Spis treści

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 znacząco wyewoluowały od ich wprowadzenia po raz pierwszy do SQL Server w kolejnych wersjach, ale rozwinęła się też ich definicja w standardzie SQL. W niniejszej książce omówione zostaną możliwości funkcji okna, które są dostępne w SQL Server, a także takie, które choć wchodzą w skład standardu języka SQL, nie zostały jeszcze zaimplementowane w 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.

- 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 „Rozpoznawanie wzorców wierszy w SQL” opisuje potężną koncepcję analityki danych wprowadzoną do standardu SQL, nazywaną *rozpoznawaniem wzorców wierszy* (row-pattern recognition – *RPR*), którą można traktować jako kolejny krok w ewolucji funkcji okna. Ta koncepcja nie jest jeszcze dostępna w dialekcie T-SQL, ale jak wspomniałem, książka omawia ważne funkcje analityczne występujące w standardzie, nawet jeśli nie są one jeszcze zaimplementowane w T-SQL.
- Rozdział 5 „Optymalizacja funkcji okna” zawiera szczegółowe omówienie metod optymalizacji funkcji okna w wersji SQL Server i Azure SQL Database. 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 6 „Rozwiązania T-SQL wykorzystujące funkcje okna” zawiera omówienie praktycznych zastosowań funkcji okna do realizowania typowych wymagań biznesowych.

Wymagania systemowe

Funkcje okna stanowią element podstawowego silnika bazy danych Microsoft SQL Server i Azure SQL Database, w związku z tym są dostępne we wszystkich edycjach produktu. Aby uruchomić prezentowane w tej książce fragmenty kodu, trzeba posiadać dostęp do instancji SQL Server 2019 lub późniejszej wersji (dowolne wydanie) albo Azure SQL Database, w której zainstalowana została przykładowa baza danych TSQLV5. Osoby, które nie mają dostępu do serwera SQL Server, mogą skorzystać z wersji próbnych oferowanych przez firmę Microsoft. Szczegóły można znaleźć na stronie *https://www.microsoft.com/en-us/sql-server/sql-server-downloads*.

Jako narzędzia klienckiego w celu połączenia się z silnikiem bazy danych i wysyłania kodu można użyć albo SQL Server Management Studio (SSMS), albo Azure Data Studio. Użyłem tego pierwszego do wygenerowania graficznych planów wykonania kwerend, które omawiam w książce. Narzędzie SSMS można pobrać ze strony *https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms*. Program Azure Data Studio dostępny jest pod adresem *https://docs.microsoft.com/en-us/sql/azure-data-studio/download*.

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:

MicrosoftPressStore.com/TSQLWindowFunctions/downloads

Na stronie tej znajduje się łącze służące do pobrania skompresowanego pliku zawierającego prezentowany w książce kod źródłowy oraz skrypt o nazwie TSQLV5.sql, który służy do stworzenia przykładowej bazy danych TSQLV5 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.

Podziękowania dla członków zespołu rozwoju produktu Microsoft SQL Server (byłych i obecnych): Conora Cunninghama, Joe Sacka, Vassilisa Papadimosa, Marca Friedmana, Craiga Freedmana, Milana Ruzica, Milana Stojica, Jovana Popovica, Borko Novakovica, 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.

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.

Aaron Bertrand i Greg Gonzales: dziękuję za moją kolumnę w portalu sqlperformance.com. SentryOne jest wspaniałą firmą ze świetnymi produktami i usługami dla społeczności.

Dziękuję wielu MVP SQL Server – Alejandro Mesa, Erlandowi Sommarskogowi, Aaronowi Bertrandowi, Paulowi White i wielu innym: to wspaniały program i jestem dumny, że do niego należę. Poziom wiedzy w tej grupie jest zadziwiający i nigdy nie mogę doczekać się kolejnych spotkań, zarówno w celu dzielenia się pomysłami, jak i pogaduszek przy piwie.

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:

MicrosoftPressStore.com/TSQLWindowFunctions/errata

Strona ta umożliwia również zgłaszanie nowo zauważonych błędów.

Dodatkowe informacje można uzyskać ze strony *MicrosoftPressStore.com/Support*. Należy mieć jednak na uwadze, że powyższe adresy nie służą do uzyskiwania pomocy technicznej w zakresie działania oprogramowania firmy Microsoft.

O autorze

Itzik Ben-Gan jest mentorem i współzałożycielem firmy SolidQ. Ma tytuł SQL Server Microsoft MVP, który otrzymał w 1999 roku. Prowadzi sesje szkoleniowe w zakresie tworzenia i optymalizowania kwerend oraz programowania w języku T-SQL w wielu różnych krajach. Jest autorem szeregu książek poświęconych językowi T-SQL, w tym *Zapytania w języku T-SQL* oraz *Podstawy języka T-SQL*, wznawiane i uzupełniane już kilka razy. Napisał wiele artykułów dla należącej do SentryOne *witryny sqlperformance.com, ITProToday* i *SolidQ Journal*. Itzik Ben-Gan pełnił funkcję prelegenta na konferencjach Tech-Ed, PASS Summit, SQLBITs oraz przedstawiał prezentacje na szeregu spotkań organizowanych przez grupy użytkowników SQL Server oraz przez firmę SolidQ. Opracował też kursy poświęcone podstawowym oraz zaawansowanym funkcjom języka T-SQL, które regularnie przeprowadza w wielu różnych krajach.

ROZDZIAŁ 1

Funkcje okna w języku SQL

Funkcje okna pomagają w realizowaniu wielu różnych zapytań, ułatwiając definiowanie intuicyjnych i efektywniejszych operacji na zbiorach. 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 podzbiorze zbiorze (lub *oknie*) wierszy i otrzymywaniu pojedynczej wartości.

