Benjamin Nevarez

Microsoft SQL Server 2014

Optymalizacja zapytań

Wrzuć piąty bieg!

Helion

Tytuł oryginału: Microsoft® SQL Server® 2014 Query Tuning & Optimization

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-283-1162-6

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/sql14o.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/sql14o
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 1

Wprowadzenie do optymalizacji zapytań

W tym rozdziale:

- **Architektura**
- **Plany wykonania**
- `SET STATISTICS TIME` i `SET STATISTICS IO`
- **Podsumowanie**

Wszyscy to przeżyliśmy: nagle dostajesz telefon z informacją o awarii aplikacji i prośbą o pilne przyłączenie się do konferencji. Po połączeniu dowiadujesz się, że aplikacja jest tak wolna, iż firma nie jest w stanie spełniać celów biznesowych, traci pieniądze i być może także klientów. Zazwyczaj też nikt nie jest w stanie zapewnić żadnych dodatkowych informacji, które mogłyby pomóc w zlokalizowaniu problemu. Co zatem powinieneś zrobić? Gdzie zacząć? A po zlokalizowaniu i naprawieniu problemu co zrobić, aby taka sytuacja nie powtórzyła się w przyszłości?

Chociaż awaria może powstać z wielu różnych powodów, włączając w to problemy ze sprzętem i z systemem operacyjnym, jako specjalista baz danych powinieneś być w stanie odpowiednio dostosować i zoptymalizować swoje bazy tak, abyś mógł szybko zlokalizować ewentualne problemy. Ta książka zapewni Ci wiedzę i narzędzia do tego potrzebne. Skupiając się na wydajności bazy SQL Servera, a w szczególności na optymalizacji i dostosowaniu zapytań, książka ta pomoże Ci, po pierwsze, dzięki optymalizacji bazy danych uniknąć problemów z wydajnością, a po drugie, szybko znaleźć i naprawić problemy, które mimo wszystko mogą wystąpić.

Jednym z najlepszych sposobów na poprawienie wydajności baz danych jest nie tylko praca z technologią, ale również zrozumienie, jak działa technologia, co można dzięki niej uzyskać, jak najlepiej ją wykorzystać, a także jakie są jej ograniczenia. Najważniejszym składnikiem bazy SQL Servera wpływającym na wydajność zapytań jest procesor zapytań, który składa się z optymalizatora zapytań i silnika wykonującego. Mając idealny optymalizator zapytań, mógłbyś po prostu przesłać dowolne zapytanie i otrzymać za każdym razem idealny plan wykonania. Idealny silnik wykonujący pozwalałby natomiast wykonać każde zapytanie w ciągu kilku milisekund. W rzeczywistości jednak optymalizacja zapytań to bardzo złożony problem, a żaden optymalizator nie znajdzie idealnego planu za każdym razem — przynajmniej w rozsądnym czasie. W przypadku rozbudowanych zapytań optymalizator zapytań może przeanalizować tylko ograniczoną liczę planów wykonania. Nawet gdyby optymalizator zapytań mógł przeanalizować wszystkie możliwe rozwiązania, kolejnym problemem byłoby podjęcie decyzji, który plan wybrać. Który będzie najbardziej wydajny? Wybór planu wiązałby się z oszacowaniem kosztu każdego z rozwiązań, co również jest bardzo skomplikowanym zadaniem.

Nie zrozum mnie źle: optymalizator zapytań SQL Servera sprawuje się naprawdę świetnie i prawie za każdym razem wybiera dobry plan wykonania. Musisz jednak zrozumieć, jakie informacje należy przekazać do optymalizatora zapytań, aby mógł dobrze wykonać swoją pracę — może to wiązać się z koniecznością zapewnienia odpowiednich indeksów lub statystyk, a także z koniecznością zdefiniowania odpowiednich więzów integralności i dobrego projektu bazy danych. SQL Server zawiera nawet narzędzia, które mogą pomóc Ci w niektórych z tych obszarów, m.in. Database Engine Tuning Advisor (DTA) czy funkcjonalności automatycznego tworzenia i aktu-

alizowania statystyk. Możesz jednak zrobić więcej, aby poprawić wydajność swoich baz, szczególnie jeżeli budujesz wysoko wydajne aplikacje. Musisz też zrozumieć przypadki, w których optymalizator zapytań może nie zwrócić dobrych wyników, i dowiedzieć się, co możesz wtedy zrobić.

Abyś więc mógł lepiej zrozumieć technologię, ten rozdział rozpoczyna się od opisu działania optymalizatora zapytań bazy SQL Servera i od wprowadzenia koncepcji dokładniej omawianych w dalszej części książki. Wyjaśniam cel istnienia zarówno optymalizatora zapytań, jak i silnika wykonującego i ich maksymalnej interakcji z pamięcią podręczną planów zapytania. W dalszej części pokazuję, jak pracować z planami wykonania, które są podstawowym narzędziem podczas pracy z procesorem zapytań.

Architektura

W sercu bazy danych SQL Servera znajdują się dwa główne komponenty: silnik przechowywania i silnik relacyjny, nazywany również procesorem zapytań. Silnik przechowywania jest odpowiedzialny za odczyt danych pomiędzy dyskiem a pamięcią w sposób optymalizujący współbieżność i pozwalający zachować integralność danych. Procesor zapytań, jak sugeruje nazwa, przyjmuje zapytania kierowane do serwera, buduje plan ich optymalnego wykonania, a następnie wykonuje go i dostarcza dane wynikowe.

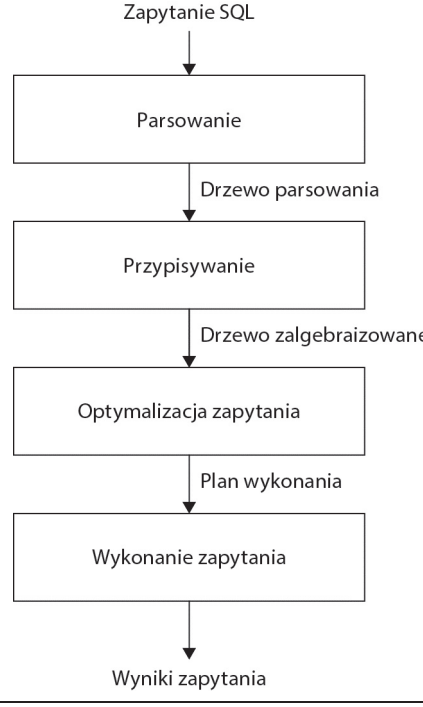
Zapytania są przekazywane do SQL Servera za pomocą języka SQL (lub T-SQL, który jest rozszerzeniem języka SQL wykorzystywanym w Micorosoft SQL Serverze). Ponieważ SQL jest językiem deklaratywnym wysokiego poziomu, definiuje tylko, jakie dane pobrać z bazy danych, nie definiuje natomiast kroków potrzebnych do ich pobrania ani algorytmów przetwarzania żądania. Dlatego, dla każdego odebranego zapytania, pierwszym krokiem wykonywanym przez procesor zapytań jest jak najszybsze określenie planu zapytania, który opisuje najlepszy możliwy sposób (lub co najmniej wydajny sposób) na wykonanie danego zapytania. Jego drugim zadaniem jest wykonanie zapytania zgodnie z planem. Każde z tych zadań jest powierzane odrębnemu komponentowi wewnątrz procesora zapytań; optymalizator zapytań opracowuje plan i przekazuje go do silnika wykonującego, który wykona zapytanie i pobierze rezultaty z bazy danych.

Optymalizator zapytań bazy SQL Servera działa na podstawie kosztów. Dla danego zapytania analizuje kilka potencjalnych planów zapytania, szacuje koszt każdego z nich i wybiera plan o najniższym koszcie. W rzeczy samej, biorąc pod uwagę fakt, że optymalizator nie może przeanalizować wszystkich możliwych planów wykonania zapytania, musi znaleźć równowagę pomiędzy czasem optymalizacji i jakością wybranego planu.

W związku z tym jest to komponent serwera, który ma największy wpływ na wydajność bazy. W końcu wybór dobrego lub złego planu zapytania może stanowić różnicę

pomiędzy wykonaniem zapytania w ciągu milisekund a wykonaniem go w ciągu minut lub nawet godzin. Naturalnie, lepsze zrozumienie sposobu działania optymalizatora zapytań może pomóc zarówno administratorom baz, jak i programistom w pisaniu lepszych zapytań i w zapewnianiu optymalizatorowi informacji potrzebnych do jego jak najlepszego działania. Ta książka pokaże Ci, jak wykorzystać nowo poznaną wiedzę na temat architektury optymalizatora zapytań, a ponadto przekaże Ci wiedzę i narzędzia do radzenia sobie z sytuacjami, w których optymalizator nie produkuje dobrego planu.

Aby dotrzeć do optymalnego planu zapytania, procesor zapytań wykonuje kilka kroków. Cały proces przetwarzania zapytania został przedstawiony na rysunku 1.1.



Rysunek 1.1. Proces przetwarzania zapytania

Procesowi temu dokładniej przyjrzymy się w rozdziale 3., ale teraz omówię pokrótce poszczególne kroki:

1. **Parsowanie i przypisywanie**. Zapytanie jest parsowane i przypisywane. Zakładając, że zapytanie jest poprawne, wynikiem tej fazy jest drzewo logiczne, którego każdy węzeł reprezentuje operację logiczną, jaką musi wykonać zapytanie, taką jak na przykład czytanie tabeli lub wykonanie złączenia.

2. **Optymalizacja zapytania**. Drzewo logiczne jest wykorzystywane w procesie optymalizacji zapytania, który, ogólnie rzecz ujmując, składa się z następujących kroków:
   - **Generowanie możliwych planów zapytania**. Korzystając z drzewa logicznego, optymalizator określa kilka możliwych sposobów wykonania zapytania (czyli możliwe plany wykonania). Plan wykonania to, ogólnie mówiąc, zestaw fizycznych operacji (takich jak *Index Seek* [przeszukanie indeksu] lub *Nested Loops Join* [złączenie z zagnieżdżoną pętlą]), które mogą zostać wykonane w celu osiągnięcia wymaganego rezultatu opisanego w drzewie logicznym.
   - **Określenie kosztu każdego z planów**. Chociaż optymalizator nie generuje wszystkich możliwych planów zapytania, określa koszt zasobów i czasu każdego z wygenerowanych planów. Plan, którego szacowany koszt będzie najniższy, zostanie wybrany i przekazany dalej do silnika wykonującego.
3. **Wykonywanie zapytań i przechowywanie planów**. Zapytanie jest wykonywane przez silnik wykonujący zgodnie z wybranym planem; plan może być przechowywany w pamięci podręcznej planów.

Parsowanie i przypisywanie

Parsowanie i przypisywanie to pierwsze operacje wykonywane po przesłaniu zapytania do instancji SQL Servera. Parsowanie pozwala upewnić się, czy zapytanie T-SQL ma poprawną składnię, i przekształca je w drzewo, a dokładniej w drzewo logicznych operatorów reprezentujących kroki wysokiego poziomu prowadzące do wykonania zapytania. Na początku te operatory będą blisko związane z pierwotną składnią zapytania i będą zawierały takie operacje jak „pobierz dane z tabeli `Klient`", „pobierz dane z tabeli `Kontakt`", „wykonaj złączenie wewnętrzne" i tak dalej. W trakcie procesu optymalizacji będą wykorzystywane różne drzewa reprezentujące zapytanie, a drzewo to będzie miało różne nazwy, dopóki nie zostanie wykorzystane do inicjalizacji struktury Memo.

Przypisywanie jest związane z rozwiązywaniem nazw. Podczas operacji przypisywania SQL Server upewnia się, czy wszystkie nazwy obiektów istnieją, i przypisuje każdą tabelę i kolumnę na drzewie parsowania do powiązanych obiektów w katalogu systemowym. Wynik tego procesu nazywany jest *drzewem zalgebraizowanym*. Drzewo to jest następnie przesyłane do optymalizatora zapytań.

Optymalizacja zapytań

Kolejny krok to proces optymalizacji, który jest w zasadzie generowaniem potencjalnych planów wykonywania i wyborem najlepszego z nich na podstawie szacowanych kosztów ich wykonania. Jak już wspomniałem, SQL Server wykorzystuje model szacowania kosztów do określenia kosztu wykonania każdego z planów.

