Komentarze

Bazy danych: Optymalizacja

Spis treści

[Refleksje 3](#_Toc42704194)

[Do 157760 - Here we go again 3](#_Toc42704195)

[To nie jest prawda: 1 3](#_Toc42704196)

[Dammit 3](#_Toc42704197)

[Do 156440 3](#_Toc42704198)

[QUERY 2 - Upmanship! 3](#_Toc42704199)

[Insert i O(logN) 4](#_Toc42704200)

[Do 160761 4](#_Toc42704201)

[Ciekawostka 4](#_Toc42704202)

[Poprawa czasu o x ms 5](#_Toc42704203)

[Do 160764 5](#_Toc42704204)

[Query 1 + do 160766 7](#_Toc42704205)

[DO 160769 8](#_Toc42704206)

[DO 160777 8](#_Toc42704207)

[To nie jest prawda: 2 8](#_Toc42704208)

[Do 160781 8](#_Toc42704209)

[Do 160798 + często powtarzający się błąd 9](#_Toc42704210)

[Ciekawostka 9](#_Toc42704211)

[+ jeszcze coś 9](#_Toc42704212)

# Refleksje

Tu macie zbiorcze refleksje

## Do 157760 - Here we go again

Znowu przypomnijcie mi żebym tego nie słał do dziekanatu:

*W bazach danych istnieje warunek nazwany SARGABLE, który z pomocą odpowiedniego indeksu jest w stanie przyśpieszyć wykonanie zapytania. Do takich warunków należą* ***=, <, >, BETWEEN*** *itd. Dla stringów takim zapytaniem jest ‘dupa%’, w przeciwieństwie do ‘%dupa’ oraz ‘%dupa%’.*

### To nie jest prawda: 1

*Przyśpieszenie operacji wpisywania do bazy jest niemożliwe poprzez wprowadzenie indeksów.*

Akurat może dobrałem niefortunnie komendę wstawiająca dane :

1. Tam są SELECTY-y – te z MAX() ale jak słusznie zauważyłeś – niewiele można z nimi zrobić
2. W środowisku współbieżnym indeks kastrujący zachowujący losowy porządek może zmniejszyć liczbę blokad – tym samym przyśpieszyć ogólny TpS – ale to jest miecz obosieczny: jeśli BD nie mieści się w RAM, to odczyty losowo uporządkowanych danych zaczynają spowalniać inserty. Optymalnie jest to zrobić tak: <https://www.slideshare.net/chris1adkin/super-scaling-singleton-inserts-53947279>

Ale nikt tego nie robi :P

### Dammit

To kawał sprawozdania – otworzyłem jako pierwsze i postawiło baaaardzo wysoko poprzeczkę.

## Do 156440

Brakuje mi komentarza na co patrzę

## QUERY 2 - Upmanship!

Na razie nikt nie poszedł w tym kierunku, ale zadanie 2 można zrobić w paskudny sposób. Można postawić **widok indeksowany** (widok materializowany / MQT).

Koncept jest taki:

CREATE VIEW MyView AS

SELECT

m.Name AS MovieName,

u.Name AS UserName,

AVG(r.rating) AS Rating

FROM

ratings r

INNER JOIN movies m ON r.movieId = m.movieId

INNER JOIN user\_info u ON u.userId = r.userId

GROUP BY

m.Name,

u.Name;

Co tworzy na razie tylko widok. Potem stawiamy indeks na widoku:

CREATE INDEX MyViewIndex ON MyView(UserName, MovieName, Rating);

I odpowiednik zapytania:

SELECT

Name,

Rating

FROM

MyView WITH(NOEXPAND);

Pisałem to z głowy – więc może to wymagać szlifów.

## Insert i O(logN)

To ma miejsce tylko dla indeksów klastrowanych.

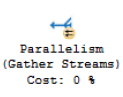
## Do 160761

*Jak można zauważyć indeksy nie pomogły, a wręcz pogorszyły sprawę, gdyż koszt zapytania dla bazy danych z indeksami w porównaniu do bazy danych bez indeksów wynosi 59% do 41%. Próbowałem pozbyć się tego problemu używając innych indeksów, jednak wynik zawsze był taki sam – niekorzystny*

( ͡° ͜ʖ ͡°) – bo źle dobrałeś indeksy. 51%/49% to nie poprawa wydajności.

## Ciekawostka

Jak w planie zapytania pojawiają się operatory <- <- np.



To bardzo źle. Baza danych nie wie jak robić żeby było szybko, więc uruchamia kilka wątków – zazwyczaj winę za to ponosi skan.

## Poprawa czasu o x ms

Takie zdania nie są fortunne – jeśli skan wykonuje się złożonością O(n), a lookup O(logN)… to zapytanie które wykonuje się nieoptymalnie (poprzez skany) w np. 287 ms powinno dać się zbić do ok. 3 ms (przyjmując podstawę 10, w półprawdziej rzeczywistości podstawa to rozmiar bloku czyli 8k, w rzeczywistości podstawa zależy od rozmiaru klucza sortowania indeksów / rozmiaru tabeli).

