**Лабораторна робота 7-8 (2 заняття).**

**Оптимізація запитів та використання індексів.**

**Мета роботи:** навчитись створювати індекси різних типів та оцінювати доцільність їх використання.

**Задачі:**

* Познайомитись із планом виконання запиту та інструментами його отримання (очікуваний та дійсний плани виконання запитів);
* Створення простих кластерного та некластерного індексу та порівняння планів виконання запиту
* Створення складеного некластерного індексу, перевірка його роботи при застосуванні різних фільтрів по одному та декількох полях
* Використання вбудованих функцій або шаблонного порівняння при вибірці даних

1. Для початку наведемо трохи теоретичних відомостей. Індекси у БД необхідні для підвищення швидкості вибірки даних із таблиць за рахунок їх упорядкування. Як ми пам’ятаємо для упорядкованих даних можна застосовувати більш оптимальні методи пошуку даних, на зразок бінарного ділення, а для неупорядкованих даних можна користуватись лише алгоритмом повного перебору, що є максимально повільним.

Упорядкувати дані двома шляхами. Перший – змінити реальний порядок збереження даних на дисковому носії. Даному підходу відповідає кластерний тип індексу. Даний тип індексації дає максимальну швидкість при пошуку необхідних даних, якщо від відбувається за полем, за яким дані були впорядковані. Головний недолік – не можна застосувати декілька різних типів сортувань одночасно, тому може бути використаний лише у частині запитів на вибірку даних.

Для додавання альтернативних сортувань використовується другий шлях – створюється додаткова структура, у якій зберігається поле, за яким виконується сортування та посилання на рядок таблиці, і сортується все ця структура, а не самі дані. При вибірці виконується пошук у відсортовані структурі, а потім, за посиланням виконується читання основних даних. Недолік даного підходу полягає у необхідності додаткового читання даних за посиланням (що також може сповільнити виконання запиту)

Спільний недолік обох підходів – сповільнення виконання операцій модифікації даних, так як вони тепер вимагатимуть зміни як таблиці, так і цілком можливо перебудови додатково створених структур індексів для збереження відсортованості даних. Тому перша загальна рекомендація – створювати індекси для таблиць та полів, де операції читання значно перевищують операції зміни даних. Звісно, є ще багато інших аспектів, на деякі з яких спробуємо звернути увагу протягом даної роботи.

1. Далі стисло познайомимось із поняттям плану виконання запиту. Коли ми пишемо запит на SQL, то використовується декларативний підхід, тобто у тексті запиту ми говоримо, які дані хочемо отримати, але ніяким чином не вказуємо, яким способом їх потрібно отримувати. Де зберігаються дані, визначається структурою самою БД, а план запиту відповідає на питання «як можна отримати дані, використовуючи наявну структуру БД, щоб вони відповідали тексту запиту». Тобто можна сказати, що виконання будь якого запиту складається з 2-х етапів - побудови плану виконання, та безпосередньо виконання запиту згідно цього плану.

В ході першого етапу як правило створюється декілька різних планів виконання запиту, після чого оптимізатор запиту вибере найоптимальніший із точки розу мінімальної вартості затребуваних ресурсів та часових витрат. Виходячи із вищенаведеного, зрозуміло, що цей етап є ресурсоємким сам по собі і нерідко може займати більше часу, аніж саме виконання. Тому у більшості випадків, після першого створення план виконання поміщується у кеш, та буде використаний у подальшому при виконанні «типових» запитів.

Розрізняють також очікуваний та фактичний плани виконання. Перший створюється оптимізатором на етапі компіляції та не виконує жодних запитів чи пакетів. Відповідно такий план не буде містити даних щодо фактичного часу використання ресурсів. Даний підхід дуже добре підійде для оптимізації ресурсоємких запитів без фактичного їх виконання. Фактичний план повертає створений оптимізатором запиту план після завершення виконання всіх запитів та пакетів, а отже містить всі дані щодо фактично використаних ресурсів.

