Microsoft

Itzik Ben-Gan

Microsoft® SQL Server® 2012

Optymalizacja kwerend T-SQL przy użyciu funkcji okna

przełożyła Natalia Chounlamany

APN Promise, Warszawa 2012

Microsoft® SQL Server® 2012. Optymalizacja kwerend T-SQL przy użyciu funkcji okna

Authorized Polish translation of English edition of **Microsoft® SQL Server® 2012 High-Performance T-SQL Using Window Functions** ISBN: 978-0-7356-5836-3

APN PROMISE SA, biuro: ul. Kryniczna 2, 03-934 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

Książka ta przedstawia poglądy i opinie autora. Przykłady firm, produktów, osób i wydarzeń opisane w niniejszej książce są fikcyjne i nie odnoszą się do żadnych konkretnych firm, produktów, osób i wydarzeń, chyba że zostanie jednoznacznie stwierdzone, że jest inaczej. Ewentualne podobieństwo do jakiejkolwiek rzeczywistej firmy, organizacji, produktu, nazwy domeny, adresu poczty elektronicznej, logo, osoby, miejsca lub zdarzenia jest przypadkowe i niezamierzone.

ISBN: 978-83-7541-098-3

Przekład: Natalia Chounlamany
Redakcja: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Przedmowa

SQL to bardzo interesujący język programowania. Spotkania z klientami stale przypominają mi, że można patrzeć na niego z różnych perspektyw. Wiele osób rozpoczynających pracę z językiem SQL postrzega go jako prosty język programowania oferujący cztery główne konstrukcje: SELECT, INSERT, UPDATE oraz DELETE. Niektóre osoby pozostają na tym poziomie wtajemniczenia. Inni odkrywają, że można filtrować wiersze zwrócone przez kwerendę przy pomocy klauzuli WHERE i od czasu do czasu korzystają z klauzuli JOIN. Jednak tylko ci, którzy poświęcą więcej czasu na zapoznanie się z technologią SQL i jej deklaratywnym, relacyjnym i opartym na zbiorach modelem, mogą się przekonać, że jest to zaawansowany język programowania o szerokich możliwościach.

Jedno z najważniejszych rozszerzeń języka SQL stanowią dodane w wersji Microsoft SQL Server 2005 funkcje okna oferujące takie konstrukcje, jak klauzula OVER oraz nowy zestaw funkcji nazywanych funkcjami rankingowymi (m.in. ROW_NUMBER, RANK). Ten dodatek umożliwił rozwiązywanie typowych problemów w łatwiejszy, bardziej intuicyjny, a często równocześnie efektywniejszy sposób. Przez kilka kolejnych lat potrzeba rozszerzenia możliwości funkcji okna (poprzez dodanie nowych funkcji i co ważniejsze koncepcji ram/przedziałów) stanowiła najczęściej zgłaszaną sugestię dotyczącą przyszłości języka SQL. W odpowiedzi na zgłoszenia nadsyłane przez wielu różnych klientów firma Microsoft zdecydowała się na wprowadzenie kolejnych ulepszeń funkcji okna w wersji SQL Server 2012.

Gdy rozmawiam z klientami na temat nowych funkcji wersji SQL Server 2012, zawsze doradzam im poświęcenie szczególnej uwagi nowym funkcjom okna oraz zrozumieniu ich wpływu na potencjał języka SQL. Cieszę się, że czytając tę książkę, poświęcą Państwo część swojego niewątpliwie cennego czasu na poznanie tej zaawansowanej i bardzo użytecznej funkcji. Jestem przekonany, że wersja SQL Server 2012 i lektura tej książki pomogą Państwu w jeszcze efektywniejszej pracy z programem SQL Server oraz w znacznie szybszym rozwiązywaniu zarówno tych małych, jak i dużych problemów.

Życzę miłej lektury!

Tobias Ternström
Główny menedżer programu
w zespole Microsoft SQL Server Engine

Wprowadzenie

Funkcje okna stanowią jedno z najbardziej kompleksowych narzędzi dostępnych zarówno w standardowym języku SQL, jak i we wspieranym przez Microsoft SQL Server dialekcie T-SQL. Umożliwiają one realizowanie obliczeń na zbiorach wierszy w elastyczny, czytelny i efektywny sposób. Dzięki pomysłowemu projektowi funkcje okna zyskują przewagę nad wieloma alternatywnymi, bardziej tradycyjnymi rozwiązaniami. Szeroki zakres zastosowań funkcji okna sprawia, że warto poświęcić czas na ich poznanie. Funkcje okna zostały wprowadzone po raz pierwszy w wersji SQL Server 2005, natomiast w wersji SQL Server 2012 rozszerzone zostały istniejące funkcje i dodane nowe. W niniejszej książce omówione zostaną możliwości funkcji okna, które są dostępne w systemie SQL Server, a także takie, które choć wchodzą w skład standardu języka SQL, nie zostały jeszcze zaimplementowane w produkcie SQL Server.

Dla kogo przeznaczona jest ta książka

Niniejsza książka została opracowana z myślą o programistach i administratorach baz danych SQL Server, czyli osobach, które zajmują się pisaniem kwerend i rozwijaniem kodu przy użyciu języka T-SQL. Przyjęto założenie, że czytelnik posiada przynajmniej półroczne doświadczenie w implementowaniu i optymalizowaniu kwerend T-SQL.

Układ książki

W niniejszej książce omówione zostały logiczne aspekty działania funkcji okna, a także zagadnienia związane z optymalizacją funkcji okna oraz ich praktycznymi zastosowaniami. Aspektom logicznym poświęcone zostały trzy pierwsze rozdziały. W pierwszym rozdziale przedstawiona została ogólna koncepcja stosowania funkcji okna w języku SQL, w drugim omówione zostały poszczególne funkcje okna, a w trzecim funkcje uporządkowanego zbioru. Czwarty rozdział został poświęcony optymalizacji funkcji okna w systemie SQL Server 2012. Natomiast ostatni, piąty rozdział zawiera omówienie praktycznych zastosowań funkcji okna.

Rozdział 1 „Funkcje okna w języku SQL” zawiera wprowadzenie do podstawowej koncepcji funkcji okna, która została ujęta w standardzie języka SQL. Zaprezentowany został ogólny model, a następnie różne typy funkcji okna. Ponadto omówione zostały dodatkowe elementy wchodzące w skład specyfikacji okna, takie jak partycjonowanie, uporządkowanie oraz ramy.

Rozdział 2 „Szczegółowe omówienie funkcji okna" zawiera szczegółowy opis poszczególnych funkcji okna. Omówione zostały różne typy funkcji, które mogą być stosowane na oknach wierszy, takie jak funkcje agregujące, funkcje rankingowe, funkcje przesunięcia oraz funkcje rozkładu.

Rozdział 3 „Funkcje uporządkowanego zbioru" został poświęcony funkcjom przetwarzania uporządkowanych zbiorów, które zostały ujęte w standardzie języka SQL, takim jak m.in. funkcje hipotetycznego zbioru oraz funkcje rozkładu odwrotnego. Ponadto zaprezentowane zostały metody realizowania podobnych obliczeń w systemie SQL Server.

Rozdział 4 „Optymalizacja funkcji okna" zawiera szczegółowe omówienie metod optymalizacji funkcji okna w wersji SQL Server 2012. Zaprezentowane zostały ogólne zalecenia dotyczące indeksowania z myślą o zmaksymalizowaniu wydajności, wyjaśniony został mechanizm równoległego przetwarzania oraz możliwości jego ulepszenia, przedstawiony został nowy operator Window Spool i nie tylko.