Na proces optymalizacji zapytań można również patrzeć jak na proces mapowania logicznych operacji zapytania wyrażonych w pierwotnym drzewie na operacje fizyczne, które będą mogły zostać wykorzystane przez silnik wykonujący. W planach wykonania tworzonych przez optymalizator zapytań jest w zasadzie implementowana funkcjonalność silnika wykonywania; to znaczy silnik wykonywania implementuje pewną liczbę algorytmów, a optymalizator podczas tworzenia planu wykonania wybiera spośród nich. Robi to, tłumacząc pierwotne operacje logiczne na operacje fizyczne, które silnik wykonujący będzie w stanie wykonać. Plany wykonania pokazują zarówno operacje logiczne, jak i fizyczne dla każdego operatora. Te same logiczne operacje, takie jak sortowanie, przekładają się na te same operacje fizyczne, niektóre jednak mapują się na więcej niż jedną możliwą operację fizyczną. Na przykład logiczne złączenie może być mapowane na kilka fizycznych operatorów: *Nested Loops Join*, *Merge Join* (złączenie ze scalaniem) lub *Hash Join* (złączenie haszowe). Nie jest to więc proces mapowania jeden do jednego i wiąże się z bardziej skomplikowanymi działaniami, które dokładniej omówię w rozdziale 3.

Produktem końcowym procesu optymalizacji zapytania jest plan wykonania — drzewo składające się z fizycznych operatorów, które zawierają algorytmy wykonywane przez silnik wykonujący w celu pobrania żądanych wyników z bazy danych.

Generowanie możliwych planów zapytań

Podstawowym celem istnienia optymalizatora zapytań jest odnalezienie wydajnego planu wykonania zapytania. Nawet dla relatywnie prostych zapytań może istnieć ogromna liczba różnych sposobów na uzyskanie dostępu do danych w celu otrzymania tych samych rezultatów. W związku z tym optymalizator musi wybrać najlepszy plan z puli, która może być bardzo duża, a podjęcie dobrej decyzji jest ważne, ponieważ czas potrzebny na zwrócenie wyniku użytkownikowi może być bardzo różny w zależności od wybranego planu.

Zadaniem optymalizatora zapytań jest stworzenie i sprawdzenie największej możliwej liczby planów w ramach pewnych kryteriów, aby znaleźć wystarczająco dobry plan, który może, ale nie musi, być planem optymalnym. Definiujemy obszar poszukiwań dla danego zapytania jako zestaw wszystkich możliwych planów zapytania, a każdy możliwy plan zwraca te same wyniki. Teoretycznie, aby znaleźć optymalny plan wykonania dla zapytania, optymalizator kosztowy powinien znaleźć wszystkie możliwe plany istniejące w danym obszarze poszukiwań i poprawnie oszacować koszty ich wszystkich. Niestety niektóre skomplikowane zapytania mogą mieć tysiące, a nawet miliony możliwych planów, i chociaż optymalizator może zazwyczaj brać pod uwagę znaczą liczbę planów, nie jest w stanie sprawdzić ich wszystkich. Gdyby to zrobić, czas potrzebny na taką operację byłby nieakceptowalnie długi i mógłby mieć znaczący wpływ na całościowy czas wykonania zapytania.

Optymalizator zapytań musi utrzymywać równowagę pomiędzy czasem optymalizacji a jakością planu. Na przykład, jeżeli optymalizator potrzebuje 1 sekundy na znalezienie wystarczająco dobrego planu, który wykonany zostanie w minutę, nie ma sensu szukać idealnego lub najbardziej optymalnego planu, skoro zajęłoby to 5 minut plus czas wykonania zapytania. Dlatego SQL Server nie wykonuje wyczerpującego wyszukiwania, lecz próbuje znaleźć wystarczająco dobry plan najszybciej jak to możliwe. Ponieważ praca optymalizatora jest ograniczona czasem, istnieje szansa, że znaleziony plan będzie planem optymalnym, ale może też być czymś tylko zbliżonym do planu optymalnego.

Aby zbadać obszar poszukiwań, optymalizator wykorzystuje reguły transformacji i heurystykę. Generowanie potencjalnych planów zapytania jest wykonywane wewnątrz optymalizatora z wykorzystaniem reguł transformacji, a wykorzystanie heurystyki ogranicza liczbę branych pod uwagę planów, tak aby czas wykonywania optymalizacji był akceptowalny. Zestaw alternatywnych planów rozpatrywanych przez optymalizator nazywany jest obszarem planów, a same plany podczas optymalizacji przechowywane są w pamięci w komponencie o nazwie Memo. Reguły transformacji, heurystykę i strukturę Memo omówię dokładnie w rozdziale 3.

Określanie kosztu każdego z planów

Wyszukanie lub ponumerowanie potencjalnych planów to tylko jedna część procesu optymalizacji. Optymalizator zapytań musi jeszcze oszacować koszt tych planów i wybrać najmniej kosztowny z nich. Aby oszacować koszt planu, musi oszacować koszt każdego fizycznego operatora w planie, korzystając z formuł kosztowych, które uwzględniają wykorzystanie zasobów takich jak I/O, CPU i pamięć. Szacunek kosztów zależny jest w dużej mierze od algorytmu wykorzystywanego przez fizyczny operator i szacowanej liczby rekordów, które będą musiały zostać przetworzone. Szacunek liczby rekordów do przetworzenia nazywa się *szacunkiem kardynalności*.

Aby pomóc w szacunku kardynalności, SQL Server wykorzystuje i przechowuje statystyki, które zawierają informacje opisujące rozkład wartości w jednej lub większej liczbie kolumn tabeli. Kiedy koszt każdego z operatorów zostanie oszacowany z wykorzystaniem szacunku kardynalności i wymagań dotyczących zasobów, optymalizator doda do siebie wszystkie te koszty, w wyniku czego otrzyma szacowany koszt planu. Nie będę tutaj zagłębiał się w szczegóły, statystyki omówię dokładniej w rozdziale 6.

Wykonywanie zapytań i przechowywanie planów

Kiedy zapytanie zostało zoptymalizowane, plan wynikowy jest wykorzystywany przez silnik wykonujący do pobrania żądanych danych. Wygenerowany plan zapytania może być przechowywany w pamięci w magazynie planów, dzięki czemu będzie mógł zostać powtórnie wykorzystany, jeżeli to samo zapytanie zostanie wykonane powtórnie.

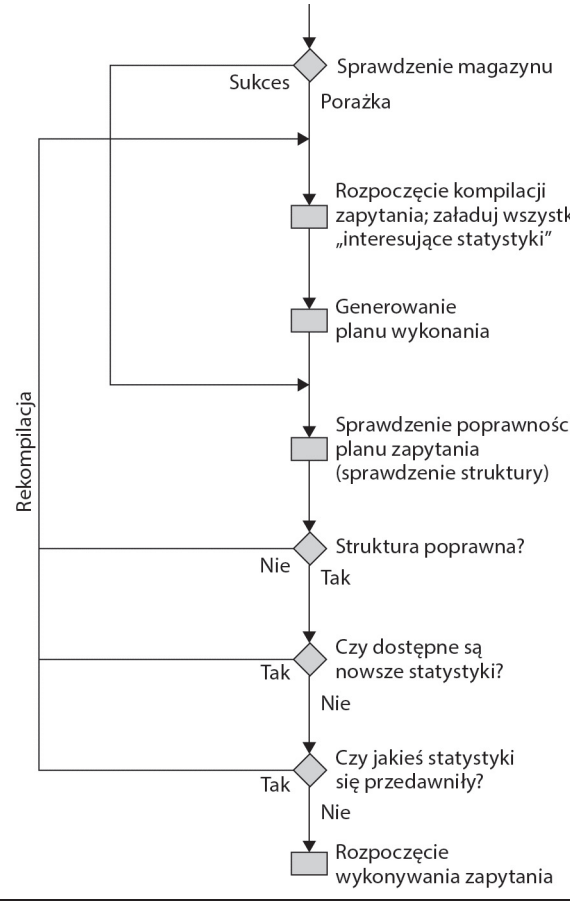
SQL Server ma pulę pamięci, która jest wykorzystywana do przechowywania zarówno stron danych, jak i planów zapytań. Większość tej pamięci wykorzystywana jest do przechowywania stron danych i nazywa się *pulą bufora*. Część tej pamięci zawiera plany wykonywania zapytań, które zostały zoptymalizowane przez optymalizator, i nazywa się *magazynem planów* (wcześniej nazywana była *magazynem procedur*). Procentowa ilość miejsca w pamięci przypisana do magazynu planów lub puli bufora jest dynamiczna i zależy od stanu systemu.

Przed optymalizowaniem zapytania SQL Server sprawdza, czy w magazynie planów nie został zapisany plan dla wykonanego zestawu zapytań. Optymalizacja zapytań to relatywnie kosztowna operacja, jeżeli więc w magazynie planów dostępny jest odpowiedni plan, proces optymalizacji może zostać pominięty, dzięki czemu system unika kosztów w postaci czasu optymalizacji, zasobów procesora i tak dalej. Jeżeli plan dla zestawu zapytań nie zostanie odnaleziony, zestaw jest kompilowany w celu wygenerowania planów dla wszystkich zapytań w procedurze przechowywanej, wyzwalaczu lub dynamicznym kodzie SQL. Optymalizacja rozpoczyna się od załadowania wszystkich interesujących statystyk. Następnie optymalizator zapytań sprawdza, czy statystyki są aktualne. Dla nieaktualnych statystyk, w przypadku domyślnych ustawień statystyk, przed przejściem do dalszego etapu optymalizacji statystyki zostaną zaktualizowane.

Kiedy plan zostanie znaleziony w magazynie planów lub zostanie stworzony nowy plan, nastąpi sprawdzenie, czy nie pojawiły się zmiany w strukturze lub statystykach. Zmiany w strukturze są weryfikowane pod kątem poprawności planu. Statystyki również są weryfikowane: optymalizator sprawdza, czy nie występują nowsze statystyki lub czy statystyki się nie przedawniły. Jeżeli z któregoś z tych powodów plan nie jest poprawny, wówczas jest odrzucany, a zestaw lub pojedyncze zapytanie jest kompilowane jeszcze raz. Takie kompilacje nazywane są *rekompilacjami*. Proces ten został pokazany na rysunku 1.2.

Plany mogą być usuwane z magazynu planów, jeżeli serwer wymaga zwolnienia pamięci lub kiedy zostały wykonane pewne instrukcje. Zmiana niektórych opcji konfiguracji (np. maksymalny poziom współbieżności) wyczyści cały magazyn planów. Podobnie niektóre instrukcje, takie jak operacje na bazie z wykorzystaniem pewnych opcji `ALTER DATABASE`, spowodują wyczyszczenie planów związanych z tą bazą.

Warto również zauważyć, że powtórne wykorzystanie istniejącego planu nie zawsze musi być dobrym rozwiązaniem dla danego zapytania oraz że w takim przypadku mogą pojawić się pewne problemy. Na przykład, w zależności od rozkładu danych w tabeli, optymalny plan zapytania może w dużej mierze zależeć od wykorzystanych parametrów. Więcej na temat tego typu problemów i samego magazynu planów dowiesz się z rozdziału 8.



Rysunek 1.2. Proces kompilacji i rekompilacji

Plany wykonania

Teraz, kiedy znamy podstawy działania procesora zapytań, czas dowiedzieć się, jak możemy wpływać na jego działanie. Podstawowym sposobem interakcji z procesorem zapytań są plany wykonania, które, jak wcześniej wspomniałem, są drzewami składającymi się z operatorów fizycznych, które z kolei zawierają algorytmy prowadzące do wygenerowania żądanych wyników z bazy danych. Biorąc pod uwagę, że w całej książce będziemy często korzystać z planów wykonania, w tym podrozdziale pokażę, jak je wyświetlać i czytać.

Możesz zażądać właściwego lub szacowanego planu zapytania dla danego zapytania, a oba typy mogą zostać wyświetlone jako grafika, tekst lub XML. Każdy z tych formatów pokazuje ten sam plan wykonania, a jedyna różnica tkwi w sposobie przedstawienia i w stopniu szczegółowości informacji w nich zawartych.