1. Для початку створимо нову таблицю, в якій будуть представлені всі можливі комбінації пар студентів. Дана операція нам необхідна, щоб отримати таблицю з відносно великою кількістю рядків, на якій можна буде проілюструвати роботу плану виконання запиту та переваги використання індексів. Почнемо із створення структури даних (рис.1)

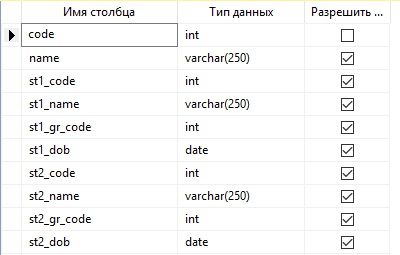


Рис.1. Структура таблиці students\_couples

Для поля code використаємо індексатор. Далі виконаємо запит, що вибере всі можливі пари студентів та запише у новостворену таблицю (рис.2)

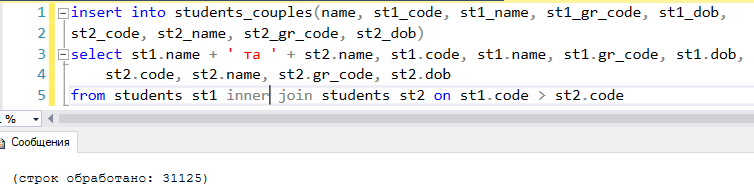


Рис.2. Наповнення таблиці students\_couples даними.

1. Переходимо нарешті до роботи із таблицею. Відкриємо нове вікно запиту на натиснемо «фактичний план виконання» (рис.3)

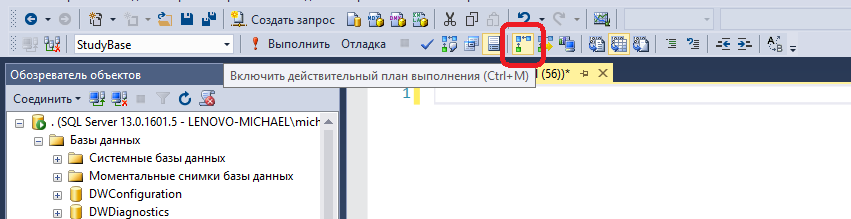


Рис.3. Статичний план виконання

Далі виконаємо вибірку всіх даних із таблиці «пари студентів». Зверніть увагу, що після виконання запиту з’являється нова вкладка – «план виконання». Перейдемо на неї (рис.4)

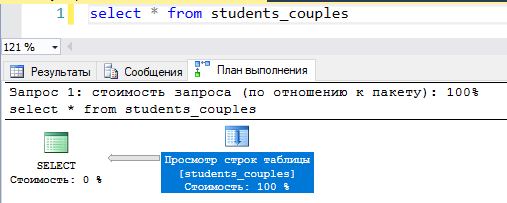


Рис.4. План виконання запиту вибірки пар студентів.

Тут можемо побачити, що 100% затрат приходиться на операцію «перегляд рядків таблиці», що відповідає повному перебору всіх рядків таблички і, як правило, є найдорожчою з усіх можливих варіантів операцій. Також підвівши мишу до даного об’єкту, можемо побачити детальний звіт про дану операцію (рис.5).

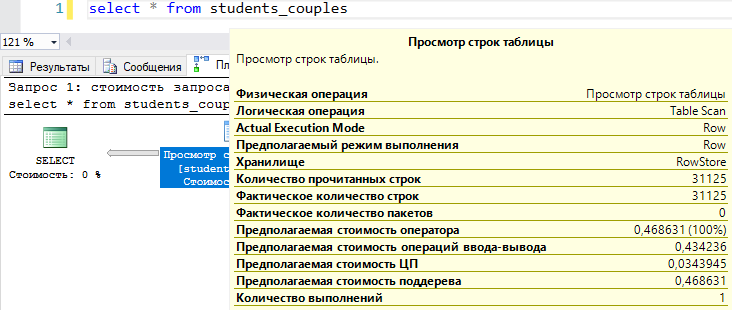


Рис.5. Детальний огляд операції «перегляд рядків таблиці»

1. Наведений вище запит, що вибирає всі записи таблиці не може бути вирішений в спосіб, відмінний від повного перебору, відповідно виконати оптимізацію даного запиту через використання індексів не є можливим. Тому наберемо інший запит, що вибирає один рядок – пару студентів із кодом 154 (рис.6).

