Andrzej PTASZNIK

ELEMENTY OPTYMALIZACJI ZAPYTAŃ SQL

Streszczenie

Współczesne bazy danych mogą przechowywać gigantyczne ilości danych i korzystać z nich mogą jednocześnie tysiące użytkowników. W bazach danych obsługujących działania operacyjne firmy krytyczne znaczenie ma szybkość realizowania zapytań. Problemy optymalizacji zapytań są złożone i wymagają od administratorów i programistów dużej wiedzy i doświadczenia. Wykład zapoznaje słuchaczy z problematyką wydajności i optymalizacji zapytań SQL. Omówiona zostanie fizyczna organizacja przechowywania danych i wprowadzone zostaną pojęcia indeksów zgrupowanych i niezgrupowanych. Zaprezentowane zostaną przykłady planów wykonania zapytań generowane przez optymalizator SQL. Na bazie przykładu omówione zostaną problemy wyboru strategii wykonania zapytania w zależności od zawartości tabel i zdefiniowanych indeksów. Wprowadzone zostanie pojecie statystyk indeksów i omówione będzie ich znaczenie przy wyborze strategii realizacji zapytania.

Abstract

Present-day databases can store vast amounts of data and can be used simultaneously by thousands of users. Query executing speed is of crucial significance in databases for operational activities of a company. The issue of query optimisation is complex and it requires immense knowledge and experience from administrators and programmers. The lecture acquaints the audience with a range of issues connected with SQL query performance and optimisation. The physical organisation of data storage will be discussed and the concepts of clustered and non-clustered indexes will be introduced. The examples of query execution plans generated by SQL optimiser will be presented. On the basis of an example, issues of the choice of query execution strategy depending on the content of tables and defined indexes will be discussed. The concept of index statistics will be introduced and their significance for the choice of query execution strategy will be discussed.

1. WPROWADZENIE

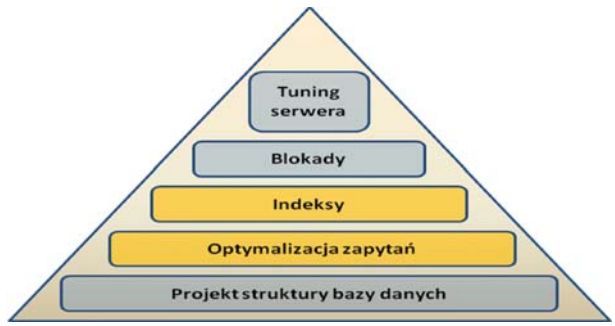
W każdym projekcie informatycznym, wykorzystującym relacyjne bazy danych, prędzej czy później pojawia się problem związany z wydajnością. Jeśli „prędzej" oznacza „przed wdrożeniem", to nie jest jeszcze tak źle. Można wtedy podjąć decyzje wiążące się z dokonywaniem zmian w projekcie bazy danych i nie będą one się wiązać

z koniecznością dbania o już istniejące dane. Gorszym wariantem jest praca na „żywym organizmie". Nie dość, że możliwości modyfikacji są ograniczone, to jeszcze trzeba starać się nie zakłócać normalnej pracy użytkowników. Gdy dodamy do tego presję czasu i stres – pojawia się obraz pracy nie do pozazdroszczenia. W każdym jednak przypadku istotne jest, żeby wiedzieć, jakie kroki podjąć, co sprawdzić, na co zwrócić szczególną uwagę, jakich narzędzi użyć i w jaki sposób aby osiągnąć cel – wzrost wydajności bazy danych do akceptowalnego poziomu. Może nam się wydawać, że takie problemy dotyczą tylko dużych projektów i baz danych, więc nie ma się co martwić na zapas. Bardzo szybko jednak można natrafić na podobne problemy nawet w prostych aplikacjach.

