LITERATURA

1. Liberty J., Kraley M., *XML od podstaw*, Translator, Warszawa 2001
2. Rizzo T., Machanic A., Dewson R., Walters R., Sack J., Skin J., *SQL Server 2005*, WNT, Warszawa 2008
3. Vieira R., *SQL Server 2005. Programowanie. Od Podstaw,* Helion, Gliwice 2007
4. Walmsley P., *Wszystko o XML Schema*, Helion, Gliwice 2007

Optymalizacja zapytań SQL

Andrzej Ptasznik
Warszawska Wyższa Szkoła Informatyki
aptaszni@wwsi.edu.pl

Streszczenie

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 będą problemy wyboru strategii wykonania zapytania w zależności od zawartości tabel i zdefiniowanych indeksów. Wykład wprowadzi pojęcie statystyk indeksów i ich znaczenie przy wyborze strategii realizacji zapytania.

Spis treści

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.

2 MODEL OPTYMALIZACJI WYDAJNOŚCI BAZY DANYCH

W świecie systemów informatycznych i komputerów od wielu lat utrzymuje się stały trend wzrostu mocy obliczeniowej, pojemności pamięci operacyjnej, pojemności i szybkości dysków twardych itp. W związku z tym, jeśli mamy do czynienia ze zbyt niską wydajnością bazy danych, to pierwszym pomysłem może być rozbudowa systemu od strony sprzętowej – a nuż, ten dodatkowy procesor lub 4 GB pamięci dodadzą bazie skrzydeł. Niestety nie zawsze to działa, lub przewidywane koszty rozbudowy są zdecydowanie nieakceptowalne. Osiągnięty efekt może także nie być zbyt długotrwały i po kolejnym miesiącu uzupełniania danych w bazie wracamy do punktu wyjścia – działa za wolno!

W takiej sytuacji warto zrobić to, od czego tak naprawdę należało zacząć – przeanalizować bazę danych pod kątem możliwości optymalizacji jej wydajności. Okazuje się, że tą drogą można otrzymać bardzo dobre rezultaty. Niestety wymaga to znacznej wiedzy i umiejętności, a także sporej dozy wyczucia, którego ot tak nie da się nauczyć. 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 rysunku 1.

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.

Pierwszym i najistotniejszym zadaniem jest jednak optymalizacja struktury bazy danych. Osiąga się ją zwykle poprzez normalizację (doprowadzenie do trzeciej postaci normalnej). Taka postać cechuje się większą liczbą tabel, krótszymi rekordami w tabelach, mniejszą podatnością na blokowanie, łatwiejszym tworzeniem zapytań bazujących na zbiorach itp. Czas poświęcony na tym etapie zwraca się bardzo szybko, podobnie jak błędy tu popełnione okrutnie mszczą się przy dalszych próbach optymalizacji wydajności.

Rysunek 1.
Model procesu optymalizacji

Bardzo istotną rzeczą jest pamiętanie o tym, że baza nie istnieje sama dla siebie. Z reguły współpracuje z jakąś aplikacją lub aplikacjami. Jakiekolwiek modyfikacje struktury bazy danych mogą wiązać się z koniecznością wprowadzania modyfikacji kodu aplikacji, a to nie jest już tak miłe. Z tego względu zalecane jest podejście zakładające utworzenie w bazie danych „warstwy abstrakcji danych", której rola polega na odcięciu aplikacji od szczegółów struktury bazy danych. Zwykle realizowane jest to za pomocą widoków, procedur składowanych czy funkcji użytkownika. Aplikacje „widzą" i korzystają tylko z tych obiektów, nie kontaktując się bezpośrednio z tabelami. Dzięki temu, w przypadku modyfikowania struktury bazy, można ukryć ten fakt przed aplikacjami – wystarczy zmodyfikować kod procedury czy widoku, aby pasował do nowej struktury tabel, a aplikacje jak z nich korzystały tak będą korzystać – zupełnie nieświadome, że dane pobierane wcześniej z dwóch tabel obecnie są rozrzucone po pięciu tabelach.