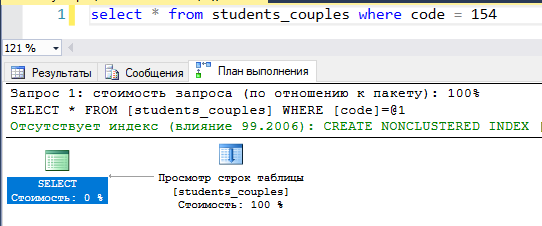


Рис.6. План виконання запиту вибірки однієї пари студентів.

За планом виконання бачимо, що незважаючи на те, що нам потрібен лише один рядок таблиці, двигун БД змушений застосувати операцію «перегляд рядків таблиці», тобто послідовно перебрати всі рядки таблиці, перевіривши для кожного умову «code = 154». Також на рис.6 бачимо, що оптимізатор запиту повідомляє нам про відсутність індексу по полю code, наявність якого могла б підвищити продуктивність запиту на 99 %.

! УВАГА. Не слід завжди виконувати поради оптимізатора запитів щодо створення індексів, так як це може негативно вплинути на виконання інших запитів.

1. Створимо індекс за полем code, але не явним чином, а через створення первинного ключа. Справа у тому, що при створенні первинного ключа SQL Server автоматично створює для нього кластерний індекс, якщо такого не було створено раніше. Дана особливість пов’язана із тим, що у переважній більшості випадків фільтрація даних із таблиці відбувається за первинним ключем (особливо у випадках використання оператора JOIN). Створюємо первинний ключ для таблиці «пари студентів» (рис.7)

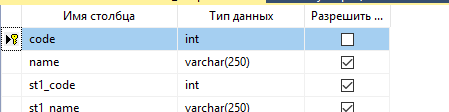


Рис.7.Створення первинного ключа для «пари студентів»

Після збереження змін, повторно виконаємо запит, що наводився на рис.6. Новий план виконання бачимо на рис.8.

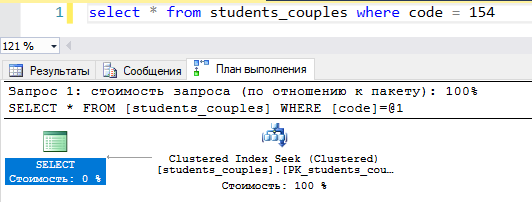
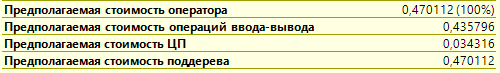


Рис.8. План виконання вибірки однієї пари студентів із використанням індексу

Даний план виконання використовує вже пошук у кластерному індексі, що є найбільш оптимальним можливих сценаріїв плану виконання запиту. Порівнявши вартість запиту до та після створення індексу наведено на рис.9.



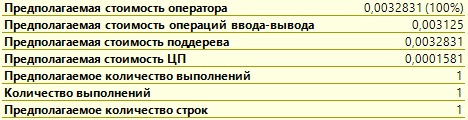


Рис.9. Порівняння продуктивності до та після створенні індексу

1. Далі виконаємо запит, що також, як і попередній, повертає один рядок, але виконує фільтрування даних за полем «name» (рис.10)

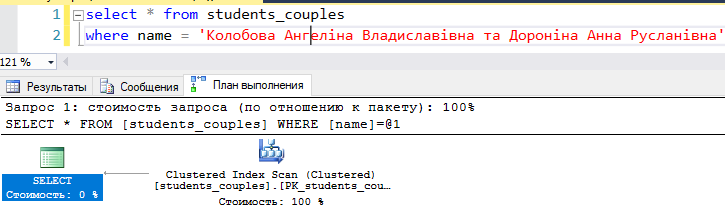
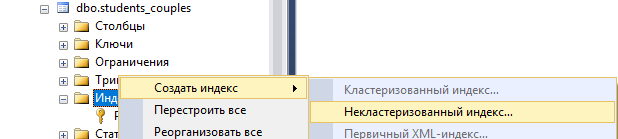


Рис.10. Вибірка одного рядку із таблиці «пари студентів» за полем «name»

