Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Оптимізація роботи СУБД MS SQL Server: від рівня сервера до рівня запитів

Текстова частина до курсової роботи
за спеціальністю „Прикладна математика”

Керівник курсової роботи
ст. викладач __Сініцина Р.Б.__
(прізвище та ініціали)

____
(підпис)

“____” __________ 2022 р.

Виконала студентка __Соколова Д.Т.__
(прізвище та ініціали)

“____” __________ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,
доктор фіз-мат наук Г.І. Малашонок

____

(підпис)

„____”________________2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Соколовій Діані Тимурівні факультету інформатики 3 курсу

ТЕМА Оптимізація роботи СУБД MS SQL Server: від рівня сервера до рівня запитів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Вступ

Теоретичні відомості

Аналіз рішень

Висновки

Список літератури

Дата видачі „___” _________ 2022 р. Керівник ________________

(підпис)

Завдання отримав ___________

(підпис)

Тема: Оптимізація роботи СУБД MS SQL Server: від рівня сервера до рівня запитів

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	22.10.2021	
2.	Огляд літератури за темою роботи.	30.11.2021	
3.	Виконання аналізу актуальності теми	10.02.2022	
4.	Аналіз рішень	05.05.2022	
5.	Написання пояснювальної роботи.	31.05.2022	
6.	Попередня демонстрація проекту науковому керівнику	04.06.2022	

Студент <u>Соколова Д.Т.</u>

Керівник <u>Сініцина Р.Б.</u>

“______” _______________

ЗМІСТ

Ст.

Індивідуальне завдання. 2

Календарний план виконання роботи. 3

Вступ. 6

РОЗДІЛ 1: Засоби діагностики роботи серверу. 8

1.1. Монітор активності. 8

1.2. Звіти продуктивності. 9

1.2.1. Інформаційна панель сервера 10

1.2.2. Панель керування продуктивністю. 11

1.3. Статистика очікування. 12

1.3.1. Планування потоків. 13

1.3.2. Sos_scheduler_yield wait 14

1.3.3. Pageiolatch wait. 15

РОЗДІЛ 2: Оптимізація та аналіз запитів 17

2.1. Індексування. 17

2.1.1. Типи індексів. 18

2.1.2. Індекси та обмеження 19

2.1.3. Чому варто використовувати індекси? 20

2.1.4. Чому надмірна кількість індексів є недоліком? 20

2.1.5 Дефрагментація індексів. 21

2.2. Аналіз та робота з запитами. 21

2.2.1. План виконання. 22

2.2.2. Статистика IO. 27

2.2.3. Сховище запитів. 28

2.2.4. Представлення динамічного управління. 30

2.2.5. Написання запитів. 30

Висновки 35

Джерела 36

ВСТУП

У сучасному світі бази даних є важливими у будь-якій галузі, оскільки вони пропонують ефективний спосіб обробки великих обсягів різних типів даних. Можливість ефективного доступу до даних дозволяє компаніям швидше приймати зважені рішення, задовольняти потреби користувачів.

Щоб зберегти своїх користувачів, будь-яка організація має працювати швидко. Для критичних середовищ затримка на пару мілісекунд в отриманні інформації може створити великі проблеми. Оскільки розміри бази даних зростають з кожним днем, нам потрібно якомога швидше отримати дані та якомога швидше записати їх назад у базу даних. Щоб переконатися, що всі операції виконуються безперебійно, ми повинні налаштувати наш сервер бази даних на високу продуктивність.

Оптимізація - це невід’ємна частина роботи, яка завжди буде актуальною, оскільки швидкість роботи впливає на якість продукту, а вимоги до якості продукту зростають з кожним роком.

Оптимізація – це процес створення чогось якомога більш досконалого, функціонального та ефективного. Оптимізація бази даних – це стратегія скорочення часу відповіді системи баз даних і надання інформації, що зберігається з ієрархічною та пов’язаною структурою, що дозволяє нам легко витягувати вміст і впорядковувати його.

Налаштування бази даних необхідне для легкої організації та доступу до даних. Чим більше у вас даних, тим повільнішим може відбуватися пошук та обробка інформації. Ось чому на оптимізацію витрачають немало сил часу та грошей не тільки на початкових стадіях розробки, а й у подальшому розвитку продукту.

Оскілки оптимізація роботи є настільки важливою, у цій курсовій роботі я вирішила більш детально розібратися та дослідити, що може спричиняти повільну роботу баз даних.

Мета курсової роботи.

Зробити дослідження роботи MS SQL Server та показати наскільки важливою є оптимізація та коректний аналіз.

Завдання.

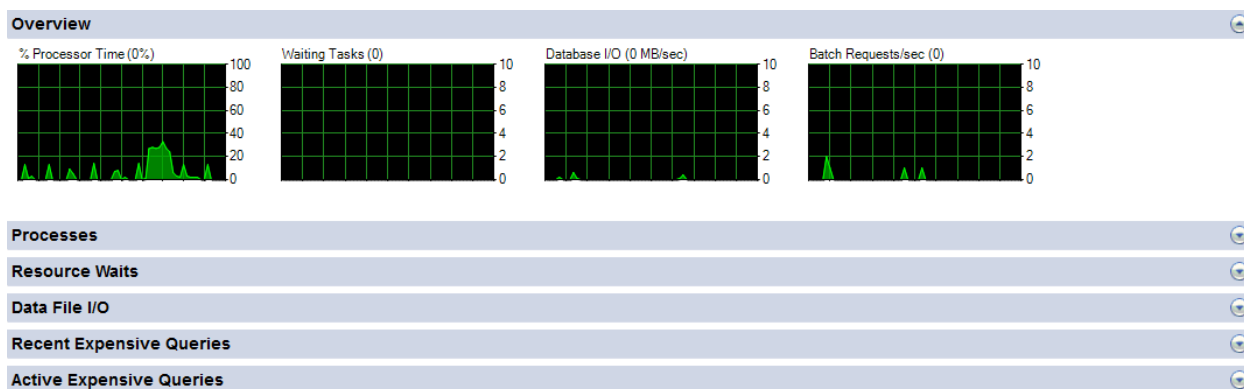
При написанні курсової роботи, було розгорнуто та налаштовано сервер на локальній машині. Для керування, налаштування та адміністрування всіх компонентів серверу було використано SQL Server Management Studio. Усі дії та аналіз були виконані на основі існуючої навчальної бази даних AdventureWorks2016_EXT.

РОЗДІЛ 1: Засоби діагностики роботи серверу

1.1 Activity Monitor (Монітор активності)

Монітор активності повідомляє нам, які поточні та останні дії є у SQL Server.

На знімку екрана нижче показано вікно огляду для монітора активності. На цьому екрані відображатимуться графіки часу процесора, завдань очікування та пакетних запитів. Як правило, чим менша кількість підрахунків, тим краща продуктивність.



У великих організаціях з величезним навантаженням може бути величезна кількість пакетних запитів із високим часом роботи процесора, але це не обов’язково вказує на проблеми з продуктивністю.

Процеси (Processes)

Після огляду потрібно зосередитися на вкладці Процеси. Вони дають нам доступ до перегляду всіх процесів, що виконуються, і переглянути, скільки процесів очікують або заблоковані. Таким чином ми можете зрозуміти, чи у нас повільно виконуються запити через якесь конкретне очікування, або запити, що займають час, блокуються іншими процесами.

Processes

S...	U...	Login	Dat...	Tas...	Com...	Appl...	Wait Tim...	Wait...	Wait...	B...	H...	Me...	Host...	Wor...
52	1	DIANA\...	tempdb	RUNNING	SELECT	Microsoft...	0					32	DIANA	default
53	1	DIANA\...	master			Microsoft...	0					0	DIANA	default
56	1	DIANA\...	Adventur...			Microsoft...	0					32	DIANA	default
57	1	NT SER...	master			SQLServ...	0					32	DIANA	default
59	1	DIANA\...	master			Microsoft...	0					32	DIANA	default
61	1	DIANA\...	Adventur...	RUNNING	SELECT	Microsoft...	0					32	DIANA	default
68	1	DIANA\...	Adventur...			Microsoft...	0					32	DIANA	default
69	1	DIANA\...	Adventur...			Microsoft...	0					32	DIANA	default
70	1	DIANA\...	master			Microsoft...	0					32	DIANA	default

Блокуються запити, які фактично призупинено через будь-який інший процес, який працює з ресурсами, від яких залежить процес. Отже, якщо є запити, які блокуються іншими процесами, потрібно перевірити наявність кореневого блокувальника, який спричиняє всі блокування, переглянувши стовпець Заблоковано (Blocked by).