Funkcje okna przeszły znaczną ewolucję w SQL Server i Azure SQL Database od ich wstępnego wprowadzania w wersji SQL Server 2005. Spróbuję w skrócie przedstawić ten rozwój. Nadal brakuje pewnych standardowych funkcjonalności, ale dzięki ulepszeniom wprowadzonym w ciągu lat produkt oferuje szerokie możliwości. W niniejszej książce zaprezentowane zostaną zarówno funkcje dostępne w SQL Server, jak i standardowe funkcjonalności, których nadal brakuje. 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 do SQL Server, ja sam coraz częściej korzystam z nich do budowania lepszych rozwiązań. Zastępuję starsze, tradycyjne konstrukcje językowe nowymi funkcjami okna, a uzyskane kwerendy są zazwyczaj prostsze i bardziej wydajne. Doszło już do tego, że obecnie wykorzystuję je w większości tworzonych przeze mnie rozwiązań bazodanowych. Ponadto standard języka SQL, systemy do zarządzania relacyjnymi bazami danych i w ogólności cała branża bazodanowa ewoluuje w kierunku rozwiązań analitycznych, a funkcje okna stanowią ważną część tego trendu. Przetwarzania transakcyjne w trybie online (OLTP), ośrodek zaintereosania branży w latach 90, przemija. Moim zdaniem funkcje okna stanowią przyszłość, jeśli chodzi o wykonywanie kwerend SQL; czas poświęcony na ich lepsze poznanie to czas dobrze zainwestowany.

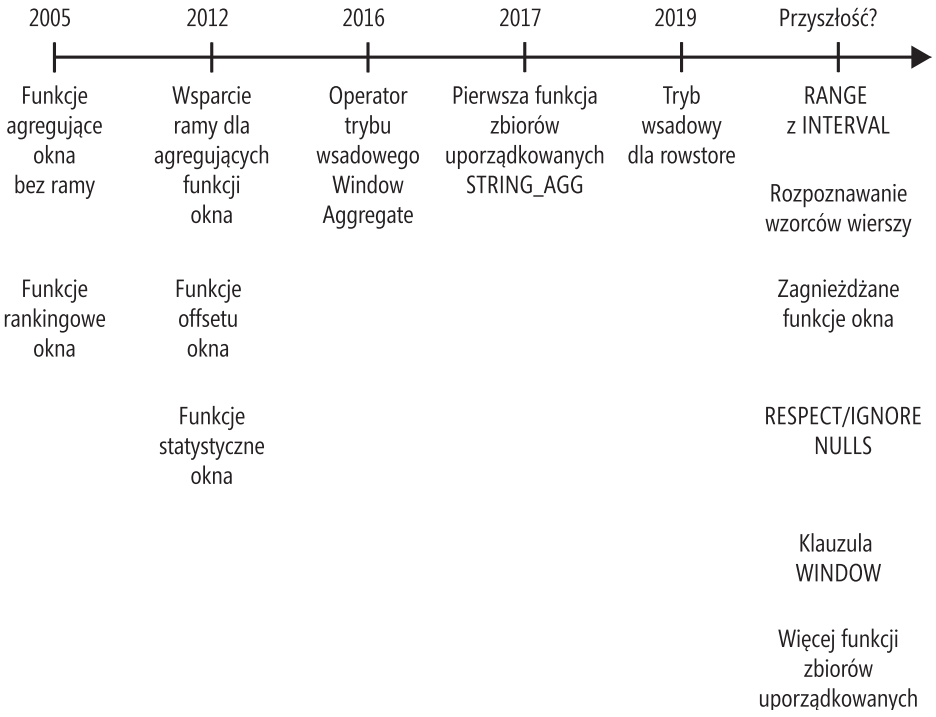
Niniejsza książka zawiera szczegółowe omówienie funkcji okna, ich optymalizacji oraz metod stosowania ich w kwerendach. Rozdział ten zaczyna poprzez wyjaśnienie tej koncepcji, a następnie przedstawia:

- Podstawy funkcji okna,
- Szybki przegląd rozwiązań wykorzystujących funkcje okna,

- Przedstawienie elementów służących do specyfikowania okna,
- Omówienie elementów kwerendy wspierających funkcje okna,
- Opis zdefiniowanej w standardzie konstrukcji umożliwiającej wielokrotne wykorzystywanie specyfikacji okna.

Ewolucja funkcji okna

Jak wspomniałem, funkcje okna przeszły znaczącą ewolucję od chwili ich włączenia do SQL Server i nadal istnieje potencjał dla dalszych udoskonaleń w przyszłości. Rysunek 1-1 ilustruje główne momenty wprowadzania funkcjonalności związanych z funkcjami okna w SQL Server oraz najważniejsze jeszcze nieobsługiwane funkcjonalności, które – miejmy nadzieję – zostaną dodane w niedalekiej przyszłości.



Rysunek 1-1 Ewolucja funkcji okna

Na tym etapie lektury niektóre z tych nazw mogą jeszcze nie być rozpoznawalne. Przedstawiłem tę oś czasową głównie z powodów referencyjnych. Po ukończeniu lektury wszystkie one będą już miały sens dla Czytelnika. Oto krótki opis kluczowych kamieni milowych, wyliczonych na rysunku 1-1:

- SQL Server 2005 wprowadza obsługę funkcji agregujących okna – ale jeszcze bez ramy okna – a także pełne wsparcie dla funkcji rankingowych okna (ROW_NUMBER, RANK, DENSE_RANK oraz NTILE).
- W wydaniach SQL Server 2008 i 2008 R2 nie pojawiły się żadne nowe możliwości okien.

