчитування	242524	46	99,98%
		Процесорний час (мс)	923	13	98,59%
		Загальний час (мс)	131	76	41,98%
3	Пошук фільмів, з якими асоціюється конкретна особа (nconst = 'nm0000123')	Кількість сканувань	1	1	0,00%
		Логічні зчитування	302530	24	99,99%
		Процесорний час (мс)	33672	30	99,91%
		Загальний час (мс)	34037	34	99,90%

б) Створено індекс idx_title_basics_titleType на полі titleType з включенням полів primaryTitle та startYear, що прискорює пошук фільмів за типом.

```
CREATE INDEX idx_title_basics_startYear
ON title_basics(startYear)
INCLUDE ([primaryTitle]);

CREATE INDEX idx_title_basics_titleType
ON title_basics(titleType) include ([primaryTitle],[startYear])
```

Рис. 3.16 Створені індекси до таблиці title_basics

в) Проведено нормалізацію поля genres. А саме:

—створено нову таблицю title_genres, яка зберігає зв'язок між tconst та окремими значеннями genre

—значення з поля genres були розділені за допомогою функції STRING_SPLIT(), яка дозволила розділити кожен рядок на окремі елементи та перенести у title_genres

—поле genres було видалено з таблиці title_basics.

```
CREATE TABLE title_genres (
    tconst NVARCHAR(12) NOT NULL,
    genre NVARCHAR(20) NOT NULL,
    FOREIGN KEY (tconst) REFERENCES title_basics(tconst)
);

INSERT INTO title_genres (tconst, genre)
SELECT
    tconst,
    TRIM(value) AS genre
FROM title_basics
CROSS APPLY STRING_SPLIT(genres, ',')
WHERE genres IS NOT NULL;

ALTER TABLE title_basics
DROP COLUMN genres;
```

Рис. 3.17 Нормалізація поля genres

г) Створено індекс idx_title_genres_tconst на полі genre у таблиці title_genres з включенням поля tconst, що дозволяє швидко знаходити всі фільми за заданим жанром.

```
CREATE INDEX idx_title_genres_tconst
ON title_genres(genre) include ([tconst])
```

Рис. 3.18 Створені індекси до таблиці title_genres

Таблиця 3.4

Порівняння продуктивності запитів до таблиці title_basics

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Пошук усіх фільмів після 2010 року (startYear >= 2010)	Кількість сканувань	1	1	0,00%
		Логічні зчитування	302526	61451	79,69%
		Процесорний час (мс)	2594	2110	18,66%
		Загальний час (мс)	19445	19215	1,18%
2	Пошук усіх серіалів (titleType = 'tvSeries')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	313713	3067	99,02%
		Процесорний час (мс)	1171	15	98,72%
		Загальний час (мс)	947	930	1,80%
3	Пошук усіх комедій (genre = 'Comedy')	Кількість сканувань	1	1	0,00%
		Логічні зчитування	302526	124157	58,96%
		Процесорний час (мс)	7484	6236	16,68%
		Загальний час (мс)	10419	8799	15,55%

5)Таблиця title_crew

Після аналізу запитів до таблиці title_crew та її структури було встановлено, що запити, які фільтрують записи за режисерами або сценаристами, працюють неефективно. Це пов’язано з тим, що поля directors і writers зберігають значення у вигляді списку ідентифікаторів nconst, розділених комами. Такий формат

унеможливлює використання індексів і змушує виконувати повільні операції пошуку за шаблоном із застосуванням конструкції LIKE '%...%', що значно погіршує продуктивність запитів.

Для підвищення продуктивності було виконано:

а)Проведено нормалізацію таблиці title_crew. А саме:

—створено нову таблицю title_crew_normalized, яка зберігає зв’язок між tconst та nconst з вказанням ролі director або writer

—значення з полей directors та writers були розділені за допомогою функції STRING_SPLIT() та перенесені у title_crew_normalized з вказанням role відповідно до того, з якого поля походить значення.