<u>Очікування ресурсу (Wait Resource)</u>

Тут знаходяться запити, які очікують на будь-який конкретний ресурс і дають інформацію про ресурс очікування, щоб ми могли перевірити тип очікування та спробувати знайти рішення для цієї проблеми.

<u>Дорогі Останні запити (Recent Expensive Queries) & Дорогі Активні запити (Active Expensive Queries)</u>

Тут ми можемо подивитись інформацію про запити, які мають високий рівень ЦП, логічні читання або високий час, що минув. Їх можна відсортувати за витраченим часом, логічним читанням і часом ЦП по одному та перевірте план виконання.

Recent Expensive Queries

Query	Executions/...	CPU (ms/sec)	Physical Reads...	Logical Writ...	Logical ...	Average...	Plan Co...	Dat...
SELECT p.[Name], SUM(th.Quantity), SUM(tha...	0	997	0	0	19264	0	1	Adventur...

Active Expensive Queries

Query	Ses...	CPU (ms/sec)	Database	Elapsed ...	Physical ...	Writes	Logical ...	Row Co...	Allocate...

<u>1.2 Звіти продуктивності (Performance Reports)</u>

Стандартні звіти – набір звітів з показниками продуктивності SQL Server, доступних у SQL Server Management Studio . Це набір вбудованих звітів, які

охоплюють найбільш поширені вимоги до продуктивності та моніторингу. Server Management Studio надає найширший діапазон звітів для рівня екземпляра SQL Server. Існує понад 20 звітів, які показують найважливіші показники продуктивності SQL Server, історію змін, використання пам'яті, активність, найпопулярніші транзакції, статистику продуктивності Service Broker і статус доставки журналу транзакцій. Їх можна використовувати для швидкого виявлення проблем з пам'яттю, найдорожчих запитів, блокування транзакцій тощо.

1.2.1. Інформаційна панель сервера (Server dashboard)

Інформаційна панель сервера – містить інформацію, яку зазвичай аналізують, коли йдеться про моніторинг продуктивності SQL Server. Він показує основну інформацію про конфігурацію екземпляра SQL Server, налаштування та дії в діаграмах і таблицях: деталі конфігурації, параметри конфігурації, які не є за замовчуванням, відомості про дії та графіки для використання процесора та здійснення введення-виведення.

Server Dashboard
on DIANA at 03.06.2022 13:05:07
SQL Server

This report provides overview data about the SQL Server instance, its configuration, and activity on it.

Configuration Details:

Server Startup Time	Jun 3 2022 12:19AM
Server Instance Name	DIANA
Product Version	15.0.2080.9
Edition	Developer Edition (64-bit)
Scheduled Agent Jobs	1

Server Collation	Ukrainian_CI_AS
Is Clustered	No
Is FullText Installed	No
Is Integrated Security Only	Yes
# Processors (used by instance)	4

Non Default Configuration Options:

Configuration Option	Run Value	Default Value
cost threshold for parallelism	40	5
max degree of parallelism	4	0
max server memory (MB)	2048	2147483647
min server memory (MB)	1024	0
remote login timeout (s)	10	20

Activity Details:

Active Sessions	1
Active Transactions	7
Active Databases	5
Total Server Memory (KB)	1128528
Idle Sessions	4

Blocked Transactions	0
Distinct Connected Logins on Sessions	2
Traces Running	1

* : "CPU Usage" and "IO Performed" charts show the cumulative share of all objects by databases.

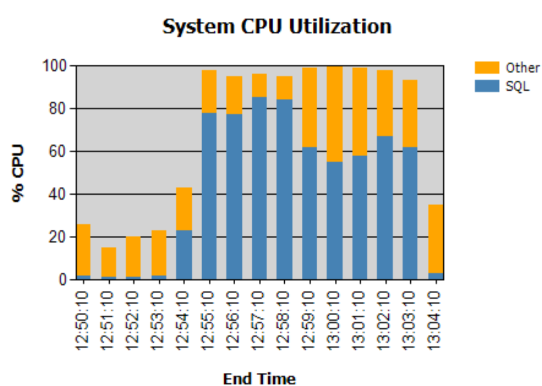
1.2.2. Панель керування продуктивністю (Performance Dashboard)

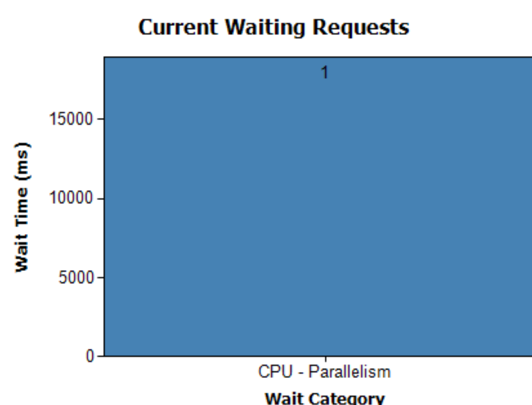
Панель керування продуктивністю була розроблена, щоб візуально забезпечити швидке уявлення про стан продуктивності SQL Server. Вона допомагає швидко визначити, чи відчуває SQL Server вузьке місце в продуктивності. А якщо таке місце виявлено, то вона збирає додаткові діагностичні дані, які можуть знадобитися для вирішення проблеми.

Microsoft SQL Server Performance Dashboard

Report Local Time: 03.06.2022 13:04:56

DIANA(15.0.2080.9 - Developer Edition (64-bit))





Current Activity

	User Requests	User Sessions
Count	1	5
Elapsed Time (ms)	6	199712
CPU Time (ms)	6(100,00%)	11339(5,68%)
Wait Time (ms)	0(0,00%)	188373(94,32%)
Cache Hit Ratio	100,000%	90,564%

Historical Information

Waits — IO Statistics

Latches

Expensive Queries

By CPU — By Duration

By Logical Reads — By Physical Reads

By Logical Writes — By CLR Time

Miscellaneous Information

Active Traces	1
Active Xevent Sessions	4
Databases	5
Missing Indexes	2

Деякі поширені проблеми з продуктивністю, які може допомогти визначити інформаційна панель продуктивності, включають:

- Вузькі місця ЦП (і які запити споживають найбільше ЦП)
- Вузькі місця введення-виводу (і які запити виконують найбільше введення-виводу)

- Рекомендації щодо індексів, створені оптимізатором запитів (відсутні індекси)
- Блокування
- Конфлікт щодо ресурсів (включаючи суперечку про фіксацію)

Панель керування продуктивністю також допомагає визначати дорогі запити, які, можливо, виконувались раніше, і доступні кілька показників для визначення високої вартості: ЦП, логічні записи, логічні читання, тривалість, фізичні читання та час CLR.

1.3. Статистика очікування

Статистика очікування - це методологія, яка поєднує всі доступні джерела системної інформації, включаючи очікування, черги та іншу інформацію про продуктивність, не пов’язану з SQL Server – є оптимальним підходом до усунення неполадок з продуктивністю.

Аналіз статистики очікування забезпечує чудову віддачу від часу, витраченого на усунення неполадок з продуктивністю, оскільки він дозволяє легко побачити області, де SQL Server проводить час в очікуванні виконання роботи. Це особливо корисно, коли використовується в поєднанні з іншими інструментами та показниками.