A jednak często pojawia się porównanie u Was: 287 ms, a po optymalizacji 230 ms.

O nie nie… to nie poprawa, to błąd pomiaru.

Problemem jest to że gro czasu w Management studio zajmuje wyświetlanie wyników – w rzeczywistości przecież webowe aplikacje nie robią printa na konsole. Czas który to zajmuje trwa. Trwa długo. Możecie wyłączyć print wyników z poziomu MS:

1. open a new query
2. in the menu *select Query / Query options*
3. select the Results pane
4. check the "discard result after execution"

I już samo to jest „niesamowitą optymalizacją”.

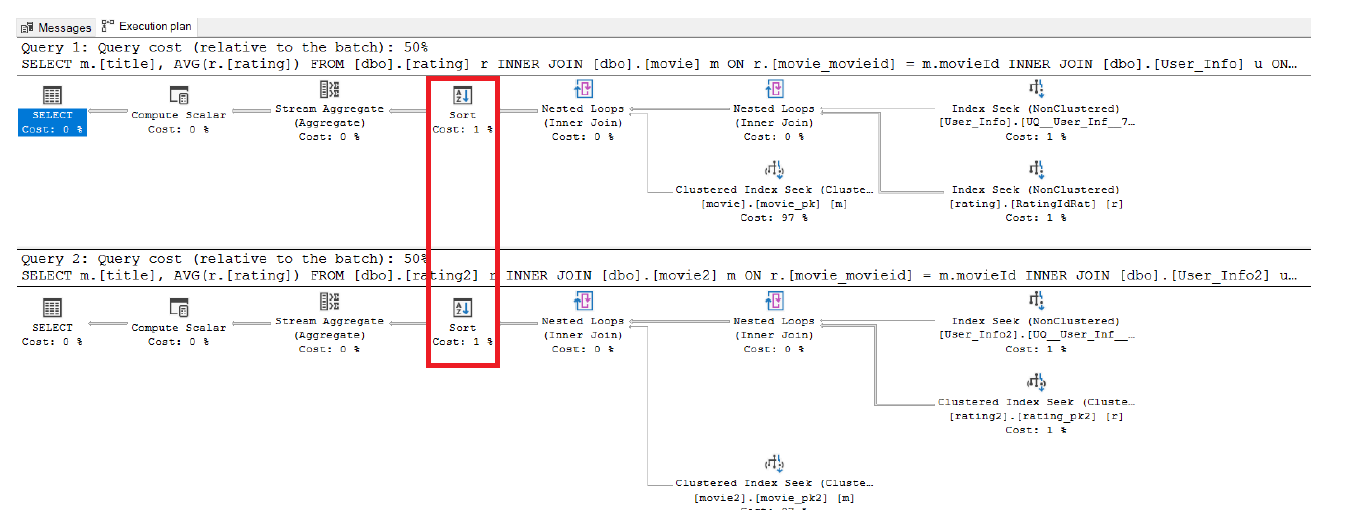
## Do 160764

Co ciekawe w 1 dobrałeś bardzo złe indeksy.

W 2 z kolei nie umiem doczytać (bo zrzut jest niewyraźny) w jakiej kolejności są łączone tabele. Masz 2 razy złączenie typu NESTED LOOP … hmm… może tu by się dało coś ugrać na wydajności?

Swoją drogą rozbawił mnie fragment: „I zastosowane indeksy nie miałby sensu w przypadku tej bazy danych i tych tabel, gdyż tylko niepotrzebnie duplikowalibyśmy dane, posortować coś posortowanego(?) – bez sensu.”

I punchline na planie wykonania zapytania:

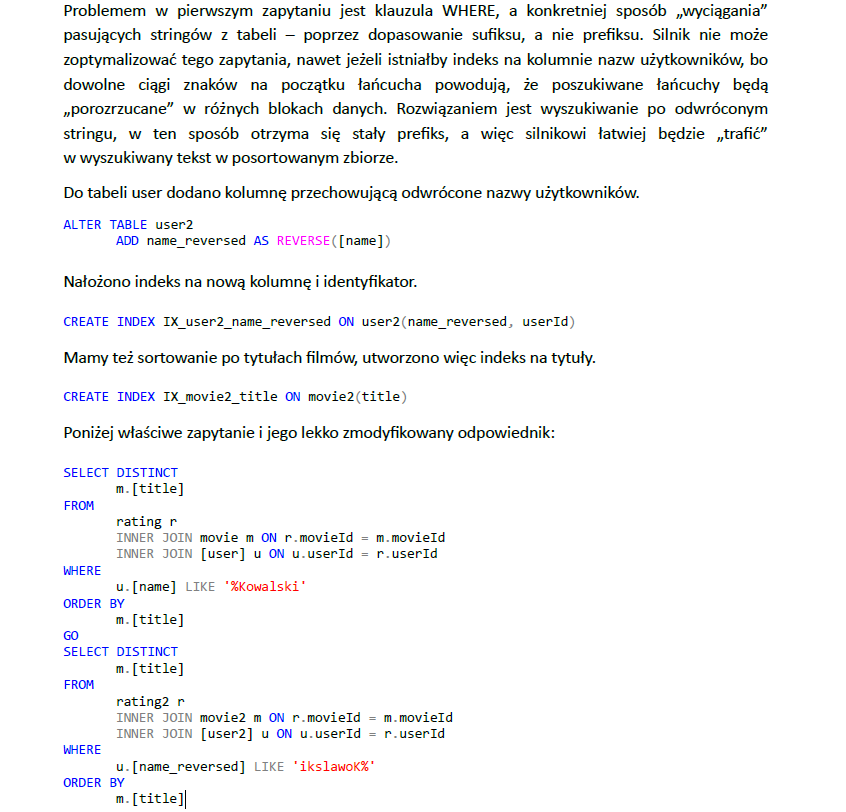


Not so sorted after all?