—табличку title_crew було видалено.

```
CREATE TABLE title_crew_normalized (
    tconst NVARCHAR(12) NOT NULL,
    nconst NVARCHAR(12) NOT NULL,
    role NVARCHAR(10) NOT NULL,
    FOREIGN KEY (tconst) REFERENCES title_basics(tconst),
    FOREIGN KEY (nconst) REFERENCES name_basics(nconst)
);

INSERT INTO title_crew_normalized (tconst, nconst, role)
SELECT
    tconst,
    value AS nconst,
    'director' AS role
FROM title_crew
CROSS APPLY STRING_SPLIT(directors, ',')
WHERE directors IS NOT NULL


INSERT INTO title_crew_normalized (tconst, nconst, role)
SELECT
    tconst,
    value AS nconst,
    'writer' AS role
FROM title_crew
CROSS APPLY STRING_SPLIT(writers, ',')
WHERE writers IS NOT NULL

drop table title_crew
```

Рис. 3.19 Нормалізація таблиці title_crew

б)Створено індекс idx_title_crew_normalized_nconst_role на полі nconst і role у таблиці title_crew_normalized з включенням поля tconst, що дозволяє швидко знаходити всі фільми, де певна особа була режисером.

```
CREATE INDEX idx_title_crew_normalized_nconst_role
ON title_crew_normalized (nconst, role) include (tconst);
```

Рис. 3.20 Створені індекси до таблиці title_crew_normalized

Таблиця 3.5

Порівняння продуктивності запитів до таблиці title_crew

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Список режисерських робіт конкретної особи (nconst = 'nm0478303')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	108541	4	99,996%
		Процесорний час (мс)	9359	13	99,86%
		Загальний час (мс)	884	63	92,87%

6)Таблиця title_episode

Після аналізу запитів до таблиці title_episode було виявлено, що найчастіше використовуваним критерієм фільтрації є поле parentTconst, яке пов’язує епізоди з відповідним серіалом. Запити також часто потребують доступу до номеру сезону (seasonNumber) та номеру епізоду (episodeNumber) для подальшого сортування або відображення.

Для підвищення продуктивності було виконано:

а) Створено індекс idx_title_episode_parentTconst на полі parentTconst з включенням полів seasonNumber та episodeNumber.

```
create nonclustered index idx_title_episode_parentTconst
on title_episode ([parentTconst]) INCLUDE ([seasonNumber],[episodeNumber])
```

Рис. 3.21 Створені індекси до таблиці title_episode

Таблиця 3.6

Порівняння продуктивності запитів до таблиці title_episode

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Список епізодів певного серіалу (parentTconst = 'tt0944947')	Кількість сканувань	13	1	92,31%
		Логічні зчитування	71317	596	99,16%
		Процесорний час (мс)	811	8	99,01%
		Загальний час (мс)	355	59	83,38%

7)Таблиця title_ratings

Після детального аналізу запитів до таблиці title_ratings було встановлено, що найбільш часто використовуються поля середнього рейтингу (averageRating) та кількості голосів (numVotes). Ці атрибути відіграють ключову роль у багатьох аналітичних запитах, оскільки дозволяють здійснювати ефективний відбір, сортування та порівняння фільмів за популярністю та якістю оцінок. Саме завдяки цим показникам користувачі мають змогу швидко знаходити найбільш рейтингові та затребувані відеопродукти. Отже, правильна індексація і оптимізація доступу до цих полів є надзвичайно важливою для забезпечення швидкої та продуктивної роботи бази даних у реальних умовах навантаження.

Для підвищення продуктивності було виконано:

а)Створено індекс idx_title_ratings_averageRating_numVotes на полях averageRating та numVotes.

```
create nonclustered index idx_title_ratings_averageRating_numVotes
on title_ratings ([averageRating],[numVotes])
```

Рис. 3.22 Створені індекси до таблиці title_ratings

Таблиця 3.7

Порівняння продуктивності запитів до таблиці title_ratings

№	Назва запиту	Метрика	До оптимізації	Після оптимізації	Результат
1	Список фільмів з оцінкою > 8	Кількість сканувань	13	1	92,31%
		Логічні зчитування	8781	1706	80,57%
		Процесорний час (мс)	484	78	83,88%
		Загальний час (мс)	1231	1193	3,09%

8)Матеріалізовані подання

Також після аналізу було вирішено створити декілька найчастіших запитів, які виконуються повторно, з однаковими параметрами фільтрації або об’єднання. Для покращення продуктивності виконання таких запитів було створено матеріалізовані подання з параметром WITH SCHEMABINDING, що дозволяє створити індекси на представленнях і забезпечити кешування результатів.