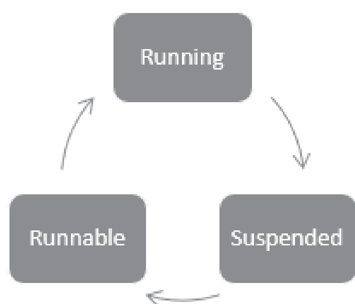
З терміну «статистика очікування» випливає, що це тип системних показників, призначених для того, щоб розповісти нам, яким чином користувачі та системні процеси змушені чекати. Статистика очікування SQL Server на найвищому концептуальному рівні згрупована у дві великі категорії: *очікування сигналу* та *очікування ресурсу* . Сигнал очікування накопичується процесами, що виконуються на SQL Server і чекають, поки ЦП стане доступним. Очікування ресурсу накопичується процесами, що виконуються на SQL Server і чекають, поки певний ресурс стане доступним, наприклад, очікування зняття блокування для певного запису.

Очікування ресурсів далі розбивається на сотні конкретних категорій. Ці категорії, які називаються *типами очікування*, дозволяють визначити більш конкретне вузьке місце на SQL Server, наприклад проблеми із блокуванням схеми, тиском кеш-пам’яті або проблеми введення-виводу резервного копіювання. Аналіз статистики очікування полягає в тому, що ми швидко можемо визначити, де найбільший тиск створюється на певні ресурси в SQL Server.

1.3.1. Планування потоків

Планування потоків у SQL Server керується самим SQL Server.

Потоки можуть бути в одному з трьох станів:



- RUNNING – тільки один потік, одночасно, може активно працювати на ЦП.
- SUSPENDED - потік очікує в waiter list, доки ресурс не стане доступним.
- RUNNABLE – ресурс доступний, але потік уже запущений, тому потік знаходиться в черзі, яка може виконуватися.

Кожному потоку надається тривалість (або квант) 4 мілісекунди для використання ЦП без перерв.

Коли процес готовий до виконання, він переходить до планувальника в стані RUNNABLE. Коли він потрапляє до початку черги, він змінюється на RUNNING і може виконуватися.

Якщо кожен планувальник запускає потік, ті, які вичерпали свій квант, добровільно поступляться, щоб дати наступному потоку в черзі виконати свій час процесора.

Вихідні потоки не потрапляють до Waiter List, оскільки вони не чекають ресурсу. Замість цього вони переходять в нижню частину черги, яка може виконуватися, і чекають, поки їх кількість буде оновлена.

Якщо більшість очікувань відноситься до цього типу і ваша система відчуває проблеми з продуктивністю, це означає, що ваша система страждає від тиску ЦП.

1.3.2. Sos_scheduler_yield wait

SOS_SCHEDULER_YIELD – є найпоширенішим очікуванням на сервері, часто спостерігається коли йде стабільне високе використання ЦП, тому цей тип очікування, не обов'язково вказує на проблему, а може свідчити про ефективне планування процесора SQL Server.

Існує дві основні причини SOS_SCHEDULER_YIELD:

- потік, що вичерпує свій плановий квант
- потоку не вдалося отримати блокування обертання

Майже в 100% випадків перший випадок – це те, що відбувається. Вкрай рідко спин-блокування стає причиною високого рівня ЦП і високого SOS_SCHEDULER_YIELD.

Єдиний спосіб довести, що спин-блокування є чи не є причиною, — це захопити стеки викликів SQL Server, коли виникає тип очікування, використовуючи розширені події та символи налагодження від Microsoft.

У випадку квантового виснаження це не є першопричиною. Це ще один симптом. Тепер нам потрібно розглянути, чому потік може постійно вичерпувати свій квант.

Потік може вичерпати свій квант лише тоді, коли він може продовжувати обробку коду SQL Server протягом чотирьох мілісекунд, не потребуючи ресурсу, яким володіє інший потік.

Найпоширеніший фрагмент коду, де може виникнути квантове виснаження та накопичити велику кількість очікувань SOS_SCHEDULER_YIELD, — це сканування індексу/таблиці, де всі необхідні сторінки файлу даних знаходяться в пам'яті, і немає жодних суперечок за доступ до цих сторінок.

Це не означає, що великі сканування є неправильними, оскільки найефективнішим способом обробки вашого робочого навантаження може бути сканування. Однак, якщо очікування SOS_SCHEDULER_YIELD є новими та незвичними та викликані великими скануваннями, вам слід дослідити, чому плани запитів використовують сканування. Можливо, хтось випустив критичний некластеризований індекс або статистика застаріла, тому вибрано неправильний план запиту. Можливо, незвичне значення параметра було передано в збережену процедуру, і план запиту викликав сканування або зміни коду відбулися без підтримки додавання індексу.

1.3.3. Pageiolatch wait

PAGEIOLATCH_SH — очікує, що сторінка файлу даних буде перенесена з диска в пул буферів, щоб її вміст можна було прочитати.

PAGEIOLATCH_EX — очікує, поки сторінка файлу даних буде перенесена з диска в пул буферів, щоб її вміст можна було змінити.

PAGEIOLATCH_SH є більш поширеним, тому якщо на сервері виникає такий тип очікування, слід зосередитися на тому, що підсистема вводу-виводу має проблему.

Перше, що потрібно зробити, це порівняти кількість і тривалість очікування PAGEIOLATCH_SH з вашим базовим рівнем. Якщо обсяг очікувань більш-менш однаковий, але тривалість кожного очікування читання стала набагато довшою, тоді проблема скоріш за все зв'язана з підсистемою введення-виводу:

- Неправильна конфігурація/несправність на рівні підсистеми вводу/виводу
- Затримка мережі
- Ще одне робоче навантаження введення-виводу викликає суперечки з нашим робочим навантаженням
- Конфігурація синхронної реплікації/відображення підсистеми введення-виводу

Якщо кількість очікувань PAGEIOLATCH_SH значно зросла в порівнянні з базовою (нормальною) величиною, а тривалість очікування також збільшилася (тобто час для читання вводу-виводу збільшився), оскільки велика кількість читань перевантажує підсистему вводу-виводу. Тоді це не проблема підсистеми вводу-виводу, а SQL Server використовує більше операцій введення-виводу, ніж слід. Тому в такій ситуації потрібно переключитися на SQL Server, щоб визначити причину додаткових операцій введення/виводу.

РОЗДІЛ 2: Оптимізація та аналіз SQL Server

2.1 Індексування

Індексування — це метод структури даних, який дозволяє швидко отримувати записи з файлу бази даних. Індексування таблиці або представлення, це один із найкращих способів покращити продуктивність запитів і програм.

Індекс - це спеціальні таблиці, які використовуються пошуковими системами для пошуку даних. Їх активне використання грає найважливішу роль підвищенні продуктивності SQL серверів.

Індекси бази даних використовуються для підвищення швидкості операцій з базою даних у таблиці з великою кількістю записів. Індекси баз даних (як кластеризовані, так і некластеризовані індекси) за своєю функціональністю дуже схожі на індекси книг. Книжковий покажчик дозволяє перейти безпосередньо до різних тем, які обговорюються в книзі. Якщо ви хочете шукати конкретну тему, ви просто перейдіть до індексу, знайдіть номер сторінки, яка містить тему, яку ви шукаєте, а потім можете перейти прямо на цю сторінку. Без покажчика вам довелося б шукати всю книгу.

Таким же чином працюють індекси бази даних. Без індексів вам довелося б шукати всю таблицю, щоб виконати конкретну операцію з базою даних. За допомогою індексів вам не потрібно переглядати всі записи таблиці. Індекс вказує безпосередньо на запис, який ви шукаєте, значно скорочуючи час виконання запиту.

Індекс:

1) Бере ключ пошуку як вхід
2) Ефективно повертає колекцію відповідних записів.

2.1.1 Типи індексів

Індекси SQL Server можна розділити на два основних типи:

- Кластерні індекси
- Некластеризовані індекси

Кластеризовані індекси сортують і зберігають рядки даних у таблиці або представленні на основі їхніх значень ключів. Це стовпці, включені у визначення індексу. У таблиці може бути тільки один кластеризований індекс, оскільки самі рядки даних можуть зберігатися лише в одному порядку. Тобто, кластеризовані індекси використовує первинний ключ для організації даних у таблиці. Кластеризований індекс гарантує, що первинний ключ зберігається в порядку зростання, який також є порядком, який таблиця зберігає в пам’яті.