Rozdział 5 „Rozwiązania T-SQL wykorzystujące funkcje okna" zawiera omówienie praktycznych zastosowań funkcji okna do realizowania typowych wymagań biznesowych.

Wymagania systemowe

Funkcje okna stanowią element podstawowego aparatu bazy danych Microsoft SQL Server 2012, w związku z tym są dostępne we wszystkich edycjach produktu. Aby uruchomić prezentowane w tej książce fragmenty kodu, trzeba posiadać dostęp do serwera SQL Server 2012 (dowolnej edycji), na którym zainstalowana została przykładowa baza danych. Osoby, które nie mają dostępu do serwera SQL Server, mogą skorzystać z wersji próbnych oferowanych przez firmę Microsoft. Szczegółowe informacje znaleźć można na stronie: *http://www.microsoft.com/sql*. Wymagania sprzętowe i programowe zostały zaprezentowane w witrynie SQL Server Books Online pod adresem: *http://msdn.microsoft.com/en-us/library/ms143506(v=sql.110).aspx*.

Fragmenty kodu

Wszystkie zaprezentowane w tej książce fragmenty kodu, a także kod służący do wygenerowania przykładowych danych, erratę, dodatkowe materiały i nie tylko znaleźć można w poniższej witrynie:

http://www.insidetsql.com

Po otworzeniu strony głównej witryny należy kliknąć hiperłącze Books, a następnie wybrać z listy niniejszą książkę. Na stronie internetowej poświęconej tej książce znajduje się łącze służące do pobrania skompresowanego pliku zawierającego prezentowany w książce kod źródłowy oraz skrypt o nazwie TSQL2012.sql, który służy do stworzenia przykładowej bazy danych TSQL2012 i wypełnienia jej danymi.

Podziękowania

Wiele osób przyczyniło się do powstania niniejszej książki w sposób bezpośredni lub pośredni i w związku z tym należą się im podziękowania.

Dziękuję Lilach za nadawanie sensu mojemu życiu, tolerowanie mnie i pomoc w korygowaniu tekstu.

Moim rodzicom Mili i Gabiemu oraz rodzeństwu Mickey i Inie dziękuję za nieustające wsparcie oraz zaakceptowanie faktu, iż mieszkam tak daleko od nich.

Podziękowania dla członków zespołu ds. rozwoju produktu Microsoft SQL Server: Tobiasa Ternströma, Lubora Kollara, Umachandara Jayachandrana, Marca Friedmana, Milana Stojica i zapewne wielu innych. Wiem, że rozszerzenie systemu SQL Server o wsparcie dla funkcji okien stanowiło duże wyzwanie. Dziękuję za Waszą pracę i za czas poświęcony na spotkania ze mną, odpisywanie na moje emaile, udzielanie odpowiedzi na moje pytania oraz rozwiewanie niejasności.

Podziękowania dla zespołu redaktorów w wydawnictwach O'Reilly oraz MSPress, a w szczególności dla Kena Jonesa, który poświęcił mi najwięcej godzin – praca z Tobą była dla mnie prawdziwą przyjemnością. Dziękuję również Benowi Ryanowi, Kristen Borg, Curtisowi Philipsowi oraz Rogerowi LeBlancowi.

Jestem wdzięczny Adamowi Machanicowi za to, że zgodził się zostać redaktorem technicznym tej książki. Byłeś doskonałą osobą do tej roli, ponieważ niewielu ludzi zna się na rozwijaniu rozwiązań SQL Server tak dobrze jak Ty.

Podziękowania dla „Q2" „Q3" oraz „Q4" Cieszę się, że miałem okazję dzielić się pomysłami z osobami, które doskonale rozumieją język SQL, a w dodatku są wspaniałymi, bezproblemowymi przyjaciółmi. Czuję, że mogę z Wami o wszystkim porozmawiać, nie martwiąc się o konwenanse i konsekwencje. Dziękuję za zaopiniowanie wczesnych wersji tekstu.

Dziękuję mojej firmie SolidQ, z którą współpracuję już od 10 lat. Jestem wdzięczny, że mogę być częścią tak wspaniałej organizacji, która dzięki stałemu rozwojowi osiągnęła obecny stan. Moi współpracownicy są dla mnie nie tylko kolegami z pracy, ale także partnerami, przyjaciółmi i rodziną. Dziękuję Fernando G. Guerrerowi, Douglasowi McDowellowi, Herbertowi Albertowi, Dejanowi Sarkowi, Gianluce Hotzowi, Jeanne Reeves, Glennowi McCoinowi, Fritzowi Lechnitzowi, Ericowi Van Soldtowi, Joelle Budd, Janowi Taylorowi, Marilyn Templeton, Berremu Walkerowi, Alberto Martinowi, Lorenie Jimenez, Ronowi Talmagerowi, Andiemu Kelly, Rushabhowi Mehta, Eladio Rincónowi, Erikowi Veermanowi, Johanowi Richardowi Waymire, Carlowi Rabelerowi, Chrisowi Randallowi, Johanowi Åhlénowi, Raoulowi Illyés, Peterowi Larssonowi, Peterowi Myersowi, Paulowi Turyle i wielu innym.

Podziękowania dla zespołu redaktorów magazynu *SQL Server Pro*: Megan Keller, Lavona Petersa, Michele Crockett, Mika Otey i zapewne wielu innych. Od ponad dekady piszę artykuły dla magazynu i jestem wdzięczny za możliwość dzielenia się wiedzą z czytelnikami.

Dziękuję posiadaczom tytułu SQL Server MVP: Alejandrowi Mesa, Erlandowi Sommarskogowi, Aaronowi Bertrandowi, Paulowi White i wielu innym, a także menedżerowi programu MVP Simonowi Tienowi. To doskonały program i uczestniczenie w nim to dla mnie zaszczyt. Poziom specjalistycznej wiedzy wśród członków tej grupy jest niesamowity i zawsze bardzo się cieszę na nasze spotkania, podczas których wymieniamy się pomysłami lub po prostu przy piwie dzielimy nowinkami z życia prywatnego. Wierzę, że zabiegi grupy SQL Server MVP oraz całej społeczności SQL Server w znacznym stopniu wpłynęły na decyzję firmy Microsoft o rozszerzeniu możliwości funkcji okna w wersji SQL Server 2012. Cieszę się, że nasza współpraca przynosi tak znaczące efekty.

Na zakończenie chciałbym podziękować swoim studentom. Nauczanie języka SQL jest moją pasją i napędza mnie do działania. Dzięki Wam mogę realizować swoje powołanie. Jestem wdzięczny za wszystkie doskonałe pytania, które motywują mnie do pogłębiania własnej wiedzy.

Errata i dodatkowe materiały

Dołożyliśmy wszelkich starań, aby zapewnić jak najlepszą jakość tej książki i dołączonych do niej materiałów. Jeśli jednak po opublikowaniu książki wykryte zostaną jakieś błędy, zostaną one opublikowane na stronie wydawnictwa Microsoft Press w witrynie oreilly.com:

http://go.microsoft.com/FWLink/?Linkid=246707

Strona ta umożliwia również zgłaszanie nowo zauważonych błędów.

Dodatkowe informacje można uzyskać od działu Microsoft Press Book Support dostępnego pod adresem *mspinput@microsoft.com*.

Należy mieć jednak na uwadze, że powyższe adresy nie służą do uzyskiwania pomocy technicznej w zakresie działania oprogramowania firmy Microsoft.