Kolejnym etapem jest optymalizacja zapytań. Głównym zagadnieniem jest tu oderwanie się od starych nawyków pisania zapytań iteracyjnych (często korzystając z kursorów) na rzecz zapytań bazujących na zbiorach. Są one bardziej wydajne oraz łatwiej skalowalne. Indeksy łączą się z poprzednim etapem pełniąc rolę pomostu pomiędzy zapytaniem a danymi. Dobrze napisane zapytania z odpowiednio dobranymi indeksami potrafią czynić cuda, a z drugiej strony żaden indeks nie naprawi kardynalnych błędów w zapytaniach czy strukturze danych. Z kolei kwestie blokad wiążą się nierozerwalnie z korzystaniem z bazy przez wielu użytkowników jednocześnie. Często zdarza się, że pozornie wydajnie działająca baza szybko traci wigor przy kolejnych jednocześnie pracujących użytkownikach. Jakiekolwiek próby na tym poziomie nie dadzą nic, jeżeli na poprzednich etapach przeoczyliśmy jakieś problemy.

Ostatni poziom to wspominane już wcześniej „rozszerzanie" serwera. Zwiększanie mocy procesora/ów, ilości pamięci czy szybkości i pojemności dysków umożliwiają osiągnięcie szybkiego efektu wzrostu wydajności. Nie pomoże to jednak w przypadku błędów popełnionych na poprzednich etapach i efekt końcowy może być mizerny, szczególnie wziąwszy pod uwagę koszty.

3 FIZYCZNA ORGANIZACJA DANYCH W SQL SERVER 2008

Skoro przekonaliśmy się już co do konieczności zwrócenia uwagi na zagadnienia w ramach optymalizowania wydajności, to możemy przejść do rzeczy i rozpocząć zgłębianie tej dziedziny. Nie uda się nam to w żaden sposób, jeśli nie zrozumiemy mechanizmów leżących u podstaw działania SQL Servera. Jednym z istotnych zagadnień jest tu sposób, w jaki dane są fizycznie przechowywane w bazie danych. Gdy myślimy o tabeli, to od razu przedstawiamy sobie coś na kształt zbioru wierszy składających się z kolumn zawierających dane różnego typu (patrz rys. 2).

ContactID	Title	FirstName	LastName	EmailAddress	Phone
1	Mr.	Gustavo	Achong	gustavo0@adventure-works.com	398-555-0132
2	Ms.	Catherine	Abel	catherine0@adventure-works.com	747-555-0171
3	Ms.	Kim	Abercrombie	kim2@adventure-works.com	334-555-0137
4	Sr.	Humberto	Acevedo	humberto0@adventure-works.com	599-555-0127

Rysunek 2.
Tabela w bazie danych

Nie zastanawiamy się, jak te dane są przechowywane fizycznie na dysku ani jaki wpływ na wydajność mogą mieć nasze decyzje podjęte przy projektowaniu tabeli. Warto jednak zadać sobie nieco trudu i zapoznać się z fizycznym sposobem przechowywania danych w bazie. Zrozumienie podstaw ułatwi później wyjaśnienie, dlaczego w takiej czy innej sytuacji wykonanie zapytania czy modyfikacji danych trwa tak długo.

Strony i obszary

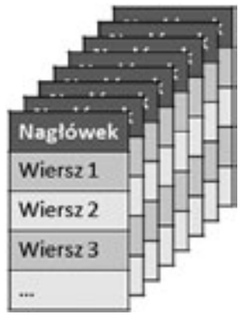
Najmniejszą jednostką przechowywania danych jest w SQL Serverze **strona** (ang. *page*). Jest to 8 KB blok składający się z nagłówka i 8060 bajtów na dane z wiersza (lub wierszy). Przy założeniu, że wiersz tabeli musi się zmieścić na stronie jasno widać, że maksymalny rozmiar wiersza to 8060 bajtów. Trochę mało? Niekoniecznie. Część danych o rozmiarze przekraczającym 8 KB 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 (ang. *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 (ang. *index*) zawierają poszczególne wpisy indeksu. W ich przypadku istotny jest limit długości klucza indeksu – 900 bajtów.
- Strony obiektów BLOB/CLOB (ang. *Binary/Character Large Object*) (ang. *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. Dla uporządkowania warto wspomnieć o pozostałych trzech: Page Free Space, Bulk Changed Map, Differential Changed Map. Pierwsza zawiera informacje o zaalokowanych stronach i wolnym miejscu na nich. Pozostałe dwa rodza-