а)Створено матеріалізоване уявлення v_cached_high_rated_movies яке містить список фільмів з середньою оцінкою більше 8 і кількістю відгуків яка перевищує 10000. Також створено унікальний кластеризований індекс idx_top_rated_drama_movies.

```
CREATE VIEW dbo.v_cached_high_rated_movies
WITH SCHEMABINDING
AS
SELECT
    b.tconst,
    b.primaryTitle,
    r.averageRating
FROM dbo.title_basics AS b
JOIN dbo.title_ratings AS r
    ON b.tconst = r.tconst
WHERE r.averageRating >= 8.0 AND r.numVotes > 10000
order by averageRating desc ,numVotes desc;

CREATE UNIQUE CLUSTERED INDEX idx_cached_view
ON dbo.v_cached_high_rated_movies (tconst);
```

Рис. 3.23 Створене матеріалізоване уявлення v_cached_high_rated_movies і індекс до нього

Таблиця 3.8

Порівняння продуктивності запита для пошуку найпопулярніших фільмів

Метрика	**До оптимізації**	**Після оптимізації**	**Матеріалізоване уявлення**
Кількість сканувань	13	1	1
Логічні зчитування	31001	19628	29
Процесорний час (мс)	186	93	3
Загальний час (мс)	273	270	65

б)Створено матеріалізоване уявлення v_movies_2024 яке містить список фільмів 2024 року. Також створено унікальний кластеризований індекс idx_v_movies_2024.

```
CREATE VIEW dbo.v_movies_2024
WITH SCHEMABINDING
AS
SELECT tconst, primaryTitle, originalTitle, isAdult, startYear, runtimeMinutes
FROM dbo.title_basics
WHERE titleType = 'movie' AND startYear = '2024';

CREATE UNIQUE CLUSTERED INDEX idx_v_movies_2024
ON dbo.v_movies_2024 (tconst);
```

Рис. 3.24 Створене матеріалізоване уявлення v_movies_2024 і індекс до нього

Таблиця 3.9

Порівняння продуктивності запита для пошуку фільмів 2024 року

Метрика	До оптимізації	Після оптимізації	Матеріалізоване уявлення
Кількість сканувань	13	1	1
Логічні зчитування	309378	136600	300
Процесорний час (мс)	1030	94	15
Загальний час (мс)	241	208	154

3.3 Аналіз ефективності оптимізації бази даних

На основі порівняння продуктивності запитів до та після оптимізації, а також при використанні матеріалізованих уявлень, можна зробити такі ключові висновки:

1)Зменшення кількості сканувань

У більшості випадків кількість сканувань таблиць скоротилася з 13 до 1, що вказує на успішне використання індексів. Це означає, що запити більше не потребують повного перегляду всієї таблиці, щоб знайти необхідні дані. Натомість завдяки правильно налаштованим індексам SQL Server може напряму звертатися до потрібних рядків. Такий підхід значно знижує навантаження на систему зберігання даних та дозволяє суттєво скоротити час вибірки при роботі з великими обсягами інформації.

2)Скорочення логічних зчитувань

Згідно з отриманими результатами, логічні зчитування зменшилися в середньому на 90%. Це покращення має безпосередній вплив на продуктивність, оскільки логічні зчитування визначають, скільки сторінок даних повинно бути прочитано з буфера або оперативної пам’яті. Менша кількість таких зчитувань означає швидшу обробку запитів і зниження ризику перевантаження системи при одночасному виконанні кількох запитів.

3)Зменшення процесорного часу

Процесорний час зменшився зі значень у тисячі та сотні мілісекунд до десятків і навіть одиничних мілісекунд,в середньому 80%. Особливо помітне покращення досягається при використанні матеріалізованих уявлень, які дозволяють попередньо зберігати результати складних обчислень і об'єднань. Таким чином, під час виконання запиту система не витрачає ресурси на повторну побудову результатів, що значно знижує навантаження на процесор та дозволяє краще обробляти паралельні запити.

4)Скорочення загального часу виконання

Майже в усіх випадках спостерігається значне покращення загального часу виконання запитів,в середньому 40%, що є ключовим показником ефективності оптимізації. Скорочення часу виконання означає не лише зручність для кінцевого користувача, але й можливість обробляти більшу кількість запитів за той самий проміжок часу. Це особливо важливо в умовах багатокористувацьких систем, де швидкість відповіді впливає на загальну якість роботи інформаційного сервісу.