W ramach niniejszego wykładu postaramy się przedstawić podstawy wiedzy potrzebnej do poruszania się w dziedzinie zagadnień związanych z wydajnością baz danych, a dokładniej – zapytań na nich wykonywanych. Zanim zaczniemy jednak wkraczać do problematyki optymalizacji zapytań SQL, postaramy się odpowiedzieć na pytanie: A po co w ogóle optymalizować? Odpowiedź na to pytanie nie jest wbrew pozorom taka oczywista. Niejako przy okazji zaprezentowany zostanie też ogólny model optymalizacji wydajności stosowany w praktyce przy realizacji zadań związanych z zapewnieniem wymaganego poziomu wydajności bazy danych. Istnieją sprawdzone w praktyce podejścia (modele) optymalizacji wydajności baz danych, lecz ich rola polega raczej na wyznaczeniu ogólnych ram i sekwencji czynności, których wykonanie należy wziąć pod uwagę przy prowadzeniu optymalizacji, niż na dostarczeniu gotowej recepty. Proces optymalizacji wydajności według przyjętego przez nas modelu składa się z kilku obszarów:

- Struktura (projekt) bazy danych
- Optymalizacja zapytań
- Indeksy
- Blokady
- Tuning serwera

Całość modelu jest przedstawiona na diagramie na rys. 1.



Rys. 1. Model procesu optymalizacji

Kolejność realizacji zadań powinna przebiegać od dołu diagramu do góry. Podobnie, liczba możliwych do osiągnięcia usprawnień jest tym większa, im niżej znajdujemy się na diagramie. Sekwencja ta nie jest przypadkowa i wzajemne zależności pomiędzy blokami powodują, że założona kolejność realizacji umożliwia uzyskanie najlepszych efektów najmniejszym kosztem. W ramach wykładu skupimy się na wyróżnionych blokach – optymalizacji zapytań i indeksach.

2. FIZYCZNA ORGANIZACJA DANYCH W SQL SERVER 2008

Warto zadać sobie nieco trudu i zapoznać się z fizycznym sposobem przechowywania danych w bazie. Zrozumienie podstaw ułatwi później zrozumienie, dlaczego w takiej czy innej sytuacji wykonanie zapytania czy modyfikacji danych trwa tak długo.

2.1. Strony i obszary

Najmniejszą jednostką przechowywania danych jest w SQL Server **strona** (ang. *page*). Jest to 8 KB blok składający się z nagłówka i 8060 bajtów na dane z wiersza (lub wierszy). Przy założeniu, że wiersz tabeli musi się zmieścić na stronie jasno widać, że maksymalny rozmiar wiersza to 8060 bajtów. Część danych o rozmiarze przekraczającym 8KB jest zapisywana na innych stronach, a w samym wierszu umieszczany jest tylko wskaźnik do pierwszej z tych stron. SQL Server rozróżnia 9 rodzajów stron przechowujących informacje o rozmaitym znaczeniu:

- Strony danych (data) zawierają wszystkie dane z wiersza, z wyjątkiem kolumn typów: text, ntext, image, nvarchar(max), varchar(max), varbinary(max), xml.
- Jeżeli wiersz nie mieści się w limicie długości 8060 bajtów, to najdłuższa z kolumn jest przenoszona do tzw. **strony przepełnienia** (strona danych), a w jej miejscu w wierszu zostaje 24 bajtowy wskaźnik.
- Strony indeksów (index) zawierają poszczególne wpisy indeksu. W ich przypadku istotny jest limit długości klucza indeksu – 900 bajtów.
- Strony obiektów BLOB/CLOB (Binary/Character Large Object) (text/image) służą do przechowywania danych o rozmiarze do 2 GB.
- Strony GAM, SGAM i IAM – wrócimy do nich w dalszej części wykładu, gdy poznamy kolejne pojęcia dotyczące fizycznego przechowywania danych.