Переваги та використання кластерного індексу SQL Server

Кластеризований індекс, визначений у таблиці, має багато переваг, але головною перевагою є прискорення виконання запитів. Запити, які містять ключові стовпці індексу в пропозиції WHERE, використовують структуру індексу, щоб перейти безпосередньо до даних таблиці. Кластеризований індекс також усуває потребу в додатковому пошуку для отримання решти даних стовпця під час запитів на основі значень ключа індексу. Це те, що не вірно для інших типів індексів. Ви також можете позбутися від необхідності сортування даних. Якщо речення ORDER BY запиту засноване на значеннях ключа індексу, сортування не потрібно, оскільки дані вже впорядковані за цими значеннями.

Недоліки кластерного індексу SQL Server

У кластерних індексах є кілька недоліків. Підтримка структури індексу щодо будь-якої операції DML (INSERT, UPDATE, DELETE) вимагає певних витрат. Це особливо вірно, якщо ви оновлюєте фактичні значення ключа в індексі, оскільки в цьому випадку всі пов’язані дані таблиці також мають бути

переміщені, оскільки вони зберігаються в кінцевому вузлі запису індексу. У кожному випадку це вплине на продуктивність вашого запиту DML.

Некластеризовані індекси – це відсортовані посилання для певного поля з основної таблиці, які утримують покажчики на вихідні записи таблиці. Некластеризований індекс не сортує фізичні дані всередині таблиці. Фактично, некластеризований індекс зберігається в одному місці, а дані таблиці — в іншому.

Переваги і недоліки некластеризованого індексу подібні до кластерного індексу, про який ми згадували вище, головною перевагою є прискорення виконання запитів. Однак є дві відмінності. Перший полягає в тому, що ви можете мати кілька некластеризованих індексів, визначених в одній таблиці. Це дозволяє вам індексувати різні стовпці, що може допомогти запитам з різними стовпцями в пункті WHERE, що дозволяє швидше отримувати дані, і в пропозиції ORDER BY, щоб усунути потребу в сортуванні. По-друге, хоча для некластеризованого індексу є накладні витрати, коли справа доходить до операцій DML, вони менші, ніж його кластеризований аналог.

Це робить запити повільніше, ніж кластеризовані індекси, але зазвичай набагато швидше, ніж неіндексований стовпець.

Як кластеризовані, так і некластеризовані індекси можуть бути унікальними. Це означає, що два рядки не можуть мати однакове значення для ключа індексу. В іншому випадку індекс не є унікальним, і кілька рядків можуть мати однакове значення ключа.

Індекси автоматично підтримуються для таблиці або представлення кожного разу, коли дані таблиці змінюються.

2.1.2 Індекси та обмеження

Індекси створюються автоматично, коли в стовпцях таблиці визначено обмеження PRIMARY KEY та UNIQUE. Наприклад, коли ви створюєте таблицю з обмеженням UNIQUE, Database Engine автоматично створює

некластеризований індекс. Якщо ви налаштуєте ПЕРВИННИЙ КЛЮЧ, Database Engine автоматично створює кластеризований індекс, якщо кластеризований індекс уже не існує. Коли ви намагаєтеся застосувати обмеження PRIMARY KEY для наявної таблиці, а кластеризований індекс уже існує для цієї таблиці, SQL Server застосовує первинний ключ за допомогою некластерного індексу.

2.1.3 Чому варто використовувати індекси?

- Це допомагає вам зменшити загальну кількість операцій введення-виводу, необхідних для отримання цих даних, тому вам не потрібно звертатися до рядка в базі даних зі структури індексу.
- Пропонує користувачам швидший пошук і отримання даних.
- Індексування також допомагає зменшити табличний простір, оскільки вам не потрібно посилатися на рядок у таблиці, оскільки немає необхідності зберігати ROWID в індексі. Таким чином ви зможете зменшити простір таблиці.
- Ви не можете сортувати дані в провідних вузлах, оскільки їх класифікує значення первинного ключа.

2.1.4 Чому надмірна кількість індексів є недоліком?

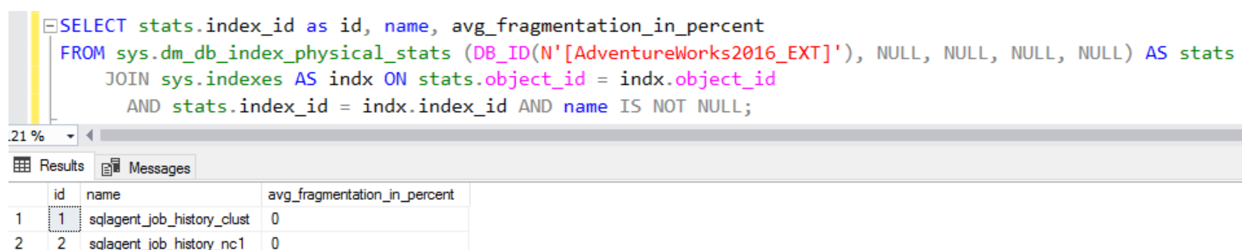
- Операції INSERT, UPDATE та DELETE займуть більше часу, оскільки індекси повинні оновлюватися з кожною зміною даних.
- Занадто багато записів (через непотрібні оновлення індексу) диск зношується раніше.
- Індекси займають місце в пам'яті та на диску. Отже, чим більше індексів, тим більше потрібно жорсткого диска та оперативної пам’яті. Це впливає на продуктивність запитів. Якщо якісь стовпці не використовуватимуться, такий індекс сповільнить запит.
- Коли в частині запиту є кілька стовпців (після WHERE), оптимізатор SQL намагається вгадати, який індекс буде найкращим для швидкого

результату. Якщо індексів занадто багато, оптимізатор може не дати найкращих результатів.

2.1.5 Дефрагментація індексів

Індекси у таблицях можуть стати фрагментованими, що може призвести до низької продуктивності будь-яких запитів, які використовують ці індекси. Регулярний графік обслуговування повинен включати реорганізацію індексів у ваших найбільш змінених таблицях.

Запустивши наступний запит для своєї бази даних, ми зможемо побачити індекси з найбільшим середнім відсотком фрагментації.



У результаті ми бачимо, що на даний момент в базі даних немає фрагментованих індексів.

У випадку, якщо у базі даних є індекси, які занадто фрагментовані, їх потрібно реорганізувати за допомогою базового сценарію ALTER. Це треба виконати у таблицях з набору результатів, які мають найбільшу фрагментацію, а потім виконайте даний запит.

```
SELECT
'ALTER INDEX ALL ON ' + table_name + ' REORGANIZE;
GO'
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_CATALOG = '[AdventureWorks2016_EXT]'
```

2.2 Аналіз та робота з запитами

Налаштування продуктивності СУБД – багаторівневий процес, і оптимізація запитів є лише одним із його аспектів. Під оптимізацією запиту розуміються дії, що призводять до того, що оптимізатор запитів вибирає найкращий процедурний план виконання. Найчастіше це зводиться до створення або розбудови відповідних індексів, оновлення статистики і діям, які не призводять до переписування самого запиту. Щоб виконувати таку

оптимізацію, необхідно вміти оперувати деякими інструментами та методами, які завжди можна використовувати, щоб мати чітке уявлення про роботу запитів.

Повільні запити є однією з найпоширеніших проблем у кожній організації, яка має справу з величезними обсягами даних. І найскладніша проблема, полягає в тому, як знайти повільне виконання запитів і з'ясувати, що є справжньою причиною проблеми з продуктивністю.

Є деякі інструменти та методи, які адміністратор баз даних завжди може використовувати, щоб мати чітке уявлення про роботу запитів.

Згадані вище інструменти, методи та поради є найпоширенішими рішеннями для усунення несправностей повільно виконуваних запитів.