Kiedy wykonujemy żądanie wyświetlenia szacowanego planu wykonywania, zapytanie nie jest wykonywane; wyświetlany plan jest tym, który najprawdopodobniej zostałby wykorzystany, gdyby zapytanie zostało wykonane (należy jednak pamiętać, że rekompilacja może spowodować wygenerowanie innego planu zapytania — dokładniej omówię tę kwestię w dalszej części). Gdy jednak zażądamy właściwego planu, zapytanie musi zostać wykonane, a plan zostanie wyświetlony wraz z wynikami zapytania. Mimo wszystko wykorzystanie planu szacowanego ma kilka zalet, możemy bowiem na przykład przejrzeć plan dla zapytania, które wykonuje się bardzo długo bez konieczności jego wykonywania, lub wyświetlić plan zapytania dla operacji aktualizacji bez konieczności zmieniania bazy danych.

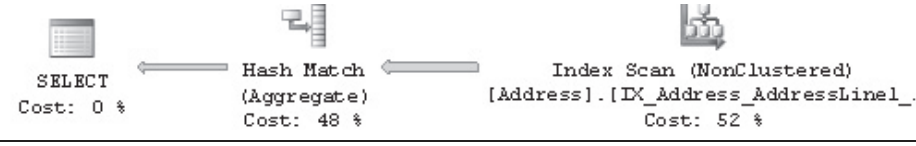
Plany graficzne

W programie SQL Server Management Studio możesz wyświetlać plany graficzne, klikając przycisk *Display Estimated Execution Plan* (wyświetlenie szacowanego planu zapytania) lub *Include Actual Execution Plan* (uwzględnienie właściwego planu zapytania) na pasku narzędzi. Kliknięcie przycisku *Display Estimated Execution Plan* wyświetli plan zapytania od razu, bez wykonywania zapytania. Aby uzyskać właściwy plan, musisz kliknąć przycisk *Include Actual Execution Plan*, a następnie wykonać zapytanie i przejść do zakładki *Execution plan* (plan wykonania).

Aby uzyskać przykład, przekopiuj poniższe zapytanie do edytora zapytań programu SQL Server Management Studio, wybierz bazę `AdventureWorks2012`, kliknij przycisk *Include Actual Execution Plan*, a następnie wykonaj zapytanie:

```
SELECT DISTINCT(City) FROM Person.Address
```

W panelu rezultatów wybierz zakładkę *Execution plan*. Wyświetlony zostanie plan przedstawiony na rysunku 1.3.



Rysunek 1.3. Graficzny plan wykonania

Każdy węzeł w drzewie jest reprezentowany przez ikonę przedstawiającą logiczny i fizyczny operator, taki jak *Index Scan* czy *Hash Aggregate*, zgodnie z rysunkiem 1.3. Pierwsza ikona to element języka o nazwie *Result operator* (operator rezultatu) i reprezentuje polecenie `SELECT`, a zazwyczaj jest też głównym elementem planu.

Operatory implementują podstawową funkcję lub operację silnika wykonującego; na przykład logiczna operacja złączenia może być implementowana przez jedną z trzech fizycznych operacji złączenia (*Nested Loops Join*, *Merge Join*, *Hash Join*). Oczywiście,

UWAGA

W tej książce znajdziesz ogromną liczbę przykładowych zapytań SQL — wszystkie zapytania tworzone były dla bazy `AdventureWorks2012`, chociaż rozdział 9. wykorzystuje również bazę `AdventureWorksDW2012`. Kod był testowany na bazie *SQL Server 2014 RTM*. Te bazy danych nie są dołączane do standardowej instalacji MS SQL Servera, ale możesz je pobrać ze strony CodePlex. Musisz pobrać rodzinę baz przykładowych dla SQL Servera 2012 (w momencie pisania tej książki przykładowe bazy dla MS SQL Servera 2014 nie istniały). Podczas instalacji możesz zainstalować wszystkie bazy lub tylko `AdventureWorks2012` i `AdventureWorksDW2012`.

w silniku wykonującym jest zaimplementowanych znacznie więcej operatorów, a całą ich listę znajdziesz pod adresem *http://msdn.microsoft.com/en-us/library/ms191158(v=sql.110).aspx*. Ikony operatorów logicznych i fizycznych wyświetlane są w kolorze niebieskim, z wyjątkiem operatorów kursora, które są żółte, i elementów języka, które są zielone:

 Operator logiczny/fizyczny

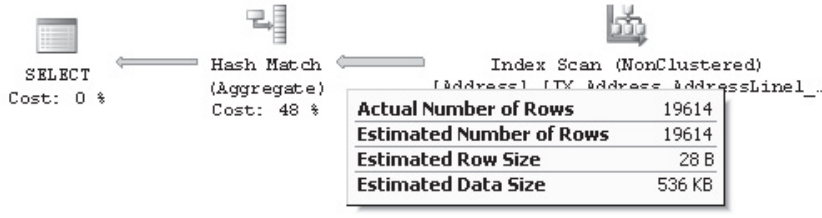
 Element języka

 Kursor

Optymalizator zapytań buduje plan wykonania, wybierając spośród operatorów, które mogą odczytywać rekordy z bazy, jak na przykład operator *Index Scan* przedstawiony na poprzednim planie; mogą też odczytywać rekordy z innego operatora, jak na przykład operator *Hash Aggregate* odczytujący rekordy z operatora *Index Scan*.

Każdy węzeł jest związany z węzłem nadrzędnym, z którym jest połączony strzałką, a dane płyną od operatora potomnego do nadrzędnego, przy czym szerokość strzałki jest proporcjonalna do liczby rekordów. Kiedy operator wykonuje pewne funkcje na przeczytanych rekordach, rezultaty zaś są przekazywane do węzła nadrzędnego. Możesz najechać kursorem myszy, aby uzyskać więcej informacji na temat ilości danych — zostaną zaprezentowane w oknie podpowiedzi. Jeżeli, na przykład, najedziesz na strzałkę pomiędzy operacjami *Index Scan* i *Hash Aggregate* przedstawioną na rysunku 1.3, otrzymasz informacje o danych przepływających pomiędzy tymi operacjami (zobacz rysunek 1.4).

Operator *Index Scan* odczytuje 19 614 rekordów i przesyła je do operatora *Hash Aggregate*. Z kolei operator *Hash Aggregate* wykonuje pewne operacje na danych i do swojego węzła nadrzędnego przesyła 575 rekordów, co również możesz zobaczyć, najeżdżając kursorem myszy na strzałkę pomiędzy tymi operacjami.



Rysunek 1.4. Przepływ danych pomiędzy operatorami Index Scan i Hash Aggregate

W tym planie operacja *Index Scan* czyta wszystkie 19 614 wierszy indeksu, a operacja *Hash Aggregate* wykonuje operacje mające wybrać niepowtarzające się nazwy miast, których jest 575 — te nazwy zostaną wyświetlone w oknie rezultatów programu Management Studio. Zauważ także, że oprócz rzeczywistej liczby rekordów otrzymujesz też informacje o szacowanej liczbie wierszy, która jest szacunkiem kardynalności wykonywanym przez optymalizator dla tego operatora. Porównanie rzeczywistej i szacowanej liczby rekordów może pomóc w wykryciu błędów szacunku kardynalności, które mogą wpłynąć na jakość planów wykonywania (więcej na ten temat dowiesz się z rozdziału 6.).

Aby były w stanie wykonywać swoje zadanie, operatory fizyczne muszą implementować przynajmniej trzy poniższe metody:

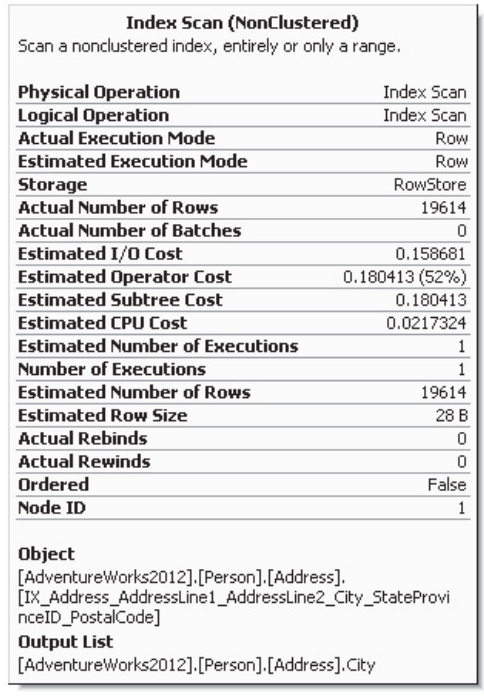
- **`Open()`** — inicjalizuje operator, może zawierać zadania przygotowujące wymagane struktury danych.
- **`GetRow()`** — wykonuje żądanie rekordu z operatora.
- **`Close()`** — wykonuje operacje sprzątające i zamyka operator po zakończeniu jego roli.

Operator pobiera rekordy z innych operatorów za pomocą metody `GetRow()`, co oznacza również, że wykonanie planu rozpoczyna się od lewej do prawej. Ponieważ `GetRow()` tworzy tylko jeden wiersz w danym momencie, liczba rekordów wyświetlona w planie zapytania jest również liczbą wywołań metody dla danego operatora; do oznaczenia końca zestawu danych wynikowych wykorzystywane jest jedno dodatkowe wywołanie metody `GetRow()`. W poprzednim przykładzie operator *Hash Aggregate* wywołuje na operatorze *Index Scan* metodę `Open()` raz, metodę `GetRow()` 19 615 razy i raz metodę `Close()`.

UWAGA

Na razie wyjaśnimy tradycyjny tryb przetwarzania zapytań, w którym operatory przetwarzają jeden wiersz w danym momencie. Ten tryb przetwarzania był wykorzystywany we wszystkich wersjach SQL Servera od wersji 7.0. W rozdziale 9. wspomnę o nowym trybie przetwarzania partiami, wprowadzonym w SQL Serverze 2012 i wykorzystywanym przez operatory związane z indeksami magazynów kolumn.

Możesz również najechać kursorem myszy na operator, aby uzyskać o nim więcej informacji. Na przykład rysunek 1.5 przedstawia informacje o operatorze *Index Scan*; zauważ, że zawiera między innymi opis operatora i dane na temat szacowanych kosztów, takich jak koszt I/O, CPU, operatora i poddrzewa (*subtree*).



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	19614
Actual Number of Batches	0
Estimated I/O Cost	0.158681
Estimated Operator Cost	0.180413 (52%)
Estimated Subtree Cost	0.180413
Estimated CPU Cost	0.0217324
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	19614
Estimated Row Size	28 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Object
[AdventureWorks2012].[Person].[Address].[IX_Address_AddressLine1_AddressLine2_City_StateProvinceID_PostalCode]

Output List
[AdventureWorks2012].[Person].[Address].City

Rysunek 1.5. Okno z informacjami o operatorze Index Scan

Niektóre z tych właściwości omawiam w tabeli 1.1, inne wyjaśnię w dalszej części książki.

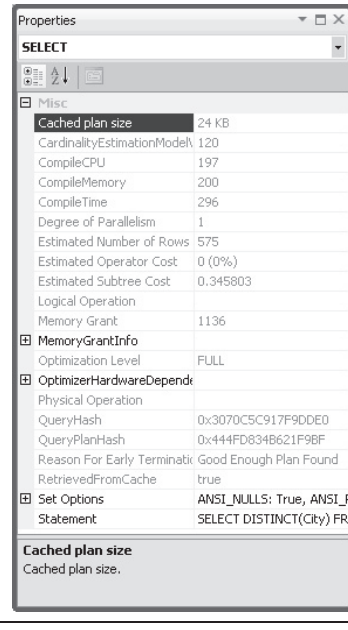
UWAGA

Warto nadmienić, że koszt wyrażony jest w wewnętrznych jednostkach, które nie powinny być utożsamiane z sekundami czy milisekundami.

Koszt każdego z operatorów jest również przedstawiany relatywnie, jako wartość procentowa całego planu (zobacz rysunek 1.3). Na przykład koszt operacji *Index Scan* wynosi 52% kosztu całego planu. Dodatkowe informacje z operatora lub całego zapytania można pozyskać za pomocą okna *Properties* (właściwości). Wybór ikony SELECT i otwarcie okna *Properties* z menu *View* (widok) lub naciśnięcie klawisza *F4* pokaże właściwości całego zapytania (zobacz rysunek 1.6).