Як бачимо, запит начебто використовує кластерний індекс за полем «code», але операція вже не «index seek» а «index scan». Це означає, що виконується повний перебір індексу, а не пошук по ньому, тому що поле code не приймає участі у фільтрі where. Створимо додатковий некластерний індекс для поля «name» (рис.11)



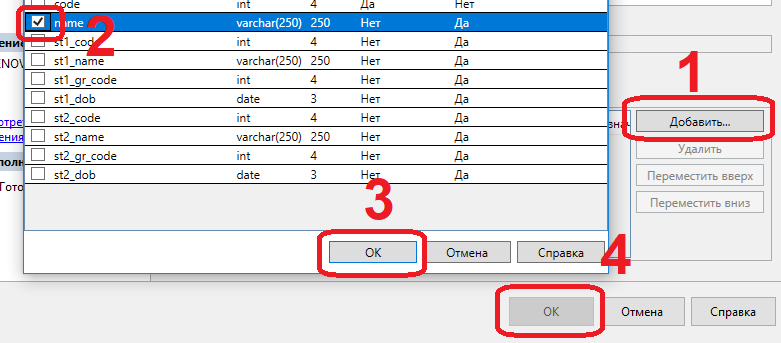


Рис.11. Створення некластерного індексу для поля name

Після створення даного індексу план виконання виглядатиме наступним чином (рис.12)

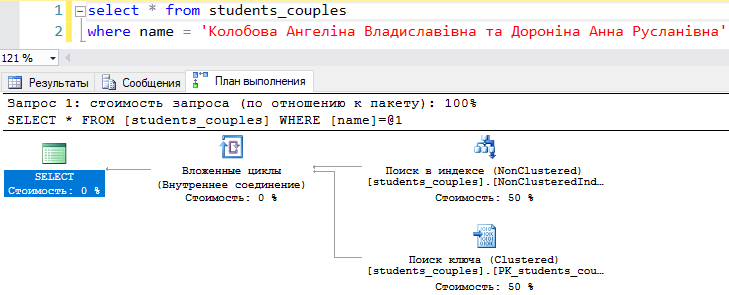


Рис.12. План виконання вибірки одного рядку із таблиці «пари студентів» за полем «name» після додавання кластерного індексу

Бачимо, що вартість запиту розділилась 50 на 50 між пошуком у новоствореному індексі та подальшою вибіркою основних даних із кластерного.

Виконаємо невеликі зміни у запиті – виберемо не всі поля таблички, а лише поле name (рис.13)

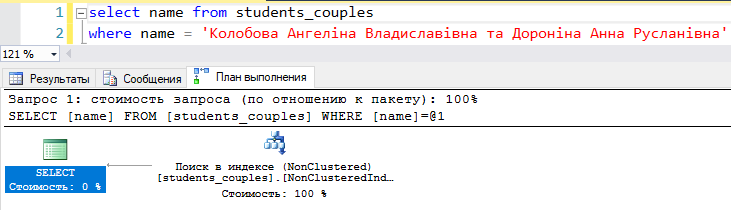


Рис.13. План виконання при виборі лише поля name

Як бачимо, тут немає етапу пошуку даних у кластарному індексі, тому що у даному випадку індекс за полем name є самодостатнім, оскільки у ньому представлені всі поля, що фігурують у запиті. Даний варіант є більш бажаним, оскільки виключає додаткову операцію пошуку даних, та відповідно, є більш швидким.

1. При створенні індексу є механізм представлення поля таблиці у індексі, навіть якщо не потрібно виконувати впорядкування за цим полем. Так, наприклад, виконаємо запит (рис.14).

