DANUTA MENDRALA
MARCIN SZELIGA

WYDANIE III

SQL

PRAKTYCZNY KURS

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/pksq3v
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
https://ftp.helion.pl/przyklady/pksq3v.zip

ISBN: 978-83-283-9236-6

Printed in Poland.

Spis treści

Rozdział 9.

Wydajność zapytań

- W jaki sposób serwery bazodanowe wykonują zapytania?
- W jakiej kolejności wykonywane są poszczególne klauzule zapytań?
- Czym jest plan wykonania zapytania i jak go odczytać?
- Co oznacza akronim SARG?
- Jakie indeksy są przydatne do wyszukiwania wierszy?
- Co to znaczy, że indeks zawiera jakieś zapytanie?
- Które kolumny powinny być poindeksowane w celu poprawy wydajności złączeń?
- Jak poprawić wydajność zapytań grupujących lub partycjonujących wiersze?
- Co oznacza akronim POC?

Wykonywanie zapytań przez serwery bazodanowe

Chociaż szczegóły sposobów, w jakie poszczególne serwery bazodanowe wykonują zapytania, są różne, główne etapy tego procesu wyglądają podobnie. Ponieważ podstawowa wiedza na ten temat jest niezbędna do pisania wydajnych zapytań, poniżej przedstawione zostały poszczególne operacje wykonywane przez SQL Server:

1. Aplikacja kliencka łączy się z serwerem bazodanowym. Po udanym połączeniu nawiązana zostaje dwukierunkowa sesja, w ramach której odbywać się będzie komunikacja pomiędzy serwerem a klientem.
2. Klient wysyła do serwera instrukcję języka SQL.

3. Instrukcja ta zostaje odebrana, a następnie serwer bazodanowy musi opracować plan jej wykonania. Przeprowadzana w tym celu optymalizacja jest bardzo skomplikowanym i zależnym od danego serwera (a nawet od jego wersji) procesem, którego omówienie wykracza poza zakres tej książki. Powinieneś jednak wiedzieć, że optymalizacja jest bardzo czasochłonnym, silnie obciążającym procesor i wymagającym dużej ilości pamięci procesem, a więc wiele serwerów bazodanowych przechowuje w pamięci raz zoptymalizowane plany wykonania zapytań w celu ich ponownego użycia.
4. Dysponując znalezionym planem wykonania, serwer bazodanowy może przystąpić do wykonywania instrukcji. Prawie zawsze wiąże się to z koniecznością odczytania pewnych danych.
5. Serwery bazodanowe przechowują dane w specjalnych jednostkach, w przypadku serwera SQL są to ośmiokilobajtowe strony. Strony są też jednostkami odczytu i zapisu danych, a więc serwer zapisuje oraz odczytuje jedną lub więcej stron, a nie poszczególne wiersze czy całe tabele. Do wykonania odebranej od klienta instrukcji serwer będzie więc musiał odczytać zawierające niezbędne do wykonania zapytania strony.
6. Jeżeli strony te znajdowały się już w buforze (pamięci RAM), wykonana zostanie operacja logicznego odczytu. W przeciwnym wypadku strony zostaną wczytane z dysku do bufora — taki odczyt nazywany jest odczytem fizycznym. Ponieważ pamięć jest znacznie szybsza od dysków, fizyczne odczyty są bardzo mało wydajne[1]. W związku z tym, żeby zapewnić jak najwyższą wydajność, należy zminimalizować liczbę fizycznych odczytów poprzez wyposażenie serwera w odpowiednią (wystarczającą do zbuforowania wszystkich danych) ilość pamięci RAM.
7. Wynik wykonania instrukcji języka SQL (w przypadku zapytania będzie to zbiór wierszy) jest wysyłany do aplikacji klienckiej.

Kolejność wykonywania klauzul zapytania

Chociaż serwery bazodanowe optymalizują zapytania przed ich wykonaniem, proces ten nie może mieć wpływu na wynik zapytania (wykonanie zapytania według różnych planów, np. poprzez złączenie tabel w różnej kolejności albo poprzez pogrupowanie wierszy, a następnie złączenie otrzymanych w ten sposób grup czy też złączenie tabel, a następnie pogrupowanie wierszy, musi zawsze skutkować zwróceniem tego samego wyniku).

Poszczególne klauzule instrukcji `SELECT` są wykonywane zawsze w tej samej kolejności. Jeżeli któraś z opcjonalnych klauzul nie występuje, dany krok jest po prostu pomijany.

[1] Automatyczne buforowanie danych jest powodem, dla którego ponowne wykonanie tego samego zapytania może być znacznie szybsze.

Rezultatem wykonania każdego kroku jest zbiór pośredni — wirtualna tabela, która nie jest dostępna dla użytkownika. **Kolejny krok jest wykonywany tylko w oparciu o zbiór pośredni będący rezultatem wykonania poprzedniego kroku**, a zbiór pośredni, rezultat wykonania ostatniego kroku, jest zwracany użytkownikowi.

Tak więc chociaż faktyczny sposób, w jaki serwer bazodanowy wykona nasze zapytanie, może być (i najczęściej będzie) inny niż przedstawiona poniżej ogólna kolejność wykonywania zapytań, warto znać tę logiczną kolejność wykonywania poszczególnych klauzul.

W przypadku zapytań niegrupujących danych ich klauzule wykonywane są w następującej kolejności:

1. Najpierw serwer musi pobrać potrzebne do wykonania zapytania dane, a więc wykona klauzulę `FROM` oraz, jeżeli istnieją, klauzule `JOIN` i operatory `APPLY`.
2. Następnie wybrane zostaną wiersze spełniające warunki klauzuli `WHERE`.
3. Dla otrzymanych w wyniku wykonania dwóch poprzednich punktów wierszy wykonane zostaną wyrażenia zdefiniowane w klauzuli `SELECT`.
4. Na końcu otrzymane wiersze zostaną posortowane, o ile w zapytaniu wystąpiła klauzula `ORDER BY`.