je są wykorzystywane do oznaczania danych zmodyfikowanych w ramach operacji typu bulk oraz do oznaczania zmian od ostatnio wykonanej kopii zapasowej.

Podstawową jednostką alokacji nie jest jednak w SQL Serverze strona, tylko zbiór ośmiu stron zwany **obszarem** (ang. *extent*) – rysunek 3.

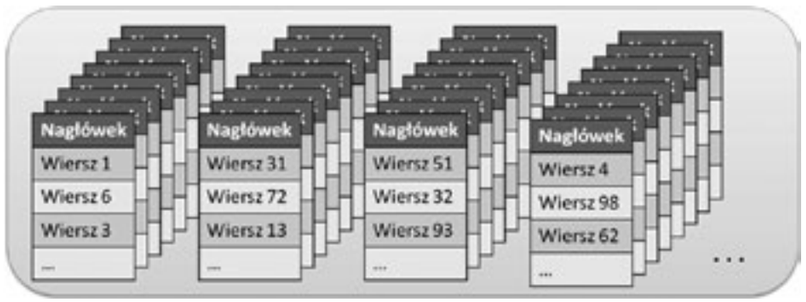


Rysunek 3.
Obszar

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** (ang. *uniform*), lub do wielu obiektów – stają się wtedy **obszarami mieszanymi** (ang. *mixed*). Jeżeli SQL Server alokuje miejsce na nowe dane, to najmniejszą jednostką jest właśnie obszar.

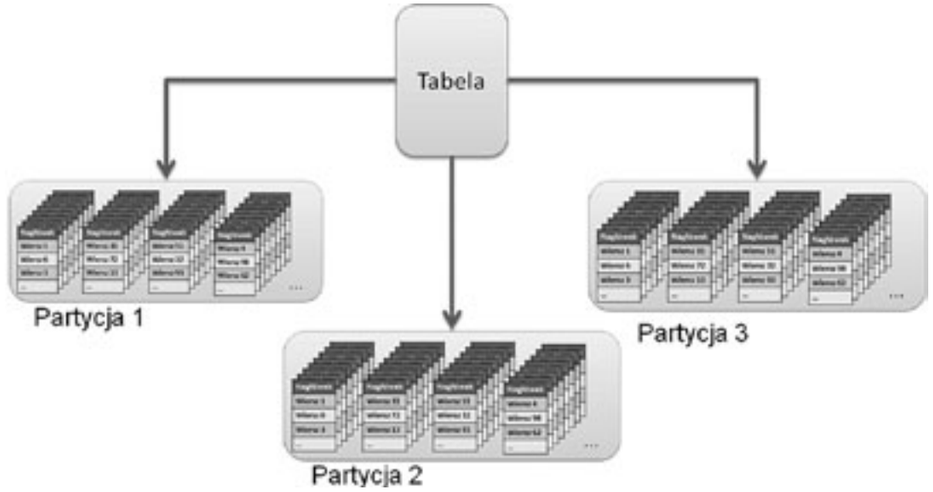
Sterty

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 rysunku 4.



Rysunek 4.
Przykładowa sterta

Dodatkowo tabela może zostać podzielona na partycje (względy wydajnościowe – zrównoleglenie operacji wejścia/wyjścia). W takim przypadku każda z partycji zawiera własną stertę. Wszystkie razem tworzą zbiór danych tabeli (rys. 5).