- SQL Server 2012 wprowadza obsługę agregujących funkcji okna z ramką, funkcji offsetu (LAG, LEAD, FIRST_VALUE i LAST_VALUE) oraz funkcji statystycznych (PERCENT_RANK, CUME_DIST, PERCENTILE_CONT i PERCENTILE_DISC).
- Wydanie SQL Server 2014 nie wprowadza nowych możliwości.
- SQL Server 2016 wprowadza usprawnienia optymalizacyjne dla funkcji okna wraz z całkowicie nowym operatorem trybu wsadowego Window Aggregate. Jednak aby móc skorzystać z tego usprawnienia, konieczna jest obecność indeksu *columnstore* dla przynajmniej jednej spośród tabel uczestniczących w kwerendzie.
- W wydaniu SQL Server 2017 wprowadzono obsługę pierwszej funkcji zbiorów uporządkowanych STRING_AGG, która stosuje konkatenację łańcuchów tekstowych jako funkcję grupującą agregacji.
- SQL Server 2019 wprowadza obsługę trybu wsadowego dla *rowstore*, pozwalając na użycie operatorów trybu wsadowego wobec danych *rowstore* i zdejmując wcześniejsze wymaganie dysponowania indeksem *columnstore* w odpytywanych danych.
- W chwili pisania tych słów kilka ważnych funkcjonalności okien nadal nie jest obsługiwanych w SQL Server. Należą do nich jednostka ramki okna RANGE z typem INTERVAL, rozpoznawanie wzorców wierszy, zagnieżdżanie funkcji okna, możliwość określania opcji RESPECT | IGNORE NULLS, klauzuli WINDOW w celu ponownego wykorzystania definicji okna, więcej funkcji uporządkowanych zbiorów i kilka innych.

Podstawy funkcji okna

Zanim zagłębimy się w mechanizmy działania funkcji okna, pomocne będzie poznanie kontekstu i podstaw tych funkcji, które wyjaśnię w tym podpunkcie. Przedstawię różnice między dwiema podstawowymi metodami wykonywania kwerend: przy użyciu zbiorów oraz przy użyciu kursora (w sposób iteratywny). Pokażę, że funkcje okna stanowią w pewnym sensie pomost między tymi dwiema metodami. Na koniec przedstawione zostaną korzyści płynące ze stosowania funkcji okna oraz wady alternatywnych rozwiązań.

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. SQL używa do tego klauzuli OVER, która służy do wyznaczania zakresu okna. Przyjrzyjmy się przykładowej kwerendzie:

```
USE TSQLV5;

SELECT orderid, orderdate, val,
  RANK() OVER(ORDER BY val DESC) AS rnk
```

```
FROM Sales.OrderValues
ORDER BY rnk;
```

Dodatkowe informacje Więcej informacji o bazie danych TSQLV5 wykorzystywanej w przykładach oraz materiałach pomocniczych zawiera Wprowadzenie do książki.

Oto skrócone wyjście tej kwerendy:

```
orderid  orderdate  val       rnk
-------- ---------- --------- ----
10865    2019-02-02 16387.50  1
10981    2019-03-27 15810.00  2
11030    2019-04-17 12615.05  3
10889    2019-02-16 11380.00  4
10417    2018-01-16 11188.40  5
10817    2019-01-06 10952.85  6
10897    2019-02-19 10835.24  7
10479    2018-03-19 10495.60  8
10540    2018-05-19 10191.70  9
10691    2018-10-03 10164.80  10
...
```

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.

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.

Najważniejsze jest to, że koncepcyjnie funkcja okna klauzula OVER definiuje okno, w którym wykonywana będzie funkcja, względem bieżącego wiersza. Jest to spełnione dla wszystkich wierszy w zbiorze wyników kwerendy. Innymi słowy, dla każdego wiersza klauzula OVER definiuje okno niezależne od okien dla 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. Na razie chciałem po prostu zasiać ziarno w umyśle Czytelnika.

Funkcje okna zostały po raz pierwszy wprowadzone do standardu języka SQL w rozszerzonym dokumencie dla wersji SQL:1999, który omawiał tzw. „funkcje OLAP", jak je ówcześnie nazywano. Możliwości funkcji okna były stopniowo rozszerzane w kolejnych wersjach standardu. W chwili publikowania tej książki były to wersje SQL:2003, SQL:2008, SQL:2011 i SQL:2016. Ostatnia wersja standardu SQL obejmuje bardzo zaawansowaną i obszerną specyfikację funkcji okna. Zawiera też powiązane funkcjonalności analityczne, takie jak rozpoznawanie wzorców wierszy, co świadczy o tym, że komitet standaryzacyjny wierzy w tę koncepcję. Jeśli dotychczasowy trend się utrzyma, standard będzie obejmował coraz szerszy zestaw funkcji okien i więcej możliwości analitycznych.

Uwaga Dokumenty zawierające specyfikację standardu języka SQL można zakupić w witrynach organizacji ISO lub ANSI. Na przykład podstawowy dokument standardu SQL:2016, omawiający kluczowe konstrukcje języka można zakupić pod adresem *https://www.iso.org/standard/63556.html*.

Standardowy język SQL wspiera kilka typów funkcji okna: agregujące, rankingowe, rozkładu (statystyczne) oraz przesunięć (offsetu). 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.
- *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.
- *Funkcje rozkładów* czy też *statystyczne* to PERCENT_RANK, CUME_DIST, PERCENTILE_CONT oraz PERCENTILE_DISC. Funkcje te stosują obliczenia statystyczne, takie jak wyliczanie centyli, ranking percentylowy i rozkład kumulatywny.

- *Funkcje offsetu* to LAG, LEAD, FIRST_VALUE, LAST_VALUE oraz NTH_VALUE. SQL Server obsługuje pierwsze czterech funkcji. Natomiast funkcja NTH_VALUE nie jest dostępna w żadnej wersji programu, nawet w najnowszej wersji SQL Server 2019.

Uwaga 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
  - Deduplikacja danych
  - Zwracanie *n* pierwszych wierszy dla każdej grupy
  - Obliczanie sum bieżących
  - Wykonywanie operacji na interwałach, takie jak pakowanie interwałów i obliczanie maksymalnej liczby równoległych interwałów
  - Identyfikowanie luk i wysp
  - Obliczanie centyli
  - Wyliczanie wartości modalnej dla rozkładu
  - Sortowanie hierarchii
  - Realizowanie operacji przestawiania
  - Obliczanie tzw. efektu świeżości
- Mam prawie trzydziestoletnie doświadczenie w tworzeniu kwerend SQL i po kilkunastoletnim doświadczeniu w stosowaniu funkcji okna uważam, że są one efektywniejsze i bardziej intuicyjne niż alternatywne, tradycyjne metody.
- Funkcje okna mogą być poddawane optymalizacji, jak będzie się można przekonać w kolejnych rozdziałach.