Wykonanie zapytania grupującego wymaga wykonania dodatkowych operacji:

1. Punkty pierwszy i drugi są wykonywane w taki sam sposób jak dla zapytań niegrupujących.
2. Przed wykonaniem klauzuli `SELECT` wiersze kandydujące są grupowane (wykonywana jest klauzula `GROUP BY`).
3. Otrzymane w wyniku grupowania wiersze są filtrowane, o ile w zapytaniu wystąpiła klauzula `HAVING`.
4. Dla otrzymanych w wyniku wykonania poprzednich punktów wierszy wykonane zostaną wyrażenia zdefiniowane w klauzuli `SELECT`.
5. Na końcu otrzymane wiersze zostaną posortowane, o ile w zapytaniu wystąpiła klauzula `ORDER BY`.

Plany wykonania zapytań

Umiejętność czytania i analizowania planów wykonania zapytań jest niezbędna każdemu, kto chce pisać nie tylko poprawne (zwracające właściwe wyniki), ale również wydajne (zwracające te wyniki w najszybszy możliwy sposób) zapytania. Wynika to z faktu, że to samo zapytanie może być wykonane na bardzo dużo różnych sposobów (liczba możliwych sposobów wykonania tego samego zapytania zależy od samego zapytania i od serwera bazodanowego, ale nawet nieskomplikowane zapytania odczytujące dane z kliku tabel mogą być wykonane na kilkadziesiąt, a nawet kilkaset

różnych sposobów). Zadaniem serwera bazodanowego jest znalezienie wystarczająco dobrego sposobu (planu) wykonania zapytania. Żeby mu to umożliwić, powinniśmy nie tylko unikać typowych błędów (takich jak używanie argumentów uniemożliwiających efektywne skorzystanie z indeksów), ale również pisać zapytania w sposób ułatwiający znalezienie optymalnych planów ich wykonania. Najprostszym sposobem sprawdzenia, czy nasze zapytanie wykonywane jest wydajnie, jest właśnie analiza planu jego wykonania.

SQL Server pozwala odczytywać plany wykonania zapytań w formie tekstowej oraz graficznej. Dodatkowo możliwe jest poznanie szacowanych oraz faktycznych planów wykonania. Ponieważ szczegółowe przedstawienie kwestii analizy planów wykonania zapytań wykracza poza zakres tej książki, ograniczymy się do przedstawiania faktycznych planów wykonania zapytań w formie graficznej.

Zacznijmy od wyjaśnienia tego, czym jest plan wykonania zapytania. Plan wykonania reprezentuje opracowaną przez serwer bazodanowy strategię odczytania i przetworzenia danych na potrzeby wykonania zapytania. Składa się ona z serii iteratorów, z których każdy wykonuje pojedynczą operację, taką jak odczytanie danych, posortowanie wierszy, złączenie tabel itd.

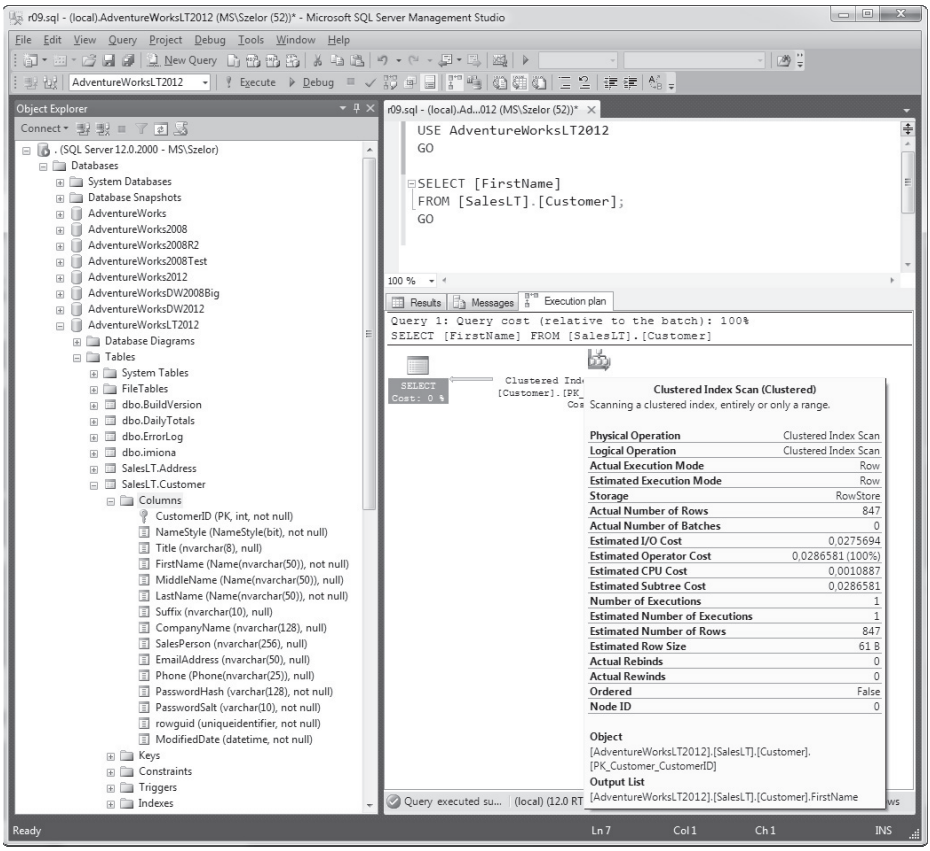
Plany wykonania zapytań można czytać w dwóch kierunkach:

1. Od lewa do prawa — ten kierunek reprezentuje logikę wykonywania zapytania;
2. Lub od prawa do lewa — ta kolejność repetuje przepływ danych pomiędzy kolejnymi iteratorami.