Sortowanie to droga operacja – być może to podpowiedź z której strony trzeba było gryźć zapytanie.

## Query 1 + do 160766

PIoter dobrze opisał co i dlaczego (Jakub też, ale po połowie sprawozdań czas odsłonić karty jak trzeba było rozwiązać zadanie – BTW: Jakub – kudos za SARGABLE).



+ spostrzeżenie ze IO Statistics, że: **liczba logicznych odczytów ([spadła] z 627 do 3).** Po jednym odczycie na źródło danich ratings/movie2/users2 (a raczej indeks). Co ciekawe można było to zrobić jeszcze odrobinę lepiej. Lepiej pod względem metadanych – postawić unikatowy indeks IX\_user2\_name\_reversed, bo skoro name jest unikatowe to tym bardziej odwrócony ciąg znaków.

Ma to tylko o tyle znaczenie, że zapytanie musimy „gdzieś zacząć”. Kolumny unikatowe z predykatem = są bardzo dobrym punktem startu dla BD. Więc jeśli BD ma (indeks name\_reversed/userId) [tu userId jeśli jest PK to może być niejawne, bo na liściach indeksów i tak są PK wstawiane) to pozostaje kwestia efektownego przeskoczenia z tabeli users na ratings. Efektowne znaczy z użyciem **merge-join –** czyli zgodności sortowania.

Dlatego IMO najlepsze indeksy w tym wypadku (bez analizowania synergii z DML-ami) to:

CREATE INDEX IX\_user2\_name\_reversed ON user2(name\_reversed, userId)

CREATE INDEX IX\_ratings ON ratings(userId, movieId)

+ PK na movie(Id)

## DO 160769

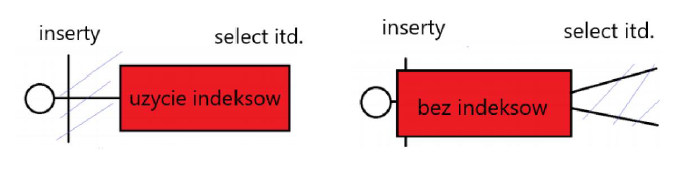
create INDEX RR ON ratings2 (rating)

Ten indeks nie będzie użyty -dlatego jego usunięcie przynosi zysk :0

## DO 160777

W zapytaniu 1 jest jakiś podejrzany HASH MATCH na planie.

## To nie jest prawda: 2



Indeksy mogą przyśpieszyć inserty, ale replikowalność testu który to wykazuje jest bólem … *‘%dupa%’*

Dlatego na tej liście INSERT jest tylko sprawdzeniem czy nie popsuliście za bardzo wydajności.

W momencie gdy macie 2 wątki (agentów/procesy) które wykonują równolegle INSERT to na różnych poziomach izolacji indeksy zaczynają grać główne skrzypce przy wydajności DML-ek.

## Do 160781

**Wnioski ogólne**

*Triggery są przydatne w komercyjnych bazach danych z potencjałem do rozrastania, ponieważ mogą*

*pomóc utrzymać porządek w bazie zabezpieczając niepożądane operacje. W praktyce działają jak*

*zwykłe funkcje tylko są wywoływane w określonych momentach, np. w momencie zapytań do bazy*

*lub zmian w niej.*

## Do 160798 + często powtarzający się błąd

Miałeś okazję do potestowania na dobrym wolumenie danych: 27 GB to chyba powiedziałbym że sweet-spot do strojenia: nie za dużo, nie za mało.

Problem, że dobrałeś sobie tak dane w zapytaniu, że miałeś 0 rows affected. To problematyczne, bo nie wiadomo czy ta informacja nie pochodzi ze **statystyk bazodanowych** – specjalnych tabel które szacują koszt zapytań. Jeśli by tak było, to optymalizujesz konkretny przypadek: kiedy baza danych **wie** że wynik nic nie zwraca…

### Ciekawostka

…. Co otwiera anegdotę dlaczego zapytanie 3 wygląda jak wygląda: oryginalnie tam była procedura do wstawiania danych i problem: <https://blog.sqlauthority.com/2019/12/19/sql-server-parameter-sniffing-simplest-example/>

Ale na pewnym etapie stwierdziłem, że nie będę złośliwy.

### + jeszcze coś

Optymalizujesz złe zapytanie :P. Interesował mnie przypadek suffixu (lub sprytnego przejścia na prefix).