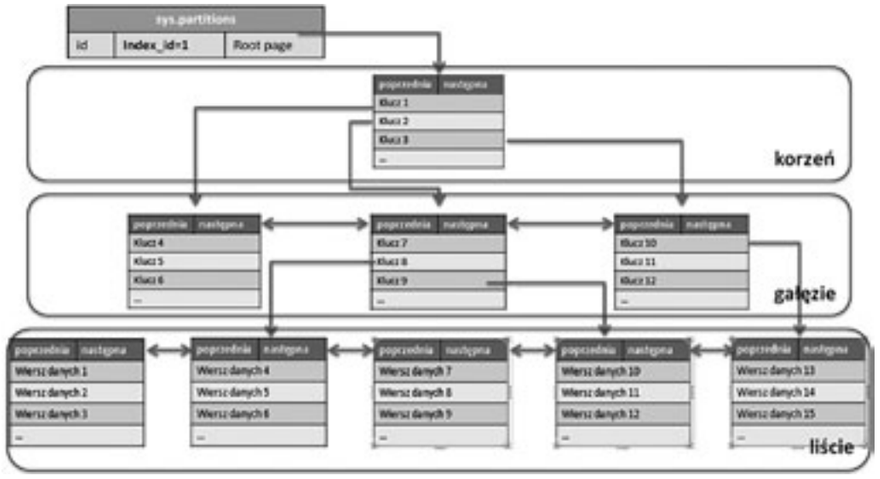
Rysunek 5.
Tabela, partycje, sterty

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. Kluczem do uzyskania dostępu do danych z tabeli jest możliwość dostania się do strony IAM tej tabeli. Informacje na temat lokalizacji stron IAM dla poszczególnych obiektów znajdują się we wpisach w tabelach systemowych. Jako że nie zaleca się „szperania" bezpośrednio w tych tabelach, zostały udostępnione specjalne widoki, które zawierają potrzebne nam dane. W przypadku stron IAM jest to widok sys.partitions. Wpisy w nim zawarte składają się m.in. z kolumny index_id określającej rodzaj obiektu (sterta, indeks zgrupowany, indeks niezgrupowany, obiekty LOB), kolumn wskazujących id obiektu i partycji oraz wskaźnika do strony IAM obiektu.

Indeksy zgrupowane i niezgrupowane

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, więc muszą istnieć jakieś inne, bardziej efektywne mechanizmy wyszukiwania. Rzeczywiście istnieją – są to **indeksy**, występujące w dwóch podstawowych wariantach jako indeksy: **zgrupowane** (ang. *clustered*) i **niezgrupowane** (ang. *nonclustered*)

Indeks zgrupowany ma postać drzewa zrównoważonego (ang. *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.



Rysunek 6.
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.

Istotną rzeczą przy podejmowaniu decyzji o utworzeniu indeksu zgrupowanego jest wybranie właściwej kolumny (kolumn). Długość klucza powinna być jak najmniejsza, co umożliwia zmieszczenie większej liczby wpisów indeksu na jednej stronie, co z kolei przenosi się na zmniejszenie liczby stron całości indeksu i w efekcie mniej operacji wejścia/wyjścia do wykonania przez serwer. Żeby indeks zgrupowany korzystnie wpływał na wydajność przy dodawaniu nowych wierszy do mocno wykorzystywanej tabeli, klucz powinien przyjmować dla kolejnych wpisów wartości rosnące (zwykle stosowana jest tu kolumna z cechą *identity*). Indeks daje duży zysk wydajności, gdy jego klucz jest możliwie wysoko selektywny (co oznacza mniejszą liczbę kluczy o tej samej wartości – duplikatów). Istotny jest także fakt, że kolumny klucza indeksu zgrupowanego nie powinny być raczej modyfikowane, gdyż pociąga to za sobą konieczność modyfikowania nie tylko stron indeksu, ale także porządkowania stron danych.