5)Ефективність нормалізації даних

Перенесення багатозначних полів у нормалізовані таблиці дозволило здійснювати фільтрацію по індексах, уникнути повного сканування таблиці і підвищити ефективність з'єднань.Також така нормалізація дозволила зменшити дублювання інформації та зробити базу даних більш гнучкою.

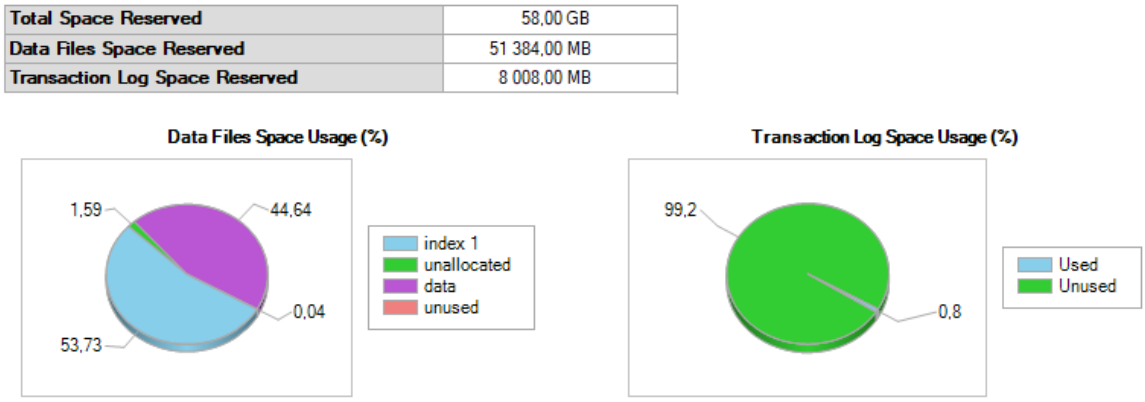


Рис. 3.25 Обсяг даних БД після оптимізації

Однак, як наслідок усіх покращень, загальний обсяг даних у базі збільшився з 24 ГБ до 58 ГБ, при цьому більша частина додаткового обсягу припадає на

індекси. Це є типовим компромісом між швидкістю виконання запитів та розміром бази даних.

Таким чином, запропоновані зміни значно покращили продуктивність системи і можуть вважатися успішними з точки зору оптимізації запитів і архітектури бази даних.

ВИСНОВКИ

У ході виконання дипломної роботи було реалізовано повноцінний цикл, що охоплює дослідження, проектування, реалізацію та оптимізацію великомасштабної реляційної бази даних. Робота поєднала глибоке теоретичне опрацювання сучасних методів підвищення продуктивності баз даних з їх практичним застосуванням у реальному середовищі. Це дозволило не лише сформувати ефективну структуру зберігання даних, але й продемонструвати конкретні результати, що свідчать про значне покращення продуктивності.

На теоретичному рівні було розглянуто ключові аспекти побудови баз даних: нормалізацію, особливості роботи з індексами, стратегії оптимізації запитів, використання кешування та матеріалізованих уявлень. Отримані знання стали основою для розумного і виваженого перепроектування бази даних.

У практичній частині робота зосереджувалась на аналізі і оптимізації обробці даних великої інформаційної системи IMDb. Було виявлено, що основними бар'єрами для швидкодії запитів є відсутність індексації, наявність полів із багатозначними значеннями, а також значне навантаження на ресурси під час виконання операцій. У відповідь на ці виклики здійснено оптимізацію бази даних, що включала нормалізацію таблиць, а також створення індексів і матеріалізованих уявлень, які відповідають типу навантаження та характеру запитів.

Оптимізаційні заходи, проведені в межах дослідження, дали чіткі та вимірювані результати, що підтверджують ефективність обраної стратегії підвищення продуктивності. Одним із найпомітніших досягнень стало зменшення кількості сканувань таблиць — у більшості випадків їхня кількість зменшилася з 13 до 1. Це свідчить про вдало реалізовану індексацію, яка дозволила уникнути повного перегляду таблиць і забезпечила точковий доступ до необхідних даних.