Tabela 1.1. Właściwości operatorów

Właściwość	Opis
Physical Operation	Algorytm fizyczny dla węzła.
Logical Operation	Operator algebry relacyjnej reprezentowany przez węzeł.
Actual Number of Rows	Liczba rekordów wytworzonych przez operator.
Estimated I/O Cost	Szacunkowy koszt operacji I/O. Nie wszystkie operacje związane są z kosztem I/O.
Estimated Operator Cost	Koszt operacji oszacowany przez optymalizator zapytań. Jest to szacowany koszt I/O i CPU. Zawiera także koszt operacji jako procentową wartość całego kosztu zapytania wyświetlony w nawiasach.
Estimated Subtree Cost	Szacowany narastający koszt wykonania tej operacji i wszystkich operacji poprzedzających ją w tym samym poddrzewie.
Estimated CPU Cost	Szacowany koszt procesora dla tej operacji.
Estimated Number of Executions	Szacowana liczba wywołań tego operatora podczas wykonywania bieżącego zapytania.
Number of Executions	Liczba wywołań tego operatora po wykonaniu zapytania.
Estimated Number of Rows	Szacowana liczba rekordów wytworzona przez operator (szacunek kardynalności).
Estimated Row Size	Szacowany średni rozmiar rekordu przetwarzanego przez ten operator.



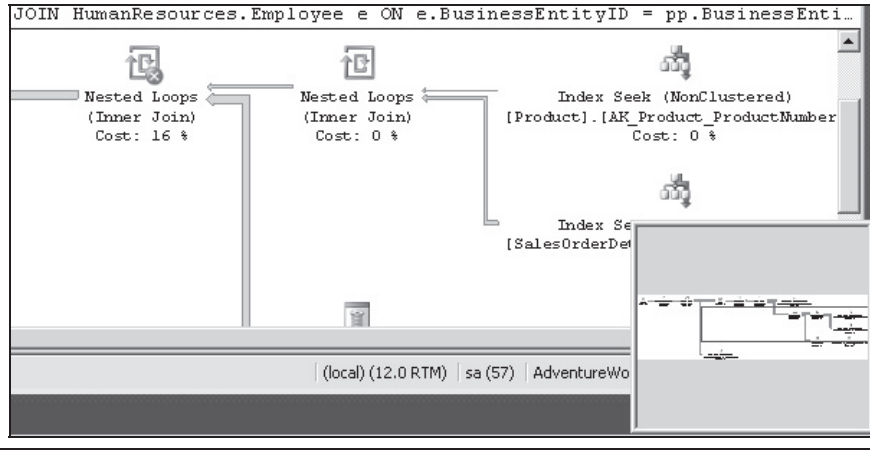
Rysunek 1.6. Okno właściwości zapytania

Tabela 1.2 zawiera większość właściwości z rysunku 1.6. W zależności od zapytania mogą pojawić się inne, opcjonalne właściwości (na przykład *Parameter List* lub *Warnings*).

Tabela 1.2. Właściwości zapytań

Właściwość	Opis
Cached plan size	Ilość pamięci w kilobajtach w magazynie planów wykorzystana przez plan zapytania.
CompileCPU	Czas procesora w milisekundach wykorzystany do skompilowania zapytania.
CompileMemory	Pamięć w kilobajtach wykorzystana do skompilowania zapytania.
CompileTime	Czas w milisekundach wykorzystany do skompilowania zapytania.
Degree of Parallelism	Liczba wątków, które mogą zostać wykorzystane do wykonania zapytania, jeżeli procesor zapytań wybierze plan równoległy.
Memory Grant	Ilość pamięci w kilobajtach udzielonej do uruchomienia tego zapytania.
MemoryGrantInfo	Informacje dotyczące szacunku na temat udzielonej pamięci i informacje o rzeczywistej ilości udzielonej pamięci.
Optimization Level	Poziom optymalizacji wykorzystany do skompilowania tego zapytania. Wyświetlany jako *StatementOptmLevel* na planie w formacie XML. Zostanie dokładniej omówiony w dalszej części tego podrozdziału.
OptimizerHardwareDependentProperties	Właściwości uzależnione od platformy sprzętowej, które mają wpływ na szacunek kosztów (a co za tym idzie, wybór planu) z punktu widzenia optymalizatora zapytań.
QueryHash	Wartość hasza binarnego obliczona na zapytaniu i wykorzystywana do identyfikacji zapytań o podobnej logice.
QueryPlanHash	Wartość hasza binarnego obliczona na planie zapytania i wykorzystywana do identyfikacji podobnych planów.
Reason For Early Termination Of Statement Optimization	Na planie w formacie XML wyświetlana jako *StatementOptmEarlyAbortReason*. Zostanie dokładniej omówiona w dalszej części tego podrozdziału.
RetrievedFromCache	Wskazuje, czy plan został pobrany z magazynu planów.
Set Options	Status opcji `SET` mających wpływ na koszt zapytania. Na planie w formacie XML wyświetlana jako *StatementSetOptions*. Te opcje to: `NSI_NULLS`, `ANSI_PADDING`, `ANSI_WARNINGS`, `ARITHABORT`, `CONCAT_NULL_YIELDS_NULL`, `NUMERIC_ROUNDABORT` i `QUOTED_IDENTIFIER`.
Statement	Tekst zapytania SQL.

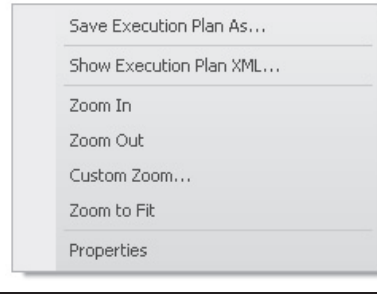
SQL Server udostępnia też funkcjonalność oddalania planu, którą możesz wykorzystać podczas nawigacji po dużych graficznych planach, które mogą nie mieścić się na ekranie. Możesz uzyskać dostęp do tego narzędzia, klikając ikonę plusa zlokalizowaną w prawym dolnym rogu zakładki planu wykonania. Przykład został przedstawiony na rysunku 1.7. Popularnym narzędziem do pracy z planami zapytań jest też SQL Sentry Plan Explorer. Możesz go pobrać za darmo pod adresem *http://sqlsentry.net/plan-explorer*.



Rysunek 1.7. Funkcjonalność oddalania planu zapytania

XML

Kiedy już wyświetliłeś plan graficzny, możesz również z łatwością wyświetlić go w formacie XML. Po prostu kliknij prawym przyciskiem myszy w dowolnym miejscu okna planu wykonywania i w menu kontekstowym wybierz opcję *Show Execution Plan XML…* (pokaż plan wykonania w formacie XML), zgodnie z rysunkiem 1.8. Wówczas zostanie otwarty edytor XML i wyświetli się plan XML. Jak widać, możesz łatwo przełączać się pomiędzy wersjami graficzną i XML.



Rysunek 1.8. Menu kontekstowe dla planu wykonania

Jeżeli będzie to potrzebne, możesz zapisywać plany graficzne w pliku, wybierając opcję *Save Execution Plan As…* (zapisz plan wykonania jako) z menu kontekstowego przedstawionego na rysunku 1.8. Plan, zapisywany zazwyczaj z rozszerzeniem *.sqlplan*, to tak naprawdę dokument XML zawierający plan w formacie XML, który może zostać odczytany przez program Management Studio i przekształcony w plan graficzny. Możesz powtórnie załadować ten plik, wybierając z menu *File* (plik) opcję *Open* (otwórz), a wówczas plan zostanie otwarty w nowym oknie i będzie się zachowywał dokładnie tak jak poprzednio. Plany zapytania w formacie XML mogą również być wykorzystywane w zapytaniu z podpowiedzią `USEPLAN`, co omówię w rozdziale 10.

Tabela 1.3 zawiera różne polecenia, jakie możesz wykorzystać do uzyskania szacowanego albo rzeczywistego planu w formacie tekstowym, graficznym lub XML.

Tabela 1.3. Polecenia pozwalające wyświetlać plany zapytań

	Szacowany plan wykonania	Rzeczywisty plan wykonania
Text Plan	`SET SHOWPLAN_TEXT` `SET SHOWPLAN_ALL`	`SET STATISTICS PROFILE`
Graphic Plan	Management Studio	Management Studio
XML Plan	`SET SHOWPLAN_XML`	`SET STATISTICS XML`

UWAGA

Jeżeli uruchomisz którąkolwiek z opcji z tabeli 1.3, korzystając z klauzuli `ON`, opcja ta będzie stosowana do wszystkich kolejnych instrukcji, do czasu ręcznego wyłączenia jej za pomocą klauzuli `OFF`.

Aby wyświetlić plan XML, możesz skorzystać z poniższych poleceń:

```
SET SHOWPLAN_XML ON
GO
SELECT DISTINCT(City) FROM Person.Address
GO
SET SHOWPLAN_XML OFF
```

To zapytanie pozwala wyświetlić wynik z pojedynczym wierszem z jedną kolumną o nazwie *Microsoft SQL Server 2005 XML Showplan* zawierający dane w formacie XML zaczynające się od:

```
<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004 ...
```

Kliknięcie linku spowoduje wyświetlenie planu graficznego, a plan XML będzie można wyświetlić, stosując tę samą metodę co poprzednio.

Możesz przejrzeć podstawową strukturę planu XML dzięki poniższemu ćwiczeniu. Bardzo proste zapytanie stworzy podstawową strukturę XML, tutaj jednak przedstawiam zapytanie, które pozwoli wygenerować dwie dodatkowe części: brakujące indeksy i listę parametrów. Wykonaj poniższe zapytanie i otwórz plan XML:

```
SELECT * FROM Sales.SalesOrderDetail
WHERE OrderQty = 1
```

Zwiń elementy <MissingIndexes>, <RelOp> i <ParameterList>, klikając znak minusa (–) po ich lewej stronie, dzięki czemu będziesz mógł zobaczyć całą strukturę. Powinieneś zobaczyć coś podobnego jak na rysunku 1.9.

Rysunek 1.9. Plan wykonania w formacie XML

Jak widzisz, główne komponenty planu XML to elementy <StmtSimple>, <Statement ↪SetOptions> i <QueryPlan>. Te trzy elementy zawierają po kilka atrybutów, niektóre z nich już objaśniłem, kiedy omawiałem plany graficzne. Ponadto element <QueryPlan> zawiera również inne elementy, takie jak <MissingIndexes>, <MemoryGrantInfo>, <Optimizer ↪HardwareDependentProperties>, <RelOp>, <ParameterList>, a także takie, które nie zostały pokazane na rysunku 1.9 (na przykład <Warnings>), a które omówię w dalszej części tego podrozdziału. Na przykład element <StmtSimple> wygląda tak:

```
<StmtSimple StatementCompId="1" StatementEstRows="68089" StatementId="1"
↪StatementOptmLevel="FULL" CardinalityEstimationModelVersion="70"
↪StatementSubTreeCost="1.13478" StatementText="SELECT * FROM [Sales].[SalesOrderDetail]
↪WHERE [OrderQty]=@1" StatementType="SELECT" QueryHash="0x42CFD97ABC9592DD"
↪QueryPlanHash="0xC5F6C30459CD7C41" RetrievedFromCache="false">
```

A element <QueryPlan> tak:

```
<QueryPlan DegreeOfParallelism="1" CachedPlanSize="32" CompileTime="3" CompileCPU="3"
↪CompileMemory="264">
```

Jak wspomniałem, atrybuty tych i innych elementów omówiłem już w podrozdziale dotyczącym planów graficznych. Inne wyjaśnię w dalszej części tego podrozdziału lub w innych podrozdziałach tej książki.

Plany tekstowe

Jak widzisz w tabeli 1.3, dwa polecenia pozwalają otrzymać szacowany plan tekstowy: `SET SHOWPLAN_TEXT` i `SET SHOWPLAN_ALL`. Oba polecenia powodują wyświetlenie szacowanego planu tekstowego, ale `SET SHOWPLAN_ALL` pokazuje pewne dodatkowe informacje, zawierające szacowaną liczbę wierszy, szacowany koszt CPU, szacowany koszt I/O i szacowany koszt operatora. Najnowsze wersje dokumentacji Books Online, włączając w to te dotyczące SQL Servera 2014, wskazują jednak, że wszystkie formy planu tekstowego zostaną wycofane w przyszłych wersjach SQL Servera i dlatego zalecanym sposobem wyświetlania planu jest XML.