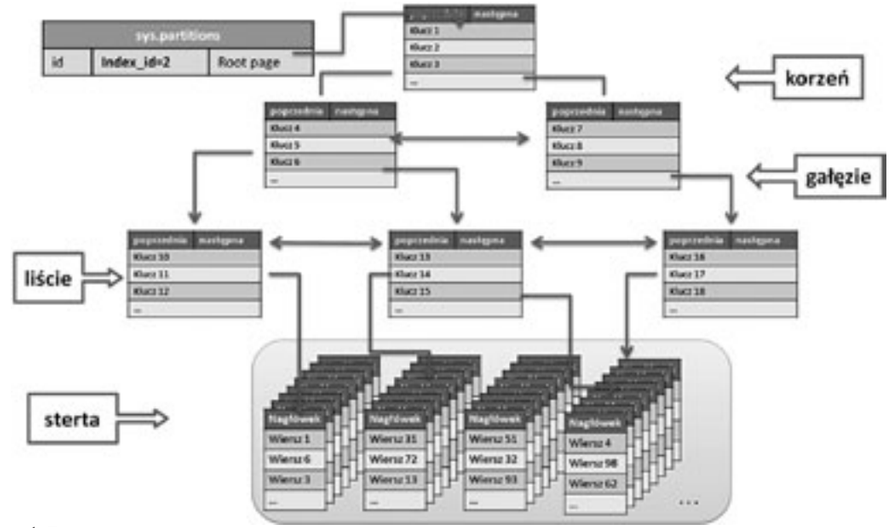
Indeksy zgrupowane nie wyczerpują możliwości budowania tego typu struktur w SQL Serverze 2008. Drugim typem indeksów są **indeksy niezgrupowane**. Ich budowa odbiega nieco od budowy indeksu zgrupowanego, a do tego indeksy niezgrupowane mogą być tworzone na bazie sterty lub istniejącego indeksu zgrupowanego. 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.

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 wtedy, gdy

tabela nie ma 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).

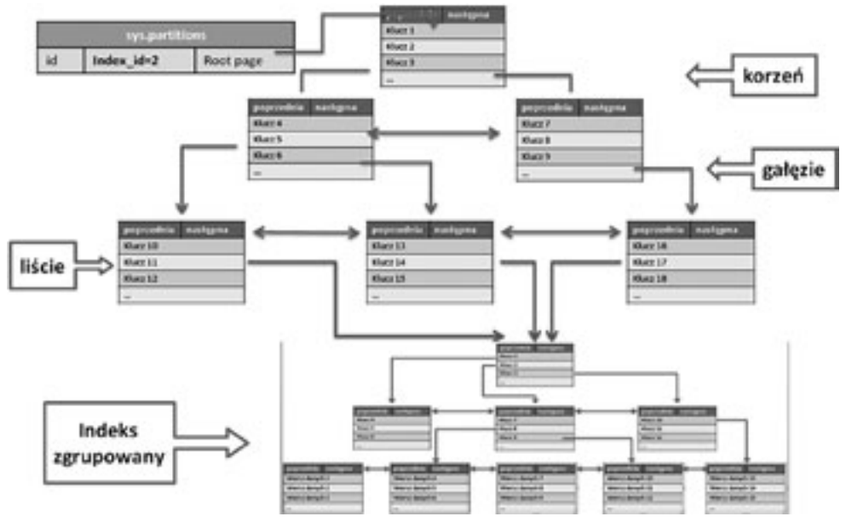


Rysunek 7.
Indeksy niezgrupowane – sterty w liściach

W przypadku budowania indeksów niezgrupowanych, szczególnie przy 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 liczba 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).