Państwa opinia jest dla nas cenna

Wydawnictwo Microsoft Press na pierwszym miejscu stawia satysfakcję czytelników, dlatego Państwa zdanie jest dla nas bardzo ważne. Prosimy o wyrażenie opinii na temat tej książki na poniższej stronie:

http://www.microsoft.com/learning/booksurvey

Ankieta jest krótka i zapewniamy, że czytamy każdy komentarz oraz sugestię. Z góry dziękujemy za wszystkie opinie!

Komentarze i pomysły dotyczące tej książki oraz wątpliwości, których nie rozwiały wspomniane strony internetowe, prosimy przesyłać na adres e-mail:

itzik@SolidQ.com

ROZDZIAŁ 1

Funkcje okna w języku SQL

Funkcje okna są stosowane na zbiorach wierszy definiowanych przy użyciu klauzuli OVER. Funkcje okna służą głównie do celów analitycznych i umożliwiają wyznaczanie sum bieżących i średnich ruchomych, identyfikowanie wysp i luk w danych oraz wykonywanie innych obliczeń. Funkcje okna bazują na koncepcji *wyznaczania zakresu okna*, która została udokumentowana w standardzie języka SQL (wspieranym zarówno przez ISO, jak i ANSI). Koncepcja ta polega na stosowaniu różnych obliczeń na zbiorze (lub *oknie*) wierszy i otrzymywaniu pojedynczej wartości. Funkcje okna pomagają w realizowaniu wielu różnych zapytań, ułatwiając definiowanie intuicyjnych i efektywniejszych operacji na zbiorach.

Proces zapewniania wsparcia dla standardowych funkcji okna w Microsoft SQL Server przebiegał dwuetapowo. Najpierw w wersji SQL Server 2005 wprowadzone zostało częściowe wsparcie dla standardowych funkcji, a następnie w wersji SQL Server 2012 dodane zostały nowe możliwości. Nadal brakuje pewnych standardowych funkcji, ale dzięki ulepszeniom wprowadzonym w wersji SQL Server 2012 produkt oferuje szerokie możliwości. W niniejszej książce zaprezentowane zostaną zarówno funkcje dostępne w programie SQL Server, jak i standardowe funkcje, które nie zostały jeszcze zaimplementowane. Wprowadzając każdą kolejną funkcję będę informować, czy jest ona dostępna w programie SQL Server i w której wersji została dodana.

Od momentu, kiedy funkcje okna zostały wprowadzone w wersji SQL Server 2005, autor niniejszej książki coraz częściej wykorzystuje je do budowania lepszych rozwiązań. Co więcej, zwiększa prostotę i efektywność istniejących systemów, zastępując starsze, tradycyjne konstrukcje językowe nowymi funkcjami okna. Funkcje okna są tak poręczne, że obecnie są one wykorzystywane w większości tworzonych przez autora tej książki rozwiązań bazodanowych. Standardowy język SQL oraz systemy do zarządzania relacyjnymi bazami danych również ewoluują w kierunku rozwiązań analitycznych i funkcje okna stanowią ważną część tego trendu. Zdaniem autora tej książki funkcje okna są przyszłością rozwiązań wykonujących kwerendy SQL i w związku z tym warto zainwestować czas w ich lepsze poznanie.

Niniejsza książka zawiera szczegółowe omówienie funkcji okna, ich optymalizacji oraz metod stosowania ich w kwerendach. Na początku tego rozdziału przedstawiona zostanie ogólna koncepcja. Następnie zaprezentowane zostanie wprowadzenie do funkcji okna, krótki przegląd opartych na nich rozwiązań, przedstawienie elementów służących do specyfikowania okna, omówienie elementów kwerendy wspierających funkcje

okna oraz opis zdefiniowanej w standardzie konstrukcji umożliwiającej wielokrotne wykorzystywanie specyfikacji okna.

Wprowadzenie do funkcji okna

Zanim zagłębimy się w mechanizmy działania funkcji okna, zaprezentuję ogólne informacje, które mogą pomóc w zrozumieniu tej koncepcji. W niniejszej sekcji wyjaśnione zostaną różnice między dwiema podstawowymi metodami wykonywania kwerend: przy użyciu zbiorów oraz przy użyciu kursora (w sposób iteratywny). Będzie się można dowiedzieć, dlaczego funkcje okna stanowią w pewnym sensie pomost między tymi dwiema metodami. Ponadto przedstawione zostaną korzyści płynące ze stosowania funkcji okna oraz wady alternatywnych rozwiązań. Należy mieć na uwadze, że chociaż funkcje okna stanowią odpowiedź na wiele problemów, zdarzają się sytuacje, kiedy inne techniki okazują się bardziej wydajne. Rozdział czwarty zatytułowany „Optymalizacja funkcji okna" zawiera szczegółowe omówienie metod optymalizacji funkcji okna oraz sytuacji, w których ich zastosowanie zapewnia największą wydajność.

Omówienie funkcji okna

Funkcja okna to funkcja stosowana na zbiorze wierszy. *Okno* (ang. window) to termin wykorzystywany w standardzie języka SQL do opisania kontekstu, w jakim wykonywana jest dana funkcja. W języku SQL dostępna jest klauzula OVER, która służy do wyznaczania zakresu okna. Przyjrzyjmy się przykładowej kwerendzie:

Dodatkowe informacje Więcej informacji o bazie danych TSQL2012 wykorzystywanej w prezentowanych przykładach oraz materiałach pomocniczych znaleźć można we Wprowadzeniu do tej książki.

```
USE TSQL2012;

SELECT orderid, orderdate, val,
  RANK() OVER(ORDER BY val DESC) AS rnk
FROM Sales.OrderValues
ORDER BY rnk;
```

Oto skrócony wynik wykonania kwerendy:

```
orderid  orderdate               val       rnk
-------- ----------------------- --------- ---
10865    2008-02-02 00:00:00.000 16387.50  1
10981    2008-03-27 00:00:00.000 15810.00  2
11030    2008-04-17 00:00:00.000 12615.05  3
10889    2008-02-16 00:00:00.000 11380.00  4
10417    2007-01-16 00:00:00.000 11188.40  5
10817    2008-01-06 00:00:00.000 10952.85  6
```

```
10897    2008-02-19 00:00:00.000 10835.24  7
10479    2007-03-19 00:00:00.000 10495.60  8
10540    2007-05-19 00:00:00.000 10191.70  9
10691    2007-10-03 00:00:00.000 10164.80  10
...
```

Klauzula OVER służy do specyfikowania okna, czyli definiowania takich aspektów jak m.in. zbiór wierszy, do którego odnosi się bieżący wiersz bądź kolejność wierszy (nazywana także porządkiem). Jeśli specyfikacja okna nie zawiera żadnych elementów ograniczających zbiór wierszy (jak w tym przykładzie), zbiór wierszy w oknie jest taki sam jak zbiór wyników kwerendy.

Uwaga A oto bardziej precyzyjna definicja: okno to zbiór wierszy (inaczej relacja) stanowiący parametr wejściowy dla tej fazy logicznego przetwarzania kwerendy, w której pojawia się funkcja okna. Na tym etapie poznawania funkcji okna powyższa definicja może być mało zrozumiała. W związku z tym na razie dla uproszczenia przyjmiemy, że zakres okna jest taki sam jak końcowy zbiór wyników kwerendy. W dalszej części książki zaprezentuję bardziej precyzyjne wyjaśnienie.

Aby móc zbudować ranking wierszy, trzeba rzecz jasna zdefiniować porządek, czyli sposób określania ich kolejności. W zaprezentowanym przykładzie wiersze zostały uporządkowane według kolumny *val* w kolejności malejącej.

