SPRAWDŹ NOWE MOŻLIWOŚCI T-SQL I SQL SERVER!

Apress

T-SQL dla zaawansowanych

Przewodnik programisty

Wydanie IV

Miguel Cebollero, Michael Coles, Jay Natarajan

Helion

Tytuł oryginału: Pro T-SQL Programmer's Guide, 4th Edition

Tłumaczenie: Andrzej Stefański

ISBN: 978-83-283-2247-9

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/tsqlz4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 9

Wyrażenia nazwane i funkcje okna

SQL Server 2014 kontynuuje obsługę wyjątkowo użytecznych wyrażeń nazwanych, które po raz pierwszy pojawiły się w SQL Server 2005. CTE mogą uprościć Twoje zapytania, co sprawi, że staną się łatwiejsze do analizy i utrzymania. SQL Server obsługuje też odwołania do tego samego CTE, co pozwala na tworzenie bardzo potężnych zapytań rekurencyjnych.

Dodatkowo SQL Server obsługuje funkcje okna, które pozwalają na dzielenie wyników i dodawanie numerowania oraz wartości rankingowych do wierszy w uzyskanych za ich pomocą zbiorach wyników. Rozdział ten zaczyna się od omówienia możliwości i korzyści powiązanych ze stosowania CTE, a kończy opisem dostępnych w SQL Server funkcji okna.

Wyrażenia nazwane

Wyrażenia nazwane (**CTE** — ang. *common table expressions*) są użytecznym dodatkiem do SQL Server. CTE to coś w rodzaju tymczasowej tabeli zawierającej nazwany zbiór wyników, dostępny jedynie dla pojedynczego zapytania lub wyrażenia DML albo do jawnego usunięcia. CTE tworzy się w tym samym wierszu kodu, co wyrażenie `SELECT` lub wyrażenie DML, które z niego korzystają, podczas gdy budowanie i użycie tabeli tymczasowej to zazwyczaj dwuetapowy proces.
W porównaniu z trwałymi tabelami i widokami CTE mają kilka zalet. Oto one.

- CTE są tymczasowe i istnieją jedynie w czasie działania pojedynczego zapytania lub wyrażenia DML. Oznacza to, że nie musisz tworzyć trwałych obiektów bazodanowych, takich jak widoki.
- Do pojedynczego CTE można odwoływać się wiele razy za pomocą nazwy w pojedynczym zapytaniu lub wyrażeniu DML, co czyni Twój kod łatwiejszym do zarządzania. Tabele pomocnicze muszą być przepisywane w całości przy każdym odwołaniu do nich.
- Można korzystać z CTE do grupowania za pomocą kolumn tworzonych ze zbioru skalarnego lub za pomocą funkcji niedeterministycznej.
- CTE mogą odwoływać się do siebie, co jest użytecznym mechanizmem rekurencyjnym.
- Zapytania odwołujące się do CTE można wykorzystywać do definiowania kursora.

CTE mogą być bardzo różne — od prostych do bardzo skomplikowanych konstrukcji. Wszystkie CTE rozpoczynają się od słowa kluczowego `WITH`, po którym umieszczana jest nazwa CTE i lista zwracanych przez zapytanie kolumn. Następnie umieszczane jest słowo kluczowe `AS` i treść CTE, którą tworzy zapytanie lub wyrażenie DML zakończone średnikiem w przypadku ciągu złożonego z większej liczby wyrażeń. Na listingu 9.1 pokazany jest bardzo prosty przykład służący do demonstracji podstawowej składni.

Listing 9.1. *Proste CTE*

```
WITH PobierzNazwiskaCTE ( BusinessEntityID, FirstName, MiddleName, LastName )
AS
(
  SELECT BusinessEntityID, FirstName, MiddleName, LastName
  FROM Person.Person
)
SELECT
  BusinessEntityID,
  FirstName,
  MiddleName,
  LastName
FROM PobierzNazwiskaCTE;
```

Na listingu 9.1 jest definiowane CTE o nazwie `PobierzNazwiskaCTE`, które zwraca kolumny `BusinessEntityID`, `FirstName`, `MiddleName` i `LastName`. Treść CTE zawiera proste zapytanie do tabeli `Person.Person` bazy danych *AdventureWorks2014*. Bezpośrednio po CTE znajduje się związane z nim wyrażenie `SELECT`. Wyrażenie `SELECT` odwołuje się do CTE w klauzuli `FROM`.