Rysunek 8.
Indeksy niezgrupowane – indeksy zgrupowane w liściach

Indeksy pokrywające

Żeby zademonstrować sposób działania indeksów pokrywających, załóżmy następującą sytuację. W bazie istnieje tabela zawierająca dane klientów. W jej skład wchodzi kilka kolumn: ID, Nazwisko, Imie, Email, DataOstatniegoZamowienia. Na tabeli został stworzony indeks zgrupowany na kolumnie ID oraz indeks niezgrupowany na kolumnie Nazwisko. Jeżeli w takim przypadku realizowane będzie zapytanie, które co prawda w klauzuli WHERE zawiera warunek tylko dla kolumny Nazwisko (zawartej w indeksie niezgrupowanym), ale na liście kolumn wyjściowych zawiera także inne kolumny (w naszym przypadku kolumna Email), to indeks niezgrupowany nie zostanie wykorzystany, gdyż wartości kolumn spoza indeksu muszą zostać pobrane ze stron danych. Zapytanie zostanie zrealizowane poprzez skanowanie indeksu zgrupowanego. Jeżeli usuniemy z listy kolumn wyjściowych kolumnę Email i wykonamy zapytanie ponownie, to tym razem indeks niezgrupowany okaże się przydatny i zostanie na nim wykonana operacja wyszukiwania w indeksie (ang. *index seek*). Będzie ona mniej kosztowna od skanowania indeksu zgrupowanego, gdyż nie wymaga dostępu do stron danych. Żeby osiągnąć ten sam efekt z kolumną Email na liście wyjściowej należy dodać ją do indeksu niezgrupowanego (jako część klucza lub nie). Po takiej modyfikacji osiągniemy założony cel – zapytanie zostanie zrealizowane z wykorzystaniem operacji wyszukiwania w indeksie niezgrupowanym.

Mechanizm indeksów pokrywających wygląda bardzo fajnie i nie jest trudny w zastosowaniu. Jest jednak druga strona medalu. Zwykle zapytań jest więcej niż jedno i zwracają więcej kolumn. Rozbudowywanie indeksów (zarówno ich liczba, jak i liczba kolumn w nich zawartych) prowadzi do znacznego wzrostu rozmiaru bazy danych oraz spadku wydajności przy modyfikowaniu danych z tej tabeli. W skrajnych przypadkach tworzymy przecież kopie poszczególnych wierszy na stronach indeksu, a co za tym idzie liczba operacji wejścia/wyjścia staje się zbliżona do tej potrzebnej do skanowania tabeli czy indeksu zgrupowanego.

4 PLANY WYKONANIA ZAPYTANIA

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 mógłby 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 i 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.** To przetłumaczenie wybranego planu wykonania do postaci kodu wykonywalnego przez silnik bazodanowy.
- **Określenie metod fizycznego dostępu do danych.** To 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. Realizuje się to za pomocą ustawienia na „ON” jednej z opcji SHOWPLAN_TEXT, SHOWPLAN_XML, SHOWPLAN_ALL. SQL Server, a właściwie narzędzie SQL Server Management Studio, umożliwia podejrzenie graficznej reprezentacji planu wykonania dla zapytania

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 na 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.

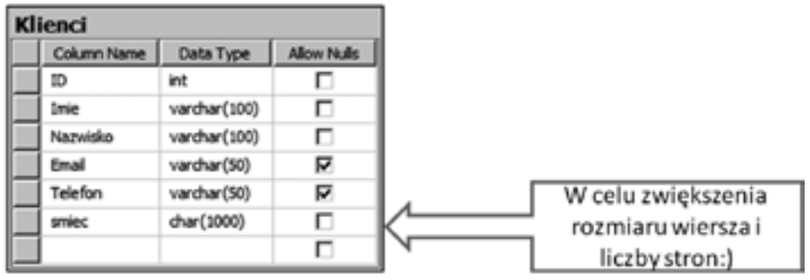
5 STATYSTYKI

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.

Ponieważ 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.

6 OPTYMALIZACJA PRZYKŁADOWEGO ZAPYTANIA

Przejdźmy teraz do kilku przykładów wykonywania zapytań przy różnych kombinacjach istniejących indeksów. Za każdym razem spróbujemy przyjrzeć się kosztom wykonania zapytania i szczegółom przyjętych planów wykonania. Na kolejnych przykładach postaramy się zademonstrować wpływ indeksów na plan wykonania zapytania i jego całkowity koszt. Zapytania będą dotyczyły tabeli Klienci (rys. 9), w której nie ma żadnego indeksu.