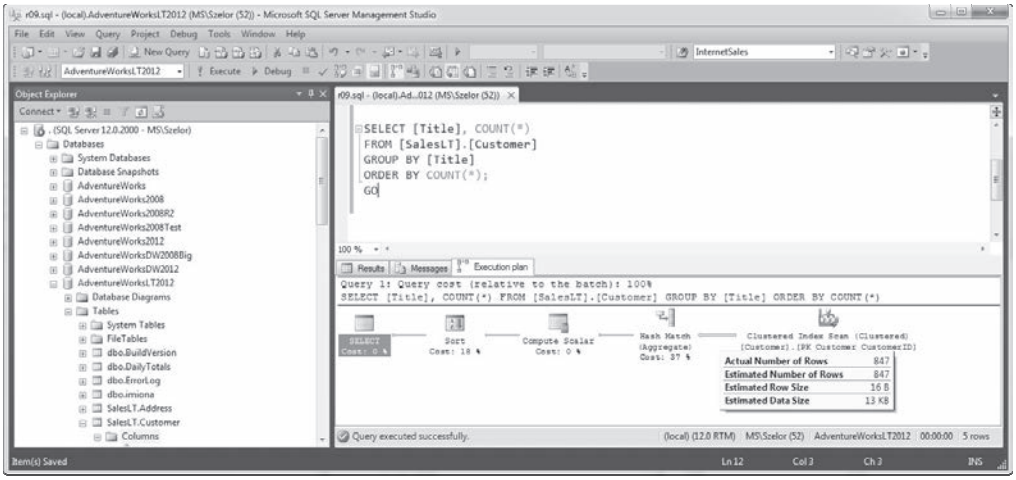
Żeby wyświetlić graficzny plan wykonania zapytania, należy kliknąć znajdujący się na pasku zadań konsoli SSMS przycisk *Include Actual Execution Plan* lub nacisnąć kombinację klawiszy *Ctrl+M*, a następnie uruchomić analizowane zapytanie (rysunek 9.1).

Przyjrzyjmy się nieco bardziej rozbudowanemu planowi wykonania zapytania. Przedstawiony na rysunku 9.2. plan wykonania należy przeczytać następująco:

1. Przeskanowany został indeks zgrupowany (tabela `[SalesLT].[Customer]`).
2. Odczytanych w ten sposób 847 wierszy (umieszczając kursor myszki nad reprezentującymi przepływ danych strzałkami, wyświetlimy dodatkowe informacje na temat liczby i rozmiaru wierszy) zostało pogrupowanych.
3. Dla każdej grupy wyliczona została funkcja `COUNT`.
4. Wiersze zostały posortowane.
5. Na końcu wiersze zostały zwrócone do aplikacji klienckiej.



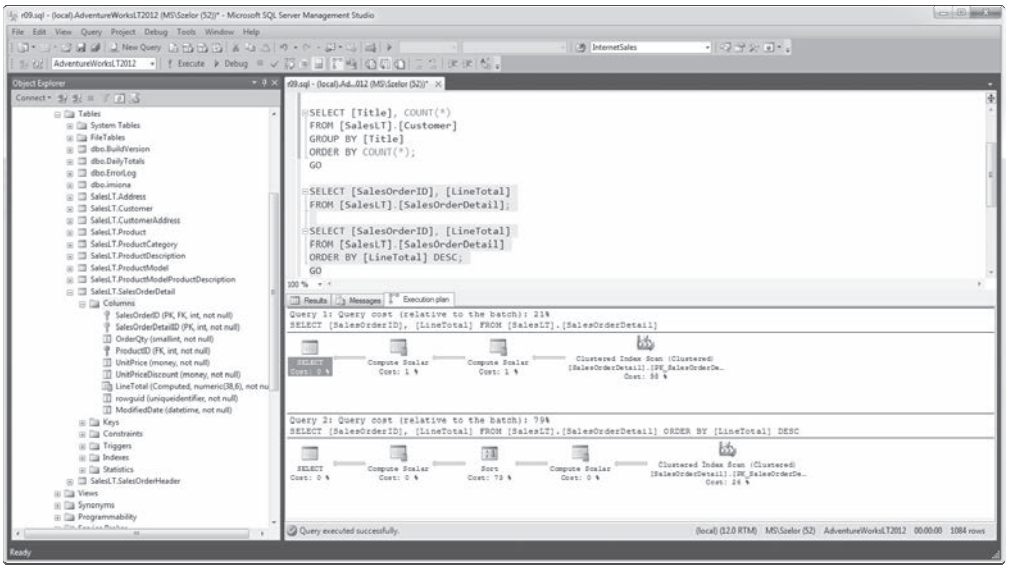
Rysunek 9.1. *Najprostszy plan wykonania zapytania — potrzebne dane zostały odczytane za pomocą iteratora Clustered Index Scan (szczegóły operacji możemy poznać, umieszczając kursor myszki nad danym iteratorem), a następnie zwrócone do aplikacji klienckiej*



Rysunek 9.2. *Składający się z pięciu iteratorów plan wykonania zapytania grupującego dane*

SQL Server ocenia i podaje koszt wykonania każdej operacji — np. w poprzednim przykładzie koszt pogrupowania wierszy wyniósł 37% kosztów wykonania całego zapytania. Ta wiedza pozwala nam na wdrożenie względnie prostej, a jednocześnie skutecznej metody optymalizacji zapytań, polegającej na eliminowaniu z planów wykonania najkosztowniejszych operacji.