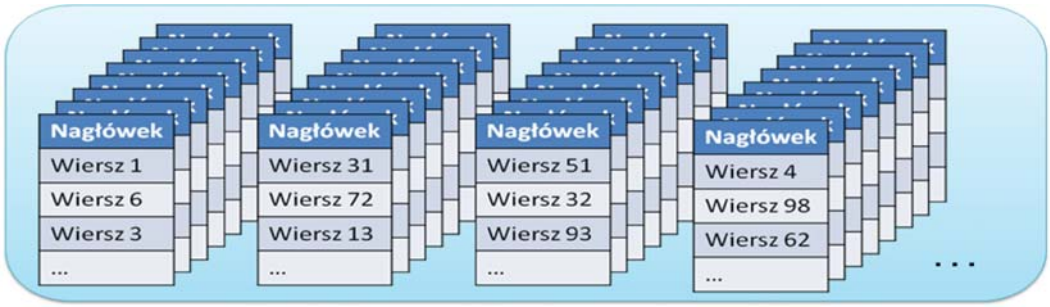
Wymieniliśmy tylko 6 rodzajów stron, żeby niepotrzebnie nie komplikować dalszych rozważań. Podstawową jednostką alokacji nie jest jednak w SQL Server strona, tylko zbiór ośmiu stron zwany **obszarem** (ang. *extent*) – patrz rys. 2.

Rys. 2. Obszar

Jest tak ze względu na fakt, iż 8 KB to trochę za mało jak na operacje w systemie plików, a 64 KB to akurat jednostka alokacji w systemie plików NTFS. Obszary mogą zawierać strony należące do jednego obiektu (tabeli czy indeksu) – nazywamy je wtedy jednolitymi (uniform), lub do wielu obiektów – stają się wtedy obszarami mieszanymi (mixed). Jeżeli SQL Server alokuje miejsce na nowe dane to najmniejszą jednostką jest właśnie obszar.

2.2. Sterty

Jeżeli tabela nie zawiera żadnego indeksu, to jej dane tworzą **stertę** – nieuporządkowaną listę stron należących do tej tabeli. Wszelkie operacje wyszukiwania na stercie odbywają się wolno, gdyż wymagają zawsze przejrzenia wszystkich stron. Inaczej w żaden sposób serwer nie jest w stanie stwierdzić, czy np. odnalazł już wszystkie wiersze zawierające dane klientów o nazwisku Kowalski. Stertę można wyobrazić sobie jak na rys. 3.



Rys. 3. Przykładowa sterta

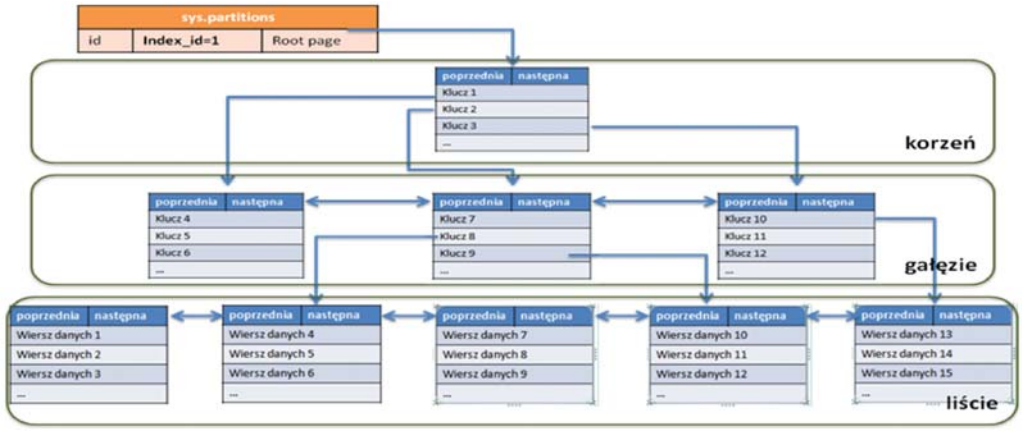
Gdy SQL Server alokuje miejsce w plikach bazy danych, wypełnia je obszarami, które wstępnie są oznaczone jako wolne. Podobnie wszystkie strony w obszarach są

oznaczone jako puste. W jaki sposób przechowywane są informacje na temat tego, czy dany obszar lub strona są wolne lub należą do jakiegoś obiektu? Służą do tego specjalne strony – GAM, SGAM i IAM. Zawierają one informacje o zajętości poszczególnych obszarów w postaci map bitowych (GAM, SGAM) lub o przynależności obszarów do obiektów (tabel, indeksów) – IAM.