Przeładowane WITH

Słowo kluczowe `WITH` jest w SQL Server przeładowane, co oznacza, że wykorzystywane jest na kilka różnych sposobów w odmiennych zadaniach w T-SQL. Wykorzystywane jest między innymi przy określaniu dodatkowych opcji w wyrażeniach DDL `CREATE`, przy dodawaniu podpowiedzi do zapytań i wyrażeń DML oraz przy deklarowaniu przestrzeni nazw XML w klauzuli `WITH XMLNAMESPACES`. Obecnie jest też wykorzystywane jako słowo kluczowe rozpoczynające definicję CTE. Jeśli zatem CTE nie jest pierwszym wyrażeniem w ciągu, poprzednie wyrażenie musi być zakończone średnikiem. Jest to jeden z powodów tego, że zdecydowanie należy korzystać ze średnika kończącego wyrażenie w tworzonym kodzie.

Proste CTE mają kilka ograniczeń przy definiowaniu i deklarowaniu.

- Do CTE muszą być dołączone pojedyncze wyrażenia `INSERT`, `DELETE`, `UPDATE` lub `SELECT`.
- Wszystkie kolumny zwracane przez CTE muszą mieć unikalne nazwy. Jeśli wszystkie kolumny zwracane przez zapytanie w treści CTE mają unikalne nazwy, możesz pominąć listę kolumn w deklaracji CTE.
- CTE może odwoływać się do innego, wcześniej zdefiniowanego CTE w tej samej klauzuli `WITH`, ale nie może odwoływać się do CTE zdefiniowanego później (nazywa się to odwołaniem do przodu).
- W CTE nie można korzystać ze słów kluczowych, klauzul i opcji, takich jak `COMPUTE`, `COMPUTE BY`, `FOR BROWSE`, `INTO` i `OPTION` (podpowiedź do zapytania). Nie możesz też korzystać z `ORDER BY`, jeśli nie użyjesz klauzuli `TOP`.
- W nierekurencyjnym CTE można zdefiniować wiele CTE. Wszystkie definicje muszą być połączone za pomocą jednego z operatorów działających na zbiorach; są to `UNION ALL`, `UNION`, `INTERSECT`, `EXCEPT`.
- Jak wspomniano w ramce „Przeładowane WITH”, jeśli CTE nie jest pierwszym wyrażeniem w ciągu, poprzedzające je wyrażenie musi być zakończone średnikiem.

Pamiętaj o tych ograniczeniach podczas tworzenia CTE.

Wiele zapytań nazwanych

Możesz definiować wiele CTE dla pojedynczego zapytania lub wyrażenia DML, oddzielając definicje CTE za pomocą przecinków. Głównym powodem, by to robić, jest możliwość uproszczenia kodu, co ułatwia jego analizę i zarządzanie. Wyrażenia nazwane oferują sposoby wizualnego rozdzielenia kodów na mniejsze bloki funkcjonalne, co upraszcza jego tworzenie i debugowanie. Zapytanie z listingu 9.2 zawiera wiele CTE, przy czym drugie CTE odwołuje się do pierwszego. Wyniki jego działania zostały pokazane na rysunku 9.1.

Listing 9.2. *Wiele CTE*

```
WITH PobierzNazwiskaCTE ( BusinessEntityID, FirstName, MiddleName,LastName )
  AS (
    SELECT BusinessEntityID, FirstName, MiddleName, LastName
    FROM Person.Person
  ),

PobierzKontaktCTE ( BusinessEntityID, FirstName, MiddleName, LastName,
                Email, HomePhoneNumber )
  AS (
    SELECT gn.BusinessEntityID, gn.FirstName
         , gn.MiddleName, gn.LastName
         , ea.EmailAddress, pp.PhoneNumber
    FROM PobierzNazwiskaCTE gn
    LEFT JOIN Person.EmailAddress ea
      ON gn.BusinessEntityID = ea.BusinessEntityID
    LEFT JOIN Person.PersonPhone pp
      ON gn.BusinessEntityID = pp.BusinessEntityID
    AND pp.PhoneNumberTypeID = 2
  )

SELECT BusinessEntityID, FirstName
     , MiddleName, LastName
     , Email, HomePhoneNumber
FROM PobierzKontaktCTE;
```