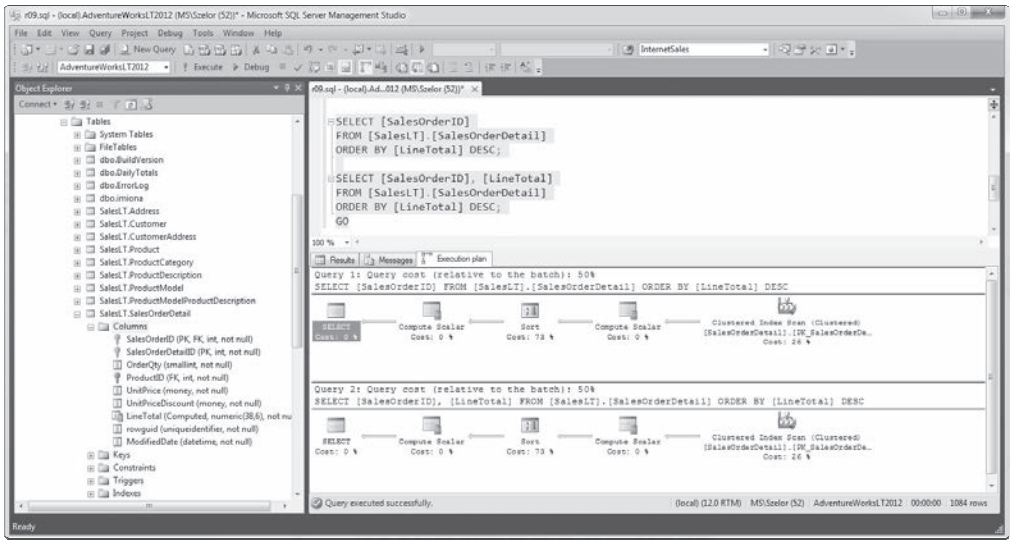
Co więcej, jeżeli jednocześnie uruchomimy kilka zapytań (np. kilka wersji tego samego zapytania), serwer SQL zwróci informację o procentowym rozkładzie kosztów wykonania całego wsadu (zbiór jednocześnie wysłanych do serwera zapytań nazywa się wsadem). Rysunek 9.3 ilustruje tę funkcjonalność.



Rysunek 9.3. *Drugie zapytanie (to z klauzulą ORDER BY) okazało się cztery razy bardziej kosztowne od pierwszego. Różnica wynika z tego, że wykonując drugie zapytanie, serwer musiał dodać kosztowny iterator SORT*

Przekonaliśmy się właśnie, że **sortowanie może wielokrotnie wydłużyć czas wykonania zapytania**, a więc jeżeli wynik zapytania nie musi być posortowany, nie należy umieszczać w nim klauzuli `ORDER BY`.

Porównajmy jeszcze koszty wykonania zapytania, w którym użyta do sortowania kolumna nie została dołączona do wyniku z zapytaniem zawierającym tę kolumnę (rysunek 9.4) — okaże się, że są one identyczne, a więc **niewymienienie w klauzuli** `SELECT` **kolumn użytych do sortowania nie skraca czasu wykonania zapytania**.



Rysunek 9.4. *Chociaż analiza planów wykonań zapytań przez konkretny serwer bazodanowy nie jest tematem tej książki, to warto wiedzieć, że kolumny użyte do sortowania muszą być odczytane tak samo jak kolumny wymienione w klauzuli SELECT. Z tego powodu większość serwerów bazodanowych wykona oba powyższe zapytania w tym samym czasie, a koszt wykonania każdego z nich przez serwer SQL wyniósł dokładnie 50% kosztu wykonania całego wsadu*

Wydajne wyszukiwanie danych za pomocą argumentów SARG

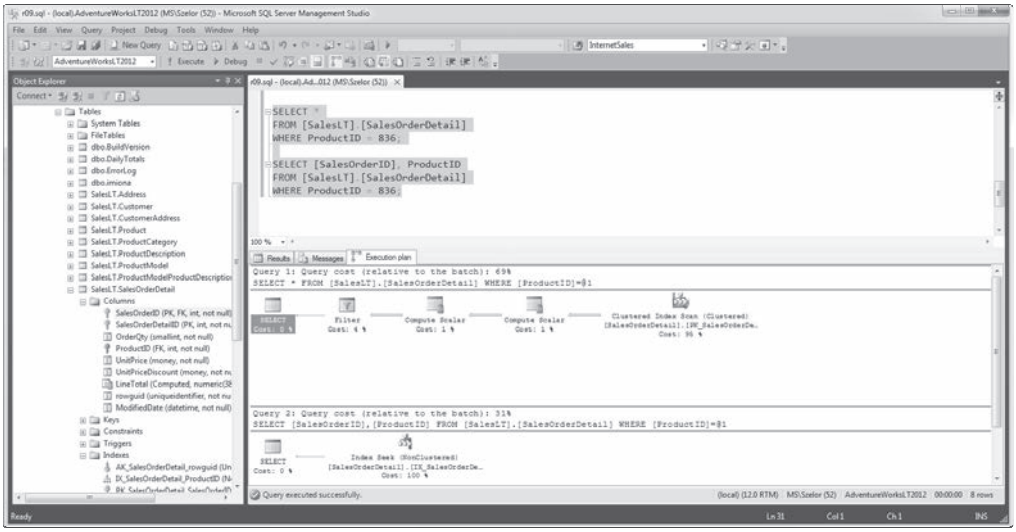
Dane mogą być zapisane:

1. W tabelach — dla uproszczenia i zgodnie z relacyjnym modelem baz danych przyjmijmy, że kolejność wierszy tabeli jest nieistotna, a więc nie są one w żaden sposób posortowane[2].
2. W indeksach — aby uniknąć analizowania różnic pomiędzy poszczególnymi serwerami bazodanowymi, przyjmijmy, że indeks przypomina skorowidz książki — dołączaną na jej końcu, alfabetycznie uporządkowaną listę haseł z odnośnikami do numerów stron, na których dane hasło jest opisane. W przypadku baz danych odpowiednikiem hasła będzie wybrana kolumna (lub kolumny) tabeli, czyli tzw. klucz indeksu, a odnośnikiem wskaźnik[3]. Podsumowując, **każdy klucz indeksu posiada wskaźnik do wiersza tabeli zawierającego wartości pozostałych (niekluczowych) kolumn, a klucze indeksu są zawsze posortowane**.