W tym przykładzie zastosowana została funkcja RANK, która wylicza pozycję bieżącego wiersza w rankingu na podstawie wejściowego zbioru wierszy oraz porządku sortowania. Jeśli zdefiniowany został porządek malejący (jak w tym przykładzie), pozycja bieżącego wiersza w rankingu jest wyliczana jako zwiększona o jeden liczba wierszy w odpowiednim zbiorze, których wartość porządkowa jest większa niż wartość porządkowa bieżącego wiersza. Przyjrzyjmy się przykładowemu wierszowi w wyniku kwerendy, np. wierszowi o pozycji 5 w rankingu. Pozycja ta wynosi 5, ponieważ w danym porządku (według kolumny *val* w kolejności malejącej) zbiór wyników kwerendy zawiera cztery wiersze o wartości atrybutu *val* większej niż wartość bieżącego wiersza (11188.40), a pozycja w rankingu stanowi sumę tej liczby wierszy plus 1.

Trzeba mieć świadomość, że zgodnie z koncepcją funkcji okna klauzula OVER służy do definiowania, w jakim oknie wykonywana będzie funkcja dla bieżącego wiersza. To samo odnosi się do wszystkich wierszy w zbiorze wyników kwerendy. Innymi słowy, dla każdego wiersza klauzula OVER definiuje okno niezależne od okien pozostałych wierszy. Jest to przełomowa koncepcja, której pełne przyswojenie wymaga czasu, jednak stanowi klucz do pełnego zrozumienia szerokich możliwości i ogromnego potencjału funkcji okien. Zagadnienie to nie jest proste, więc jeszcze do niego powrócimy.

Funkcje okna zostały po raz pierwszy wprowadzone do standardu języka SQL w rozszerzonym dokumencie dla wersji SQL:1999, w którym nosiły one jeszcze nazwę

„funkcji OLAP". Możliwości funkcji okna były stopniowo rozszerzane w kolejnych wersjach standardu (w momencie publikowania tej książki były to wersje SQL:2003, SQL:2008 oraz SQL:2011). Najnowsza wersja standardu SQL obejmuje bardzo zaawansowaną i obszerną specyfikację funkcji okna, co świadczy o tym, że organizacja zatwierdzająca standard popiera tę koncepcję. Jeśli dotychczasowy trend się utrzyma, standard będzie obejmował coraz szerszy zestaw funkcji okien o coraz większych możliwościach.

Uwaga Dokumenty zawierające specyfikację standardu języka SQL są dostępne do kupienia w witrynach organizacji ISO lub ANSI. Na przykład podstawowy dokument SQL:2011 można zakupić w witrynie ANSI pod adresem: http://webstore.ansi.org/RecordDetail.aspx?sku=ISO%2fIEC+9075-2%3a2011.

Standardowy język SQL wspiera kilka typów funkcji okna: agregujące, rankingowe, rozkładu oraz przesunięcia. Należy jednak pamiętać, że jest to ogólna koncepcja, w związki z czym w kolejnych wersjach standardu mogą pojawiać się nowe typy funkcji okna.

Agregujące funkcje okna to powszechnie znane funkcje agregujące stosowane zazwyczaj w kwerendach grupujących, takie jak SUM, COUNT, MIN czy MAX. Funkcja agregująca musi być wykonywana na zbiorze, który może być definiowany przy użyciu kwerendy grupującej lub specyfikacji okna. W wersji SQL Server 2005 wprowadzone zostało częściowe wsparcie dla agregujących funkcji okna, natomiast w wersji SQL Server 2012 dodane zostały nowe możliwości.

Funkcje rankingu to RANK, DENSE_RANK, ROW_NUMBER oraz NTILE. Obecnie w dokumentacji standardu dwie pierwsze funkcje należą do innej kategorii niż dwie kolejne (przyczyny takiej klasyfikacji wyjaśnię później). Jednak dla uproszczenia zaliczymy wszystkie cztery funkcje do tej samej kategorii, podobnie jak w oficjalnej dokumentacji produktu SQL Server. Pełna funkcjonalność czterech funkcji rankingowych została wprowadzona już w wersji SQL Server 2005.

Funkcje rozkładu to PERCENT_RANK, CUME_DIST, PERCENTILE_CONT oraz PERCENTILE_DISC. Wsparcie dla tych czterech funkcji zostało wprowadzone w wersji SQL Server 2012.

Funkcje przesunięcia to LAG, LEAD, FIRST_VALUE, LAST_VALUE oraz NTH_VALUE. W wersji SQL Server 2012 wprowadzone zostało wsparcie dla pierwszych czterech funkcji. Natomiast funkcja NTH_VALUE nie jest dostępna w żadnej wersji programu, nawet w najnowszej wersji SQL Server 2012.

Szczegółowe omówienie poszczególnych funkcji i ich zastosowań przedstawię w rozdziale 2 zatytułowanym „Szczegółowe omówienie funkcji okna".

Większość nowych koncepcji, metod czy narzędzi, nawet gdy są one lepsze i łatwiejsze w użyciu niż dotychczas stosowane rozwiązania, spotyka się na początku z pewnym oporem. Nowe rozwiązania często wydają się skomplikowane. Aby zachęcić

czytelników, którzy nie mają doświadczenia w korzystaniu z funkcji okien, do zainwestowania czasu w poznanie tych nowych narzędzi, przedstawię kilka ich zalet:

- Funkcje okna pomagają w realizowaniu różnego rodzaju kwerend. Jak wspomniałem wcześniej, autor tej książki wykorzystuje funkcje okna w większości implementowanych rozwiązań bazodanowych. Szczegółowe przykłady zastosowań zostaną zaprezentowane w ostatnim, piątym rozdziale – po omówieniu ogólnej koncepcji i metod optymalizacji. Na razie wystarczy mieć świadomość, że funkcje okna pomagają w rozwiązaniu między innymi następujących problemów:
  - Stronicowanie
  - Usuwanie powtarzających się danych
  - Zwracanie *n* pierwszych wierszy dla każdej z grup
  - Obliczanie sum bieżących
  - Realizowanie operacji na interwałach czasu np. dopasowywanie interwałów czasu i obliczanie maksymalnej liczby równoległych sesji
  - Identyfikowanie luk i wysp
  - Obliczanie centyli
  - Wyliczanie wartości modalnej dla rozkładu
  - Sortowanie hierarchii
  - Realizowanie operacji przestawiania
  - Analizowanie tzw. efektu świeżości
- Majac prawie dwudziestoletnie doświadczenie w konstruowaniu kwerend SQL oraz kilkuletnie doświadczenie w stosowaniu funkcji okna, uważam, że funkcje okna są efektywniejsze i bardziej intuicyjne niż alternatywne metody, choć potrzeba trochę czasu na przyzwyczajenie się do tej koncepcji.
- Funkcje okna mogą być poddawane optymalizacji, jak będzie się można przekonać w kolejnych rozdziałach.

Podsumowując, ponieważ funkcje okien w języku SQL stanowią nową koncepcję, przyzwyczajenie się do ich stosowania może zająć trochę czasu. Jednak warto zainwestować ten czas, gdyż funkcje okien są prostym i intuicyjnym w użyciu narzędziem. Przypomnijmy sobie, jak trudno było przyzwyczaić się do innych technologii, bez których obecnie nie wyobrażamy sobie życia.