3. INDEKSY

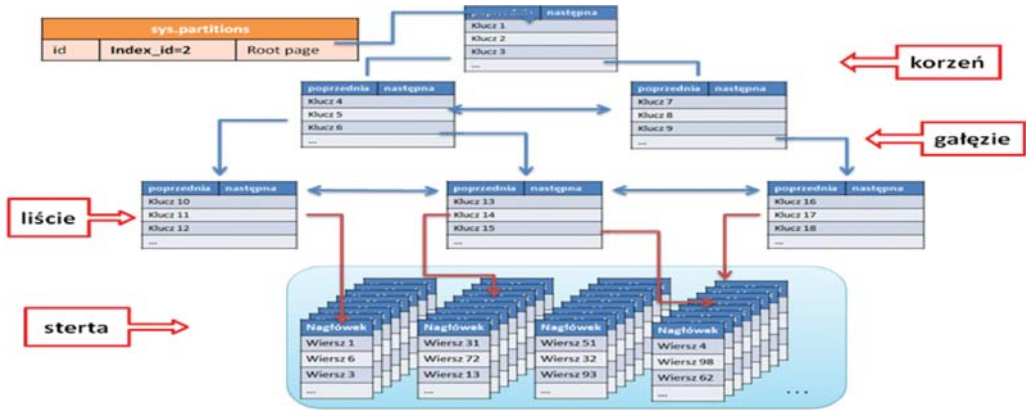
Poznaliśmy już w zarysie sposób przechowywania danych w tabeli, dla której nie stworzono indeksów. Cechą charakterystyczną był fakt nieuporządkowania stron i wierszy należących do jednej tabeli, co wymuszało przy każdej operacji wyszukiwania danych w tabeli przeszukanie wszystkich wierszy. Taka operacja nosi nazwę **skanowania tabeli** (ang. *table scan*). Jest ona bardzo kosztowna (w sensie zasobów) i wymaga częstego sięgania do danych z dysku, tym częściej im więcej danych znajduje się w tabeli. Taki mechanizm jest skrajnie nieefektywny. Operacji skanowania tabeli można uniknąć dzięki indeksom, występującym w dwóch podstawowych wariantach jako indeksy: **zgrupowane** (ang. *clustered*) i **niezgrupowane** (ang. *nonclustered*)

Indeks zgrupowany ma postać drzewa zrównoważonego (*B-tree*). Na poziomie korzenia i gałęzi znajdują się strony indeksu zawierające kolejne wartości klucza indeksu uporządkowane rosnąco. Na poziomie liści znajdują się podobnie uporządkowane strony z danymi tabeli. To właśnie jest cechą charakterystyczną indeksu zgrupowanego – powoduje on fizyczne uporządkowanie wierszy w tabeli, rosnąco według wartości klucza indeksu (wskazanej kolumny lub kolumn). Z tego względu oczywiste jest ograniczenie do jednego indeksu zgrupowanego dla tabeli.