Крім того, спостерігалося значне скорочення логічних зчитувань — в середньому близько 90%. Це зменшило навантаження на оперативну пам’ять і суттєво прискорило виконання запитів.

Процесорний час також зменшився — зі значень у тисячі та сотні мілісекунд до десятків і навіть одиничних мілісекунд,в середньому 80%. Особливо помітні результати були досягнуті при застосуванні матеріалізованих уявлень, які дозволили уникнути повторного виконання складних обчислень і значно знизити витрати ресурсів.

Загальний час виконання запитів у багатьох випадках також скоротився, що є критично важливим для систем, орієнтованих на швидке отримання результатів при роботі з великими обсягами даних. У середньому зменшення загального часу виконання становило приблизно 40%, залежно від типу запиту.

У сукупності ці результати свідчать про високий рівень досягнутої оптимізації, яка значно покращила продуктивність системи. База даних стала більш ефективною, швидкою та готовою до обробки великих обсягів інформації у реальних умовах.

Попри те, що обсяг бази даних збільшився із 24 ГБ до 58 ГБ унаслідок створення індексів та додаткових структур, це розширення є цілком виправдане зважаючи на підвищення ефективності.

Однак, варто підкреслити, що процес оптимізації бази даних не є одноразовим актом. Ефективність зберігання та обробки даних залежить від постійного моніторингу навантаження, характеру запитів та змін у бізнес-вимогах. У цьому контексті регулярний аналіз продуктивності, переіндексація, оновлення матеріалізованих уявлень і адаптація схеми до нових викликів залишаються критично важливими для стабільної та масштабованої роботи системи.

Таким чином, результати, отримані у цій роботі, мають не лише академічну, а й практичну цінність. Запропоновані підходи можуть бути легко адаптовані до інших інформаційних систем, що працюють з великими наборами даних, і здатні забезпечити високу продуктивність. Це підтверджує актуальність і доцільність проведеного дослідження як з наукової, так і з прикладної точки зору.

ПЕРЕЛІК ПОСИЛАНЬ

1. Учасники проектів Вікімедіа. Великі дані – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Великі_дані (дата звернення: 14.05.2025).
2. Що таке Big Data? - Характеристики, приклади, методи обробки великих даних. Sigma Software University. URL: https://university.sigma.software/what-is-big-data/ (дата звернення: 14.05.2025).
3. A guide to database optimization for high traffic | last9. Last9: High Cardinality Observability at Scale | Last9. URL: https://last9.io/blog/a-guide-to-database-optimization/ (дата звернення: 14.05.2025).
4. Kaniganti S. T. Optimizing database performance for high-load systems. Journal of engineering and applied sciences technology. 2022. С. 1–12. URL: https://doi.org/10.47363/jeast/2022(4)244 (дата звернення: 14.05.2025).
5. Contributors to Wikimedia projects. Database normalization - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Database_normalization (дата звернення: 14.05.2025).
6. Effectively handling large datasets. Blog - Dataiku. URL: https://blog.dataiku.com/effectively-handling-large-datasets (дата звернення: 14.05.2025).
7. GeeksforGeeks. SQL Indexes - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/sql-indexes/ (дата звернення: 14.05.2025).
8. How to optimize SQL for large data sets | built in. Built In. URL: https://builtin.com/articles/optimize-sql-for-large-data-sets (дата звернення: 14.05.2025).
9. IMDb Non-Commercial Datasets. URL: https://developer.imdb.com/non-commercial-datasets/ (дата звернення: 14.05.2025).
10. Import flat file to SQL - SQL server. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/sql/relational-

databases/import-export/import-flat-file-wizard?view=sql-server-ver16 (дата звернення: 14.05.2025).