Język deklaratywny i optymalizacja

Niektórzy czytelnicy mogą zastanawiać się, skąd w języku deklaratywnym, takim jak SQL, w którym logicznie deklarujemy żądanie, a nie opisujemy sposób jego osiągnięcia, biorą się różnice wydajności pomiędzy dwiema postaciami tego samego żądania – na przykład w kwerendzie z funkcją okna i bez niej? Dlaczego implementacja SQL, taka jak program SQL Server z jego dialektem 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 obydwu?

Sytuacja ta wynika z kilku przyczyn. Po pierwsze optymalizator programu SQL Server nie jest doskonały. Proszę nie traktować tego jako nadmierną krytykę – optymalizator SQL Server robi fantastyczną robotę, jest komponentem o ogromnych możliwościach. Jednak nie jest możliwe ujęcie w jednym programie 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.

Projekt funkcji okna, którym zajmiemy się w dalszej części książki, często sam z siebie pozwala na lepszą optymalizację, niż alternatywne metody zrealizowania tego samego zadania.

Tym, co trzeba zapamiętać z tych wywodów, to fakt, że konieczne będzie dokonanie świadomego wysiłku, aby przestawić się z tradycyjnego języka SQL na posługiwanie się funkcjami okna, gdyż jest to odmienna koncepcja. Warto to jednak zrobić, aby uzyskać prawdziwe korzyści. Przyzwyczajenie się do funkcji okna SQL wymaga nieco czasu. Jednak gdy już to zrobimy, okaże się, że funkcje okien są prostym i intuicyjnym w użyciu narzędziem. Przypomnijmy sobie, jak trudno było przyzwyczaić się do tych wszystkich nowoczesnych gadżetów, bez których obecnie nie wyobrażamy sobie życia.

Programowanie w oparciu o zbiory lub przy użyciu iteracji/kursora

Ludzie często charakteryzują rozwiązania T-SQL jako albo deklaratywne (oparte na zbiorach), albo bazujące na iteracji (kursorze). Większość programistów T-SQL zgadza się, że w miarę możliwości należy trzymać się pierwszej metody. Jednak 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. Luka pomiędzy myśleniem iteracyjnym a opartym na zbiorach jest naprawdę ogromna. Można ją jednak zamknąć, choć zdecydowanie nie jest to łatwe, i właśnie w tym wielką rolę odgrywają 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 – na matematyce. Niektórzy twierdzą, że matematyka to prawda absolutna. Dzięki tak silnym matematycznym fundamentom model relacyjny jest bardzo silny i 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 jest efektem próby stworzenia języka opartego na modelu relacyjnym. Nie jest idealny i pod wieloma względami odbiega od modelu relacyjnego, jednak dostarcza narzędzi, dzięki którym osoby znające model relacyjny 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. To 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 na samą jej interpretację mógłbym poświęcić kilkadziesiąt 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.

- **Nieobecność porządku** 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 w 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 SQL Server nie daje takiej gwarancji. Aby zapewnić zaprezentowanie wierszy w zbiorze wyników 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.

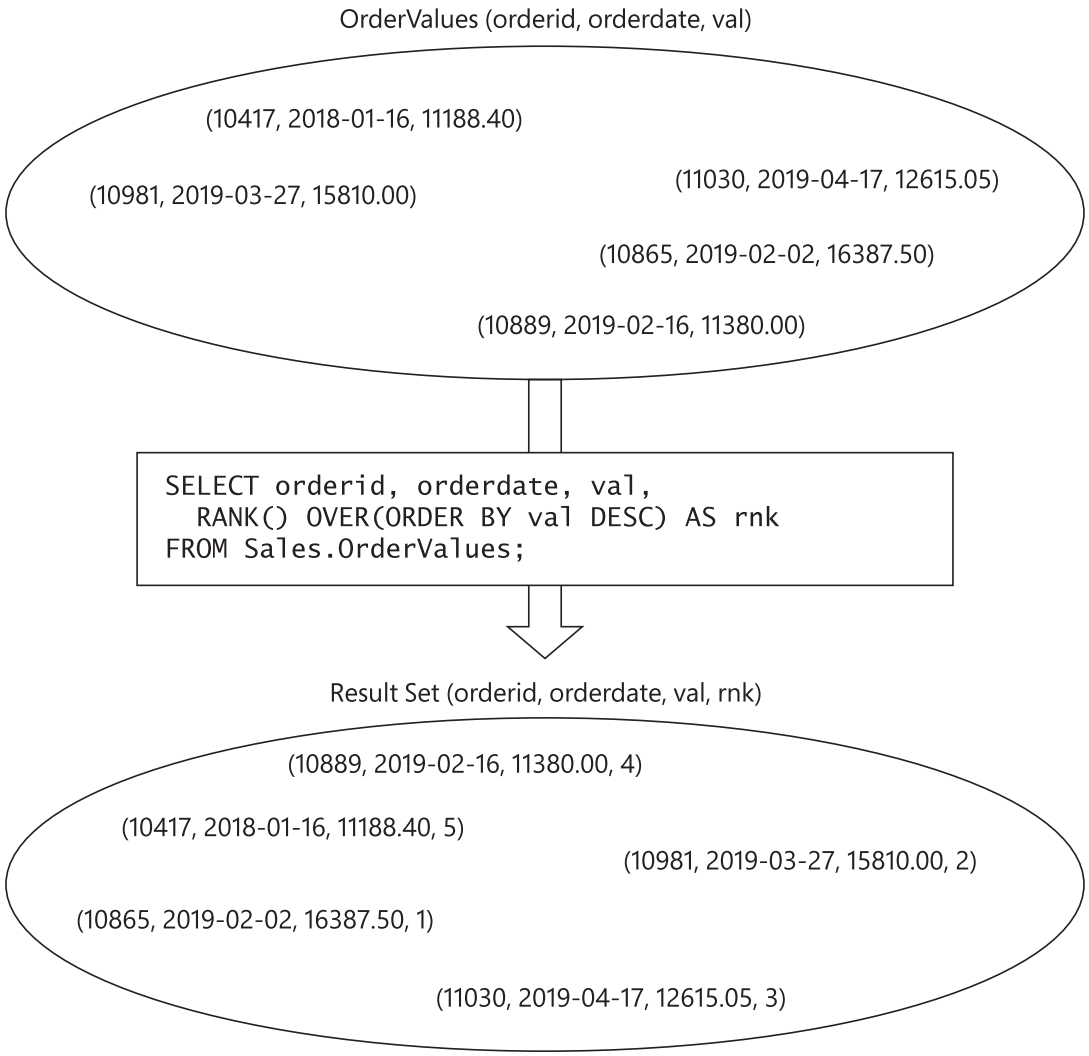
Aby pisać efektywne kwerendy, trzeba rozumieć naturę języka SQL i myśleć w kategoriach zbiorów. Z pomocą przychodzą funkcje okna, które ułatwiają przejście między iteracyjnym sposobem myślenia (pojedyncze wiersze w określonej kolejności) a sposobem myślenia w oparciu o zbiory (postrzeganie zbioru jako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