Rys. 4. Indeks zgrupowany

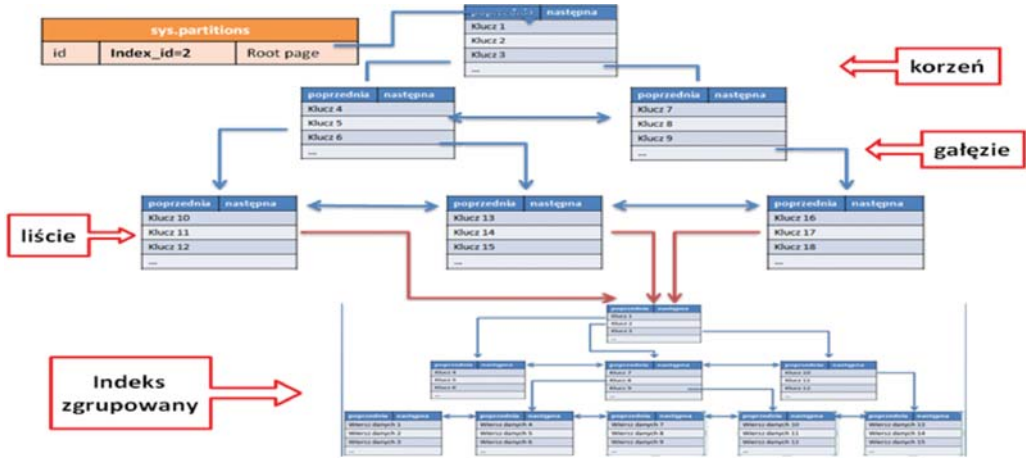
Specyfika indeksu zgrupowanego polega na fizycznym porządkowaniu danych z tabeli według wartości klucza indeksu. W związku z tym jasne jest, że indeks ten będzie szczególnie przydatny przy zapytaniach operujących na zakresach danych, grupujących dane, oraz korzystających z danych z wielu kolumn. W takich przypadkach indeks zgrupowany zapewnia znaczny wzrost wydajności w stosunku do sterty lub indeksu niezgrupowanego. Indeksy zgrupowane nie wyczerpują możliwości budowania tego typu struktur w SQL Serverze 2008. Drugim typem indeksów są **indeksy niezgrupowane**. Ich budowa odbiega nieco od budowy indeksu zgrupowanego, a do tego indeksy niezgrupowane mogą być tworzone w oparciu o stertę lub istniejący indeks zgrupowany. Dla jednej tabeli można utworzyć do 248 indeksów niezgrupowanych. Indeks niezgrupowany różni się od zgrupowanego przede wszystkim tym, że w swojej strukturze na poziomie liści ma także strony indeksu (a nie strony danych). W przypadku budowania indeksu niezgrupowanego na stercie, strony te oprócz wartości klucza indeksu zawierają wskaźniki do konkretnych stron na stercie, które dopiero zawierają odpowiednie dane.



Rys. 5. Indeksy niezgrupowane – na bazie sterty

Indeksy niezgrupowane mają strukturę zbliżoną do zgrupowanych. Zasadnicza różnica polega na zawartości liści indeksu. O ile indeksy zgrupowane mają w tym miejscu strony danych, to indeksy niezgrupowane – strony indeksu. Strony te zależnie od wariantu indeksu niezgrupowanego zawierają oprócz klucza różne informacje. Indeksy niezgrupowane mogą być tworzone w oparciu o stertę. Jest to możliwe tylko w przypadku, gdy tabela nie posiada indeksu zgrupowanego. W takim przypadku liście indeksu zawierają wskaźniki do konkretnych stron na stercie. Indeks niezgrupowany tworzony na tabeli zawierającej już indeks zgrupowany, jest tworzony nieco inaczej. Korzeń, gałęzie i liście zawierają strony indeksu, ale liście zamiast

wskaźników do stron na stercie zawierają wartości klucza indeksu zgrupowanego. Każde wyszukanie w oparciu o indeks niezgrupowany, po dojściu do poziomu liści, zaczyna dalsze przetwarzanie od korzenia indeksu zgrupowanego (wyszukiwany jest klucz zawarty w liściu).