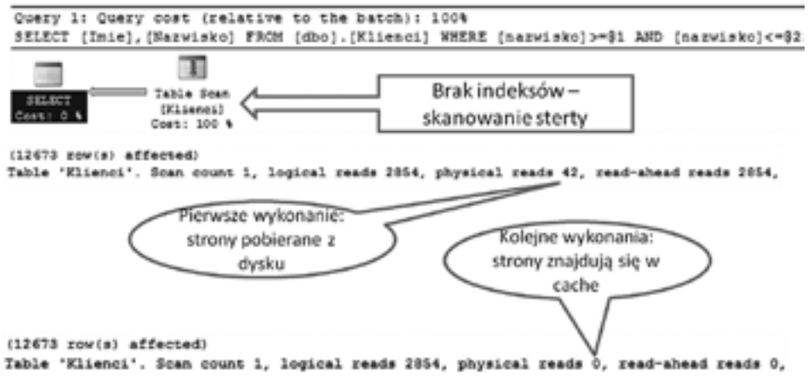


Rysunek 9.
Przykładowa tabela – Klienci

Z tego powodu można przewidywać, że operacją wykorzystaną do realizacji zapytania będzie skanowanie tabeli. Przykładowe zapytanie ma postać:

```
SELECT
  Imie
 ,Nazwisko
FROM
  dbo.Klienci
WHERE
  nazwisko BETWEEN 'F' AND 'T'
```

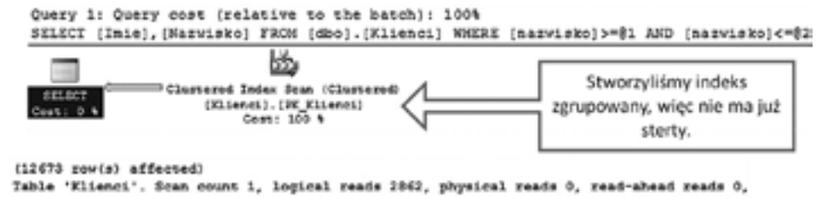
Po dwukrotnym wykonaniu zapytania uzyskaliśmy rezultaty, jak na rysunku 10.



Rysunek 10.
Efekty wykonania przykładowego zapytania

Zgodnie z oczekiwaniami, do realizacji zapytania została wykorzystana operacja skanowania tabeli. Przy pierwszym wykonaniu konieczne było pobranie stron danych z dysku (liczba fizycznych odczytów większa od 0). Każde następne wykonanie korzysta już ze stron umieszczonych w pamięci cache, czego przejawem jest zerowa wartość fizycznych odczytów. Całkowity koszt zapytania realizowanego według tego planu jest równy 2,1385.

Pierwszym etapem naszych działań jest utworzenie indeksu zgrupowanego na kolumnie ID. Nie przyczyni się to w znaczącym stopniu do zwiększenia wydajności, ale spowoduje zmianę planu wykonania. Skoro utworzenie indeksu zgrupowanego powoduje fizyczne uporządkowanie stron danych (i likwidacje sterty), to plan wykonania powinien zawierać wykonanie innej operacji niż skanowanie tabeli. Po wykonaniu zapytania stwierdzamy, że faktycznie tak jest (patrz rysunek 11).



Rysunek 11.
Efekty wykonania przykładowego zapytania po utworzeniu indeksu zgrupowanego

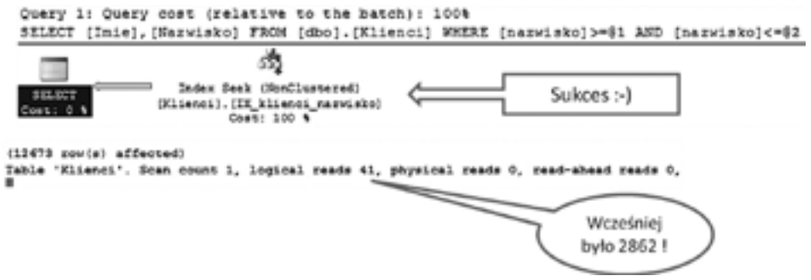
Tym razem serwer skorzystał z operacji skanowania indeksu zgrupowanego. Nie jest to żaden skok wydajnościowy, bo i tak przejrzane muszą być wszystkie strony danych, gdyż nasze kryterium wyszukiwania nie jest kolumną zawartą w indeksie.