Programowanie w oparciu o zbiory lub przy użyciu iteracji/kursora

Kwerendy T-SQL często dzielone są na dwie główne kategorie: rozwiązań opartych na zbiorach oraz opartych na iteracji/kursorze. Większość programistów T-SQL zgadza się, że w miarę możliwości należy trzymać się pierwszej metody, aczkolwiek kursory

są nadal powszechnie stosowane. W związku z tym pojawia się kilka interesujących pytań. Dlaczego zaleca się stosowanie rozwiązań opartych na zbiorach? I dlaczego mimo to tak wielu programistów sięga po metody iteracyjne? Jakie przeszkody uniemożliwiają im stosowanie zalecanego podejścia?

Aby zrozumieć ten problem, trzeba najpierw posiąść podstawową znajomość języka T-SQL i zrozumieć, na czym tak naprawdę polegają metody oparte na zbiorach. W rezultacie można się przekonać, że wiele osób postrzega techniki wykonywania operacji na zbiorach jako mało intuicyjne, w przeciwieństwie do technik opartych na iteracji. Wynika to z natury ludzkiego myślenia, co wyjaśnię za chwilę. Metody oparte na zbiorach wymagają zupełnie innego sposobu myślenia niż metody iteracyjne.

Język deklaratywny i optymalizacja

Niektórzy czytelnicy mogą zastanawiać się, skąd w języku deklaratywnym, takim jak SQL, biorą się różnice wydajności pomiędzy dwiema postaciami tego samego żądania, takimi jak kwerenda z funkcją okna i bez niej. W gruncie rzeczy żądanie zawiera jedynie logiczną deklarację, nie specyfikuje sposobu jej realizacji. Dlaczego implementacja systemu SQL, taka jak program SQL Server z językiem T-SQL, nie rozpoznaje, że dwie z pozoru różne kwerendy reprezentują w rzeczywistości to samo żądanie i w efekcie nie generuje tego samego planu wykonania dla obu kwerend?

Sytuacja ta wynika z kilku przyczyn. Po pierwsze optymalizator programu SQL Server nie jest idealny. Optymalizator SQL Server jest fantastycznym komponentem o ogromnych możliwościach, jednak nie obejmuje wszystkich potencjalnych reguł optymalizacji. Po drugie optymalizator musi zakończyć proces optymalizacji kwerendy w ograniczonym czasie. W przeciwnym przypadku optymalizacja mogłaby trwać tak długo, że w efekcie wydłużyłaby czas wykonania kwerendy, zamiast go skrócić. W efekcie mogłoby dojść do absurdalnej sytuacji, w której proces realizacji kwerendy, zamiast opracować niekoniecznie najbardziej optymalny plan wykonania w ciągu kilkudziesięciu milisekund i zrealizować go w ciągu kilku sekund, przez wiele godzin analizowałby wszystkie możliwe plany wykonania w nadziei na odnalezienie tego, który pozwoli oszczędzić kilka sekund. Dlatego ze względów praktycznych optymalizator musi ograniczyć czas poświęcony na proces optymalizacji. W oparciu o takie czynniki, jak m.in. rozmiar tabel wykorzystywanych w kwerendzie, system SQL Server oblicza dwie wartości progowe dla kwerendy: jedna to koszt oceniany jako *wystarczająco dobry*, natomiast druga to maksymalny czas, po którym proces optymalizacji zostanie zatrzymany. Jeśli przekroczona zostanie którakolwiek z tych wartości progowych, proces optymalizacji zostaje zatrzymany i system SQL Server wykorzystuje najlepszy z odnalezionych do tej pory planów wykonania.

Tak ogromne różnice powodują, że niełatwo jest zrezygnować z jednego podejścia na rzecz drugiego. I tu na pomoc przychodzą funkcje okna. Są one doskonałym narzędziem, które stanowi w pewnym sensie pomost między tymi dwiema różnymi szkołami i pozwala na łagodniejsze przejście do sposobu myślenia, jakiego wymagają metody oparte na zbiorach.

Na początku wyjaśnię, na czym polega realizowanie kwerend T-SQL opartych na zbiorach. T-SQL stanowi dialekt standardowego języka SQL (ISO/ANSI). Język SQL opiera się (lub aspiruje do tego) na modelu relacyjnym, który jest matematycznym modelem zarządzania danymi opracowanym i opublikowanym po raz pierwszy przez Edgara F. Codda na przełomie lat sześćdziesiątych i siedemdziesiątych. Model relacyjny bazuje na dwóch podstawowych teoriach matematycznych: teorii zbiorów oraz logice predykatów. W świecie informatyki wiele teorii zostało opracowanych intuicyjnie i w konsekwencji ulega stałym zmianom – do tego stopnia, że czasem czuję się, jakbym ścigał się z własnym cieniem. Model relacyjny to bezpieczna przystań w świecie informatyki, ponieważ bazuje na dużo mocniejszym fundamencie, a mianowicie na matematyce. Niektórzy twierdzą, że matematyka to prawda absolutna. Dzięki tak silnym matematycznym fundamentom model relacyjny jest bardzo stabilny. Ewoluuje, ale nie tak szybko jak inne działy informatyki. Model relacyjny już od kilkudziesięciu lat utrzymuje swoją pozycję i nadal stanowi główny motor wiodących platform bazodanowych, nazywanych *systemami zarządzania relacyjnymi bazami danych*.

Język SQL powstał w wyniku próby stworzenia języka opartego na modelu relacyjnym. Język SQL nie jest idealny i pod wieloma względami odbiega od modelu relacyjnego, jednak dostarcza narzędzia, dzięki któremu osoby zaznajomione z modelem relacyjnym mogą wykorzystywać język SQL w sposób relacyjny. Niewątpliwie stanowi on najważniejszy język wykorzystywany w dzisiejszych systemach zarządzania relacyjnymi bazami danych.

Jednak jak już wspomniałem, myślenie w sposób relacyjny nie jest intuicyjne dla wielu osób. Wynika to między innymi z dużej różnicy między podejściem iteracyjnym a tym opartym na zbiorach. Największe trudności napotykają programiści języków proceduralnych, w których interakcja z danymi w plikach następuje w sposób iteracyjny zaprezentowany na przykładzie następującego pseudokodu:

```
otwórz plik
pobierz pierwszy rekord
dopóki nie koniec pliku
  początek
    przetwórz rekord
    pobierz następny rekord
  koniec
```

Dane w plikach (a dokładniej w plikach ISAM, czyli plikach metody indeksowanego dostępu sekwencyjnego) są przechowywane w określonej kolejności. Rekordy są pobierane z pliku zgodnie z tą kolejnością i w dodatku pojedynczo. W efekcie przyzwyczajamy się do traktowania danych jak uporządkowanych rekordów przetwarzanych jeden

po drugim. Ten sposób myślenia przypomina realizowanie operacji przy użyciu kursorów w języku T-SQL, dlatego metody oparte na kursorach i iteracji wydają się tak znajome programistom z doświadczeniem w korzystywaniu z języków proceduralnych.