Rys. 6. Indeksy niezgrupowane – na bazie indeksu zgrupowanego

W przypadku budowania indeksów niezgrupowanych, szczególnie na dużych tabelach, warto dobrze zaplanować tę czynność, szczególnie gdy planowane jest też utworzenie indeksu zgrupowanego. Niewzięcie tego pod uwagę może powodować konieczność przebudowywania indeksów niezgrupowanych w związku z dodaniem lub usunięciem indeksu zgrupowanego. W sporym uproszczeniu rola indeksów sprowadza się do ograniczenia liczby operacji wejścia/wyjścia niezbędnych do realizacji zapytania. SQL Server nie odczytuje poszczególnych obszarów potrzebnych do realizacji zapytania z dysku za każdym razem. Zawiera rozbudowany bufor pamięci podręcznej, do której trafiają kolejne odczytywane z dysku obszary. Ze względu na ograniczony rozmiar bufora, strony nieużywane lub używane rzadziej są zastępowane tymi, z których zapytania korzystają częściej.

Przy korzystaniu z indeksów niezgrupowanych istnieje jeszcze jedna możliwość dalszego ograniczania liczby operacji wejścia/wyjścia. Polega ona na tym, że do indeksu (dokładnie do stron liści indeksu) dodawane są dodatkowe kolumny. Jeżeli liście indeksu niezgrupowanego zawierają wszystkie kolumny zwracane przez zapytanie, to nie ma w ogóle konieczności sięgania do stron z danymi. W takim przypadku mamy do czynienia z tak zwanym **indeksem pokrywającym**. Dodawanie kolumn do indeksu niezgrupowanego może polegać na dodawaniu kolejnych kolumn do klucza (występuje tu ograniczenie do 16 kolumn w kluczu i 900 bajtów długości klucza),

albo na dodawaniu kolumn „niekluczowych" do indeksu (nie wliczają się one do długości klucza). Trzeba jednak pamiętać, że tworzenie indeksów pokrywających dla kolejnych zapytań nie prowadzi do niczego dobrego, gdyż po pierwsze rośnie ilość danych (wartości kolumn są przecież kopiowane do stron indeksu), a po drugie drastycznie spada wydajność modyfikowania danych (pociąga za sobą konieczność naniesienia zmian we wszystkich indeksach).

4. STATYSTYKI

Sam fakt istnienia takiego czy innego indeksu nie powoduje, że od razu staje się on kandydatem do skorzystania w ramach realizacji zapytania. W trakcie optymalizacji zapytania potrzebne są dodatkowe informacje na temat indeksów – **statystyki indeksów**. Sensowność skorzystania z indeksu można ocenić tylko w połączeniu z informacjami o liczbie wierszy w tabeli oraz o rozkładzie wystąpień poszczególnych wartości lub zakresów wartości w danych zawartych w kolumnie. Przykładowo mamy tabele klientów, w której 80% klientów nosi nazwisko Kowalski, a jedynie dwóch Nowak. Na podstawie samego faktu istnienia indeksu na kolumnie nazwisko trudno ocenić, czy sensownie jest go wykorzystać przy wyszukiwaniu Kowalskich lub Nowaków. Po przejrzeniu statystyk może okazać się, że dla Kowalskiego nie ma co zaprzątać sobie głowy indeksami, natomiast w przypadku Nowaka może to znacznie poprawić wydajność. Jako, że dane zawarte w tabelach zwykle się zmieniają (pojawiają się nowe, istniejące są modyfikowane lub usuwane), istotne jest także aktualizowanie statystyk. Optymalizator zapytań podejmujący decyzje na podstawie nieaktualnych statystyk działa jak pilot samolotu, któremu przyrządy pokładowe pokazują wskazania sprzed 5 minut. Skutki mogą być opłakane. Z tego powodu, jeżeli mamy do czynienia z sytuacją, gdy do tej pory zapytanie wykonywało się zadowalająco szybko, a nagle wydajność spadła, pierwszym krokiem do wykonania jest właśnie uaktualnienie statystyk. Warto o tym pamiętać, bo może to nam oszczędzić sporo czasu.