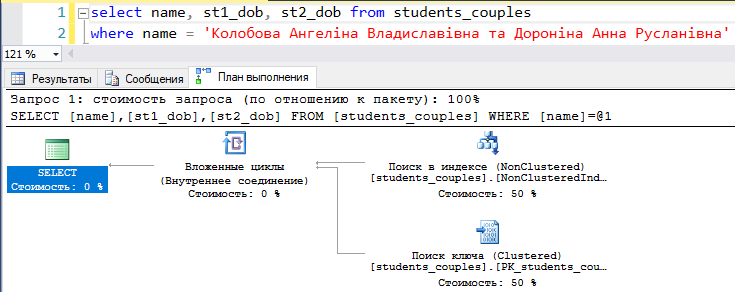


Рис.14. Вибірка полів name, st1\_dob, st2\_dob із таблиці «пари студентів»

Бачимо, що повернулася перація «пошук ключа», що потрібна для вибірки полів st1\_dob та st2\_dob із вартістю 50%, тобто запит став у 2 рази повільнішим. Уникнути цього можна, представивши поля st1\_dob та st2\_dob безпосередньо у індексі. Для цього зайдемо до вікна редагування індексу на на вкладці «включено стовбці» додамо поля st1\_dob та st2\_dob (рис.15)

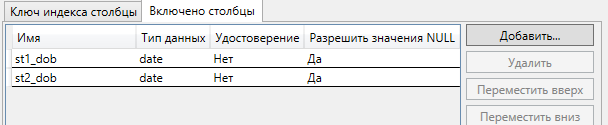


Рис.15. Додавання включених стовбців до індексу

Виконаємо запит (рис.14) повторно та переглянемо результат (рис.16)

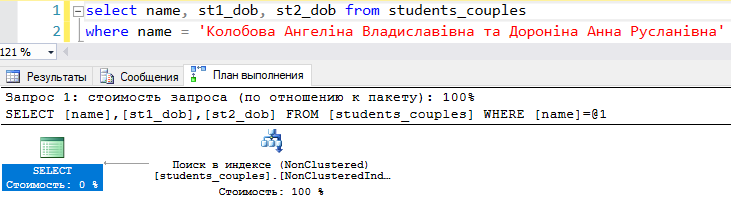


Рис.16. Вибірка полів name, st1\_dob, st2\_dob із таблиці «пари студентів» після представлення полів у індексі

Бачимо, що всі дані знову беруться у некластерному індексі, операція «пошук ключа» не виконується, та відповідно запит знову став у 2 ази швидшим.

! УВАГА. Слід обережно ставитись до представлення полів у індексі, оскільки це веде до підвищення ймовірності необхідності перебудови дерева індексу при зміні даних таблиці, що може значно сповільнити операції модифікації даних.

1. Слід розуміти, що створення індексу не завжди приводить до його використання у запиті, навіть за умови виконання фільтрації за полем індексу. Проілюструємо це на прикладі. Створимо індекс для поля st1\_dob та виконаємо вибірку із фільтрацією за цим полем (рис.17)

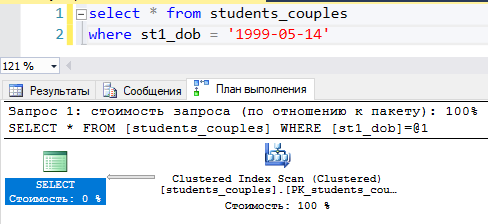


Рис.17. План виконання для вибірки даних із умовою по даті народження першого студента

Як видно, виконується повне сканування кластерного індексу (повний перебір даних), а створений індекс по полю st1\_dob, що мав би пришвидшити виконання запиту, взагалі не використовується. Це відбувається тому, що даний запит вибирає вже не 1 рядок, а 235, тобто при використанні індекса потребуватиме 235 операцій «пошук ключа» для вибірки даних за посиланням з кластерного індексу. За оцінкою оптимізатора повний перебір даних пройде швидше.

Переконаємось у цьому, примусивши оптимізатор запиту використати наш індекс (рис.18)

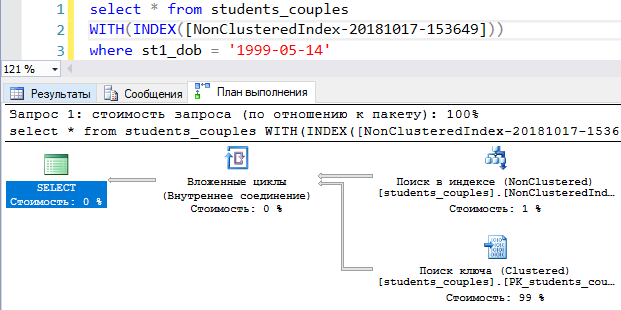


Рис.18. Примусове використання індексу

Як бачимо, скоріше за все оптимізатор запиту виявився правим, відмовившись від використання даного індексу, оскільки 99% вартості приходиться саме на операцію «пошук ключа».