Results | Messages

	BusinessEntityID	FirstName	MiddleName	LastName	Email	HomePhoneNumber
133	20216	Alyssa	M	Alexander	alyssa64@adventure-works.com	NULL
134	3057	Amanda	L	Alexander	amanda40@adventure-works.com	NULL
135	7458	Ana	R	Alexander	ana17@adventure-works.com	1 (11) 500 555-0181
136	7678	Angela	NULL	Alexander	angela21@adventure-works.com	NULL
137	7161	Angelica	NULL	Alexander	angelica18@adventure-works.com	214-555-0150
138	3683	Anna	NULL	Alexander	anna44@adventure-works.com	150-555-0156
139	13220	Antonio	NULL	Alexander	antonio19@adventure-works.com	NULL
140	7719	Arianna	NULL	Alexander	arianna17@adventure-works.com	140-555-0167
141	12597	Ashley	J	Alexander	ashley46@adventure-works.com	NULL
142	6266	Austin	NULL	Alexander	austin15@adventure-works.com	1 (11) 500 555-0140
143	6726	Benjamin	A	Alexander	benjamin20@adventure-works.com	240-555-0151
144	6919	Brandon	NULL	Alexander	brandon16@adventure-works.com	780-555-0152
145	19276	Brianna	NULL	Alexander	brianna64@adventure-works.com	242-555-0137
146	7210	Brittany	NULL	Alexander	brittany17@adventure-works.com	804-555-0125
147	5561	Caleb	NULL	Alexander	caleb15@adventure-works.com	670-555-0141
148	6315	Cameron	NULL	Alexander	cameron14@adventure-works.com	399-555-0196
149	6800	Caroline	NULL	Alexander	caroline20@adventure-works.com	NULL
150	13268	Carson	G	Alexander	carson18@adventure-works.com	NULL

Rysunek 9.1. *Część wyniku zapytania z wieloma CTE*

Zwiększona czytelność CTE

Dzięki wykorzystaniu CTE możesz tworzyć bardziej czytelne zapytania niż przy użyciu zagnieżdżonych podzapytań. Dla celów demonstracyjnych poniższe zapytanie służy do uzyskania takich samych wyników jak zapytanie korzystające z CTE pokazane na listingu 9.2, ale za pomocą zagnieżdżonych podzapytań.

```
SELECT
  gn.BusinessEntityID,
  gn.FirstName,
  gn.MiddleName,
  gn.LastName,
  gn.EmailAddress,
  gn.HomePhoneNumber
FROM
(
  SELECT
    p.BusinessEntityID,
    p.FirstName,
    p.MiddleName,
    p.LastName,
    ea.EmailAddress,
    ea.HomePhoneNumber
  FROM Person.Person p
  LEFT JOIN
  (
    SELECT
      ea.BusinessEntityID,
      ea.EmailAddress,
      pp.HomePhoneNumber
    FROM Person.EmailAddress ea
    LEFT JOIN
    (
      SELECT
        pp.BusinessEntityID,
        pp.PhoneNumber AS HomePhoneNumber,
        pp.PhoneNumberTypeID
      FROM Person.PersonPhone pp
    ) pp
    ON ea.BusinessEntityID = pp.BusinessEntityID
      AND pp.PhoneNumberTypeID = 2
  ) ea
  ON p.BusinessEntityID = ea.BusinessEntityID
) gn
```

Korzystająca z CTE wersja tego zapytania pokazana na listingu 9.2 upraszcza i hermetyzuje kod zapytania oraz jest dużo prostsza do czytania i zrozumienia niż wersja korzystająca z zagnieżdżonych podzapytań, co sprawia, że jest łatwiejsza do debugowania i utrzymania w długim horyzoncie czasowym.

Przykład z listingu 9.2 zawiera dwa CTE o nazwach `PobierzNazwiskaCTE` i `PobierzKontaktCTE`. Zapytanie `PobierzNazwiskaCTE` zapożyczone jest z listingu 9.1; pobiera ono nazwiska z tabeli `Person.Person`.

```
WITH PobierzNazwiskaCTE ( BusinessEntityID, FirstName, MiddleName,LastName )
  AS (
    SELECT BusinessEntityID, FirstName, MiddleName, LastName
    FROM Person.Person
  ),
```