Relacyjne, oparte na zbiorach podejście do przetwarzania danych jest zupełnie inne. Aby ułatwić jego zrozumienie, zacznijmy od definicji *zbioru* wprowadzonej przez twórcę teorii Georga Cantora:

> Zbiorem *M* jest spojenie w całość określonych rozróżnialnych podmiotów naszej poglądowości czy myśli, które nazywamy elementami danego zbioru *m*.
>
> *– Joseph W. Dauben,* Georg Cantor *(Princeton University Press, 1990)*

Ta z pozoru prosta definicja jest w rzeczywistości tak złożona, że jej interpretacji mógłbym poświęcić kilka stron. Jednak na potrzeby tej książki skoncentruję się na dwóch kluczowych aspektach: jednym, który został bezpośrednio wyrażony w definicji oraz drugim, który z niej wynika:

- **Całość** Przyjrzyjmy się wykorzystaniu terminu *całość*. Zbiór powinien być postrzegany i traktowany jako całość. Nasza uwaga powinna skoncentrować się na zbiorze jako całości, zamiast na poszczególnych elementach zbioru. Przetwarzanie iteracyjne nie jest zgodne z tą koncepcją, ponieważ rekordy pliku lub kursor są przetwarzane pojedynczo. Tabela w bazie danych SQL reprezentuje (choć nie w doskonały sposób) relację z modelu relacyjnego. Relacja stanowi zbiór podobnych do siebie elementów (tzn. elementów posiadających takie same atrybuty). Gdy podejmujemy interakcję z tabelami przy użyciu kwerend opartych na zbiorach, traktujemy tabelę jako całość, a nie jak pojedyncze wiersze (krotki relacji) – zarówno pod względem sposobu deklarowania żądań SQL, jak i sposobu postrzegania zbioru. Niektóre osoby mają duże trudności z zaadaptowaniem tego sposobu myślenia.
- **Porządek** Jak można zauważyć, definicja nie wspomina nic o kolejności elementów. Wynika to z faktu, iż elementy w zbiorze nie są uporządkowane. Niektórym osobom trudno przywyknąć do tego faktu. Pliki i kursory definiują określoną kolejność rekordów i pobierając pojedyncze rekordy możemy polegać na tej kolejności. Wiersze w tabeli nie mają kolejności, ponieważ tabela stanowi zbiór. Osoby, które tego nie rozumieją, często mylą logiczną warstwę modelu danych i języka z fizyczną warstwą implementacji. Zakładają, że jeśli istnieje określony indeks na tabeli, mogą mieć pewność, że podczas wykonywania kwerend na tej tabeli dane będą zawsze przetwarzane zgodnie z porządkiem zdefiniowanym w indeksie. Co więcej, czasem od spełnienia tego założenia uzależniają poprawne działanie rozwiązania. Oczywiście w SQL Server nie można mieć takiej pewności. Aby zagwarantować, że wiersze w zbiorze wyników zostaną zaprezentowane w określonej kolejności, musimy dodać do kwerendy klauzulę ORDER BY. Jednak

dodając tę klauzulę powinniśmy mieć świadomość, że otrzymany wynik nie jest relacyjny, ponieważ definiuje określoną kolejność.

Aby pisać efektywne kwerendy, trzeba rozumieć naturę języka SQL i myśleć w kategoriach zbiorów. Z pomocą przychodzą funkcje okna, które ułatwiają przejście między iteracyjnym sposobem myślenia (pojedyncze wiersze w określonej kolejności) a sposobem myślenia w oparciu o zbiory (postrzeganie zbioru jako nieuporządkowanej całości). A wszystko to dzięki pomysłowemu projektowi funkcji okna.

Gdy zachodzi potrzeba określenia kolejności, funkcje okna wspierają klauzulę ORDER BY. Jednak sam fakt, iż funkcja określa kolejność, nie oznacza, że jest ona niezgodna z założeniami modelu relacyjnego. Zarówno dane wejściowe, jak i dane wyjściowe funkcji okna mają charakter relacyjny (nie wymagają i nie gwarantują żadnej kolejności). Porządek stanowi jedynie element specyfikacji obliczenia i skutkuje dodaniem atrybutu do wynikowej relacji. Nie ma pewności, że wiersze w zbiorze wyników będą uporządkowane w takiej samej kolejności, jaka wykorzystana została w oknie funkcji. Co więcej, różne funkcje okna w tej samej kwerendzie mogą definiować różne porządki. Zasadniczo ten rodzaj uporządkowania nie ma nic wspólnego z kolejnością wierszy w prezentowanym wyniku wykonania kwerendy. Jak widać na rysunku 1-1, zarówno dane wejściowe, jak i dane wyjściowe kwerendy z funkcją okna są relacyjne, mimo iż specyfikacja funkcji okna zawiera definicję porządku.

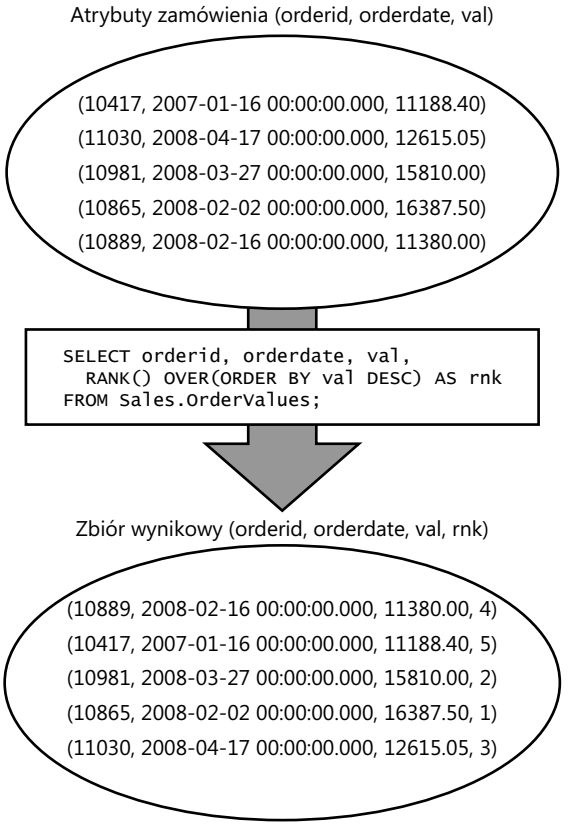
Zastosowane na ilustracji owale oraz różne pozycje wierszy w zbiorze wejściowym i wyjściowym służą do podkreślenia faktu, iż kolejność wierszy nie ma znaczenia.

Istnieje dodatkowy aspekt funkcji okna, który pomaga w stopniowym przejściu z iteratywnego sposobu myślenia do myślenia relacyjnego. Nauczycielom wprowadzającym nowe zagadnienie zdarza się czasem nieco naciągnąć prawdę. Zdają oni sobie sprawę z tego, że studenci nie są w stanie zrozumieć szczegółowego, kompleksowego omówienia. Czasami można osiągnąć lepsze rezultaty rozpoczynając od wyjaśnienia koncepcji w uproszczony, choć nie do końca prawdziwy sposób. W efekcie studenci oswajają się z ogólną ideą. Prawdziwą, szczegółową wersję można przedstawić w przyszłości, gdy studenci będą już lepiej na nią przygotowani.

W analogiczny sposób można przedstawić mechanizm wykonywania funkcji okna. Można wyjaśnić ogólną koncepcję w uproszczony sposób, który, choć nie jest całkowicie zgodny z prawdą, daje dobre efekty. Uproszczona, nieprawdziwa procedura bazuje na metodzie przetwarzania pojedynczych wierszy w określonej kolejności. W rzeczywistości procedura wykonywania funkcji okna bazuje na metodzie przetwarzania zbiorów. Jednak prawdziwa procedura jest bardziej skomplikowana i aby ją zrozumieć, trzeba osiągnąć pewien stopień wtajemniczenia.