Do wyświetlenia planu tekstowego możesz wykorzystać poniższy kod:

```
SET SHOWPLAN_TEXT ON
GO
SELECT DISTINCT(City) FROM Person.Address
GO
SET SHOWPLAN_TEXT OFF
GO
```

Kod ten spowoduje wyświetlenie dwóch zestawów wyników: pierwszy będzie zawierał tekst zapytania T-SQL, drugi natomiast będzie zawierał poniższy plan tekstowy (wyedytowany tak, aby zmieścił się na stronie), pokazujący te same operatory *Hash Aggregate* i *Index Scan*, które widniały na planie graficznym z rysunku 1.3:

```
|--Hash Match(Aggregate, HASH:([Person].[Address].[City]), RESIDUAL …
|--Index Scan(OBJECT:([AdventureWorks].[Person].[Address]. [IX_Address …
```

`SET SHOWPLAN_ALL` i `SET STATISTICS PROFILE` mogą zwracać więcej informacji niż `SET SHOWPLAN_TEXT`. Możesz także, zgodnie z tabelą 1.3, skorzystać z polecenia `SET SHOWPLAN_ALL` do wyświetlenia samego planu i `SET STATISTICS PROFILE` do wykonania zapytania. Uruchom poniższy przykład:

```
SET SHOWPLAN_ALL ON
GO
SELECT DISTINCT(City) FROM Person.Address
GO
SET SHOWPLAN_ALL OFF
GO
```

Wynik został pokazany na rysunku 1.10.

	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp
1	SELECT DISTINCT(City) FROM Person.Address	1	1	0	NULL	NULL
2	|--Hash Match(Aggregate, HASH:([AdventureWorks...	1	2	1	Hash Match	Aggregate
3	|--Index Scan(OBJECT:([AdventureWorks2012].[...	1	3	2	Index Scan	Index Scan

Rysunek 1.10. Wynik operacji SET SHOWPLAN_ALL

Ponieważ `SET STATISTICS PROFILE` powoduje wykonanie zapytania, pozwala w łatwy sposób wyszukać problemy w szacowaniu kardynalności. Porównanie wielu operatorów jednocześnie jest bardzo proste, natomiast na planie graficznym lub XML może to być bardziej skomplikowane. Teraz uruchom poniższy kod:

```
SET STATISTICS PROFILE ON
GO
SELECT * FROM Sales.SalesOrderDetail
WHERE OrderQty * UnitPrice > 25000
GO
SET STATISTICS PROFILE OFF
GO
```

Wynik został pokazany na rysunku 1.11.

	Rows	EstimateRows	Executes	StmtText
1	5	36395.1	1	SELECT * FROM [Sales].[SalesOrderDetail] WHERE [OrderQty]*[UnitPrice]>@1
2	5	36395.1	1	|--Filter(WHERE:([Expr1003]>($25000.0000)))
3	0	121317	0	|--Compute Scalar(DEFINE:([AdventureWorks2012].[Sales].[SalesOrderDetail].[Li...
4	0	121317	0	|--Compute Scalar(DEFINE:([AdventureWorks2012].[Sales].[SalesOrderDetail]....
5	121317	121317	1	|--Clustered Index Scan(OBJECT:([AdventureWorks2012].[Sales].[SalesOr...

Rysunek 1.11. Wynik operacji SET STATISTICS PROFILE

Zauważ, że kolumna *EstimateRows* została ręcznie przeniesiona w Management Studio obok kolumny *Rows*, co pozwala na ich łatwe porównanie. W tym konkretnym przykładzie możesz zobaczyć dużą różnicę w szacunku kardynalności dla operacji *Filter* (filtr) — oszacowane zostało 36 395,1, podczas gdy rzeczywiście w operacji uczestniczyło tylko 5 wierszy.

Dodatkowe właściwości planów

Interesującym sposobem na naukę komponentów planów wykonania, także tych z przyszłych wersji SQL Servera, jest oglądanie schematu showplan. Plany XML muszą stosować się do opublikowanego schematu XSD. Aktualna i starsze wersje są dostępne pod adresem *http://schemas.microsoft.com/sqlserver/2004/07/showplan/*, który znajdziesz także na początku każdego planu w formacie XML. Aktualnie uruchomienie tego adresu w przeglądarce wyświetli linki do schematów dla wersji SQL Server 2014 RTM (jako *Current version* — aktualna wersja), SQL Server 2012 RTM, SQL Server 2008 RTM, SQL Server 2005 SP2 i SQL Server 2005 RTM.

Omówienie wszystkich elementów i atrybutów planu wykonywania zajęłoby wiele stron, dlatego omówię tylko kilka bardziej interesujących. Operatory wykorzystywane w planach wykonywania omówię dokładniej w rozdziale 4. Rozpocznijmy od atrybutów `StatementOptmLevel`, `StatementOptmEarlyAbortReason` i `CardinalityEstimationModelVersion` elementu `<StmtSimple>`.

Chociaż atrybuty te dotyczą zagadnień omawianych dokładniej w dalszej części książki, warto przedstawić je już teraz. `StatementOptmLevel` dotyczy poziomu optymalizacji zapytania i może przyjmować wartości `TRIVIAL` (trywialna) lub `FULL` (pełna). Proces optymalizacji dla prostych zapytań niewymagających szacowania kosztów może być kosztowny, aby więc uniknąć tych kosztów dla prostych zapytań, SQL Server wykorzystuje optymalizację trywialną. Jeżeli zapytanie nie kwalifikuje się do optymalizacji trywialnej, wykonana zostanie pełna optymalizacja. Na przykład w SQL Serverze 2014 następujące zapytanie będzie podległo optymalizacji trywialnej:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 43666
```

Możesz wykorzystać nieudokumentowaną (a tym samym niewspieraną) flagę 8757 do przetestowania zachowania z wyłączoną optymalizacją trywialną:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 43666
OPTION (QUERYTRACEON 8757)
```

Podpowiedź zapytania `QUERYTRACEON` jest wykorzystywana do przypisywania flag do zapytań. Po uruchomieniu poprzedniego zapytania SQL Server wykona pełną optymalizację, co możesz potwierdzić, sprawdzając wartość właściwości `StatementOptmLevel` w planie zapytania. Zauważ, że chociaż podpowiedź zapytania `QUERYTRACEON` jest dobrze znana, aktualnie wspierana jest tylko w niewielkim stopniu. W chwili pisania tej książki `QUERYTRACEON` jest wspierana tylko podczas stosowania flag udokumentowanych w artykule dostępnym pod adresem *http://support.microsoft.com/kb/2801413*.

UWAGA

W tej książce znajdziesz wiele nieudokumentowanych i niewspieranych funkcjonalności. Możesz korzystać z nich w środowisku testowym, do szukania problemów lub do nauki nowej technologii. Nie są jednak przeznaczone do wykorzystania w środowiskach produkcyjnych i nie są wspierane przez Microsoft. Będę wskazywał, które polecenia lub flagi są nieudokumentowane lub niewspierane.

Z drugiej strony, atrybut `StatementOptmEarlyAbortReason` (powód wczesnego zakończenia optymalizacji) może przyjmować wartości `GoodEnoughPlanFound` (znaleziono wystarczająco dobry plan), `TimeOut` (przekroczenie dozwolonego czasu) i `MemoryLimit ↪Exceeded` (przekroczenie limitu pamięci) i pojawia się tylko wtedy, kiedy optymalizacja została zakończona przedwcześnie (w starszych wersjach SQL, aby zobaczyć tę informację, trzeba było skorzystać z nieudokumentowanej flagi 8675). Ponieważ celem optymalizatora jest stworzenie wystarczająco dobrego planu najszybciej jak to możliwe, optymalizator na początku działania wylicza dwie wartości. Pierwsza z nich to koszt wystarczająco dobrego zapytania, a druga to maksymalny czas, który może

być wykorzystany do stworzenia planu. Jeżeli podczas procesu optymalizacji zostanie znaleziony plan o koszcie wykonania niższym niż wcześniej obliczona wartość graniczna, optymalizacja jest przerywana, a znaleziony plan zostanie zwrócony z wartością `GoodEnoughPlanFound`. Jeśli jednak proces optymalizacji przekroczy pierwotnie założony czas, optymalizacja również zostanie przerwana i zwrócony zostanie najlepszy do tej pory plan z właściwością `StatementOptmEarlyAbortReason` o wartości `TimeOut`. Wartości `GoodEnoughPlanFound` i `TimeOut` nie oznaczają problemu — we wszystkich trzech (włączając w to `MemoryLimitExceeded`) przypadkach plan zapytania będzie poprawny. Jednak w przypadku `MemoryLimitExceeded` plan może nie być optymalny. Wówczas być może będziesz musiał uprościć zapytanie lub zwiększyć ilość dostępnej pamięci. Te i inne szczegóły procesu optymalizacji zapytania omówię w rozdziale 3.

Na przykład, nawet jeżeli poniższe zapytanie wykonuje złączenie czterech tabel i wymaga sortowania, i tak kończy się przedwcześnie z komunikatem o odnalezieniu wystarczająco dobrego planu:

```
SELECT pm.ProductModelID, pm.Name, Description, pl.CultureID, cl.Name AS Language
FROM Production.ProductModel AS pm
  JOIN Production.ProductModelProductDescriptionCulture AS pl
    ON pm.ProductModelID = pl.ProductModelID
  JOIN Production.Culture AS cl
    ON cl.CultureID = pl.CultureID
  JOIN Production.ProductDescription AS pd
    ON pd.ProductDescriptionID = pl.ProductDescriptionID
ORDER BY pm.ProductModelID
```

Atrybut `CardinalityEstimationModelVersion` (wersja modelu szacowania kardynalności) odnosi się do wersji modelu szacowania kardynalności wykorzystanej w optymalizatorze zapytań. SQL Server 2014 wykorzystuje nowy sposób szacowania kardynalności, masz jednak możliwość korzystania również ze starego sposobu, poprzez zmianę trybu kompatybilności bazy danych lub dzięki flagom 2312 i 9481. Więcej szczegółów dotyczących obu modeli szacowania kardynalności znajdziesz w rozdziale 6.

Opcjonalny atrybut `NonParallelPlanReason` (powód wykorzystania planu nierównoległego) elementu `QueryPlan`, który został wprowadzony w SQL Serverze 2012, zawiera informację, dlaczego plan równoległy nie może zostać wybrany dla zapytania. Chociaż lista możliwych wartości nie jest udokumentowana, najczęściej spotykane są poniższe:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 43666
OPTION (MAXDOP 1)
```

Ponieważ skorzystaliśmy z `MAXDOP 1`, właściwość przyjmie wartość:

```
NonParallelPlanReason="MaxDOPSetToOne"
```

Wykorzystanie tej funkcji:

```
SELECT CustomerID,('AW' + dbo.ufnLeadingZeros(CustomerID))
AS GenerateAccountNumber
```

```
FROM Sales.Customer
ORDER BY CustomerID;
```

wygeneruje następującą wartość:

```
NonParallelPlanReason="CouldNotGenerateValidParallelPlan"
```

Jeżeli w systemie z jednym procesorem spróbujesz wykonać poniższe zapytanie:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 43666
OPTION (MAXDOP 8)
```

otrzymasz taki wynik:

```
NonParallelPlanReason="EstimatedDOPIsOne"
```

W schemacie XSD znajduje się wprowadzony również w SQL Serverze 2012 element `OptimizerHardwareDependentProperties` (właściwości optymalizatora uzależnione od sprzętu), zawierający właściwości uzależnione od sprzętu, które mogą mieć wpływ na wybór planu. Element ten ma poniższe udokumentowane właściwości:

- **`EstimatedAvailableMemoryGrant`** — szacowana ilość pamięci (w kB), która będzie dostępna w momencie wykonania zapytania.
- **`EstimatedPagesCached`** — szacowana liczba stron danych, które pozostaną w puli bufora, jeżeli zapytanie będzie musiało powtórnie czytać.
- **`EstimatedAvailableDegreeOfParallelism`** — szacowana liczba procesorów, które będą mogły być wykorzystane do wykonania zapytania, gdyby optymalizator wybrał plan równoległy.