Przede wszystkim, 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 – przynajmniej koncepcyjnie. Rysunek 1-2 stanowi próbę zilustrowania idei, że zarówno dane wejściowe, jak i dane wyjściowe kwerendy z funkcją okna są relacyjne, mimo iż specyfikacja funkcji okna zawiera definicję porządku. Użyte na ilustracji owale oraz różne pozycje wierszy w zbiorze wejściowym i wyjściowym służą do podkreślenia faktu, iż kolejność wierszy nie ma znaczenia.

Jest jeszcze inny 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.

Tak też jest w przypadku prób przedstawienia, jak funkcje okna są obliczane. Oto prosty sposób wyjaśnienia, który wprawdzie nie jest tak naprawdę poprawny koncepcyjnie, ale prowadzi do poprawnych wyników. Ta podstawowa metoda wykorzystuje podejście wiersz po wierszu, w określonym porządku. Później przejdziemy do pogłębionej, koncepcyjnie poprawnej metody wyjaśnienia idei, ale umysł musi się najpierw przyzwyczaić („dojrzeć"), aby ją zrozumieć. Ta pogłębiona metoda używa podejścia opartego na zbiorach.



Rysunek 1-2 Dane wejściowe i wyjściowe kwerendy z funkcją okna

Aby zademonstrować, co mam na myśli, rozważmy następującą kwerendę:

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

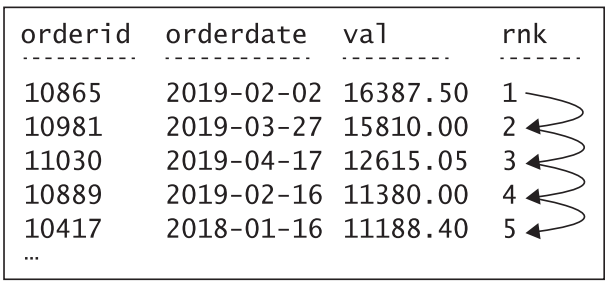
Oto skrócony wynik wykonania przedstawionej kwerendy (kolejność wierszy nie jest gwarantowana):

```
orderid  orderdate  val       rnk
-------- ---------- --------- ----
10865    2019-02-02 16387.50  1
10981    2019-03-27 15810.00  2
11030    2019-04-17 12615.05  3
10889    2019-02-16 11380.00  4
10417    2018-01-16 11188.40  5
...
```

Poniższy przykład w pseudokodzie pokazuje uproszczony sposób myślenia, jak wyznaczana jest pozycja w rankingu :

```
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 1-3 stanowi graficzną ilustrację tego sposobu myślenia.

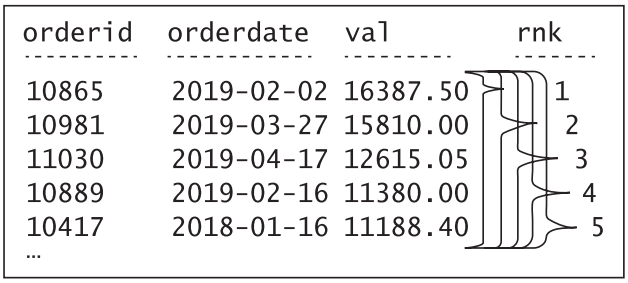


Rysunek 1-3 Uproszczone rozumienie wyznaczania pozycji w rankingu

Jak wspomniałem, choć zaprezentowana procedura prowadzi do uzyskania prawidłowego wyniku, nie jest w pełni zgodna z rzeczywistością. Sytuację dodatkowo komplikuje fakt, iż pokazane uproszczone rozumowanie przypomina to, jak SQL Server fizycznie realizowuje tę operację. 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 jest niezależny

od fizycznej implementacji w silniku 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 1-4 ilustruje tę koncepcję na przykładzie zaprezentowanej wcześniej kwerendy z funkcją RANK.



Rysunek 1-4 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 w porównaniu do alternatywnych, bardziej tradycyjnych metod realizowania tych samych obliczeń, takich jak np. kwerendy grupujące bądź kwerendy podrzędne. W tym podrozdziale przedstawię kilka prostych przykładów. Warto mieć świadomość, że istnieją dodatkowe, znaczące różnice poza przedstawionymi tu zaletami, ale byłoby jeszcze za wcześnie na ich omawianie.

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

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 bazie tej kwerendy, a następnie złączeniu wyrażenia tabelarycznego z tabelą bazową w celu dopasowania szczegółowych wierszy do agregacji. Listing 1-1 prezentuje przykładową implementację tej metody:

Listing 1-1 Łączenie szczegółów i wyników w kwerendzie grupującej

```
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, jak w listingu 1-2:

Listing 1-2 Mieszanie szczegółów i wyników w dwóch kwerendach grupujących

```
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. Listing 1-3 pokazuje alternatywne rozwiązanie wykorzystujące podkwerendy w ostatnich dwóch kwerendach grupujących:

Listing 1-3 Mieszanie szczegółów i wyników skalarnych podkwerend agregujących

```
-- 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;
```

Jak widać, wersje wykorzystujące funkcje okna są dużo krótsze i bardziej zrozumiałe. Co więcej, optymalizator SQL Server zawiera kod wyszukujący funkcje o tej samej specyfikacji okna. W przypadku ich odnalezienia SQL Server wykorzystuje w nich jedną i tę samą operację dostępu do danych (niezależnie od wybranego sposobu skanowania). Na przykład w zaprezentowanej uprzednio kwerendzie SQL Server wykorzysta jedną operację dostępu do danych do wyliczenia dwóch pierwszych funkcji (sumy i średniej partycjonowanej według atrybutu *custid*) oraz drugą operację dostępu do danych do wyliczenia dwóch ostatnich funkcji (sumy i średniej bez partycjonowania). Ten

proces optymalizacji zostanie szczegółowo omówiony w rozdziale 5, „Optymalizacja funkcji okna".

Inną przewagą funkcji okna nad kwerendami podrzędnymi jest to, że początkowe okno przed zastosowaniem ograniczeń ma taki sam zakres jak zbiór wyników kwerendy. To oznacza, że pokrywa się ono ze zbiorem wyników po zastosowaniu operatorów (np. JOIN), filtrów, grupowania i innych. Taki zbiór wyników uzyskujemy ze względu na fazę logicznego przetwarzania kwerendy, w której wykonane zostają funkcje okna (zagadnienie to zostanie omówione w dalszej części tego rozdziału). Natomiast kwerenda podrzędna rozpoczyna od samego początku, nie od zbioru wyników zewnętrznej kwerendy. W konsekwencji, jeśli chcemy, aby kwerenda podrzędna przetwarzała te same wiersze co zbiór wyników kwerendy zewnętrznej, musimy powtórzyć w niej wszystkie konstrukcje wykorzystywane w kwerendzie zewnętrznej. Dla przykładu załóżmy, że chcemy, aby wyliczenia udziału procentowego w sumie całkowitej oraz różnicy dla średniej odnosiły się jedynie dla zamówień złożonych w roku 2018. Jeśli wykorzystujemy funkcje okna, wystarczy, że dodamy do kwerendy jeden filtr:

```
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
WHERE orderdate >= '20180101'
  AND orderdate < '20190101';
```

Wszystkie funkcje okna rozpoczynają przetwarzanie od zbioru po zastosowaniu filtra, w przeciwieństwie do kwerend podrzędnych, które rozpoczynają od początku i dlatego wymagają powtórzenia filtra we wszystkich kwerendach podrzędnych, jak w Listingu 4:

Listing 1-4 Powtarzanie filtru we wszystkich podkwerendach

```
SELECT orderid, custid, val,
  CAST(100. * val /
        (SELECT SUM(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid
           AND orderdate >= '20180101'
           AND orderdate < '20190101') AS NUMERIC(5, 2)) AS pctcust,
  val - (SELECT AVG(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE O2.custid = O1.custid
           AND orderdate >= '20180101'
           AND orderdate < '20190101') AS diffcust,
  CAST(100. * val /
        (SELECT SUM(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE orderdate >= '20180101'
```

```
          AND orderdate < '20190101') AS NUMERIC(5, 2)) AS pctall,
  val - (SELECT AVG(O2.val)
         FROM Sales.OrderValues AS O2
         WHERE orderdate >= '20180101'
           AND orderdate < '20190101') AS diffall
FROM Sales.OrderValues AS O1
WHERE orderdate >= '20180101'
  AND orderdate < '20190101';
```

Oczywiście moglibyśmy wprowadzić pewne usprawnienia, np. zdefiniować wspólne wyrażenie tabelaryczne (Common Table Expression – CTE) odpowiedzialne za filtrowanie, a następnie odwołać się do tego wyrażenia CTE w kwerendzie zewnętrznej oraz w kwerendach podrzędnych. Jednak funkcje okna nie wymagają mechanizmów pomocniczych tego rodzaju, ponieważ są wykonywane na zbiorze wyników kwerendy. Dodatkowe informacje o tym aspekcie funkcji okna zaprezentuję w dalszej części tego rozdziału.

Jak wspomniałem wcześniej, funkcje okna są zwykle łatwiejsze w optymalizacji niż alternatywne rozwiązania. Oczywiście zdarzają się wyjątki. Optymalizacja funkcji okna zostanie omówiona w rozdziale 5, natomiast w rozdziale 6 zaprezentuję szereg przykładów ilustrujących efektywne zastosowania funkcji okna.

Przedsmak rozwiązań wykorzystujących funkcje okna

Cztery pierwsze rozdziały książki poświęcone zostały opisowi funkcji okna i ich zastosowań. Zaprezentowany materiał ma charakter techniczny i niektóre osoby mogą postrzegać jego lekturę jako nieco nużącą. Czytelników z reguły dużo bardziej interesują metody stosowania funkcji do rozwiązywania praktycznych problemów, które zostaną zademonstrowane w ostatnim rozdziale. Jednak aby uświadomić sobie ogromny potencjał funkcji okna, trzeba zdawać sobie sprawę z szerokiej gamy ich zastosowań. Chciałbym przekonać czytelników, że warto poświęcić czas na zapoznanie się z aspektami technicznymi i cierpliwie kontynuować lekturę aż do osiągnięcia bardziej interesującej części książki. W tym celu już teraz pokażę przedsmak rozwiązań wykorzystujących funkcje okna.

Wyobraźmy sobie, że mamy za zadanie pobrać dane z tabeli, w której w określonej kolumnie przechowywana jest *sekwencja* (ciąg) wartości, oraz zidentyfikować ciągłe przedziały istniejących wartości. Ten problem jest również nazywany *problemem wysp*. Dane w sekwencji mogą być typu numerycznego, czasowego (najczęściej stosowanego) lub innego typu wspierającego *porządek liniowy*. Sekwencja może zawierać niepowtarzalne wartości lub zezwalać na powtórzenia. Interwałem (różnicą) może być dowolna ustalona stała zgodna z typem kolumny (np. liczba całkowita 1, liczba całkowita 7, interwał czasowy 1 dzień, interwał czasowy 2 tygodnie). W rozdziale 6 omówię różne warianty tego ogólnego problemu. Ponieważ przykład ten służy jedynie do wstępnego

zademonstrowania możliwości funkcji okna, zajmę się najprostszym wariantem: sekwencją numeryczną z liczbą całkowitą 1 pełniącą rolę interwału. Następujący kod służy do wygenerowania przykładowych danych dla tego zadania:

```
SET NOCOUNT ON;
USE TSQLV5;

IF OBJECT_ID('dbo.T1', 'U') IS NOT NULL DROP TABLE dbo.T1;
GO

CREATE TABLE dbo.T1
(
  col1 INT NOT NULL
    CONSTRAINT PK_T1 PRIMARY KEY
);

INSERT INTO dbo.T1(col1)
  VALUES(2),(3),(11),(12),(13),(27),(33),(34),(35),(42);
GO
```

Jak widać, w sekwencji w kolumnie *col1* w tabeli T1 występują pewne luki. Naszym zadaniem jest zidentyfikowanie przedziałów wartości (zwanych również *wyspami*) i zwrócenie początku (*start_range*) oraz końca (*end_range*) dla każdej z wysp. Oto wynik, jaki chcemy osiągnąć:

```
start_range end_range
----------- -----------
2           3
11          13
27          27
33          35
42          42
```

Tego typu problem pojawia w wielu rzeczywistych systemach. Przykładowe warianty tego problemu wiążą się z identyfikowaniem: okresów dostępności, okresów aktywności (np. sprzedaży), okresów, w których spełnione było określone kryterium (np. okresów, w których wartość akcji przekracza określoną wartość progową), zakresów tablic rejestracyjnych będących w użytku itp. Celowo uprościłem prezentowany przykładowy problem, aby móc skoncentrować się na sposobie jego rozwiązywania. Metoda służąca do rozwiązania bardziej zaawansowanych przypadków tylko w niewielkim stopniu różni się od metody służącej do rozwiązania prostego wariantu. Zachęcam czytelników do próby samodzielnego wymyślenia efektywnego rozwiązania opartego na zbiorach przed zapoznaniem się z zademonstrowanym tutaj przykładem. Można rozpocząć od wymyślenia jakiegokolwiek działającego rozwiązania, a następnie wypełnić tabelę znaczącą liczbą wierszy (powiedzmy 10 milionów) i sprawdzić jego wydajność.

Zanim zaprezentuję rozwiązanie oparte na funkcjach okna, pokażę jedno z rozwiązań wykorzystujących bardziej tradycyjne konstrukcje językowe. Zaczniemy od metody opartej na kwerendach podrzędnych. Aby zrozumieć strategię pierwszego rozwiązania, przeanalizujmy wartości w sekwencji T1.col1, do której dodamy atrybut logiczny, który jeszcze nie istnieje i który będzie pełnił rolę identyfikatora grupy:

```
col1  grp
----- ---
2     a
3     a
11    b
12    b
13    b
27    c
33    d
34    d
35    d
42    e
```

Atrybut *grp* jeszcze nie istnieje. Jego wartość służyć będzie do identyfikowania wysp. To oznacza, że musi być on jednakowy dla wszystkich członków tej samej wyspy i różny od wartości przypisanych innym wyspom. Gdy uda nam się wygenerować spełniający te warunki identyfikator grupy, będziemy mogli pogrupować dane wynikowe według atrybutu *grp* i zwrócić wartości maksymalne oraz minimalne kolumny *col1* w każdej z grup (wysp). Jeden ze sposobów generowania identyfikatora grupy przy użyciu tradycyjnych konstrukcji językowych polega na wyliczeniu dla każdej bieżącej wartości *col1* minimalnej wartości *col1*, która jest większa lub równa bieżącej i nie ma kolejnej wartości.

Na przykład dla wartości 2 minimalna wartość kolumny *col1*, która jest większa lub równa 2 i która pojawia się tuż przed brakującą wartością, wynosi 3. Gdy przeprowadzimy te same obliczenia dla wartości 3, również otrzymamy wartość 3. W związku z tym liczba 3 stanowi identyfikator grupy dla wyspy, która rozpoczyna się od wartości 2 i kończy na wartości 3. Dla wyspy rozpoczynającej się od wartości 11 i kończącej wartością 13 identyfikator grupy dla wszystkich należących do niej elementów wynosi 13. Jak widzimy, identyfikatorem grupy dla wszystkich elementów wchodzących w skład danej wyspy jest ostatni element tej wyspy.

Oto kod T-SQL implementujący tę koncepcję:

```
SELECT col1,
  (SELECT MIN(B.col1)
    FROM dbo.T1 AS B
    WHERE B.col1 >= A.col1
      -- czy to ostatni wiersz w tej grupie?
      AND NOT EXISTS
        (SELECT *
         FROM dbo.T1 AS C
```

```
         WHERE C.col1 = B.col1 + 1)) AS grp
FROM dbo.T1 AS A;
```

Kwerenda ta wygeneruje następujące dane wynikowe:

```
col1        grp
----------- -----------
2           3
3           3
11          13
12          13
13          13
27          27
33          35
34          35
35          35
42          42
```

Ostatnia część rozwiązania jest dość prosta. W zewnętrznej kwerendzie grupujemy dane według identyfikatora grupy z zaprezentowanej uprzednio kwerendy podrzędnej i zwracamy minimalną oraz maksymalną wartość atrybutu *col1* dla każdej z grup w następujący sposób:

```
SELECT MIN(col1) AS start_range, MAX(col1) AS end_range
FROM (SELECT col1,
        (SELECT MIN(B.col1)
         FROM dbo.T1 AS B
         WHERE B.col1 >= A.col1
           AND NOT EXISTS
             (SELECT *
              FROM dbo.T1 AS C
              WHERE C.col1 = B.col1 + 1)) AS grp
      FROM dbo.T1 AS A) AS D
GROUP BY grp;
```

Rozwiązanie to pociąga za sobą dwa zasadnicze problemy. Po pierwsze jego logika jest mało czytelna. Po drugie ma ono bardzo długi czas wykonania. Nie chcę na razie zagłębiać się w szczegóły planów wykonania kwerendy, ponieważ przyjdzie jeszcze na to czas. Na razie wystarczy mieć świadomość, że dla każdego wiersza w tabeli SQL Server realizuje niemal dwie pełne operacje skanowania danych. W konsekwencji przetworzenie sekwencji 10 milionów wierszy stanowi kolosalne przedsięwzięcie.

W kolejnym prezentowanym rozwiązaniu również będziemy wyznaczać identyfikator grupy, lecz tym razem przy użyciu funkcji okna. Rozpoczynamy od wyznaczenia numerów wierszy w oparciu o kolejność wartości w kolumnie *col1* przy użyciu funkcji ROW_NUMBER. Funkcja ROW_NUMBER zostanie szczegółowo omówiona w dalszej części książki, na razie wystarczy mieć świadomość, że dla każdej partycji generuje ona szereg unikatowych, kolejnych liczb całkowitych (rozpoczynając od liczby 1) w oparciu o określony porządek wierszy.

Poniższa kwerenda zwraca wartości atrybutu *col1* i numery wierszy uporządkowanych według kolumny *col1*:

```
SELECT col1, ROW_NUMBER() OVER(ORDER BY col1) AS rownum
FROM dbo.T1;

col1        rownum
----------- --------------------
2           1
3           2
11          3
12          4
13          5
27          6
33          7
34          8
35          9
42          10
```

A teraz skoncentrujmy uwagę na dwóch sekwencjach. Pierwsza (*col1*) to sekwencja z lukami, a druga (*rownum*) to sekwencja bez luk. Zastanówmy się, w jaki sposób zależność między wartościami w tych dwóch sekwencjach może pomóc w identyfikowaniu wysp. W zakresie wyspy wartości w obu sekwencjach wzrastają o ten sam ustalony interwał (1). W związku z tym różnica między nimi będzie wartością stałą. Przy wchodzeniu w zakres następnej wyspy wartość kolumny *col1* wzrasta o ponad 1, natomiast wartość kolumny *rownum* nadal wzrasta tylko o 1, w związku z tym różnica między wartościami tych kolumn zwiększa się. W związku z tym różnica między dwiema kolumnami stanowi wartość stałą i unikatową dla każdej wyspy. Poniższa kwerenda służy do wyliczania tej różnicy:

```
SELECT col1, col1 - ROW_NUMBER() OVER(ORDER BY col1) AS diff
FROM dbo.T1;

col1        diff
----------- --------------------
2           1
3           1
11          8
12          8
13          8
27          21
33          26
34          26
35          26
42          32
```

Jak widzimy, wyznaczona różnica spełnia oba wymagania stawiane identyfikatorowi grupy i w związku z tym możemy użyć jej do tego celu. Kolejny etap jest taki sam jak w przypadku poprzedniego rozwiązania. Grupujemy wiersze według identyfikatora

grupy i zwracamy minimalną oraz maksymalną wartość kolumny *col1* dla każdej grupy w następujący sposób:

```
WITH C AS
(
  SELECT col1,
    -- różnica jest stała i unikatowa dla każdej wyspy
    col1 - ROW_NUMBER() OVER(ORDER BY col1) AS grp
  FROM dbo.T1
)
SELECT MIN(col1) AS startrange, MAX(col1) AS endrange
FROM C
GROUP BY grp;
```

Powyższe rozwiązanie jest krótkie i proste. Oczywiście, jak zawsze, warto dodać komentarze pomocnicze, aby ułatwić zrozumienie rozwiązania osobom, które mają z nim styczność po raz pierwszy.

Co więcej, zaprezentowane rozwiązanie jest bardzo efektywne. Przypisanie numerów wierszy wymaga dużo mniej nakładu pracy niż poprzednie rozwiązanie. Wystarczy jedna operacja skanowania w kolejności indeksu na kolumnie *col1* oraz operator zwiększający licznik. Podczas testów wydajności uruchomionych na tabeli zawierającej 10 milionów wierszy kwerenda zakończyła działanie po 3 sekundach. Inne rozwiązania potrzebowały dużo więcej czasu.

Mam nadzieję, że ten przykład zastosowań funkcji okna wystarczy, aby zaintrygować czytelników i pomóc w dostrzeżeniu ogromnych możliwości tych funkcji. Na razie powrócę do analizowania aspektów technicznych, ale w dalszej części książki będzie można zobaczyć wiele innych przykładów.

Elementy specyfikacji funkcji okna

Specyfikacja sposobu działania funkcji okna jest umieszczana w klauzuli OVER i może zawierać wiele elementów. Trzy podstawowe elementy to partycjonowanie, porządek oraz ramy. Nie wszystkie funkcje okna wspierają wszystkie trzy elementy. Opisując poszczególne elementy wymienię funkcje, w których można je stosować.

Partycjonowanie okna

Element partycjonowania jest implementowany przy użyciu klauzuli PARTITION BY i wspierany przez wszystkie funkcje okna. Element ten służy do ograniczania okna bieżącego obliczenia do tych wierszy ze zbioru wyników kwerendy, które mają w kolumnach partycjonowania te same wartości, co bieżący wiersz. Na przykład, jeśli funkcja wykorzystuje partycjonowanie PARTITION BY *custid*, a wartość atrybutu *custid* dla bieżącego wiersza wynosi 1, okno dla bieżącego wiersza stanowią wszystkie wiersze ze zbioru wyników kwerendy o wartości 1 w kolumnie *custid*. Analogicznie, gdy