Отже можна зробити висновок, що створення такого індексу буде мати негативний ефект для швидкодії роботи із БД, оскільки навіть якщо дозволити оптимізатору не використовувати його при вибірці даних, він все одно буде сповільнювати операції зміни даних, що торкатимуться поля «st1\_dob»

1. Далі розглянемо використання декількох полів при створенні індексу. Щоб не плутатись, видалимо всі попередньо створені некластерні індекси та створимо новий, що включатиме 4 поля (рис.19)

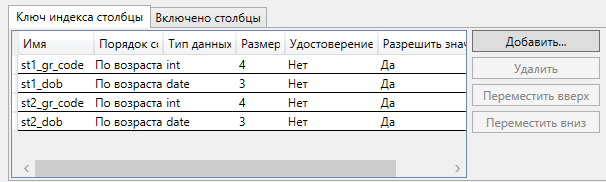


Рис.19. Створення складеного індексу

Виконаємо наступний запит, що виконує фільтрацію за всіма полями створеного індексу (рис.20)

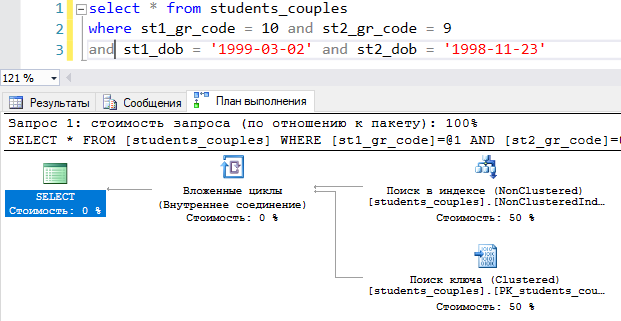


Рис.20. Як бачимо, створений індекс успішно використовується

Далі приберемо із запиту умови щодо кодів груп, але залишимо щодо дат народження і виконаємо його (рис.21)

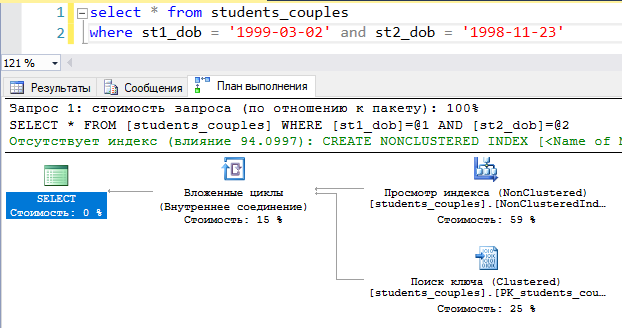


Рис.21. Використання фільтрації за частиною полів запиту

Індекс начебто і використовується, але операція змінила назву з «пошук в індексі» та «перегляд індексу». Також запит став виконуватись довше, у чому можна переконатись, подивившись вартість вказаних операцій. Це можна пояснити тим, що перше поле, за яким іде сортування в індексі – це st\_gr\_code, тому не всі значення із st1\_dob = ‘1999-03-02’ будуть розміщуватись один за одним. Тому використання індексу відбувається не у повній мірі.

Розберемо ще декілька прикладів, що виконують фільтрацію за полями, що входять до нашого індексу. У наступному прикладі фільтруємо за полем st2\_dob (рис.22).

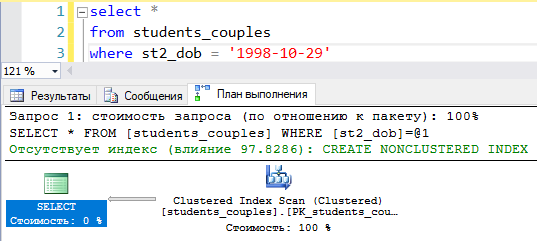


Рис.22. Приклад ігнорування індексу оптимізатором запиту

Дане поле є останнім по порядку у списку сортування, а отже має максимальну розрідженість при розташуванні в упорядкованому списку. Оптимізатор оцінює цей фактор та приймає рішення взагалі не використовувати індексу.