[2] Wiele serwerów bazodanowych porządkuje wiersze tabeli według wartości klucza podstawowego. Taka posortowana tabela nazywana jest indeksem zgrupowanym.

[3] Więcej informacji na temat indeksów znajduje się w rozdziale 13.

Jeżeli odczytywana tabela nie jest poindeksowana, serwer bazodanowy wykonując dowolne odwołujące się do tej tabeli zapytanie, będzie musiał odczytać ją w całości. Tego typu sytuacja zachodzi też, gdy odczytujemy wszystkie kolumny tabeli za pomocą symbolu *. Żeby się o tym przekonać, wystarczy porównać plany wykonania dwóch pokazanych na rysunku 9.5 zapytań:

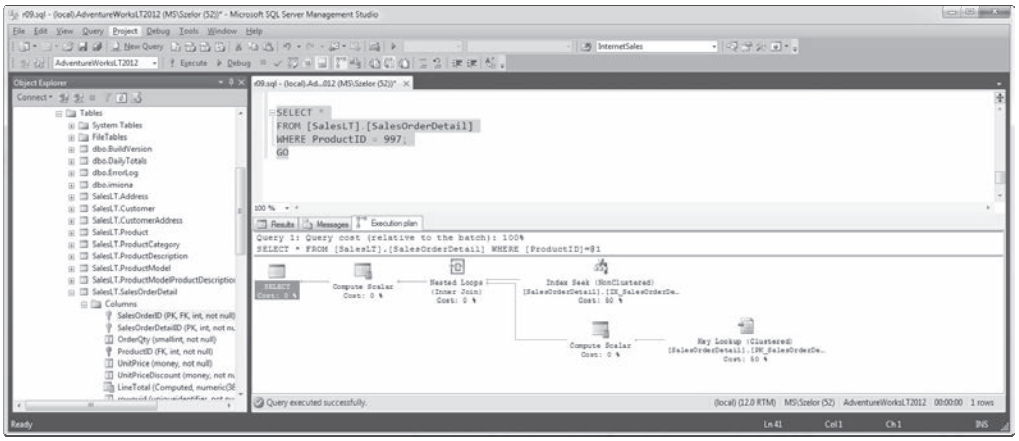


Rysunek 9.5. *Wyszukanie w indeksie kluczy spełniających warunek z klauzuli WHERE i odczytanie tylko czterech wierszy tabeli okazało się w tym przypadku trzy razy szybsze niż odczytanie całej tabeli i wybranie czterech wierszy spełniających podany warunek*

Podsumujmy — tabela `[SalesLT].[SalesOrderDetail]` ma założony indeks na kolumnie `ProductID`. Indeks ten zawiera, oprócz identyfikatorów produktów, identyfikatory zamówień. Nie zawiera natomiast pozostałych kolumn tej tabeli. Z tego powodu wykonanie pierwszego, zwracającego tylko cztery wiersze, zapytania wymagało odczytania całej tabeli, podczas gdy do znalezienia odpowiednich danych w indeksie wystarczyło go przeszukać. O takich zapytaniach jak te drugie mówimy, że zawierają się one w indeksie, ponieważ wszystkie potrzebne do ich wykonania dane zapisane są w założonym dla tabeli indeksie.

> Utworzenie zawierającego zapytanie indeksu jest najprostszym i najskuteczniejszym sposobem na poprawę wydajności tego zapytania. Nie oznacza to jednak, że powinniśmy tworzyć indeksy zawierające wszystkie zapytania. Ponieważ serwer bazodanowy automatycznie synchronizuje dane w indeksach z danymi w tabeli, liczba indeksów zdefiniowanych dla pojedynczej tabeli nie powinna przekraczać 10.

Jeżeli istnieje indeks zawierający zapytanie, serwer bazodanowy zawsze go użyje. Zupełnie inaczej wygląda użycie przez serwery bazodanowe indeksów podczas wykonywania zapytań, które się w nich nie zawierają (rysunek 9.6).



Rysunek 9.6. *Plan wykonania zapytania poprzez przeszukanie indeksu i pobranie brakujących w tym indeksie danych z tabeli*

Na planie wykonania drugiego zapytania widoczny jest operator `Key Lookup` symbolizujący odczytywanie danych z tabeli `[SalesLT].[SalesOrderDetail]`. Operacja ta musiała być przeprowadzona, bo indeks zawierał tylko identyfikatory produktów i zamówień, a zapytanie miało zwrócić wszystkie dane wybranego zamówienia. Takie sięganie po dane jest na tyle kosztowne, że **jeżeli zapytanie zwraca więcej niż kilka procent wierszy tabeli, serwery bazodanowe przestają używać indeksów i odczytują całą tabelę, tak jakby użyta do wyszukiwania kolumna nie była zindeksowana**.

Wiedząc, w jaki sposób serwery bazodanowe odczytują dane, możemy teraz uzasadnić uwagę z jednego z poprzedniego rozdziałów, w której napisaliśmy, że **używając symbolu *, zmuszamy serwer bazodanowy do odczytania wszystkich kolumn tabeli, co może wielokrotnie wydłużyć czas wykonywania zapytania**. Ponieważ większość zapytań zwraca więcej niż 1% wierszy tabeli, posługując się symbolem *, powodujemy, że serwery bazodanowe nie korzystają z istniejących indeksów.