Na przykład zapytanie:

```
SELECT DISTINCT(CustomerID)
FROM Sales.SalesOrderHeader
```

zwróci poniższą wartość:

```
<OptimizerHardwareDependentProperties EstimatedAvailableMemoryGrant="101808"
EstimatedPagesCached="8877" EstimatedAvailableDegreeOfParallelism="2" />
```

Ostrzeżenia w planach wykonania

Plany wykonania mogą również zawierać komunikaty ostrzeżeń. Plany zawierające ostrzeżenia powinny być szczegółowo analizowane, ponieważ ich wystąpienie może wskazywać na to, że optymalizator wybiera mniej optymalne plany. Przed SQL Serverem 2012 występowały tylko ostrzeżenia `ColumnsWithNoStatistics` (kolumny bez statystyk) i `NoJoinPredicate` (brak predykatów złączenia). Schemat XSD z wersji SQL Server 2012 dodaje sześć nowych ostrzeżeń iteratora i zapytania:

- `SpillToTempDb` (przekazywanie danych do bazy tymczasowej),
- `Wait` (oczekiwanie),
- `PlanAffectingConvert` (konwersje wpływające na plan),
- `SpatialGuess` (przypuszczenie przestrzenne),
- `UnmatchedIndexes` (niepasujące indeksy),
- `FullUpdateForOnlineIndexBuild` (pełna aktualizacja dla indeksu online).

Omówmy kilka z nich.

ColumnsWithNoStatistics

To ostrzeżenie oznacza, że optymalizator próbował wykorzystać statystyki, ale nie były one dostępne. Jak tłumaczyłem we wcześniejszej części rozdziału, optymalizator polega na statystykach w celu stworzenia optymalnego planu. Aby zasymulować wystąpienie ostrzeżenia, wykonaj poniższe polecenia.

Wykonaj następujące polecenie, aby usunąć (jeżeli istnieją) statystyki dla kolumny `VacationHours`:

```
DROP STATISTICS HumanResources.Employee._WA_Sys_0000000C_49C3F6B7
```

Następnie tymczasowo wyłącz automatyczne tworzenie statystyk na poziomie bazy danych:

```
ALTER DATABASE AdventureWorks2012 SET AUTO_CREATE_STATISTICS OFF
```

Wreszcie uruchom następujące zapytanie:

```
SELECT * FROM HumanResources.Employee
WHERE VacationHours = 48
```

Otrzymasz plan pokazany na rysunku 1.12.

Rysunek 1.12. Plan z ostrzeżeniem ColumnsWithNoStatistics

Zwróć uwagę na ostrzeżenie (symbol ze znakiem wykrzyknika) dla operatora *Clustered Index Scan*. Jeżeli spojrzysz na właściwości tego operatora, zobaczysz wartość: *Columns With No Statistics: [AdventureWorks2012].[HumanResources].[Employee].VacationHours*.

Nie zapomnij o powtórnym włączeniu automatycznego tworzenia statystyk (uruchom poniższe zapytanie). Nie trzeba tworzyć usuniętego wcześniej obiektu statystyk, ponieważ jeżeli będzie potrzebny, może zostać utworzony automatycznie.

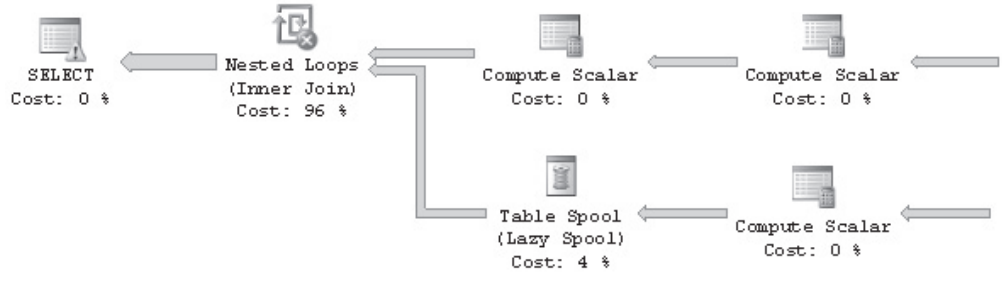
```
ALTER DATABASE AdventureWorks2012 SET AUTO_CREATE_STATISTICS ON
```

NoJoinPredicate

Wykorzystanie składni połączeń ze specyfikacji ANSI SQL-89 związane jest z ryzykiem nieumyślnego pominięcia predykatów złączenia, co z kolei wiąże się z wystąpieniem ostrzeżenia `NoJoinPredicate`. Załóżmy, że chcesz uruchomić poniższe zapytanie, ale zapomnisz o klauzuli `WHERE`:

```
SELECT * FROM Sales.SalesOrderHeader soh, Sales.SalesOrderDetail sod
WHERE soh.SalesOrderID = sod.SalesOrderID
```

Pierwszą wskazówką, że wystąpił problem, może być długi czas wykonywania zapytania, nawet dla małych tabel. Później zobaczysz także, że wynik zwracany przez zapytanie jest bardzo obszerny. Czasami dobrym sposobem na szukanie problemów z długo wykonującymi się zapytaniami jest zatrzymanie ich i zażądanie szacowanego planu zapytania. Jeżeli nie dołączysz predykatu złączenia (w klauzuli `WHERE`), otrzymasz plan zgodny z rysunkiem 1.13.



Rysunek 1.13. Plan z ostrzeżeniem NoJoinPredicate

Tym razem ostrzeżenie `NoJoinPredicate` znajduje się na operacji *Nested Loops Join* i ma inną ikonę. Zauważ, że jeżeli korzystasz ze składni złączeń ze specyfikacji ANSI SQL-92, nie możesz ominąć predykatów złączenia, ponieważ otrzymasz błąd, dlatego właśnie ta składnia jest zalecana. Na przykład ominięcie predykatu złączenia z poniższego zapytania zwróci błąd składni:

```
SELECT * FROM Sales.SalesOrderHeader soh JOIN Sales.SalesOrderDetail sod
-- ON soh.SalesOrderID = sod.SalesOrderID
```

UWAGA

Jeżeli zajdzie taka potrzeba, możesz otrzymać po jednym wierszu dla wszystkich możliwych par wierszy z dwóch tabel — taki wynik nazywa się również *wynikiem kartezjańskim* i możesz go uzyskać, stosując składnię *CROSS JOIN*.

PlanAffectingConvert

To ostrzeżenie pokazuje, że wykonane były konwersje typów, które mogły mieć wpływ na wydajność wynikowego planu zapytania. Wykonaj poniższy przykład, w którym najpierw deklarowana jest zmienna typu `nvarchar`, a następnie jest ona porównywana do kolumny typu `varchar`, `CreditCardApprovalCode`:

```
DECLARE @code nvarchar(15)
SET @code = '95555Vi4081'
SELECT * FROM Sales.SalesOrderHeader
WHERE CreditCardApprovalCode = @code
```

Dla zapytania stworzony zostanie plan zgodny z rysunkiem 1.14.

Rysunek 1.14. Plan z ostrzeżeniem PlanAffectingConvert

Ikona ostrzeżenia na operacji `SELECT` dotyczy dwóch poniższych ostrzeżeń:

```
Type conversion in expression
(CONVERT_IMPLICIT(nvarchar(15),[AdventureWorks2012].[Sales].[SalesOrderHeader].
↪[CreditCardApprovalCode],0)) may affect "CardinalityEstimate" in query plan choice,
Type conversion in expression
(CONVERT_IMPLICIT(nvarchar(15),[AdventureWorks2012].[Sales].[SalesOrderHeader].
↪[CreditCardApprovalCode],0)=[@code]) may affect "SeekPlan" in query plan choice
```

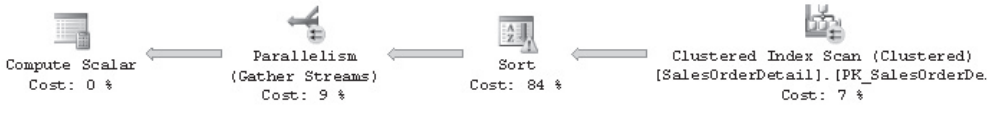
Oczywiście zalecane jest wykorzystywanie podobnych typów danych podczas porównań.

SpillToTempDb

To ostrzeżenie wskazuje, że operacja nie miała wystarczająco dużo pamięci i musiała podczas wykonywania zapytania przenieść dane na dysk, co może być problemem ze względu na dodatkowe operacje I/O. Aby zasymulować ten problem, wykonaj poniższe zapytanie:

```
SELECT * FROM Sales.SalesOrderDetail
ORDER BY UnitPrice
```

Jest to bardzo proste zapytanie, a uzyskanie ostrzeżenia jest uzależnione od ilości pamięci w systemie, być może więc będziesz musiał spróbować z większą tabelą. Wygenerowany zostanie plan jak na rysunku 1.15.



Rysunek 1.15. Plan z ostrzeżeniem SpillToTempDb

Ostrzeżenie jest pokazywane na operatorze *Sort* i zawiera komunikat *Operator used tempdb to spill data during execution with spill level 1* (operator podczas wykonywania zapytania przekazał dane do bazy tymczasowej z poziomem przekazania 1). Plan XML zawiera również to:

```
<SpillToTempDb SpillLevel="1" />
```

UnmatchedIndexes

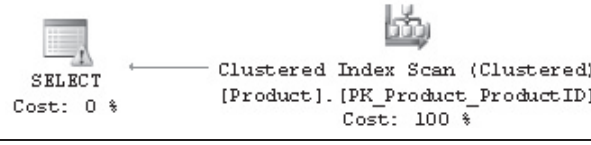
Ostrzeżenie `UnmatchedIndexes` informuje, że optymalizator nie był w stanie dopasować indeksu filtrowanego do danego zapytania (na przykład kiedy nie może zobaczyć wartości parametru). Załóżmy, że stworzyłeś poniższy indeks filtrowany:

```
CREATE INDEX IX_Color ON Production.Product(Name, ProductNumber)
WHERE Color = 'White'
```

a następnie wykonałeś następujące zapytanie:

```
DECLARE @color nvarchar(15)
SET @color = 'White'
SELECT Name, ProductNumber FROM Production.Product
WHERE Color = @color
```

Indeks *IX_Color* nie zostanie wykorzystany, a plan będzie zawierał ostrzeżenie (zobacz rysunek 1.16).



Rysunek 1.16. Plan z ostrzeżeniem UnmatchedIndexes

Na planie XML będziesz mógł zobaczyć następujące informacje (lub zaglądając do właściwości `UnmatchedIndexes` we właściwościach operatora `SELECT`):

```
<UnmatchedIndexes>
  <Parameterization>
    <Object Database="[AdventureWorks2012]" Schema="[Production]"
      Table="[Product]" Index="[IX_Color]" />
  </Parameterization>
</UnmatchedIndexes>
<Warnings UnmatchedIndexes="true" />
```

Natomiast dla poniższego zapytania indeks zostanie wykorzystany:

```
SELECT Name, ProductNumber FROM Production.Product
WHERE Color = 'White'
```

Indeksy filtrowane i element `UnmatchedIndexes` omówię dokładnie w rozdziale 5. Na razie usuń indeks, który utworzyliśmy wcześniej:

```
DROP INDEX Production.Product.IX_Color
```

UWAGA

Wiele ćwiczeń w tej książce będzie wymagało wprowadzania zmian w bazie `AdventureWorks2012`. Chociaż baza jest przywracana do stanu pierwotnego, możesz również rozważyć przywrócenie świeżej kopii bazy po wykonaniu serii ćwiczeń.

Pobieranie planów za pomocą śledzenia lub z magazynu planów

Do tej pory testowaliśmy pobieranie planów wykonania poprzez bezpośrednie wykorzystanie kodu zapytania w Management Studio. Jednakże pobieranie planu w ten sposób nie zawsze jest możliwe, a czasami będziesz musiał przechwycić plan wykonywania z innych lokalizacji (na przykład z magazynu danych lub z puli aktualnie wykonywanych zapytań). W takich przypadkach może będziesz musiał pobrać plan za pośrednictwem śledzenia, na przykład korzystając ze zdarzeń śledzenia lub z rozszerzonych zdarzeń, albo z magazynu planów za pomocą dynamicznej funkcji zarządzania (*Dynamic Management Function* — DMF) `sys.dm_exec_query_plan` lub być może z wykorzystaniem danych zebranych przez komponent SQL Server Data Collector. Przyjrzyjmy się niektórym z tych opcji.

sys.dm_exec_query_plan

Jak wspomniałem wcześniej, kiedy zapytanie jest optymalizowane, jego plan wykonywania może być przechowywany w magazynie planów, a funkcja `sys.dm_exec_query_plan` może zostać wykorzystana do zwrócenia zachowanego planu, a także dowolnego planu, który jest aktualnie wykonywany. Jeżeli jednak plan zostanie usunięty z magazynu, przestanie być dostępny, a kolumna `query_plan` zwracanej tabeli będzie miała wartość `null`.