Drugie CTE, `PobierzKontaktCTE`, łączy wyniki `PobierzNazwiskaCTE` z tabelami `Person.EmailAddress` i `Person.PersonPhone`.

```
PobierzKontaktCTE ( BusinessEntityID, FirstName, MiddleName, LastName,
                    Email, HomePhoneNumber )
  AS (
    SELECT gn.BusinessEntityID, gn.FirstName
         , gn.MiddleName, gn.LastName
         , ea.EmailAddress, pp.PhoneNumber
```

```
    FROM PobierzNazwiskaCTE gn
    LEFT JOIN Person.EmailAddress ea
      ON gn.BusinessEntityID = ea.BusinessEntityID
    LEFT JOIN Person.PersonPhone pp
      ON gn.BusinessEntityID = pp.BusinessEntityID
    AND pp.PhoneNumberTypeID = 2
  )
```

Zauważ, że słowo kluczowe `WITH` wykorzystywane jest tylko raz na początku całego wyrażenia. Druga deklaracja CTE jest oddzielona od pierwszej za pomocą przecinka i nie wymaga użycia słowa kluczowego `WITH`. W końcu warto zauważyć jak proste i czytelne jest zapytanie `SELECT` powiązane z CTE dzięki temu, że złączenia zostały przeniesione do CTE.

```
SELECT BusinessEntityID, FirstName
     , MiddleName, LastName
     , Email, HomePhoneNumber
FROM PobierzKontaktCTE;
```

■ **Wskazówka** Do CTE możesz odwołać się z treści innego CTE, z powiązanego zapytania lub wyrażenia DML. Oba typy odwołań CTE pokazane są na listingu 9.2 — `PobierzNazwiskaCTE` wywoływane jest przez `PobierzKontaktCTE`, a `PobierzKontaktCTE` jest wywoływane przez zapytanie powiązane z CTE.

Rekurencyjne zapytania nazwane

Rekurencyjne CTE to takie wyrażenie, które wywołuje samo siebie wielokrotnie, by ustalić podzbiór danych aż do momentu uzyskania pełnego wyniku. CTE może odwołać się do samego siebie, co jest użytecznym mechanizmem do odpytywania o dane hierarchiczne. Rekurencyjne CTE są podobne do nierekurencyjnych CTE, ale w treści CTE znajduje się wiele zbiorów zapytań generujących zbiory wyników, w których wiersze są połączone za pomocą operatora zbiorów `UNION ALL`. Przynajmniej jedno z zapytań z treści rekurencyjnego CTE nie może odwoływać się do CTE — zapytanie takie nazywane jest **zapytaniem kotwiczącym** (ang. *anchor query*). Rekurencyjne CTE zawierają też jedno lub więcej rekurencyjnych zapytań odwołujących się do CTE. Rekurencyjne zapytania są połączone ze sobą za pomocą zapytania kotwiczącego (lub zapytań kotwiczących) w treści CTE. Wymagają operatora `UNION ALL` najwyższego poziomu łączącego rekurencyjne i nierekurencyjne zapytania. Wiele zapytań kotwiczących można łączyć za pomocą `UNION ALL`. Rekurencja kończy się, gdy zapytanie nie zwróci żadnych wierszy. Na listingu 9.3 znajduje się proste rekurencyjne CTE pobierające zbiór wyników zawierający liczby od 1 do 10.

Listing 9.3. *Proste rekurencyjne CTE*

```
WITH Liczby (n)
AS (
SELECT 1 AS n
  UNION ALL
SELECT n + 1
  FROM Liczby
WHERE n < 10 )

SELECT n FROM Liczby;
```

CTE z listingu 9.3 zaczyna się od deklaracji definiującej nazwę CTE i zwracane kolumny.

```
WITH Liczby (n)
```

Treść CTE zawiera pojedyncze zapytanie kotwiczące zwracające pojedynczy wiersz z liczbą 1 w kolumnie *n*.

```
SELECT 1 AS n
```

Zapytanie kotwiczące jest łączone z zapytaniem rekurencyjnym za pomocą operatora zbiorów `UNION ALL`. Zapytanie rekurencyjne zawiera odwołanie wewnętrzne do CTE `Liczby`, które dodaje 1 do kolumny `n` przy każdym rekurencyjnym wywołaniu. Klauzula `WHERE` ogranicza zbiór wyników do pierwszych dziesięciu liczb.

```
SELECT n + 1
  FROM Liczby
  WHERE n < 10
```

Rekurencyjne CTE mają domyślnie ustawioną maksymalną głębokość rekurencji 100. Oznacza to, że zapytanie rekurencyjne w treści CTE może wywołać samo siebie maksymalnie 100 razy. Możesz użyć opcji `MAXRECURSION`, by zwiększyć maksymalną wartość dla każdego CTE osobno. Na listingu 9.4 znajduje się CTE z listingu 9.3 zmodyfikowane w taki sposób, by zwracało liczby od 1 do 1000. Zmodyfikowane zapytanie korzysta z opcji `MAXRECURSION` zwiększającej maksymalny poziom rekurencji. Bez opcji `MAXRECURSION` to CTE zwróciłoby błąd po 100 poziomach rekurencji.