Skoro koszt (a więc i czas) wykonania zapytań z użyciem indeksów jest wielokrotnie niższy niż koszt odczytania tych samych danych z tabel, **powinniśmy tak pisać zapytania, żeby serwery bazodanowe korzystały z indeksów przy ich wykonywaniu**.

Skrót SARG (ang. *Search ARGuments*) oznacza warunki wyszukiwania, czyli takie warunki logiczne, które pozwalają serwerom bazodanowym na wybranie właściwych wierszy poprzez przeszukanie indeksu, a nie odczytanie całej tabeli.

Ponieważ użycie indeksów może być bardzo kosztowne[4], serwery bazodanowe, zanim się na nie zdecydują, szacują koszt wykonania zapytania, używając do tego danych statystycznych. Jeżeli nie będą w stanie oszacować tego kosztu, wybiorą najbezpieczniejsze rozwiązanie, czyli odczytają tabelę.

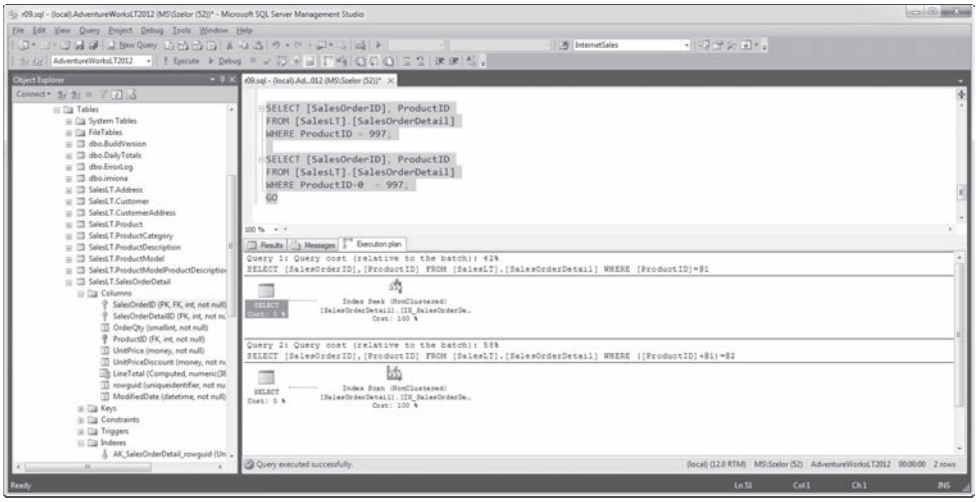
[4] Z powodu opisanej wcześniej operacji *Key Lookup*, polegającej na odczytywaniu dla każdego klucza indeksu odpowiadającego mu wiersza tabeli.

W poniższych punktach zostały opisane przypadki, w których serwer bazodanowy może nie mieć możliwości oszacowania kosztu wykonania zapytania, a więc nie skorzysta z indeksów i czas wykonania zapytań będzie wielokrotnie dłuższy:

1. Zanegowanie warunku logicznego (użycie operatora `NOT` albo operatora `<>`), o ile serwer bazodanowy nie potrafi automatycznie przekształcić warunku logicznego w równoważny warunek pozytywny. Przykładem takiego przekształcenia może być zastąpienie przez serwer SQL warunku `NOT CustomerID >3` warunkiem `CustomerID <= 3`:

```
SELECT *
FROM [SalesLT].[SalesOrderDetail]
WHERE ProductID <> 999;
```

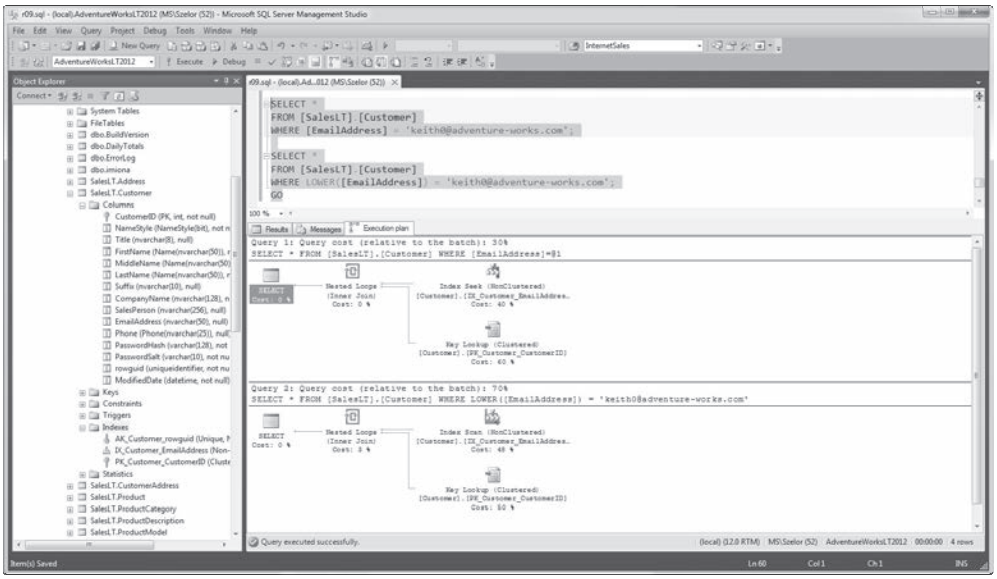
2. Umieszczenie nazwy kolumny w ramach dowolnego wyrażenia, nawet jeżeli to wyrażenie nie zmienia wyniku testu logicznego. Z tego powodu, choć oba pokazane na rysunku 9.7 zapytania zwracają te same dane, to pierwsze zostanie wykonane czterokrotnie szybciej.