У наступному прикладі виконаємо фільтрацію за першими двома полями у порядку сортування (рис.23)

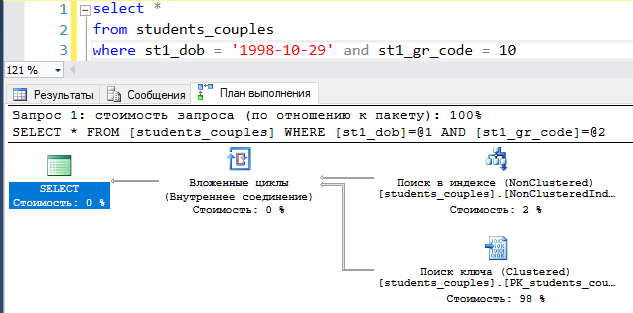


Рис.23. Використання індексу при часткові фільтрації

Незважаючи на те, що пошук відбувається лише по двом з чотирьох полів індексу, але враховуючи той факт, що це 2 перші поля по порядку при сортуванні, знову маємо операцію «пошук в індексі», та бачимо, що вина виконується майже миттєво (2% від вартості запиту).

1. Тепер розглянемо вплив використання вбудованих функцій та шаблонного порівняння (like) при вибірці даних. Видалимо попередньо створений. Видалимо попередньо створений складений некластерний індекс та створимо новий для поля «name». Далі виконаємо запит за вибірку із фільтром за «name» на рівність (рис.24). Цілком прогнозовано, що у плані виконання запиту бучимо операцію «пошук в індексі».

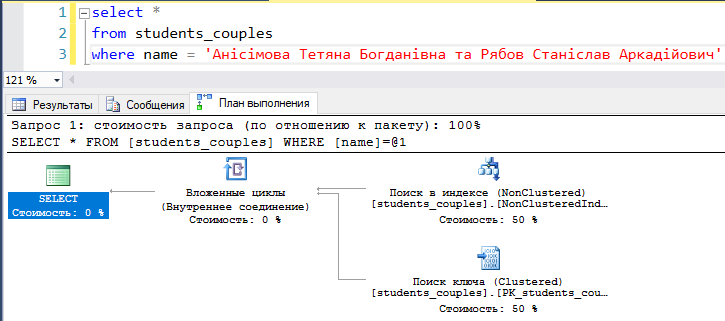


Рис.24. Фільтр на рівність за полем «name»

Далі трохи змінимо запит, та введемо у нього конструкцію like (рис.25). Даний запит вибирає той же самий один рядок, що і попередній, але відпрацьовує він помітно довше. Переглянувши план виконання стає зрозумілою причина, оскільки індекс не використовується.

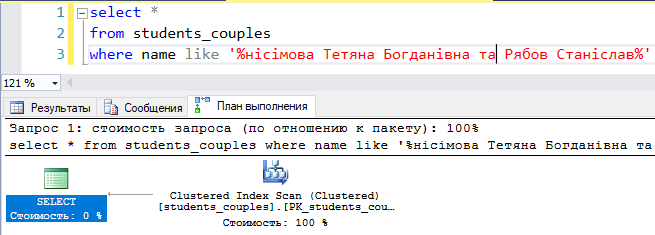


Рис.25. Фільтр за полем «name» із використанням like.

У даному випадку оптимізатор приймає рішення не використовувати індекс, оскільки рядки, що задовольняють умові шаблону, можуть розташовуватись у індексі не підряд. Для вирішення даної проблеми слід прибрати перший символ % із шаблону (якщо це вписується у логіку нашого запиту). У даному випадку результуючі рядки будуть розташовані у індексі один за одним, відповідно використання індексу дає приріст у швидкодії (рис.26)