Listing 9.4. *Rekurencyjne CTE z opcją MAXRECURSION*

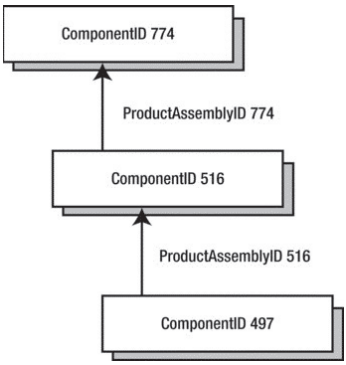
```
WITH Liczby (n)
AS (
SELECT 0 AS n
  UNION ALL
SELECT n + 1
  FROM Liczby
WHERE n < 1000 )
SELECT n
  FROM Liczby OPTION (MAXRECURSION 1000);
```

Wartość `MAXRECURSION` musi być z zakresu od `0` do `32767`. SQL Server wyrzuca wyjątek, jeśli zostanie przekroczony limit `MAXRECURSION`. Wartość `0` w zmiennej `MAXRECURSION` oznacza brak ograniczeń poziomów rekurencji dla CTE. Opcji tej należy używać ostrożnie — jeśli nie ograniczysz odpowiednio wyników w klauzuli `WHERE` zapytania, możesz znaleźć się w nieskończonej pętli.

■ **Wskazówka** Utworzenie trwałej tabeli z kolejnymi liczbami może być wydajniejsze niż korzystanie z rekurencyjnego CTE do generowania liczb szczególnie wtedy, kiedy często będziesz generował liczby w ten sposób.

Rekurencyjne CTE są użyteczne przy odpytywaniu o dane zapisane w postaci rozgałęzionego grafu. Tabela `Production.BillOfMaterials` z bazy danych *AdventureWorks* zawiera przykład takich danych. W tabeli tej znajdują się dwie ważne kolumny `ComponentID` i `ProductAssemblyID`, które odwzorowują strukturę hierarchiczną. `ComponentID` to unikalna liczba identyfikująca każdy z komponentów wykorzystywanych przez *AdventureWorks* przy tworzeniu swoich produktów. `ProductAssemblyID` oznacza bardziej złożony komponent utworzony z jednego lub większej liczby komponentów z listy. Na rysunku 9.2 pokazano relacje pomiędzy komponentami i bardziej złożonymi produktami w bazie danych *AdventureWorks*.

Rekurencyjne CTE pokazane na listingu 9.5 pobiera pełną listę części (**BOM**, ang. *bills of materials*) dla wskazanego komponentu. Komponent wykorzystany w przykładzie to srebrny 48-calowy rower Mountain-100 mający w bazie *AdventureWorks* identyfikator `ComponentID` równy `774`. Część wyników pokazana jest na rysunku 9.3.



Rysunek 9.2. *Zależności pomiędzy komponentami i produktami*

Listing 9.5. *Rekurencyjne CTE zwracające BOM*

```
DECLARE @ComponentID int = 774;

WITH BillOfMaterialsCTE
(
  BillOfMaterialsID,
  ProductAssemblyID,
  ComponentID,
  Quantity,
  Level
)
AS
(
SELECT
  bom.BillOfMaterialsID,
  bom.ProductAssemblyID,
  bom.ComponentID,
  bom.PerAssemblyQty AS Quantity,
  0 AS Level
FROM Production.BillOfMaterials bom
WHERE bom.ComponentID = @ComponentID

UNION ALL

SELECT
  bom.BillOfMaterialsID,
  bom.ProductAssemblyID,
  bom.ComponentID,
  bom.PerAssemblyQty,
  Level + 1
FROM Production.BillOfMaterials bom
INNER JOIN BillOfMaterialsCTE bomcte
  ON bom.ProductAssemblyID = bomcte.ComponentID
WHERE bom.EndDate IS NULL
)

SELECT
  bomcte.ProductAssemblyID,
  p.ProductID,
  p.ProductNumber,
  p.Name,
  p.Color,
  bomcte.Quantity,
  bomcte.Level
FROM BillOfMaterialsCTE bomcte
INNER JOIN Production.Product p
  ON bomcte.ComponentID = p.ProductID
ORDER BY bomcte.Level;
```