Rysunek 9.7. *Użycie nazwy kolumny jako części dowolnego wyrażenia oznacza, że serwer bazodanowy będzie musiał odczytać cały indeks (albo nawet całą tabelę), żeby mieć pewność, że zwrócił wszystkie wiersze spełniające tego typu warunek*

3. Użycie nazwy kolumny jako argumentu dowolnej funkcji (rysunek 9.8).
4. Użycie do wyboru wierszy wzorca zaczynającego się od symbolu wieloznacznego, np. `Name LIKE N'%b'`.

Przykładowa baza danych jest bardzo mała i powyższe zapytania odwołują się do tabel liczących zaledwie kilkaset wierszy każda. Tylko dlatego różnice w wydajności poprawnie napisanych (umożliwiających przeszukiwanie indeksu) i nieoptymalnych zapytań są tak małe. Tymczasem swoją prawdziwą siłę indeksy pokazują w przypadku dużych, liczących tysiące czy miliony wierszy, tabel. Wtedy znalezienie wybranych wierszy w indeksie nadal wymaga odczytania takiej samej ilości danych, podczas gdy odczytanie całej tabeli czy indeksu może zająć nawet kilka minut.



Rysunek 9.8. *Umieszczenie nazwy kolumny jako argumentu dowolnej (również systemowej) funkcji ma dokładnie ten sam wpływ na użycie indeksów co użycie nazwy kolumny w wyrażeniu*

Poprawa wydajności złączeń

Łączenie tabel, szczególnie tych dużych (liczących wiele wierszy i kolumn), jest kosztowną operacją — wybranie pasujących do warunku złączenia (warunku z klauzuli `ON`) wierszy w najgorszym wypadku jest operacją o złożoności obliczeniowej rzędu $O(n^2)$, gdzie n jest liczbą wierszy. Oznacza to, że jeżeli koszt złączenia 100 wierszy wyniósł 10 000, to koszt złączenia 200 wierszy wyniesie już 40 000, a więc cztery razy więcej niż w przypadku dwukrotnie mniejszych tabel.

Zmniejszyć ten koszt możemy, zakładając indeksy na używanych do łączenia tabel kolumnach — jeżeli obie użyte do złączenia kolumny są poindeksowane, złączenie tabel będzie operacją o złożoności obliczeniowej rzędu O(n). Ponieważ prawie zawsze tabele są łączone na podstawie pary kolumn klucz podstawowy – klucz obcy, to na tych kolumnach należy założyć indeksy.

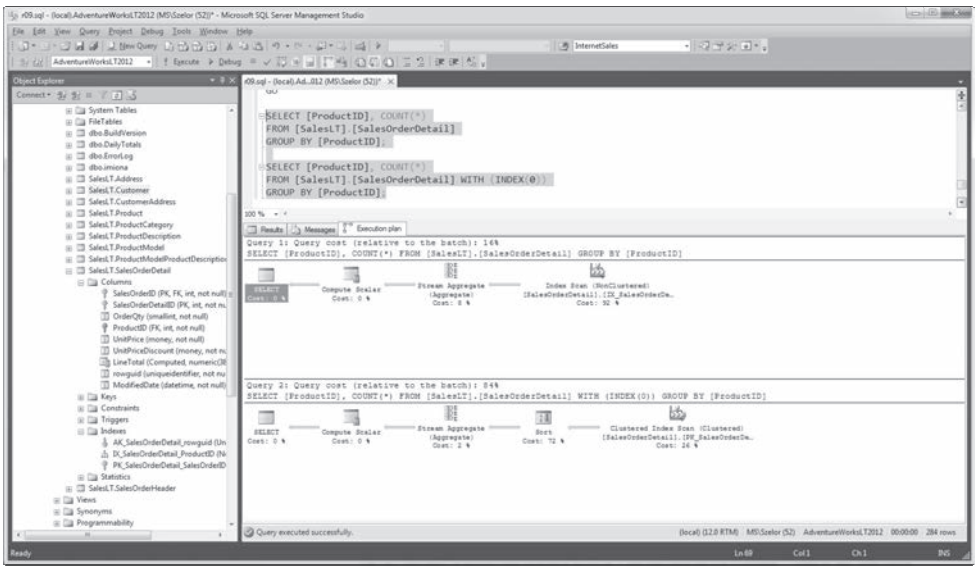
Większość serwerów bazodanowych, w tym SQL Server, automatycznie indeksuje kolumny klucza podstawowego. Oznacza to, że aby poprawić wydajność złączeń, wystarczy założyć indeksy na kolumnach kluczy obcych wszystkich tabel bazy danych.

Wydajne grupowanie i partycjonowanie danych

Grupowanie, sortowanie i wyszukiwanie danych może być bardzo kosztowne. Najprostszym i najskuteczniejszym sposobem na poprawę wydajności grupowania danych, tak samo jak sortowania czy wyszukiwania na podstawie argumentów SARG, jest użycie indeksów.

W przypadku grupowania zindeksowana powinna być każda lub co najmniej pierwsza kolumna wymieniona w klauzuli `GROUP BY`**.** Dzięki temu serwer bazodanowy będzie mógł odczytać odpowiednio posortowane dane, co znacznie uprości i przyspieszy ich pogrupowanie (grupowanie posortowanych danych wymaga wielokrotnie mniej czasu procesora i pamięci niż grupowanie nieposortowanych danych)[5]. Ponadto klucze indeksu są z reguły znacznie mniejsze niż wiersze tabeli, a więc zmniejszy się liczba odczytywanych z dysku danych.

Żeby się o tym przekonać, wykonamy dwa zapytania zwracające dokładnie tak samo pogrupowane i posortowane dane, za drugim razem zabraniając serwerowi bazodanowemu skorzystania z indeksu[6] (rysunek 9.9).