2.2.1 План виконання (Execution plan)

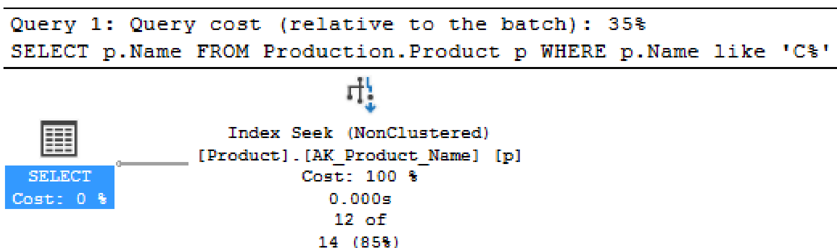
Читати план виконання запиту і розуміти, як виконуються фізичні оператори, що фігурують у плані є дуже важливим, адже тут можна побачити на що варто звернути увагу.

Index Seek та Index Scan

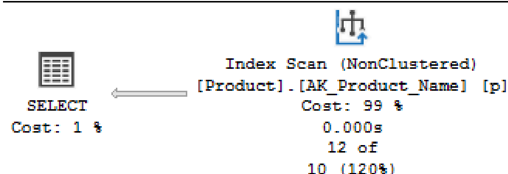
Одні з перших операторів плану виконання, на який варто звернути увагу - це пошук та сканування некластеризованого індексу.

Пошук (seek) - це добре для продуктивності, оскільки він являє собою прямий доступ SQL Server до необхідних рядків даних.

Сканування (scan) - це погано, оскільки він передбачає послідовне читання індексу для отримання великої кількості рядків, що призводить до більш повільної обробки.



Query 2: Query cost (relative to the batch): 65%
SELECT p.name FROM Production.Product p WHERE left (p.Name,1) ='c'



Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	12
Actual Number of Rows for All Executions	12
Actual Number of Batches	0
Estimated Operator Cost	0.0032978 (100%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0032978
Estimated CPU Cost	0.0001728
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	14.4
Estimated Number of Rows to be Read	14.4
Estimated Number of Rows Per Execution	14.4
Estimated Row Size	61 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	0

Predicate
[AdventureWorks2016_EXT].[Production].[Product].[Name] as [p].[Name] like N'C%'
Object
[AdventureWorks2016_EXT].[Production].[Product].[AK_Product_Name] [p]

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	504
Actual Number of Rows for All Executions	12
Actual Number of Batches	0
Estimated I/O Cost	0.0053472
Estimated Operator Cost	0.0060586 (99%)
Estimated Subtree Cost	0.0060586
Estimated CPU Cost	0.0007114
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	10
Estimated Number of Rows Per Execution	10
Estimated Number of Rows to be Read	504
Estimated Row Size	47 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Predicate
substring([AdventureWorks2016_EXT].[Production].[Product].[Name] as [p].[Name],(1),(1))=N'c'
Object
[AdventureWorks2016_EXT].[Production].[Product].[AK_Product_Name] [p]

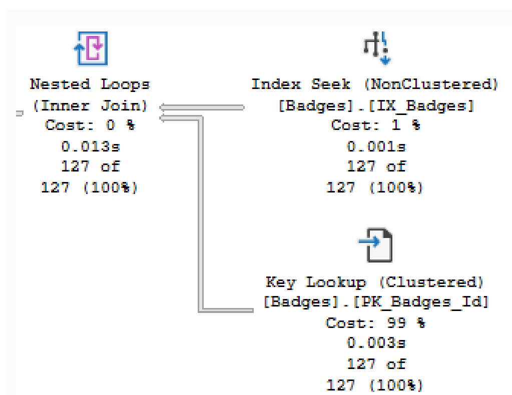
Можна зробити висновок, що в першому випадку ми використовуємо індекс, для знаходження інформації, а в другому випадку нам потрібно пройтись по всій таблиці.

RID Lookup и Key Lookup

RID Lookup (пошук ідентифікатора запису) та Key Lookup (пошук ключа) – це ще одна пара операторів, на які варто звертати увагу в плані запиту під час аналізу продуктивності.

RID Lookup – легко пофіксувати, це означає, що відсутній кластеризований індекс на таблиці і його потрібно додати. Після цього, відразу можна отримати деяке зростання продуктивності для більшості, якщо не для всіх, запитів.

Key Lookup (Пошук ключів) - має більше нюансів. SQL Server використовує цей оператор, коли він знає, що з більшою ефективністю може використовувати некластеризований індекс, а потім перейти до кластерзованого індексу для пошуку значення рядків, що залишилися, відсутні в некластеризованому індексі.



Пошук ключів – це не обов'язково погано. Звернення SQL Server до кластеризованого індексу для вилучення значень досить ефективний у порівнянні з необхідністю створювати і підтримувати абсолютно нові індекси.

Однак, якщо все, що потрібно SQL Server від операції Key Lookup, це єдиний стовпець даних, можливо, найпростіше додати цей стовпець до вашого існуючого некластеризованого індексу. Так, розмір індексу збільшиться на один стовпець, але якщо SQL Server зможе уникнути необхідності звертатися до двох індексів для отримання всіх необхідних даних, це, ймовірно, в цілому виявиться більш ефективним.

Sort

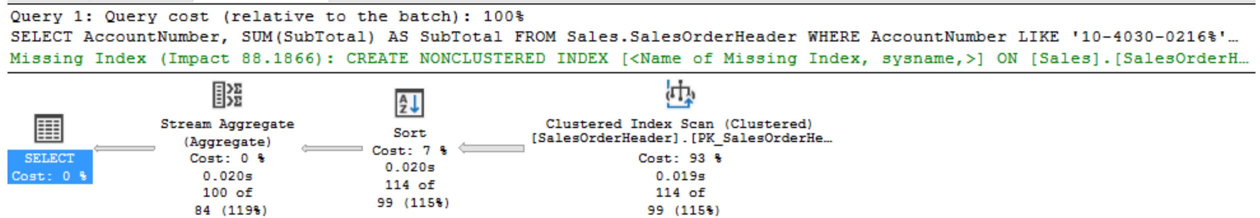
Дії оператора Sort (Сортування) прості - вони змінюють порядок рядків у потоці даних, що проходить.

Сортування є однією з найдорожчих операцій, які можуть бути в плані виконання, тому краще уникати їх, наскільки це можливо.

Одним із найпростіших способів уникнути оператора сортування - мати дані, що зберігаються у попередньо впорядкованому вигляді. Це може бути виконано створенням індексу з ключовими стовпцями, перерахованими в тому самому порядку, який використовує оператор сортування.

Якщо SQL Server повинен виконати сортування одних і тих же даних в тому самому порядку кілька разів у плані виконання, ще однією можливістю є

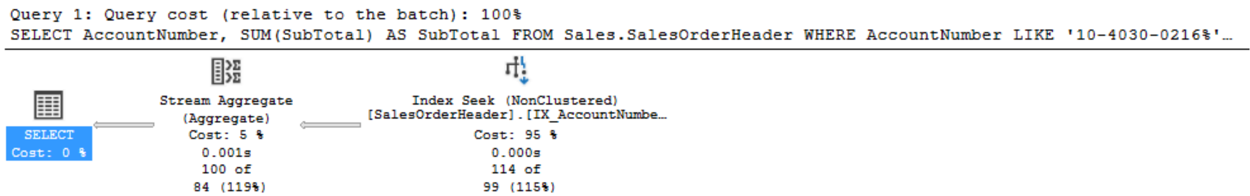
розбиття запиту на кілька етапів при використанні тимчасових індексованих таблиць для збереження даних між етапами. Заміна єдиного оператора сортування на індексну часову таблицю не дасть виграшу у продуктивності, але якщо є можливість повторно використовувати тимчасову таблицю щодо виконання вашого запиту, то отримаєте чисту економію.



Ось наприклад є запит, у якого є оператор сортування. І оптимізатор запитів

```
CREATE NONCLUSTERED INDEX [IX_AccountNumber]
ON [Sales].[SalesOrderHeader] ([AccountNumber])
INCLUDE ([SubTotal])
```

пропонує нам створити індекс, що я і зроблю. І ми бачимо, що оператор сортування у нас зникає і з'являється пошук за індексом, що пришвидшує роботу запиту.