5. PLANY WYKONANIA

Gdy zlecamy serwerowi wykonanie zapytania rozpoczyna się dość złożony proces prowadzący do określenia sposobu realizacji zapytania. Zależnie od konstrukcji samego zapytania, rozmiarów tabel, istniejących indeksów, statystyk itp. serwer tworzy kilka planów wykonania zapytania. Następnie spośród nich wybierany jest ten, który cechuje się najniższym kosztem wykonania (wyrażanym przez koszt operacji wejścia/wyjścia oraz czasu procesora). Tak wybrany plan jest następnie kompilowany

(przetwarzany na postać gotową do wykonania przez silnik bazodanowy) i przechowywany w buforze w razie gdyby mógł się przydać przy kolejnym wykonaniu podobnego zapytania. W ramach tego punktu zajmiemy się nieco dokładniej procesem wykonania zapytania przez SQL Server.

Cały proces, przebiegający od momentu przekazania zapytania do wykonania do odebrania jego rezultatów, jest dość złożony i może stanowić temat niejednego wykładu. Postaramy się choć z grubsza zasygnalizować najistotniejsze etapy tego procesu.

- **Parsowanie zapytania** – polega na zweryfikowaniu składni polecenia, wychwyceniu błędów i nieprawidłowości w jego strukturze. Jeżeli takie błędy nie występują, to efektem parsowania jest tak zwane **drzewo zapytania** (postać przeznaczona do dalszej obróbki).
- **Standaryzacja zapytania**. Na tym etapie drzewo zapytania jest doprowadzane do postaci standardowej – usuwana jest ewentualna nadmiarowość, standaryzowana jest postać podzapytań itp. Efektem tego etapu jest ustandaryzowane drzewo zapytania.
- **Optymalizacja zapytania**. Polega na wygenerowaniu kilku planów wykonania zapytania oraz przeprowadzeniu ich analizy kosztowej zakończonej wybraniem najtańszego planu wykonania.
- **Kompilacja** polega na przetłumaczeniu wybranego planu wykonania do postaci kodu wykonywalnego przez silnik bazodanowy.
- **Określenie metod fizycznego dostępu do danych** (skanowanie tabel, skanowanie indeksów, wyszukiwanie w indeksach itp.).

Proces optymalizacji zapytania składa się z kilku etapów. W ich skład wchodzą: analizowanie zapytania pod kątem kryteriów wyszukiwania i złączeń, dobranie indeksów mogących wspomóc wykonanie zapytania, oraz określenie sposobów realizacji złączeń. W ramach realizacji poszczególnych etapów optymalizator zapytań może korzystać z istniejących statystyk indeksów, generować je dla wybranych indeksów lub wręcz tworzyć nowe indeksy na potrzeby wykonania zapytania. Efektem tego procesu jest plan wykonania o najniższym koszcie, który jest następnie przekazywany do kompilacji i wykonania. Plan wykonania dla zapytania można podejrzeć w formie tekstowej, XML bądź zbioru wierszy. Opcja prezentacji graficznej postaci planu wykonania dla zapytania jest dostępna w dwóch wariantach: *Estimated Execution Plan* oraz *Actual Execution Plan*. Pierwszy z nich polega na wygenerowaniu planu wykonania dla zapytania bez jego wykonywania. Powoduje to, że część informacji w planie wykonania jest szacunkowa lub jej brakuje (np. liczba wierszy poddanych operacjom, liczba wątków zaangażowanych w wykonanie itp.). Zaletą tego wariantu

jest na pewno szybkość działania. Jest to szczególnie odczuwalne przy zapytaniach, które wykonują się dłużej niż kilkanaście sekund. Drugi wariant zawiera pełne dane na temat wykonania zapytania. Jest on zawsze wiarygodny i mamy gwarancję, że dokładnie tak zostało wykonane zapytanie. W praktyce lepiej jest pracować z faktycznymi planami wykonania, chyba, że czas potrzebny ich uzyskanie jest przeszkodą. Na diagramach reprezentujących plany wykonania zapytań może znajdować się kilkadziesiąt różnych symboli graficznych reprezentujących różne operatory (logiczne i fizyczne) oraz przebieg wykonania zapytania. Nie sposób omówić ich choćby pobieżnie w ramach tego wykładu.