Rysunek 9.9. *Pogrupowanie danych nawet małej, liczącej mniej niż tysiąc wierszy, tabeli bez indeksu okazało się pięć razy bardziej kosztowne niż użycie w tym celu odpowiedniego indeksu*

[5] Niektóre serwery bazodanowe potrafią grupować tylko posortowane dane. Jeżeli brakuje im przydatnego indeksu, przed grupowaniem dodatkowo sortują dane.

[6] Serwery bazodanowe umożliwiają określenie sposobu, w jaki zapytanie ma być wykonane — służą do tego specyficzne dla danego serwera dyrektywy optymalizatora. Dyrektywa `WITH (INDEX(0))` zabrania serwerowi SQL użycia jakiegokolwiek indeksu.

Zapytania partycjonujące dane (a więc zapytania, w których występuje klauzula `OVER`) mają większe wymagania co do indeksów. W ich przypadku najlepszymi indeksami są indeksy POC (ang. *Partitioning, Ordering, Covering*), czyli takie, w których pierwsze kolumny są kolumnami użytymi do partycjonowania danych (w klauzuli `PARTITION BY`), po nich występują kolumny użyte do ustalenia porządku wierszy (w klauzuli `ORDER BY`), a indeks zawiera też wszystkie pozostałe (wymienione w klauzuli `SELECT`) kolumny.

Podsumowanie

- Kolejność, w jakiej wpisuje się poszczególne klauzule zapytania, nie odpowiada kolejności, w której są one wykonywane przez serwery bazodanowe.
- Serwery bazodanowe optymalizują sposoby wykonania zapytań.
- Sposobem na sprawdzenie, czemu dane zapytanie działa wolniej, niż się spodziewaliśmy, jest sprawdzenie planu jego wykonania.
- Indeksy są najprostszym i wyjątkowo skutecznym sposobem na poprawę wydajności zapytań.
- Samo istnienie indeksu nie wystarczy, żeby serwer bazodanowy mógł z niego efektywnie skorzystać — zapytanie musi być poprawnie napisane, np. do wybierania wierszy powinno się używać argumentów typu SARG, a zapytanie nie powinno odczytywać więcej kolumn, niż jest to wymagane.
- Uniwersalna strategia indeksowania polega na tworzeniu indeksów zawierających zapytania, a wiec indeksów wielokolumnowych. Dla zapytań grupujących lub partycjonujących dane należy tworzyć indeksy typu POC.

Zadania

1. Poniższe zapytanie zwracające nazwy produktów i kategorii działa zbyt wolno. Jak można poprawić jego wydajność?

```
SELECT [Name]
FROM [SalesLT].[Product]
UNION
SELECT [Name]
FROM [SalesLT].[ProductCategory];
```

2. Wykonanie poniższego zapytania wymaga przeskanowania (odczytania w całości) indeksu założonego na kolumnie `UnitPrice`. Przepisz to zapytanie tak, żeby używało argumentu typu SARG.

```
SELECT [SalesOrderID]
FROM [SalesLT].[SalesOrderDetail]
WHERE [UnitPrice]*.77 > 900;
```

3. Zajdź optymalny indeks dla poniższego zapytania:

```
SELECT [DueDate], [SalesOrderID], [TotalDue],
       LAG([TotalDue]) OVER (PARTITION BY [DueDate] ORDER BY [DueDate]) as
       PreviusTotalDue
FROM [SalesLT].[SalesOrderHeader]
ORDER BY [DueDate];
```

Skorowidz

J

K

L

Ł

M

N

O

P

R

S

T

U

W

Z

PRAKTYCZNY
KURS

SQL to dziś właściwie jedyny poważny język używany do tworzenia i obsługi relacyjnych baz danych, niezależnie od tego, czy są to bazy Microsoftu, Oracle czy Sun Microsystems. I choć nie wszędzie działa dokładnie tak samo, bez jego znajomości nie ma co marzyć o swobodnym korzystaniu z bazy, nie mówiąc już o jej stworzeniu. Jeśli więc taka baza jest Ci potrzebna, jeśli chcesz zorientować się, jak ułożyć dane w sposób najwygodniejszy dla siebie albo precyzyjnie wysegregować to, czego akurat szukasz, musisz opanować SQL — inaczej serwer bazodanowy nijak Cię nie zrozumie.

Trzecie wydanie tej książki traktuje o języku SQL w wersji dla SQL Server firmy Microsoft. Autorzy szybko przeprowadzą Cię od instalacji serwera bazodanowego, przez najróżniejsze operacje na przykładowej, niewielkiej bazie AdventureWorksLT, aż po kwestie związane z tworzeniem własnej bazy i nadawaniem uprawnień jej użytkownikom. Ponadto znajdziesz tu ważne (i nowe!) informacje o partycjonowaniu danych i wydajności zapytań. Bezcennym wsparciem w trakcie nauki będą dla Ciebie zadania — ich rozwiązanie pozwoli Ci poczuć się pewniej i sprawdzić swoje wiadomości w praktyce. Jeśli chcesz rozpocząć przygodę z bazami danych albo odświeżyć wiedzę, trafiłeś doskonale!

- Teoria baz danych i standardy języka SQL
- Odczytywanie danych z wybranej tabeli
- Wybieranie i grupowanie wierszy
- Łączenie tabel i wyników zapytań
- Partycjonowanie wierszy oraz funkcje rankingu, analityczne i okienkowe
- Podzapytania i wydajność zapytań
- Transakcje i współbieżność
- Bazy danych i tabele
- Widoki i indeksy
- Nadawanie i odbieranie uprawnień

Stwórz bazę danych z językiem SQL!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-9236-6

Cena: 59,90 zł