11. Мікула М., Коцюк Ю., Мікула О. Організація баз даних та знань. Вид-во Нац. ун-ту «Острозь-ка акад.», 2021. URL: https://doi.org/10.25264/978-617-8041-08-3 (дата звернення: 14.05.2025).
12. Big data optimization: recent developments and challenges / ред. A. Emrouznejad. Cham : Springer International Publishing, 2016. URL: https://doi.org/10.1007/978-3-319-30265-2 (дата звернення: 14.05.2025).
13. Anchlia A. Enhancing query performance through relational database indexing. International journal of computer trends and technology. 2024. Т. 72, № 8. С. 130–133. URL: https://doi.org/10.14445/22312803/ijctt-v72i8p119 (дата звернення: 14.05.2025).
14. Cao K., Li W., Church R. Big data, spatial optimization, and planning. Environment and planning B: urban analytics and city science. 2020. Т. 47, № 6. С. 941–947. URL: https://doi.org/10.1177/2399808320935269 (дата звернення: 14.05.2025).
15. Holubinka V., Khudyi A. Enhancing database query performance: analysis of indexing techniques. Vìsnik Nacìonal′nogo unìversitetu "L′vìvs′ka polìtehnìka". Serìâ Ìnformacìjnì sistemi ta merežì. 2024. Т. 15. С. 65–73. URL: https://doi.org/10.23939/sisn2024.15.065 (дата звернення: 14.05.2025).
16. Improving database performance with SQL server optimization techniques / Krishna Kishor Tirupati та ін. Modern dynamics: mathematical progressions. 2024. Т. 1, № 2. С. 450–494. URL: https://doi.org/10.36676/mdmp.v1.i2.32 (дата звернення: 14.05.2025).
17. Indexing Object Database using HC-Tree. International journal of engineering and advanced technology. 2020. Т. 9, № 3. С. 3205–3208. URL: https://doi.org/10.35940/ijeat.c6088.029320 (дата звернення: 14.05.2025).
18. Kumar R. AI-Augmented database indexing for high-performance query optimization. International scientific journal of engineering and management.

2023. Т. 02, № 11. С. 1–7. URL: https://doi.org/10.55041/isjem01292 (дата звернення: 14.05.2025).

19. Roy C., Swarup Rautaray S., Pandey M. Big data optimization techniques: a survey. International journal of information engineering and electronic business. 2018. Т. 10, № 4. С. 41–48. URL: https://doi.org/10.5815/ijieeb.2018.04.06 (дата звернення: 14.05.2025).
20. Semeniuk V. Аналіз та оптимізація продуктивності баз даних у розподілених системах. Computer-integrated technologies: education, science, production. 2024. № 54. С. 199–205. URL: https://doi.org/10.36910/6775-2524-0560-2024-54-25 (дата звернення: 14.05.2025).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Бази даних для оптимізації роботи з великими обсягами інформації

Виконав студент 4 курсу
Групи ІСД-43
Омельченко А.В.
Науковий керівник:
Шахматов І.О.

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження:
Дослідити особливості зберігання та обробки великих обсягів даних у сучасних СУБД, а також реалізувати ефективні підходи до їх оптимізації задля підвищення продуктивності, масштабованості та швидкодії систем.

Об’єкт дослідження:
Процеси зберігання, обробки та оптимізації структур баз даних у СУБД при роботі з великими обсягами даних.

Предмет дослідження:
Методи аналізу, моделювання та оптимізації баз даних, що забезпечують ефективну роботу з великими наборами даних у середовищі сучасних систем управління базами даних.

Характеристика об'єкта дослідження

name.basics.tsv

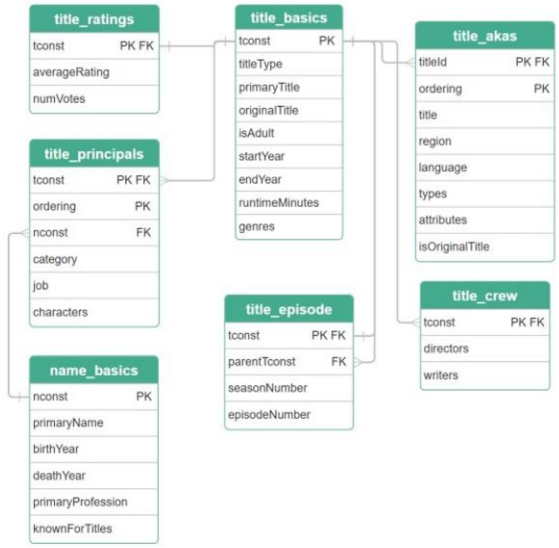
	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
2	nm0000001	Fred Astaire	1899	1987	actor,miscellaneous,producer	tt0072308,tt0050419,tt0027125,tt0031983
3	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack,archive_footage	tt0037382,tt0075213,tt0117057,tt0038355
4	nm0000003	Brigitte Bardot	1934	\N	actress,music_department,producer	tt0057345,tt0049189,tt0056404,tt0054452
5	nm0000004	John Belushi	1949	1982	actor,writer,music_department	tt0072562,tt0077975,tt0080455,tt0078723
6	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0050986,tt0069467,tt0050976,tt0083922
7	nm0000006	Ingrid Bergman	1915	1982	actress,producer,soundtrack	tt0034583,tt0038109,tt0036855,tt0038787
8	nm0000007	Humphrey Bogart	1899	1957	actor,producer,miscellaneous	tt0034583,tt0043265,tt0033870,tt0037382
9	nm0000008	Marlon Brando	1924	2004	actor,director,writer	tt0078788,tt0068646,tt0047296,tt0070849
10	nm0000009	Richard Burton	1925	1984	actor,producer,director	tt0061184,tt0087803,tt0059749,tt0057877
11	nm0000010	James Cagney	1899	1986	actor,director,producer	tt0029870,tt0031867,tt0042041,tt0034236