Następująca kwerenda służy do zilustrowania tego problemu:

```
SELECT orderid, orderdate, val,
  RANK() OVER(ORDER BY val DESC) AS rnk
FROM Sales.OrderValues;
```



Rysunek 1-1 Dane wejściowe i wyjściowe kwerendy z funkcją okna

Oto skrócony wynik wykonania przedstawionej kwerendy (kolejność wierszy nie jest gwarantowana):

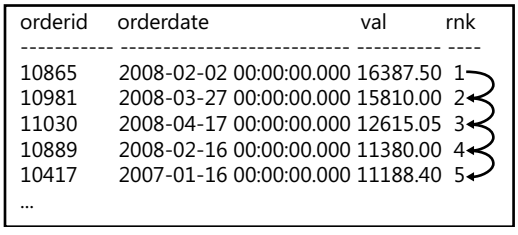
```
orderid  orderdate               val       rnk
-------- ----------------------- --------- ---
10865    2008-02-02 00:00:00.000 16387.50  1
10981    2008-03-27 00:00:00.000 15810.00  2
11030    2008-04-17 00:00:00.000 12615.05  3
10889    2008-02-16 00:00:00.000 11380.00  4
10417    2007-01-16 00:00:00.000 11188.40  5
...
```

Przykładowa uproszczona procedura wyznaczania pozycji w rankingu może zostać opisana przy pomocy pseudokodu w następujący sposób:

```
poukładaj wiersze według wartości val
wykonaj iterację po wierszach
dla każdego wiersza
  jeśli bieżący wiersz jest pierwszym w partycji zwróć 1
  w przeciwnym przypadku: jeśli wartość val równa się poprzedniej wartości val,
```

```
   zwróć poprzednią pozycję
w przeciwnym przypadku zwróć liczbę dotychczas przetworzonych wierszy
```

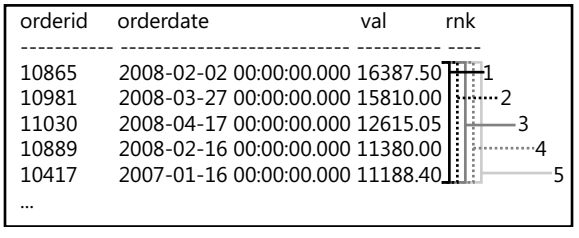
Rysunek 1-2 stanowi graficzną ilustrację tej uproszczonej procedury.



Rysunek 1-2 Uproszczona procedura wyznaczania pozycji w rankingu

Jak wspomniałem, choć zaprezentowana procedura prowadzi do uzyskania prawidłowego wyniku, nie jest całkiem zgodna z rzeczywistością. Sytuację dodatkowo komplikuje fakt, iż uproszczona procedura przypomina fizyczny proces realizowania tej operacji przez system SQL Server. Jednak niniejsza książka koncentruje się nie na fizycznej implementacji, lecz na warstwie koncepcyjnej, czyli języku i modelu logicznym. W związku z tym mówiąc o „niezgodności uproszczonej procedury z rzeczywistością" miałem na myśli to, iż z perspektywy języka obliczenia są realizowane w sposób oparty na zbiorach, a nie w sposób iteracyjny. Należy pamiętać, że język SQL jest niezależny od fizycznej implementacji w aparacie bazy danych. To warstwa fizyczna odpowiada za ustalenie, w jaki sposób obsłużyć logiczne żądanie tak, aby w jak najszybszym czasie uzyskać prawidłowy wynik.

A teraz spróbuję wyjaśnić, co mam na myśli, mówiąc o bardziej zaawansowanym i prawidłowym zrozumieniu mechanizmu działania funkcji okna z perspektywy języka. Funkcja definiuje osobne, niezależne okno dla każdego wiersza w zbiorze wyników kwerendy. Jeśli specyfikacja okna nie zawiera żadnych ograniczeń, na początku każde okno obejmuje wszystkie wiersze zawarte w zbiorze wyników kwerendy. Jednak możemy rozszerzać specyfikację okna o różne elementy (m.in. omówione w dalszej części rozdziału partycje i ramy), które ograniczają zbiór wierszy w każdym oknie. Rysunek 1-3 ilustruje tę koncepcję na przykładzie zaprezentowanej wcześniej kwerendy z funkcją RANK.



Rysunek 1-3 Zaawansowana procedura wyliczania pozycji w rankingu

Z logicznego punktu widzenia dla każdej funkcji okna i wiersza w zbiorze wyników kwerendy klauzula OVER tworzy osobne okno. W specyfikacji okna w przykładowej kwerendzie nie wprowadziłem żadnych ograniczeń, zdefiniowałem jedynie porządek przetwarzania wierszy. W związku z tym w zaprezentowanym przykładzie każde z okien będzie obejmować wszystkie wiersze ze zbioru wyników. Wszystkie okna są tworzone w tym samym czasie. W każdym z nich wyliczana jest pozycja rankingu wynosząca jeden plus liczba wierszy, które mają większą wartość atrybutu *val* niż bieżący wiersz.

Wielu osobom łatwiej przychodzi postrzeganie danych jako uporządkowanych wierszy, które są przetwarzane pojedynczo w iteracyjny sposób. I nie ma w tym nic złego, jeśli osoby te dopiero rozpoczynają pracę z funkcjami okien i uproszczone podejście pozwoli im na pisanie prawidłowych, choć prostych kwerend. W miarę upływu czasu warto stopniowo pogłębiać swoją znajomość logicznego modelu funkcji okna i zacząć postrzegać dane w kategoriach nieuporządkowanych zbiorów.

Wady rozwiązań alternatywnych dla funkcji okna

Funkcje okna mają szereg zalet, których nie posiadają alternatywne, bardziej tradycyjne metody realizowania tych samych obliczeń, takie jak np. kwerendy grupujące bądź kwerendy podrzędne. W tym podrozdziale przedstawię kilka prostych przykładów. Warto mieć świadomość, że istnieją dodatkowe, znaczące różnice, które nie zostaną na razie zaprezentowane.

Zacznijmy od tradycyjnych kwerend grupujących. Umożliwiają one uzyskiwanie dostępu do nowych informacji będących wynikiem agregacji, ale wiążą się z utratą szczegółowości.

Grupując dane musimy stosować wszystkie obliczenia w kontekście grupy. Ale co, jeśli chcemy zastosować obliczenia, które odwołują się zarówno do poszczególnych wierszy, jak i agregacji? Załóżmy na przykład, że potrzebujemy wykonać na widoku Sales.OrderValues kwerendę, która dla każdego zamówienia wylicza procentowy udział wartości bieżącego zamówienia w wartości wszystkich zamówień klienta, jak również różnicę między bieżącą a średnią wartością zamówienia danego klienta. Wartość bieżącego zamówienia stanowi element szczegółowy, natomiast suma i średnia zamówień klienta stanowią agregacje. Jeśli pogrupujemy dane według klienta, tracimy dostęp do poszczególnych zamówień. Jeden ze sposobów rozwiązywania tego typu problemów polega na stworzeniu kwerendy grupującej dane według klientów, zdefiniowaniu wspólnego wyrażenia tabelarycznego (Common Table Expression – CTE) na podstawie tej kwerendy, a następnie złączeniu wyrażenia tabelarycznego z tabelą bazową w celu dopasowania szczegółowych wierszy do agregacji. Poniżej zaprezentowałem przykładową implementację tej metody:

```
WITH Aggregates AS
(
```

```
  SELECT custid, SUM(val) AS sumval, AVG(val) AS avgval
  FROM Sales.OrderValues
  GROUP BY custid
)
SELECT O.orderid, O.custid, O.val,
  CAST(100. * O.val / A.sumval AS NUMERIC(5, 2)) AS pctcust,
  O.val - A.avgval AS diffcust
FROM Sales.OrderValues AS O
  JOIN Aggregates AS A
    ON O.custid = A.custid;
```