Na przykład poniższe zapytanie pokaże plany wykonania dla wszystkich zapytań aktualnie wykonywanych w systemie. Dynamiczny widok zarządzania (*Dynamic Management View* — DMV) `sys.dm_exec_requests`, który zwraca informacje o wszystkich aktualnie wykonywanych żądaniach, jest potrzebny do uzyskania wartości `plan_handle` (uchwyt planu), która z kolei jest niezbędna do pobrania planu poprzez wykorzystanie funkcji `sys.dm_exec_query_plan`. Wartość `plan_handle` to unikalny w obrębie systemu hasz reprezentujący konkretny plan wykonania.

```
SELECT * FROM sys.dm_exec_requests
CROSS APPLY
sys.dm_exec_query_plan(plan_handle)
```

W wyniku tego zapytania otrzymujemy rezultat zawierający kolumnę `query_plan`, która zawiera linki podobne do tych z podrozdziału dotyczącego planów XML. Jak już tłumaczyłem, kliknięcie linku spowoduje wyświetlenie planu graficznego.

W ten sam sposób poniższy przykład wyświetla wszystkie plany przechowywane w magazynie planów. Widok `sys.dm_exec_query_stats` zawiera jeden wiersz dla każdego zapytania i udostępnia wartość `plan_handle` potrzebną dla funkcji `sys.dm_exec_query_plan`.

```
SELECT * FROM sys.dm_exec_query_stats
CROSS APPLY
sys.dm_exec_query_plan(plan_handle)
```

Załóżmy teraz, że chcesz znaleźć 10 zapytań najbardziej kosztownych pod względem wykorzystania czasu procesora. Do uzyskania takiej informacji możesz wykorzystać poniższe zapytanie, które zwróci średni czas procesora w mikrosekundach dla wykonania:

```
SELECT TOP 10 total_worker_time/execution_count AS avg_cpu_time,
plan_handle, query_plan
FROM sys.dm_exec_query_stats
CROSS APPLY sys.dm_exec_query_plan(plan_handle)
ORDER BY avg_cpu_time DESC
```

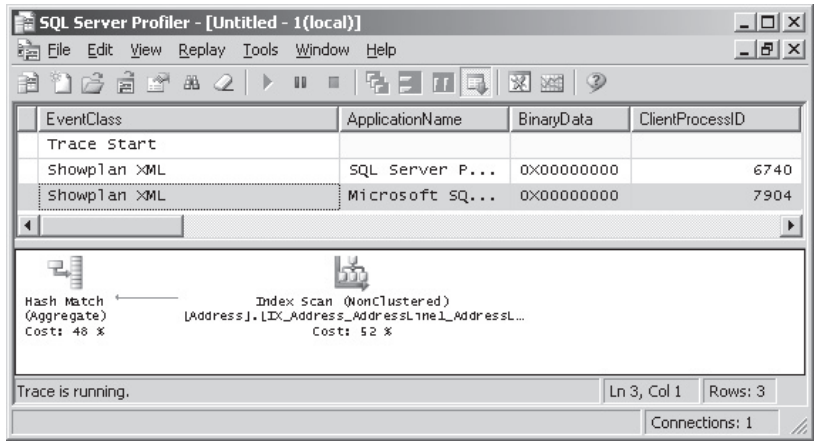
SQL Trace/Profiler

Możesz również wykorzystać program SQL Profiler do przechwytywania planów wykonania aktualnie wykonywanych zapytań. Możesz skorzystać z kategorii zdarzenia *Performance* (wydajność), która zawiera następujące zdarzenia:

- *Performance Statistics* (statystyki wydajności),
- *Showplan All* (pokaż plan — wszystko),
- *Showplan All For Query Compile* (pokaż plan — wszystko dla kompilacji zapytania),
- *Showplan Statistics Profile* (pokaż plan — profil statystyk),
- *Showplan Text* (pokaż plan — tekst),
- *Showplan Text (Unencoded)* (pokaż plan — tekst bez kodowania),
- *Showplan XML* (pokaż plan — XML),
- *Showplan XML For Query Compile* (pokaż plan — XML dla kompilacji zapytania),
- *Showplan XML Statistics Profile* (pokaż plan — XML dla profilu statystyk).

Aby prześledzić któreś z tych zdarzeń, uruchom program Profiler, połącz się ze swoją instancją SQL Servera, kliknij zakładkę *Events Selection* (wybór zdarzeń), rozwiń kategorię *Performance* i wybierz te zdarzenia, które są dla Ciebie interesujące. Możesz wybrać wszystkie kolumny lub tylko ich część, wybrać filtr kolumn i tak dalej. Kliknij przycisk *Run* (uruchom), aby rozpocząć śledzenie. Rysunek 1.17 przedstawia przykład śledzenia dla zdarzenia *Showplan XML*.

Możesz również utworzyć proces śledzenia, korzystając ze skryptu, a nawet wykorzystać program Profiler jako narzędzie do tworzenia skryptów. Aby to zrobić, zdefiniuj zdarzenia do śledzenia, uruchom i zatrzymaj proces śledzenia, a następnie wybierz *File/Export/Script Trace Definition/For SQL Server 2005 – 2014…* (plik/eksport/



Rysunek 1.17. Śledzenie w programie Profiler dla zdarzenia Showplan XML

definicja skryptu śledzenia/dla SQL Server 2005-2014…). Dzięki temu uzyskasz kod pozwalający uruchomić proces śledzenia, który będzie wymagał tylko podania nazwy pliku, w którym mają być zapisywane przechwycone dane. Część wygenerowanego kodu została pokazana na listingu poniżej:

```
/****************************************************/
/* Created by: SQL Server 2014 Profiler */
/* Date: 12/18/2013 08:37:22 AM */
/****************************************************/
-- Create a Queue
declare @rc int
declare @TraceID int
declare @maxfilesize bigint
set @maxfilesize = 5

-- Please replace the text InsertFileNameHere, with an appropriate
-- filename prefixed by a path, e.g., c:\MyFolder\MyTrace. The .trc extension
-- will be appended to the filename automatically. If you are writing from
-- remote server to local drive, please use UNC path and make sure server has
-- write access to your network share
exec @rc = sp_trace_create @TraceID output, 0, N'InsertFileNameHere', @maxfilesize,
NULL
if (@rc != 0) goto error

-- Client side File and Table cannot be scripted

-- Set the events
declare @on bit
set @on = 1
exec sp_trace_setevent @TraceID, 10, 1, @on
exec sp_trace_setevent @TraceID, 10, 9, @on
exec sp_trace_setevent @TraceID, 10, 2, @on
exec sp_trace_setevent @TraceID, 10, 66, @on
exec sp_trace_setevent @TraceID, 10, 10, @on
exec sp_trace_setevent @TraceID, 10, 3, @on
exec sp_trace_setevent @TraceID, 10, 4, @on
```

```
exec sp_trace_setevent @TraceID, 10, 6, @on
exec sp_trace_setevent @TraceID, 10, 7, @on
exec sp_trace_setevent @TraceID, 10, 8, @on
exec sp_trace_setevent @TraceID, 10, 11, @on
exec sp_trace_setevent @TraceID, 10, 12, @on
exec sp_trace_setevent @TraceID, 10, 13, @on
```

UWAGA

W wersji 2008 wszystkie zdarzenia niezwiązane z XML-em, takie jak *Showplan All* czy *Showplan Text*, zostały zdeprecjonowane. Począwszy od tej wersji, Microsoft zaleca korzystanie ze zdarzeń XML. Także SQL Trace zostało zdeprecjonowane, w wersji 2012, i zalecane jest korzystanie ze zdarzeń rozszerzonych.

Więcej szczegółów na temat korzystania z Profilera i SQL Trace znajdziesz w dokumentacji SQL Server Books Online.

Zdarzenia rozszerzone

Do przechwytywania planów zapytania możesz również wykorzystać zdarzenia rozszerzone. Chociaż Microsoft zaleca korzystanie ze zdarzeń rozszerzonych zamiast SQL Trace, jak wcześniej wspomniałem, w aktualnych wersjach SQL Servera zdarzenia przechwytujące plany zapytania są kosztowne. Dokumentacja zawiera następujące ostrzeżenie dla wszystkich trzech zdarzeń rozszerzonych dotyczących przechwytywania planów wykonania: „Wykorzystanie tego zdarzenia może mieć znaczący wpływ na spadek wydajności, więc powinno być wykorzystywane tylko do szukania problemów lub monitorowania konkretnych problemów przez krótkie okresy czasu".

Możesz stworzyć i uruchomić sesję zdarzeń rozszerzonych za pomocą polecenia `CREATE EVENT SESSION` i `ALTER EVENT SESSION`. Możesz także skorzystać z nowego interfejsu graficznego wprowadzonego w SQL Serverze 2012. Oto zdarzenia związane z planami wykonania:

- **`query_post_compilation_showplan`** — występuje po skompilowaniu polecenia SQL. To zdarzenie zwraca reprezentację szacowanego planu wykonania w formacie XML, który generowany jest po skompilowaniu zapytania.
- **`query_post_execution_showplan`** — występuje po wykonaniu polecenia SQL. To zdarzenie zwraca reprezentację właściwego planu zapytania w formacie XML.
- **`query_pre_execution_showplan`** — występuje po skompilowaniu polecenia SQL. To zdarzenie zwraca reprezentację szacowanego planu wykonania w formacie XML, który generowany jest po optymalizacji zapytania.

Załóżmy na przykład, że chcesz rozpocząć sesję śledzenia zdarzenia `query_post_execution_showplan`. Do stworzenia takiej sesji możesz wykorzystać poniższe zapytanie:

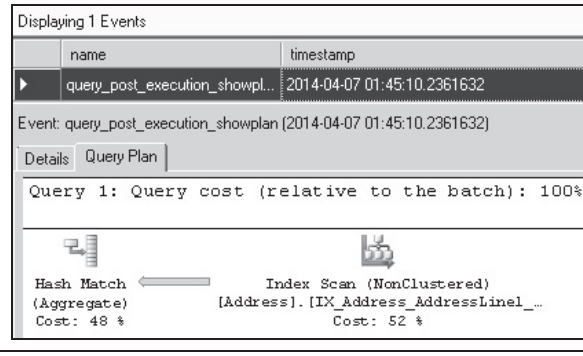
```
CREATE EVENT SESSION [test] ON SERVER
ADD EVENT sqlserver.query_post_execution_showplan(
    ACTION(sqlserver.plan_handle)
    WHERE ([sqlserver].[database_name]=N'AdventureWorks2012'))
ADD TARGET package0.ring_buffer
WITH (STARTUP_STATE=OFF)
GO
```

Zdarzenia rozszerzone omówię dokładniej w rozdziale 2. Na razie możesz zwrócić uwagę na to, że argument `ADD EVENT` zawiera nazwę zdarzenia (w tym przypadku `query_post_execution_showplan`), `ACTION` odnosi się do pól globalnych, które mają zostać przechwycone w sesji zdarzenia (w tym wypadku `plan_handle`), a `WHERE` jest wykorzystywane do zdefiniowania filtra do ograniczenia danych, które mają zostać przechwycone. Predykat `[sqlserver].[database_name]=N'AdventureWorks2012'` wskazuje, że chcemy przechwytywać zdarzenia wyłączenie dla bazy `AdventureWorks2012`. `TARGET` to konsument zdarzenia i możemy wykorzystać ten parametr do zbierania danych do analizy. W tym przypadku celem jest `ring_buffer`. Wreszcie `STARTUP_STATE` to jedna z opcji zdarzenia rozszerzonego i wykorzystywana jest do określenia, czy sesja powinna być uruchamiana automatycznie po starcie SQL Servera.