Wśród całej gamy informacji wyświetlanych w szczegółach wybranego operatora, dla nas najistotniejsze są te, związane z kosztem wykonania danego etapu. W dalszych przykładach będziemy się na nich opierać prezentując zmiany kosztu wykonania zapytania w zależności od podjętych kroków przy optymalizacji zapytania.

6. NARZĘDZIA WSPOMAGAJĄCE OPTYMALIZACJĘ

Przy optymalizowaniu zapytań trzeba brać pod uwagę wiele czynników. Jeśli dodać do tego pracę z wieloma zapytaniami, to szybko wyłania się obraz ogromu pracy do wykonania. Na szczęście istnieją narzędzia, które mogą choć trochę wspomóc nasze wysiłki. Narzędzie Database Engine Tuning Advisor jest w stanie wygenerować i wykonać wiele czynności prowadzących do podniesienia wydajności bazy danych. Proces ten jest realizowany rzecz jasna w kontekście konkretnych zapytań, gdyż nie ma możliwości optymalizowania pod kątem dowolnych zapytań. Punktem wejścia do procesu automatycznej optymalizacji jest określenie zapytań, które są wykonywane na bazie wraz z określeniem częstotliwości ich wykonywania. Najłatwiej zrobić to w ramach monitorowania działania aplikacji. Za pomocą narzędzia SQL Profiler można zebrać tzw. **ślad** zawierający informacje o wszystkich wykonywanych na bazie zapytaniach. Plik z takimi informacjami może stanowić dane wejściowe dla Database Engine Tuning Advisora. Na ich podstawie narzędzie jest w stanie określić zapytania najistotniejsze dla funkcjonowania aplikacji i skupić się na optymalizowaniu pod ich kątem. Narzędzie zawiera wiele opcji umożliwiających sterowanie procesem optymalizacji. Można na przykład określić zbiór mechanizmów, które mają być wykorzystane do zwiększenia wydajności (indeksy, widoki indeksowane itp.). Można również określić, czy optymalizacja ma pozostawić istniejące indeksy bez zmian, czy zaplanować wszystkie od początku. Rezultatem pracy narzędzia jest lista poleceń do wykonania na bazie danych (służą one do tworzenia zaplanowanych indeksów, usuwania niepotrzebnych itp.). Istotne jest, że przedstawiony przez narzędzie plan z reguły przyczynia się do podniesienia wydajności. Często można na tym

zakończyć dalsze prace. Jeśli jednak mamy więcej pomysłów na zwiększenie wydajności, to wynik prac narzędzia zawsze można traktować jako dobry punkt wyjścia do dalszej analizy prowadzonej już „ręcznie".

W ramach tego wykładu zaledwie rozpoczęliśmy omawianie zagadnień związanych z optymalizacją zapytań i optymalizacją wydajności SQL Servera jako taką. Celem było przedstawienie pewnych podstawowych zagadnień i mechanizmów niezbędnych do zrozumienia podstawowych zasad rządzących w dziedzinie sposobów realizacji zapytań przez SQL Server

Literatura

1. K. Delany: *MS SQL Server 2005 od środka: Dostrajanie i optymalizacja zapytań*, APN PROMISE, Warszawa 2008
2. T. Rizzo, A. Machanic, R. Dewson, R. Walters, J. Sack, J. Skinner: *SQL Server 2005*, WNT, Warszawa 2008
3. R. Vieira: *SQL Server 2005. Programowanie. Od Podstaw,* Helion, Gliwice 2007
4. I. Ben-Gan, L. Kollar, D. Sarka: *MS SQL Server 2005 od środka: Zapytania w języku T-SQL,* APN PROMISE, Warszawa 2006