Oto skrócony wynik wygenerowany przez kwerendę:

```
orderid  custid  val     pctcust  diffcust
-------- ------- ------- -------- ------------
10835    1       845.80  19.79    133.633334
10643    1       814.50  19.06    102.333334
10952    1       471.20  11.03    -240.966666
10692    1       878.00  20.55    165.833334
11011    1       933.50  21.85    221.333334
10702    1       330.00  7.72     -382.166666
10625    2       479.75  34.20    129.012500
10759    2       320.00  22.81    -30.737500
10926    2       514.40  36.67    163.662500
10308    2       88.80   6.33     -261.937500
...
```

A teraz wyobraźmy sobie, że chcemy dodać współczynnik procentowego udziału w całkowitej sumie i różnicę dla średniej wartości wszystkich zamówień. W tym celu musimy dodać kolejne wyrażenie tabelaryczne:

```
WITH CustAggregates AS
(
  SELECT custid, SUM(val) AS sumval, AVG(val) AS avgval
  FROM Sales.OrderValues
  GROUP BY custid
),
GrandAggregates AS
(
  SELECT SUM(val) AS sumval, AVG(val) AS avgval
  FROM Sales.OrderValues
)
SELECT O.orderid, O.custid, O.val,
  CAST(100. * O.val / CA.sumval AS NUMERIC(5, 2)) AS pctcust,
  O.val - CA.avgval AS diffcust,
  CAST(100. * O.val / GA.sumval AS NUMERIC(5, 2)) AS pctall,
  O.val - GA.avgval AS diffall
FROM Sales.OrderValues AS O
  JOIN CustAggregates AS CA
    ON O.custid = CA.custid
  CROSS JOIN GrandAggregates AS GA;
```

Oto wynik wykonania tej kwerendy:

```
orderid  custid  val     pctcust  diffcust     pctall  diffall
-------- ------- ------- -------- ------------ ------- -------------
10835    1       845.80  19.79    133.633334   0.07    -679.252072
10643    1       814.50  19.06    102.333334   0.06    -710.552072
10952    1       471.20  11.03    -240.966666  0.04    -1053.852072
10692    1       878.00  20.55    165.833334   0.07    -647.052072
11011    1       933.50  21.85    221.333334   0.07    -591.552072
10702    1       330.00  7.72     -382.166666  0.03    -1195.052072
10625    2       479.75  34.20    129.012500   0.04    -1045.302072
10759    2       320.00  22.81    -30.737500   0.03    -1205.052072
10926    2       514.40  36.67    163.662500   0.04    -1010.652072
10308    2       88.80   6.33     -261.937500  0.01    -1436.252072
...
```

Jak widać, kwerenda uzyskana przy pomocy tej metody jest skomplikowana i wymaga zdefiniowania dodatkowych wyrażeń tabelarycznych oraz złączeń.

Podobne obliczenia mogą zostać przeprowadzone także w inny sposób – przy użyciu osobnej kwerendy podrzędnej dla każdego z obliczeń. Oto alternatywne rozwiązanie wykorzystujące kwerendy podrzędne do wykonania zadania realizowanego uprzednio przez dwie kwerendy grupujące:

```
-- kwerendy podrzędne z danymi szczegółowymi i agregacjami dla klienta
SELECT orderid, custid, val,
  CAST(100. * val /
        (SELECT SUM(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid) AS NUMERIC(5, 2)) AS pctcust,
  val - (SELECT AVG(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid) AS diffcust
FROM Sales.OrderValues AS O1;

-- kwerendy podrzędne z danymi szczegółowymi oraz agregacjami dla klienta
-- i całościowymi
SELECT orderid, custid, val,
  CAST(100. * val /
        (SELECT SUM(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid) AS NUMERIC(5, 2)) AS pctcust,
  val - (SELECT AVG(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid) AS diffcust,
  CAST(100. * val /
        (SELECT SUM(O2.val)
         FROM Sales.OrderValues AS O2) AS NUMERIC(5, 2)) AS pctall,
  val - (SELECT AVG(O2.val)
         FROM Sales.OrderValues AS O2) AS diffall
FROM Sales.OrderValues AS O1;
```

Rozwiązanie oparte na kwerendach podrzędnych ma dwie zasadnicze wady. Po pierwsze, jego kod jest długi i złożony. Po drugie, obecnie optymalizator SQL Server nie identyfikuje przypadków, gdy wiele kwerend podrzędnych uzyskuje dostęp do tego samego zbioru wierszy, w związku z tym każda kwerenda podrzędna wymagać będzie osobnej operacji dostępu do danych. W konsekwencji, im więcej zastosujemy kwerend podrzędnych, tym więcej operacji dostępu do danych zostanie przeprowadzonych. W odróżnieniu od poprzedniej sytuacji, problem ten nie wynika z natury języka, lecz ze sposobu optymalizowania kwerend podrzędnych w programie SQL Server.

Jak pamiętamy, funkcje okna definiują okno, czyli zbiór wierszy przetwarzanych przez funkcję. Funkcje agregacji zostały zaprojektowane z myślą o stosowaniu ich na zbiorze wierszy, dlatego specyfikacja okna w połączeniu z funkcjami agregacji może być lepszą opcją niż wykorzystanie kwerend grupujących bądź podrzędnych. W przypadku wykorzystania agregujących funkcji okna nie tracimy szczegółowych informacji. Do definiowania okna dla funkcji używamy klauzuli OVER. Na przykład, do obliczenia sumy wartości dla wszystkich wierszy ze zbioru wyników kwerendy możemy użyć następującego prostego wyrażenia:

```
SUM(val) OVER()
```

Jeśli nie ograniczymy okna (zastosujemy pusty nawias), rozpoczynamy od zbioru wyników kwerendy.

Do obliczenia sumy wartości (*val*) wszystkich wierszy ze zbioru wyników kwerendy, które mają jednakowy identyfikator klienta (*custid*) jak bieżący wiersz, wykorzystujemy element partycjonowania w funkcji okna (który zostanie omówiony później), przeprowadzając partycjonowanie okna według atrybutu *custid* w następujący sposób:

```
SUM(val) OVER(PARTITION BY custid)
```

Zwracam uwagę, że termin *partycjonowanie* sugeruje filtrowanie, a nie grupowanie.

Przy pomocy funkcji okna możemy pobrać informacje szczegółowe i zagregowane dla klientów, zwracając procentowy udział wartości bieżącego zamówienia w sumie wszystkich zamówień klienta, jak również różnicę między tą wartością a średnią wartością zamówienia klienta (funkcje okna zostały pogrubione):

```
SELECT orderid, custid, val,
  CAST(100. * val / SUM(val) OVER(PARTITION BY custid) AS NUMERIC(5, 2)) AS pctcust,
  val - AVG(val) OVER(PARTITION BY custid) AS diffcust
FROM Sales.OrderValues;
```

A oto kolejna kwerenda, do której dodałem współczynnik procentowego udziału w sumie całkowitej oraz różnicę dla średniej wartości wszystkich zamówień:

```
SELECT orderid, custid, val,
  CAST(100. * val / SUM(val) OVER(PARTITION BY custid) AS NUMERIC(5, 2)) AS pctcust,
  val - AVG(val) OVER(PARTITION BY custid) AS diffcust,
  CAST(100. * val / SUM(val) OVER() AS NUMERIC(5, 2)) AS pctall,
  val - AVG(val) OVER() AS diffall
FROM Sales.OrderValues;
```