Вигляд вмісту файла датасета IMDb

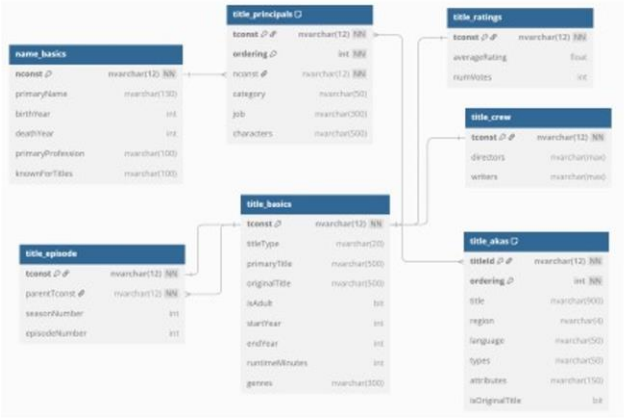
name.basics.tsv.gz

- nconst (string) - alphanumeric unique identifier of the name/person
- primaryName (string)– name by which the person is most often credited
- birthYear – in YYYY format
- deathYear – in YYYY format if applicable, else '\N'
- primaryProfession (array of strings)– the top-3 professions of the person
- knownForTitles (array of tconsts) – titles the person is known for

Опис вмісту файла датасета IMDb



Логічна модель



Фізична модель

Реалізація в СУБД

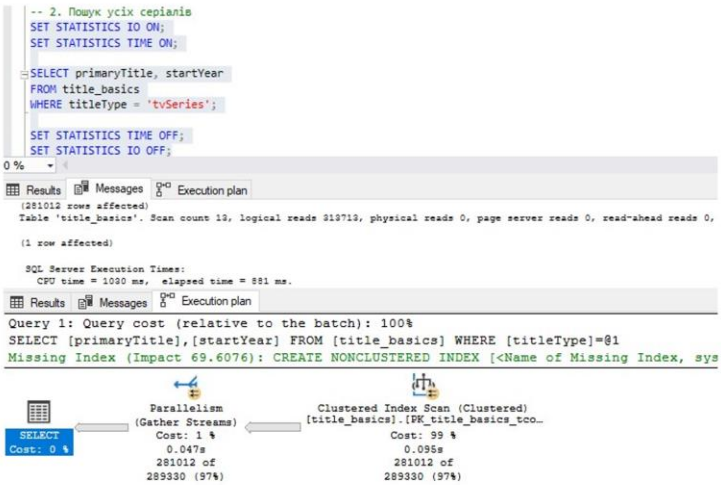
Total Space Reserved	23,70 GB
Data Files Space Reserved	19 784,00 MB
Transaction Log Space Reserved	4 488,00 MB

Загальний обсяг бази даних

Назва таблиці	Кількість записів
title_principals	92 343 976
title_akas	52 035 728
name_basics	14 376 758
title_basics	11 630 899
title_crew	11 630 712
title_episode	8 949 408
title_ratings	1 565 445