Po stworzeniu sesji możesz ją uruchomić, korzystając z polecenia `ALTER EVENT SESSION`:

```
ALTER EVENT SESSION [test]
ON SERVER
STATE=START
```

Do podglądu danych przechwyconych przez sesję zdarzenia rozszerzonego możesz skorzystać z funkcjonalności *Watch Live Data* (podgląd danych na żywo), wprowadzonej w SQL Serverze 2012. Aby to zrobić, w oknie *Object Explorer* (eksplorator obiektów) rozwiń katalog *Management/Extended Events/Sessions*, a następnie kliknij prawym przyciskiem na sesji i wybierz *Watch Live Data*. Rysunek 1.18 pokazuje przykład przechwycenia planu wykonania.



Rysunek 1.18. Funkcjonalność podglądania danych na żywo

Aby zobaczyć te dane, możesz również wykorzystać poniższy kod:

```
SELECT
  event_data.value('(event/@name)[1]', 'varchar(50)') AS event_name,
  event_data.value('(event/action[@name="plan_handle"]/value)[1]',
    'varchar(max)') as plan_handle,
  event_data.query('event/data[@name="showplan_xml"]/value/*') as showplan_xml,
  event_data.value('(event/action[@name="sql_text"]/value)[1]',
    'varchar(max)') AS sql_text
FROM( SELECT evnt.query('.') AS event_data
  FROM
    ( SELECT CAST(target_data AS xml) AS target_data
      FROM sys.dm_xe_sessions AS s
      JOIN sys.dm_xe_session_targets AS t
        ON s.address = t.event_session_address
      WHERE s.name = 'test' AND t.target_name = 'ring_buffer'
    ) AS data
  CROSS APPLY target_data.nodes('RingBufferTarget/event') AS xevent(evnt)
  ) AS xevent(event_data)
```

Kiedy skończysz testy, musisz zatrzymać lub usunąć sesję. Uruchom następujące polecenia:

```
ALTER EVENT SESSION [test]
ON SERVER
STATE=STOP
GO
DROP EVENT SESSION [test] ON SERVER
```

Jest jeszcze kilka innych narzędzi SQL Servera pozwalających przeglądać plany, w tym narzędzie Data Collector, wprowadzone w SQL Serverze 2008. Omówię je w rozdziale 2.

Usuwanie planów z magazynu planów

Możesz wykorzystać kilka różnych poleceń do usuwania planów z magazynu planów. Te polecenia, dokładniej omówione w rozdziale 8., mogą być użyteczne podczas testowania i nie powinny być uruchamiane w środowisku produkcyjnym, jeżeli nie jesteśmy pewni, jaki efekt chcemy osiągnąć. Polecenie `DBCC FREEPROCCACHE` jest wykorzystywane do usuwania wszystkich wpisów z magazynu. Może również przyjmować uchwyt do planu lub uchwyt SQL w celu usuwania tylko pojedynczych planów, może też przyjmować nazwę puli *Resource Governor* (zarządca zasobów) i usuwać plany z nią związane. Polecenia `DBCC FREESYSTEMCACHE` można użyć do usunięcia wszystkich elementów z magazynu planów lub tylko elementów skojarzonych z nazwą puli *Resource Governor*. `DBCC FLUSHPROCINDB` można wykorzystać do usunięcia wszystkich planów dla konkretnej bazy danych.

Chociaż nie jest ono związane z magazynem planów, można też wykorzystać polecenie `DBCC DROPCLEANBUFFERS` do usunięcia wszystkich buforów z puli buforów. Możesz skorzystać z tego polecenia, gdy zechcesz zasymulować uruchomienie zapytania z pustą pamięcią podręczną, tak jak zrobimy to w kolejnym podrozdziale.

SET STATISTICS TIME i SET STATISTICS IO

Zamkniemy ten rozdział dwoma poleceniami, które mogą udzielić Ci dodatkowych informacji o Twoich zapytaniach i które pozwolą Ci skorzystać z dodatkowych technik poprawiania wydajności zapytań. Mogą być doskonałym uzupełnieniem planów wykonania, jeżeli chodzi o informacje na temat wykonania i optymalizacji zapytań. Często zdarza się, że programiści próbują porównywać koszt zapytania z wydajnością zapytania. Nie powinieneś zakładać bezpośredniej korelacji pomiędzy szacowanym kosztem zapytania a wydajnością planu. Koszt to wewnętrzna jednostka wykorzystywana przez optymalizator zapytań i nie powinna być stosowana do porównywania wydajności planu; zamiast tego możesz użyć `SET STATISTICS TIME` i `SET STATISTICS IO`. Ten podrozdział opisuje oba polecenia.

Polecenie `SET STATISTICS TIME` możesz wykorzystać do sprawdzanie, jak długo, w milisekundach, trwa parsowanie, kompilacja i wykonanie zapytania. Na przykład uruchom poniższe polecenie:

```
SET STATISTICS TIME ON
```

a następnie poniższe zapytanie:

```
SELECT DISTINCT(CustomerID)
FROM Sales.SalesOrderHeader
```

Aby zobaczyć wynik, będziesz musiał przełączyć się do zakładki *Messages* (komunikaty) okna wyników; zobaczysz wynik podobny do tego:

```
SQL Server parse and compile time:
  CPU time = 16 ms, elapsed time = 226 ms.
SQL Server Execution Times:
  CPU time = 16 ms, elapsed time = 148 ms.
```

Fragment „parse and compile” odnosi się do czasu potrzebnego na optymalizację zapytania. Polecenie `SET STATISTICS TIME` będzie włączone dla wszystkich wykonywanych zapytań. Możesz wyłączyć śledzenie czasu następującym poleceniem:

```
SET STATISTICS TIME OFF
```

Jak już wspomniałem, informacje dotyczące parsowania i kompilacji można także zobaczyć na planie wykonania:

```
<QueryPlan DegreeOfParallelism="1" CachedPlanSize="16" CompileTime="226" CompileCPU="9"
➥CompileMemory="232">
```

Oczywiście, jeżeli potrzebujesz tylko informacji o czasie wykonywania każdego zapytania, możesz otrzymać tę informację bezpośrednio w edytorze zapytań Management Studio Query Editor.

Polecenie `SET STATISTICS IO` pozwala uzyskać informację o ilości operacji dyskowych generowanych przez zapytanie. Aby je uruchomić, wykonaj następujące polecenie:

```
SET STATISTICS IO ON
```

Wykonaj poniższe polecenie, aby opróżnić bufory z puli buforów i zyskać dzięki temu pewność, że dla tej tabeli żadne strony nie są załadowane do pamięci:

```
DBCC DROPCLEANBUFFERS
```

Następnie wykonaj poniższe zapytanie:

```
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID = 870
```

Otrzymasz wynik podobny do tego:

```
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 3,
↪read-ahead reads 1277, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Oto definicje tych pozycji (wszystkie korzystają ze stron o rozmiarze 8 kB):

- ***Logical reads*** — liczba stron przeczytanych z puli bufora.
- ***Physical reads*** — liczba stron przeczytanych z dysku.
- ***Read-ahead reads*** — *Read-ahead* to mechanizm optymalizacji wydajności, który przewiduje, jakie strony będą potrzebne, i odczytuje je z dysku. Może przeczytać do 64 stron z jednego pliku danych.
- ***Lob logical reads*** — liczba stron LOB (ang. *large object* — duże obiekty) przeczytanych z puli bufora.
- ***Lob physical reads*** — liczba stron LOB przeczytanych z dysku.
- ***Lob read-ahead reads*** — liczba stron LOB przeczytanych z dysku za pomocą mechanizmu *Read-ahead*.

Jeżeli teraz uruchomisz to samo zapytanie, nie będzie żadnych fizycznych odczytów mechanizmu *Read-ahead*, a wynik będzie podobny do poniższego:

```
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0,
↪read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Atrybut *scan count* jest definiowany jako liczba wyszukań lub skanów rozpoczętych po osiągnięciu poziomu liścia (czyli najniższego poziomu indeksu). Jedyny przypadek, w którym wartość ta będzie równa 0, to taki, kiedy szukasz jednej wartości dla indeksu unikalnego, jak w poniższym przykładzie:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 51119
```

Jeżeli wypróbujesz poniższe zapytanie, w którym `SalesOrderID` będzie zdefiniowane jako indeks nieunikalny mogący zwracać więcej niż jeden wiersz, zobaczysz, że atrybut *scan count* przyjmie wartość 1.

```
SELECT * FROM Sales.SalesOrderDetail
WHERE SalesOrderID = 51119
```

Wreszcie dla poniższego przykładu atrybut *scan count* przyjmie wartość 4, ponieważ SQL Server będzie musiał uruchomić cztery wyszukiwania:

```
SELECT * FROM Sales.SalesOrderHeader
WHERE SalesOrderID IN (51119, 43664, 63371, 75119)
```

Podsumowanie

W tym rozdziale pokazałem, w jaki sposób lepsze zrozumienie tego, co robi procesor zapytań, może pomóc zarówno administratorom baz, jak i programistom pisać lepsze zapytania i zapewniać optymalizatorowi zapytań informacje, których potrzebuje do tworzenia wydajnych planów zapytania. Pokazałem też, jak możesz wykorzystać nową wiedzę dotyczącą działania procesora zapytań i narzędzi wbudowanych w SQL Server do analizowania przypadków, w których zapytania nie zachowują się zgodnie z oczekiwaniami. Na tej podstawie przedstawiłem najważniejsze informacje dotyczące optymalizatora zapytań, silnika wykonywania i magazynu planów, a komponenty te omówię dokładniej w dalszej części książki.

Ponieważ plany zapytań będziemy wykorzystywać w całej książce, pokazałem również, jak je czytać, jakie są ich najważniejsze właściwości i jak pobierać je ze źródeł takich jak magazyn planów czy za pomocą śledzenia aktywności serwera. Ta część powinna dać Ci wystarczającą wiedzę do zrozumienia reszty książki. Wprowadziłem także pojęcie operatorów zapytań, ale omówię je dokładniej w rozdziale 4. i w dalszej części książki.

W następnym rozdziale pokażę Ci dodatkowe narzędzia pozwalające poprawić wydajność zapytań oraz techniki takie jak SQL Trace, zdarzenia rozszerzone i widoki DMV, które pozwolą ocenić, jakie zapytania zużywają najwięcej zasobów, i pomogą odnaleźć inne problemy związane z wydajnością.

Skorowidz

L

M

N

O

P

Q

R

S

Ś

T

U

W

X

Z

Microsoft SQL Server 2014 to najnowsza wersja serwera bazodanowego firmy Microsoft. Ta platforma jest rozwijana od ponad 25 lat, a każda kolejna wersja wprowadza serię ulepszeń — zarówno w obszarze możliwości, jak i wydajności. Jednak sam rozwój serwera nie wystarczy, żeby błyskawicznie wyciągać z bazy danych kluczowe informacje. Konieczna jest także optymalizacja parametrów jego pracy oraz zadawanych zapytań SQL.

Jak to zrobić? Na to i wiele innych pytań odpowiada ta niepowtarzalna książka. Została ona w całości poświęcona optymalizacji serwera oraz zapytań SQL. W kolejnych rozdziałach znajdziesz bezcenne informacje na temat rozwiązywania problemów z zapytaniami oraz optymalizacją zapytań. Ponadto zrozumiesz sposób działania optymalizatora, zalety dynamicznych widoków oraz znaczenie dobrze wybranych indeksów. Microsoft SQL Server zbiera liczne informacje na temat swojej pracy — możesz je wykorzystać do podkręcenia jego osiągów. Ta książka jest lekturą obowiązkową każdego administratora, który ma do czynienia z Microsoft SQL Server!

Dzięki tej książce:

- zrozumiesz sposób działania optymalizatora zapytań
- przeanalizujesz i zoptymalizujesz parametry pracy serwera
- użyjesz dynamicznych widoków
- przekonasz się o wpływie indeksów na wydajność bazy
- w pełni wykorzystasz możliwości Microsoft SQL Server 2014

Zmuś Microsoft SQL Server do pracy na najwyższych obrotach!

Benjamin Nevarez — MVP w dziedzinie SQL Server. Niezależny konsultant. Ma ponad 15-letnie doświadczenie w pracy z bazą danych SQL Server. Główny obszar jego zainteresowań obejmuje optymalizację zapytań SQL.

Helion

36680 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1162-6

9 788328 311626

Informatyka w najlepszym wydaniu

cena: 69,00 zł