<u>З'єднання</u>

- З'єднання злиттям (Merge join)

Найефективніший з операторів логічного з'єднання. Оптимізатор вибирає використання з'єднання злиттям, коли вхідні дані вже відсортовані або SQL Server може виконати сортування даних із відносно невеликою вартістю. Операція не застосовується, якщо вхідні дані не відсортовані.

- З'єднання вкладеними циклами (Nested Loops Join)

Трапляються часто. Виконують досить ефективне з'єднання невеликих наборів даних. З'єднання вкладеними циклами не вимагає сортування вхідних даних. Однак продуктивність можна покращити за допомогою сортування

вхідного джерела даних; SQL Server зможе вибрати ефективніший оператор, якщо обидва входи відсортовані. Операція не застосовується, якщо дані надто великі для зберігання пам'яті.

- З’єднання з хешуванням (Hash Match Join)

Операція використовується завжди, коли неможливо застосувати інші види з'єднання. Вона вибирається оптимізатором запитів з однієї із двох причин:

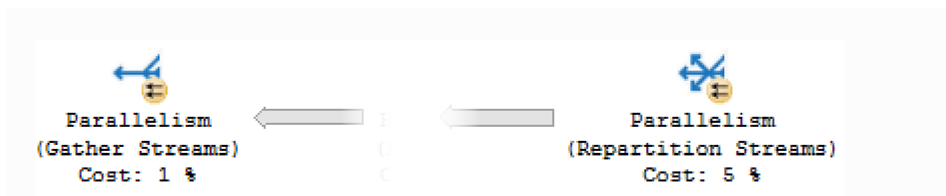
1) Комплекти даних, що з'єднуються, настільки великі, що вони можуть бути оброблені тільки за допомогою Hash Match Join.
2) Набори даних не впорядковані по стовпцях з'єднання, і SQL Server вважає, що обчислення хешів і цикл по них буде швидше, ніж сортування даних.

При першому сценарії важко оптимізувати виконання запиту, якщо не знайти способу з'єднувати менші обсяги даних.

При другому сценарії, якщо є деякий спосіб отримати дані в упорядкованому вигляді до з'єднання, типу зумовленого порядку сортування в індексі, то можливо, що SQL Server вибере замість цієї операції швидший алгоритм з'єднання.

Оператори Hash Match Join досить ефективні тоді, коли дані не скидають у tempdb.

Parallelism



Оператори паралелізму зазвичай вважаються хорошими речами: SQL Server дробить ваші дані на безліч частин для асинхронної обробки на багатьох процесорах, скорочуючи загальний час роботи, необхідне для виконання вашого запиту.

Однак паралелізм може стати поганим, якщо більшість або всі запити використовують його. При паралелізмі процесори, як і раніше, виконують той самий обсяг роботи, що і без нього, тим самим забираючи ресурси в інших запитів, які можуть бути запущені, плюс накладається додаткове навантаження на SQL Server по дробленню та подальшому об'єднанню всіх даних з безлічі ниток виконання.

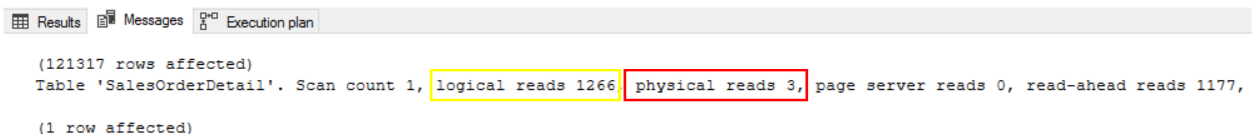
Якщо паралелізм є вузьким місцем продуктивності, можна розглянути питання про зміну порогового значення вартості для налаштування паралелізму, якщо воно встановлено надто низьким.

2.2.2. Статистика IO

Це дозволяє нам побачити, скільки логічних і фізичних читань виконується під час виконання запиту.

SET STATISTICS IO ON

Після цього ми будемо бачити додаткові дані, в панель повідомлень



Логічні зчитування говорять нам, скільки читань було зроблено з буферного кешу. Це число, на яке ми будемо посилатися щоразу, коли говоримо про те, за скільки читань відповідає запит або скільки IO він спричиняє.

Фізичні зчитування говорять нам, скільки даних було прочитано з пристрою зберігання даних, оскільки вони ще не були присутні в пам’яті. Це може бути корисною ознакою проблем з буферним кешом або ємністю пам’яті, якщо дані дуже часто зчитуються з пристроїв зберігання даних, а не з пам’яті.

Загалом, IO може бути основною причиною затримок і вузьких місць під час аналізу повільних запитів.

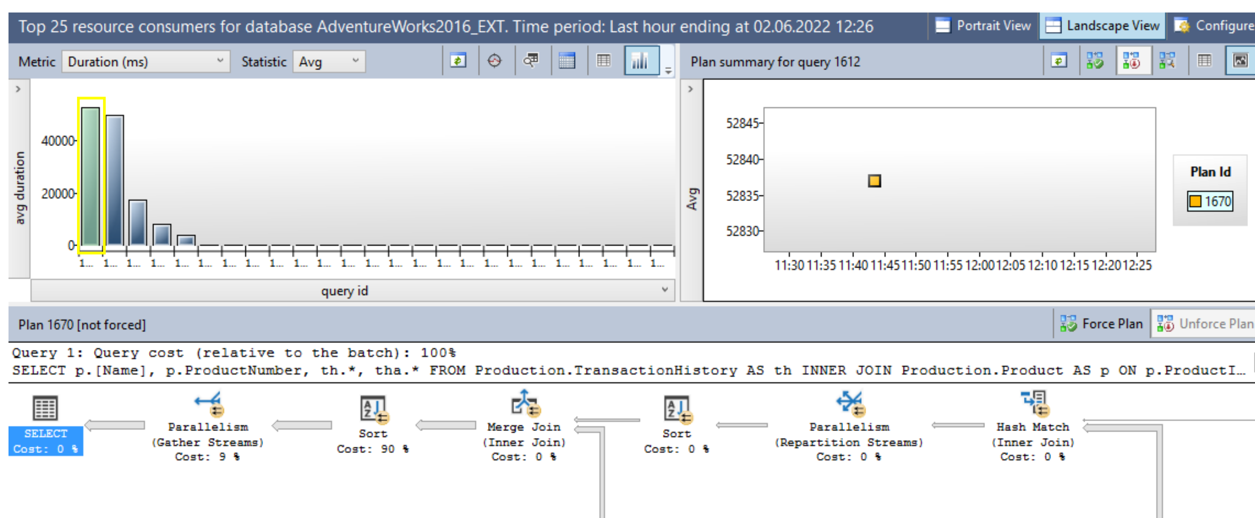
2.2.3. Сховище запитів (Query Store)

Сховище запитів SQL Server (Query Store) — це, по суті, «чорний ящик», який фіксує історію виконаних запитів, статистику виконання запиту під час виконання, плани виконання тощо щодо певної бази даних. Ця інформація допомагає виявити проблеми з продуктивністю, спричинені змінами плану запитів, та усунути неполадки, швидко знаходячи відмінності в продуктивності, навіть після перезавантаження або оновлення SQL Server. Усі дані, які збирає SQL Server Query Store, зберігаються на диску.

Коли функція зберігання запитів SQL Server може бути корисною:

- Для знаходження найдорожчих запитів для CPU, I/O, Memory тощо.
- Для отримання повної історії виконання запитів
- Для отримання інформації про регресії запиту (новий план виконання, згенерований механізмом запитів, гірший за попередній). Що дозволяє швидко знаходити регресію продуктивності та виправляйте її, форсувавши попередній план запиту, продуктивність якого набагато краща, ніж нещодавно згенерований план
- Для визначення, скільки разів запит було виконано в заданому діапазоні часу

Розберемо на прикладі. Вибравши запит, який займає найбільше часу, ми можемо побачити план виконання запиту в нижньому вікні.



Ми перевіримо, чому запити виконуються повільно, і яку частину запиту потрібно виправити (якщо потрібно).