Кількість записів в таблицях

Аналіз продуктивності



Приклад аналіза запита

Оптимізація

```sql
CREATE INDEX idx_title_basics_startYear
ON title_basics(startYear)
INCLUDE ([primaryTitle]);

CREATE INDEX idx_title_basics_titleType
ON title_basics(titleType) include ([primaryTitle],[startYear])
```

1.Створення індексів

```sql
CREATE VIEW dbo.v_cached_high_rated_movies
WITH SCHEMABINDING
AS
SELECT
    b.tconst,
    b.primaryTitle,
    r.averageRating
FROM dbo.title_basics AS b
JOIN dbo.title_ratings AS r
    ON b.tconst = r.tconst
WHERE r.averageRating >= 8.0 AND r.numVotes > 10000
order by averageRating desc ,numVotes desc;

CREATE UNIQUE CLUSTERED INDEX idx_cached_view
ON dbo.v_cached_high_rated_movies (tconst);
```

2.Нормалізація таблиць

```sql
CREATE TABLE title_genres (
    tconst NVARCHAR(12) NOT NULL,
    genre NVARCHAR(20) NOT NULL,
    FOREIGN KEY (tconst) REFERENCES title_basics(tconst)
);

INSERT INTO title_genres (tconst, genre)
SELECT
    tconst,
    TRIM(value) AS genre
FROM title_basics
CROSS APPLY STRING_SPLIT(genres, ',')
WHERE genres IS NOT NULL;

ALTER TABLE title_basics
DROP COLUMN genres;
```

3.Створення матеріалізованих уявлень

Результат оптимізації

№	Назва запиту	Скорочення сканувань (%)	Скорочення зчитувань (%)	Скорочення CPU (%)	Скорочення часу (%)
1	Пошук усіх акторів	92,31	83,51	76,13	2,52
2	Пошук фільмів за участі конкретної особи	92,31	99,66	99,83	88,85
3	Пошук усіх фільмів за мовою 'es'	92,31	92,15	78,24	3,39
4	Пошук усіх фільмів для регіону 'JP'	92,31	92,76	75,61	2,14
5	Пошук усіх осіб за роком народження	92,31	99,94	99,49	89,93
6	Пошук усіх осіб за роком смерті	92,31	99,98	98,59	41,98
7	Пошук фільмів, з якими асоціюється особа	0	99,99	99,91	99,9
8	Пошук фільмів після 2010 року	0	79,69	18,66	1,18
9	Пошук усіх серіалів	92,31	99,02	98,72	1,8
10	Пошук усіх комедій	0	58,96	16,68	15,55
11	Список режисерських робіт	92,31	99,99	99,86	92,87
12	Список епізодів серіалу	92,31	99,16	99,01	83,38
13	Список фільмів з оцінкою > 8	92,31	80,57	83,88	3,09
	Підсумок	71,01	91,18	80,35	40,51

Total Space Reserved	58,00 GB
Data Files Space Reserved	51 384,00 MB
Transaction Log Space Reserved	8 008,00 MB

Загальний обсяг бази даних після оптимізації

Висновки

- У результаті виконання роботи вдалося суттєво підвищити продуктивність бази данихIMDb шляхом впровадження індексів, нормалізації таблиць і використання матеріалізованих уявлень. Це дозволило зменшити кількість сканувань таблиць у 13 разів, логічні зчитування — в середньому на 90%, а процесорний час— на 80%. Загальний час виконання запитів скоротився приблизно на 40%, що критично важливо для систем, які працюють із великими обсягами даних.
- Незважаючи на збільшення розміру бази з 24 до 58 ГБ, оптимізація виявилася ефективною й обґрунтованою. Результати мають не лише академічну, а й практичну цінність, оскільки обрані підходи легко масштабуються та можуть бути адаптовані до інших систем. Важливо пам’ятати, що оптимізація— це постійний процес, який потребує регулярного моніторингу та адаптації до змін.

Апробація

- 1.VI Міжнародна науково-практична конференція для студентів, аспірантів, докторантів, молодих учених «НАУКА ТА ОСВІТА В ДОСЛІДЖЕННЯХ МОЛОДИХ УЧЕНИХ» 16.05.25- **ЗАСТОСУВАННЯ АНАЛІТИКИ ВЕЛИКИХ ДАНИХ У ІНДИВІДУАЛЬНІЙ ОСВІТНІЙ ТРАЄКТОРІЇ**

Дякую за увагу!