Rozpocznijmy teraz działania ukierunkowane na obniżenie kosztów realizacji zapytania. Pierwszy pomysł – stwórzmy indeks niezgrupowany na kolumnie Nazwisko, która jest wykorzystywana jako kryterium wyszukiwania. Powinno to spowodować wykorzystanie tego indeksu do wyszukania wierszy z nazwiskami z określonego przez nas zakresu. Niestety po wykonaniu zapytania doznaliśmy rozczarowania – patrz rysunek 12.

Rysunek 12.
Efekt utworzenia indeksu niezgrupowanego

Plan wykonania się nie zmienił! Dlaczego? Z powodu umieszczenia na liście kolumn wyjściowych kolumny Imie. Optymalizator zapytań stwierdził, że mimo istnienia indeksu niezgrupowanego na kolumnie, po której wyszukujemy, nie warto z niego korzystać, gdyż i tak trzeba sięgnąć do stron danych, żeby pobrać wartości kolumny Imie. Z tego powodu plan wykonania nie uległ zmianie.

Skoro przemyśleliśmy już mechanizm działania zapytania i role indeksu, doprowadźmy sprawę do końca. Usuńmy istniejący indeks niezgrupowany, utwórzmy go na nowo z dodaną kolumną Imie. Po wykonaniu zapytania po raz kolejny, okazuje się, że tym razem indeks został wykorzystany (patrz rys. 13).



Rysunek 13.
Przykład wykorzystania indeksu

Odpowiednie wiersze spełniające kryteria wyszukiwania zostały zlokalizowane bardzo łatwo dzięki indeksowi niezgrupowanemu. Dodatkowo nie było konieczności sięgania do stron danych, gdyż indeks zawierał także kolumnę Imie, która była potrzebna do realizacji zapytania. Efekt jest widoczny. Koszt realizacji zapytania spadł z 2,1385 do 0,0453!

Po zakończeniu „walki" z optymalizacją prostego zapytania przez utworzenie odpowiednich indeksów można zdać sobie sprawę, iż wykonywanie tego typu operacji na prawdziwych bazach danych jest procesem złożonym i żmudnym. Do tego często nie da się pogodzić ze sobą wydajności dwóch lub więcej zapytań, bo każda poprawa wydajności w jednym psuje wydajność drugiego. Dodatkowo każdy kolejny indeks to dodatkowy problem z jego utrzymaniem oraz więcej czynności do wykonania przy modyfikacji danych. Jak sobie z tym radzić? Nie ma jednej sprawdzonej i zawsze działającej recepty. Są pewne podejścia umożliwiające realizację czynności w określonym porządku, co może się przyczynić do uniknięcia błędów lub ułatwienia wychwycenia typowych problemów. Zawsze jednak optymalizowanie wydajności pozostanie po części sztuką :-).

7 NARZĘDZIA WSPOMAGAJĄCE OPTYMALIZACJĘ

Przy optymalizowaniu zapytań trzeba brać pod uwagę mnóstwo 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 „zaorać" je i zaplanować wszystkie od początku.

Rezultatem pracy narzędzia jest lista poleceń do wykonania na bazie danych (służą one do tworzenia zaplanowanych indeksów, usuwania niepotrzebnych itp.). To co jest istotne, to fakt, 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. Ben-Gan I., Kollar L., Sarka D., *MS SQL Server 2005 od środka. Zapytania w języku T-SQL*, APN PROMISE, Warszawa 2006
2. Delany K., *MS SQL Server 2005 od środka. Dostrajanie i optymalizacja zapytań*, APN PROMISE, Warszawa 2008
3. Rizzo T., Machanic A., Dewson R., Walters R., Sack J., Skin J., *SQL Server 2005*, WNT, Warszawa 2008
4. Vieira R., *SQL Server 2005. Programowanie. Od Podstaw,* Helion, Gliwice 2007