Results | Messages

	ProductAssemblyID	ProductID	ProductNumber	Name	Color	Quantity	Level
1	NULL	774	BK-M82S-48	Mountain-100 Silver, 48	Silver	1.00	0
2	774	516	SA-M687	HL Mountain Seat Assembly	NULL	1.00	1
3	774	741	FR-M94S-52	HL Mountain Frame - Silver, 48	Silver	1.00	1
4	774	807	HS-3479	HL Headset	NULL	1.00	1
5	774	810	HB-M918	HL Mountain Handlebars	NULL	1.00	1
6	774	817	FW-M928	HL Mountain Front Wheel	Black	1.00	1
7	774	825	RW-M928	HL Mountain Rear Wheel	Black	1.00	1
8	774	894	RD-2308	Rear Derailleur	Silver	1.00	1
9	774	907	RB-9231	Rear Brakes	Silver	1.00	1
10	774	937	PD-M562	HL Mountain Pedal	Silver/Black	1.00	1
11	774	945	FD-2342	Front Derailleur	Silver	1.00	1
12	774	948	FB-9873	Front Brakes	Silver	1.00	1
13	774	951	CS-9183	HL Crankset	Black	1.00	1
14	774	952	CH-0234	Chain	Silver	1.00	1
15	774	996	BB-9108	HL Bottom Bracket	NULL	1.00	1
16	996	3	BE-2349	BB Ball Bearing	NULL	10.00	2
17	996	526	SH-9312	HL Shell	NULL	1.00	2
18	951	319	CA-7457	HL Crankarm	Black	2.00	2

Rysunek 9.3. *Część wyników rekurencyjnego CTE generującego BOM*

Tak jak wcześniejsze przykłady CTE, listing 9.3 rozpoczyna się od nazwy CTE i deklaracji listy kolumn.

```
WITH BillOfMaterialsCTE
(
  BillOfMaterialsID,
  ProductAssemblyID,
  ComponentID,
  Quantity,
  Level
)
```

Zapytanie kotwiczące po prostu pobiera z tabeli wiersz, dla którego `CoponentID` jest równy wskazanemu identyfikatorowi. Jest to komponent najwyższego poziomu z BOM, w tym przypadku 774. Zauważ, że CTE może odwoływać się do zmiennych T-SQL tak, jak `@ComponentID` w przykładzie.

```
SELECT
  bom.BillOfMaterialsID,
  bom.ProductAssemblyID,
  bom.ComponentID,
  bom.PerAssemblyQty AS Quantity,
  0 AS Level
FROM Production.BillOfMaterials bom
WHERE bom.ComponentID = @ComponentID
```

Rekurencyjne zapytania pobierają kolejne poziomy BOM z CTE, gdzie `ProductAssemblyID` każdego wiersza ma taką samą wartość jak `ComponentID` wierszy wyższego poziomu. Chodzi o to, że rekurencyjne zapytanie CTE pobiera wiersze niższych poziomów z hierarchii, korzystając z zależności hierarchicznych pokazanych wcześniej na rysunku 9.2.

```
SELECT
  bom.BillOfMaterialsID,
  bom.ProductAssemblyID,
  bom.ComponentID,
  bom.PerAssemblyQty,
  Level + 1
FROM Production.BillOfMaterials bom
INNER JOIN BillOfMaterialsCTE bomcte
  ON bom.ProductAssemblyID = bomcte.ComponentID
WHERE bom.EndDate IS NULL
```

Zapytanie nazwane ma dołączone zapytanie SELECT łączące wyniki jego działania z tabelą Production.Product, dzięki czemu pobierane są szczegółowe informacje na temat produktu, takie jak nazwa i kolor komponentu.

```
SELECT
  bomcte.ProductAssemblyID,
  p.ProductID,
  p.ProductNumber,
  p.Name,
  p.Color,
  bomcte.Quantity,
  bomcte.Level
FROM BillOfMaterialsCTE bomcte
INNER JOIN Production.Product p
  ON bomcte.ComponentID = p.ProductID
```

Ograniczenia prostych CTE opisywane wcześniej w tym rozdziale dotyczą też rekurencyjnych CTE. W rekurencyjnych CTE pojawiają się też dodatkowe ograniczenia.