По-перше, я ввімкну *SET STATISTICS TIME, IO ON*. Таким чином, ми отримаємо I/O для кожної таблиці та загальну вартість ЦП для запитів, що виконуються всередині запиту.

```
SQL Server parse and compile time:
   CPU time = 16 ms, elapsed time = 55 ms.

(1424074 rows affected)
Table 'Product'. Scan count 0, logical reads 4, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead re
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead r
Table 'Worktable'. Scan count 18, logical reads 41455, physical reads 0, page server reads 0, read-ahead reads 23799, page server re
Table 'TransactionHistory'. Scan count 5, logical reads 849, physical reads 0, page server reads 0, read-ahead reads 0, page server
Table 'TransactionHistoryArchive'. Scan count 5, logical reads 662, physical reads 0, page server reads 0, read-ahead reads 0, page
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead

 SQL Server Execution Times:
   CPU time = 13842 ms,  elapsed time = 78263 ms.
```

На наведеному вище знімку екрана ми бачимо, що найбільше I/O займає таблиця WorkTable, і цей запит займає 13842 мс часу ЦП (78263 мс час, що минув)

Далі ми активуємо фактичний план виконання для запиту, і спробуємо відповісти на питання, чому ця таблиця приймала цей I/O і який компонент плану виконання приймає більшість часу.

На що варто звернути увагу:

- Якщо в таблиці використовується Key Lookup, спробуйте видалити цей пошук, додавши ці стовпці в індекс, який використовується таблицею.
- Якщо кількість рядків, повернутих запитом, значно відрізняється від кількості рядків, повернутих операторами таблиці, спробуйте переписати запит, який фільтрує дані, використовуючи більше стовпців, щоб зменшити кількість рядків.
- Якщо розрахункові рядки і фактичні рядки мають величезну різницю, спробуйте оновити статистику таблиць, що знаходяться під лежачими умовами.

- Якщо відсутні індекси, зазначені в запиті, спробуйте оцінити індекс, і якщо цей індекс допомагає зробити запит, додайте його до відповідної таблиці.

2.2.4. Представлення динамічного управління (DMV)

Представлення динамічного управління (Dynamic Management Views) може надати нам велику кількість інформації з широкого кола тем, від стану сервера та бази даних до статистики запитів, планів виконання, останні запити та багато іншого. Ці подання можуть бути досить корисними для моніторингу загальної працездатності, визначення першопричини вузьких місць у продуктивності і налаштування продуктивності екземпляра або бази даних SQL Server.

```sql
SELECT TOP 10 SUBSTRING(qt.TEXT, (qs.statement_start_offset/2)+1,
((CASE qs.statement_end_offset
WHEN -1 THEN DATALENGTH(qt.TEXT)
ELSE qs.statement_end_offset
END - qs.statement_start_offset)/2)+1),
qs.execution_count,
qs.total_logical_reads, qs.last_logical_reads,
qs.total_logical_writes, qs.last_logical_writes,
qs.total_worker_time,
qs.last_worker_time,
qs.total_elapsed_time/1000000 total_elapsed_time_in_S,
qs.last_elapsed_time/1000000 last_elapsed_time_in_S,
qs.last_execution_time,
qp.query_plan
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY qs.total_logical_reads DESC -- logical reads
```

Наприклад, ось цей запит можна використовувати для пошуку запитів, які використовують найбільше читання, запису, робочий час (ЦП) тощо.

Подивившись на результати виконання запиту, можна зробити певні висновки, а потім, скопіювати запит з найгіршими показниками і побачити, чи є спосіб його покращити, додати індекс тощо. Таким чином, користуючись DMV, нам завжди доступна основна зведена статистика.

Results | Messages

	(No column name)	execution_count	total_logical_reads	last_logical_reads	total_logical_writes	last_logical_writes	total_worker_time	last_worker_time
1	SELECT AccountNumber, SUM(SubTotal) AS SubTotal F...	1	698	698	0	0	43952	43952
2	IF (SELECT COUNT_BIG(*) FROM (SELECT * FRO...	2	368	184	0	0	781898	441130
3	SELECT @c = COUNT_BIG(*) FROM (SELECT * FRO...	1	184	184	0	0	347792	347792
4	SELECT p.name FROM Production.Product p WHERE le...	1	6	6	0	0	149	149
5	SELECT p.Name FROM Production.Product p WHERE p...	1	2	2	0	0	245	245
6	select host_platform from sys.dm_os_host_info	4	0	0	0	0	284	58

2.2.5. Правильне та розумне написання SQL запитів

Є багато різних підказок та порад як краще писати запити, щоб вони були оптимальними з самого початку. Ось деякі найпоширеніші:

- Краще писати SELECT поля, які нас цікавлять замість SELECT *

Таким чином, ви можете звузити дані, отримані з таблиці під час запиту, збільшуючи швидкість вашого запиту. Команда SELECT * витягне всі дані з вашої таблиці, тоді як вказівка полів може скоротити час виконання запиту, забезпечуючи отримання лише необхідних даних.

- Використовуйте EXISTS() замість COUNT()

Хоча ви можете використовувати як EXIST(), так і COUNT(), щоб дізнатися, чи є в таблиці певний запис, використання EXIST() є ефективнішим.

```
IF (
    SELECT COUNT_BIG(*)
    FROM (
        SELECT * FROM [Production].[TransactionHistoryArchive] s
        CROSS JOIN
        (SELECT number FROM MASTER..spt_values) s2
        CROSS JOIN
        (SELECT number n2 FROM MASTER..spt_values) s3
    ) t
) > 100000
SELECT 1
GO
```

```
IF EXISTS(
    SELECT *
    FROM (
        SELECT * FROM [Production].[TransactionHistoryArchive] s
        CROSS JOIN
        (SELECT number FROM MASTER..spt_values) s2
        CROSS JOIN
        (SELECT number n2 FROM MASTER..spt_values) s3
    ) t
)
SELECT 1

GO
```

У той час як COUNT() шукатиме всю таблицю, щоб надати загальну кількість відповідних записів, EXIST() працюватиме лише до тих пір, поки не знайде перший запис запису в таблиці, заощаджуючи ваш час і обчислювальну потужність, а також дозволяючи оптимізувати запити SQL.

```
Table 'TransactionHistoryArchive'. Scan count 1, logical reads 158, physical reads 0, page server reads 0,
Table 'spt_values'. Scan count 2, logical reads 26, physical reads 0, page server reads 0, read-ahead reads

 SQL Server Execution Times:
  CPU time = 235 ms,  elapsed time = 239 ms.

Table 'TransactionHistoryArchive'. Scan count 1, logical reads 2, physical reads 0, page server reads 0,
Table 'spt_values'. Scan count 2, logical reads 4, physical reads 0, page server reads 0, read-ahead read

 SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.
```

- Уникайте одночасного використання DISTINCT і GROUP BY
- Уникайте одночасного використання UNION та DISTINCT
- Уникайте використання OR

DISTINCT використовується для вибору лише унікальних значень стовпця і, таким чином, усунення повторюваних значень. DISTINCT працює шляхом

ГРУПУВАННЯ всіх полів у запиті для створення чітких результатів. Однак для досягнення цієї мети потрібна велика обчислювальна потужність.

UNION сама створює окремі записи, тому нам не потрібно використовувати DISTINCT з UNION

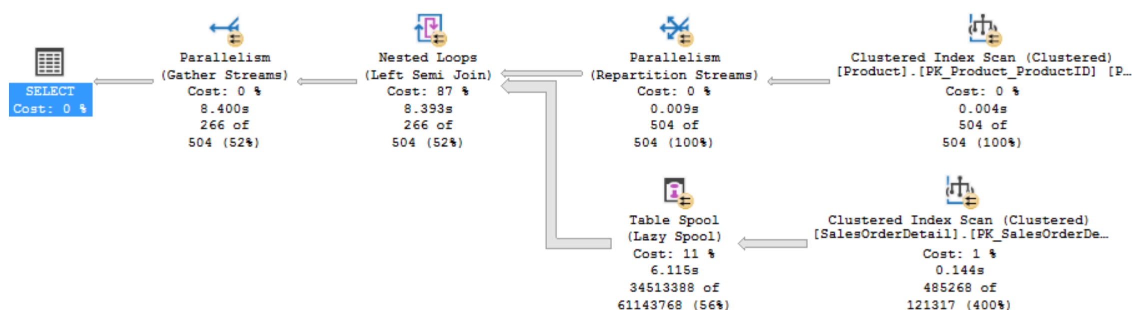
Якщо потрібно об’єднати дві або більше умов, рекомендується виключити використання оператора OR або розділити запит на частини, що розділяють вирази пошуку. SQL Server не може обробити OR в межах однієї операції. Замість цього він оцінює кожен компонент OR, що, у свою чергу, може призвести до поганої продуктивності.