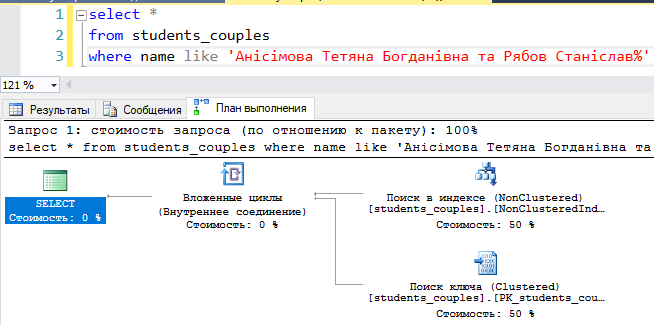


Рис.26. Зміна шаблону like для використання індексу

1. При використанні функцій у фільтрах виникає та ж проблема. Оптимізатор запиту не завжди в змозі оцінити результат роботи функції до її виконання, тому у більшості випадків індекс не буде використано. Додамо індекс по полю «st2\_dob» та виконаємо у запиті фільтрацію по цьому полю за значеннями 1 та 2 жовтня із використанням вбудованих функцій (рис.27)

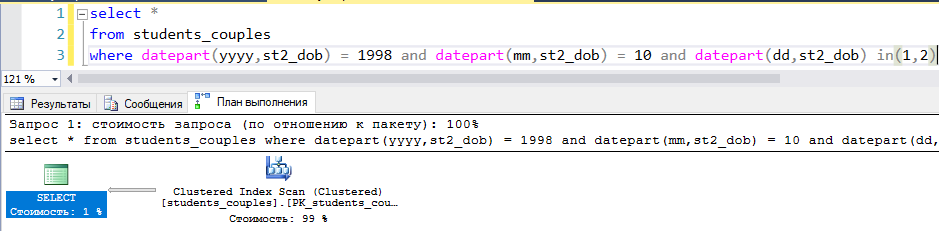


Рис.27. Використання вбудованих функцій у запитах

Як бачимо, індекс поки не використовується. Перепишімо запит, щоб він таки використав індекс (рис.28)

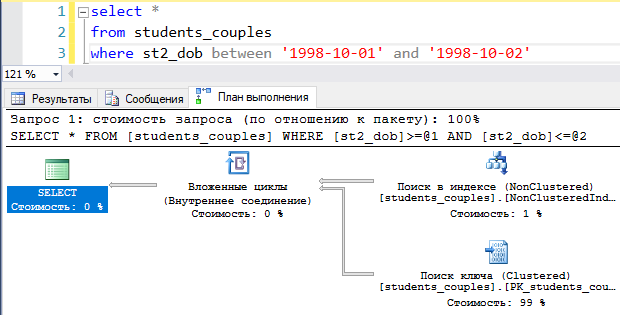


Рис.28. Зміна запиту для використання індексу

З вищенаведеного можна зробити висновок, що використання функцій у запитах слід намагатися звести до мінімуму, або використовувати до невеликих наборів даних.

1. Кінець роботи

**Завдання для індивідуального виконання.**

1. У власній БД виконати запит до таблиці без індексів та ключів та навести план виконання запиту (операція «перегляд рядків таблиці»)
2. У власній БД виконати запит до таблиці із первинним ключем без фільтрів та навести план виконання запиту (операція «сканування кластерного індексу»)
3. У власній БД виконати запит до таблиці із первинним ключем із фільтром на рівність по первинному ключу та навести план виконання запиту (операція «пошук у кластерному індексі»)
4. У власній БД створити некластерний індекс із одним полем та виконати запит із фільтром по цьому полю так, щоб аналізатор запиту використав це індекс. Навести план виконання запиту (операція «пошук у некластерному індексі»)
5. Створити та використати у запиті (та навести план виконання запиту) індекс із включеними стовбцями
6. Створити складений індекс (2 та більше поля). Виконати фільтрацію полів (або поля), що використовуються у індексі у 2-х запитах так, щоб в одному індекс був використаний, а у іншому ні. Пояснити чому. Навести плани виконання запитів.
7. Створити неоптимальний індекс, виконати запит що фільтрує за індексованим полем. Індекс має залишатись невикористаним. Використати індекс примусово. Порівняти 2 плани виконання та обґрунтувати, чому індекс не був використаний оптимізатором запиту.

Для виконання даної роботи можуть бути використані додаткові таблиці та дані (не маючі відношення до БД вашого варіанту) за умови неспівпадіння із іншими студентами групи.