Розглянемо приклад.

```
SELECT DISTINCT p.ProductID, p.Name
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
OR p.rowguid = sod.rowguid
```

Запит досить простий: 2 таблиці та об’єднання, яке перевіряє як ProductID, так і rowguid. Навіть якби жоден із цих стовпців не був проіндексований, ми очікуємо сканування таблиці продукту та сканування таблиці SalesOrderDetail. Дорого, але хоч щось ми можемо зрозуміти.

Ось результативність цього запиту:




```
(266 rows affected)
Table 'Product'. Scan count 5, logical reads 40, physical reads 1, page server reads 0, read-ahead reads 16, page
Table 'SalesOrderDetail'. Scan count 4, logical reads 5064, physical reads 2, page server reads 0, read-ahead read
Table 'Worktable'. Scan count 4, logical reads 1209220, physical reads 0, page server reads 0, read-ahead reads 0,
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page
```

Ми сканували обидві таблиці, але обробка OR потребувала величезної обчислювальної потужності. За цей час було здійснено 1,2 мільйона читань.

При тому, що Product містить лише 504 рядки, а SalesOrderDetail містить 121317 рядків, ми читаємо набагато більше даних, ніж повний вміст кожної з цих таблиць. Крім того, виконання запиту зайняло близько 9 секунд.

Можемо зробити висновок, що SQL Server не може легко обробити умову OR в кількох стовпцях. Найкращий спосіб розібратися з OR – це усунути його (якщо можливо) або розбити на менші запити. Розбиття короткого та простого запиту на довший і розтягнутий запит може здатися не елегантним, але коли вирішуєте проблеми OR, це часто є найкращим вибором.

В даному випадку, якщо взяти кожен компонент OR і перетворили його на власний оператор SELECT, тобто розділити попередній запит на два запити SELECT і об’єднати їх за допомогою оператора UNION, таким чином запит буде оптимізовано.

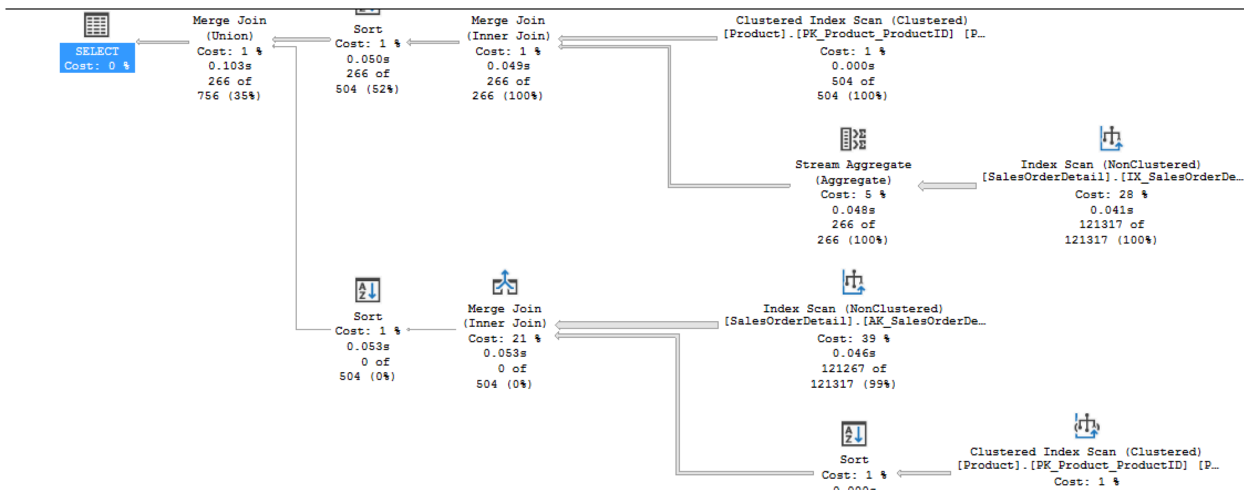
```
SELECT p.ProductID, p.Name
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
UNION
SELECT p.ProductID, p.Name
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
ON p.rowguid = sod.rowguid
```

UNION в свою чергу об'єднує набір результатів і видаляє дублікати, тому DISTINCT не потрібен.

Ось результативність:

```
(266 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0,
Table 'Product'. Scan count 2, logical reads 30, physical reads 0, page server reads 0, read-ahead reads 0,
Table 'SalesOrderDetail'. Scan count 2, logical reads 754, physical reads 4, page server reads 0, read-ahead
```

Кількість читань зменшено з 1,2 мільйона до 754, а запит виконується менше ніж за секунду, а не за 9 секунд.



Незважаючи на те, що в плані виконання все ще є велика кількість сканованих індексів, але, незважаючи на необхідність сканування таблиць чотири рази, щоб задовольнити наш запит, продуктивність набагато краща, ніж раніше.

Висновки

В процесі написання курсової роботи я детальніше ознайомився з MS SQL Server та його можливостями, також був виконаний аналіз на різних рівнях абстракції та розглянуті різні варіанти оптимізації роботи.

Я ознайомилася з різними засобами діагностики такими, як монітор активності та звіти продуктивності, які вбудовані в SQL Server Management Studio та дозволяють провести оцінку роботи сервера у будь-який момент. Також я дізналася, що таке статистика очікування, та як вона може допомогти при пошуку проблем з продуктивністю.

Я зрозуміла наскільки необхідна правильна робота з індексами, розібралася чому не варто зловживати їх використанням та як їх нестача може впливати на швидкість роботи бази даних.

Я навчилася правильно читати план виконання запиту, завдяки якому можна виявляти слабкі місця у роботі запитів.

Сховище запитів та представлення динамічного управління – дозолило мені проаналізувати останні виконані запити, та дозволили знайти найпроблемніші з них. Я дізналася, що може спричиняти проблеми зі швидкістю виконання запитів та як їх позбутися. В додаток, було запропоновано, декілька рішень, як писати запити, щоб у подальшій роботі з ними не виникало проблем з продуктивністю.

Джерела

1. https://www.sentryone.com/white-papers/troubleshooting-sql-server-wait-stats
2. Mercioiu, Nicolae, and Victor Vladucu. "Improving SQL Server Performance." Informatica Economica 14.2 (2010).
3. https://www.sqlshack.com/troubleshooting-using-wait-stats-in-sql-server/
4. https://www.sqlshack.com/how-to-identify-slow-running-queries-in-sql-server/
5. http://www.sql-tutorial.ru/ru/book_sql_server_query_plan_explanation.html
6. Kumari, Navita. "SQL server query optimization techniques-tips for writing efficient and faster queries." International Journal of Scientific and Research Publications 2.6 (2012): 1-4.
7. https://www.mssqltips.com/sqlservertip/6274/types-of-sql-server-indexes/
8. https://www.sqlshack.com/query-optimization-techniques-in-sql-server-tips-and-tricks/
9. https://www.sqlshack.com/query-optimization-techniques-in-sql-server-the-basics/
10. https://cutt.ly/vJvBUfR
11. https://cutt.ly/3JvBHMd
12. Corlăţan, Costel Gabriel, et al. "Query optimization techniques in Microsoft SQL server." Database Systems Journal 5.2 (2014): 33-48.
13. https://www.sqlshack.com/sql-server-management-studio-performance-reports/
14. https://cutt.ly/nJbeb5e
15. https://stackify.com/performance-tuning-in-sql-server-find-slow-queries/
16. https://cloud.google.com/compute/docs/instances/sql-server/best-practices