- Rekurencyjne CTE muszą mieć w treści przynajmniej jedno zapytanie kotwiczące i przynajmniej jedno zapytanie rekurencyjne. Wszystkie zapytania kotwiczące muszą znajdować się przed zapytaniami rekurencyjnymi.
- Wszystkie zapytania kotwiczące muszą być połączone za pomocą operatorów działających na zbiorach UNION, UNION ALL, INTERSECT i EXCEPT. Przy korzystaniu z wielu zapytań kotwiczących i zapytań rekurencyjnych pierwsze zapytanie rekurencyjne i ostatnie zapytanie kotwiczące muszą być połączone za pomocą operatora UNION ALL. Dodatkowo wszystkie zapytania rekurencyjne muszą być połączone przy użyciu UNION ALL.
- Typy danych wszystkich kolumn we wszystkich zapytaniach kotwiczących oraz zapytaniach rekurencyjnych muszą być takie same.
- Klauzula FROM zapytania rekurencyjnego powinna odwoływać się do nazwy CTE tylko raz.
- Zapytania rekurencyjne nie mogą zawierać następujących operatorów i słów kluczowych: GROUP BY, HAVING, LEFT JOIN, RIGHT JOIN, OUTER JOIN i SELECT DISTINCT. Zapytania rekurencyjne nie mogą też zawierać funkcji agregujących (takich jak SUM i MAX), funkcji okna, podzapytań czy wskazówek w rekurencyjnym wywołaniu CTE.

Funkcje okna

SQL Server 2014 zawiera funkcje okna, które pozwalają dzielić wyniki zapytań i dodawać numerowanie, rankingi i funkcje agregujące do każdej części danych (partycji). Kluczowa dla funkcji okna jest klauzula OVER, która pozwala definiować partycje i w niektórych przypadkach kolejność wierszy w partycji. W tym podrozdziale omówione zostaną funkcje okna w SQL Server 2014 wraz z funkcjami rozszerzającymi klauzulę OVER, które umożliwiają numerowanie, porządkowanie i agregowanie.

Funkcja ROW_NUMBER

Funkcja ROW_NUMBER przyjmuje klauzulę OVER z klauzulą ORDER BY i opcjonalną klauzulą PARTITION BY. Kod z listingu 9.6 pobiera nazwiska z tabeli Person.Person. Klauzula OVER jest wykorzystana do podzielenia wierszy ze względu na zawartość kolumny LastName i porządkuje wiersze każdej partycji ze względu na wartości kolumn LastName, FirstName i MiddleName. Funkcja ROW_NUMBER wykorzystana jest do przypisania każdemu wierszowi liczby.

Listing 9.6. *ROW_NUMBER z partycjonowaniem*

```
SELECT
  ROW_NUMBER() OVER
  (
    PARTITION BY
    LastName
    ORDER BY
    LastName,
    FirstName,
    MiddleName
  ) AS Number,
  LastName,
  FirstName,
  MiddleName
FROM Person.Person;
```

Partycja utworzona na listingu 9.6 działa jak okno przesuwające się przez zestaw wyników (stąd nazwa funkcja okna). Klauzula `ORDER BY` porządkuje wiersze każdej partycji ze względu na wartości zapisane w kolumnach `LastName`, `FirstName` i `MiddleName`. SQL Server korzysta z funkcji `ROW_NUMBER` w każdej partycji. Powoduje to, że funkcja `ROW_NUMBER` numeruje wszystkie wiersze wyniku, zaczynając numerowanie od `1` dla każdej kolejnej wartości `LastName`, co jest widoczne na rysunku 9.4.

Results | Messages

	Number	LastName	FirstName	MiddleName
1	1	Abbas	Syed	E
2	1	Abel	Catherine	R.
3	1	Abercrombie	Kim	NULL
4	2	Abercrombie	Kim	NULL
5	3	Abercrombie	Kim	B
6	1	Abolrous	Hazem	E
7	2	Abolrous	Sam	NULL
8	1	Acevedo	Humberto	NULL
9	1	Achong	Gustavo	NULL
10	1	Ackerman	Pilar	NULL
11	2	Ackerman	Pilar	G
12	1	Adams	Aaron	B
13	2	Adams	Adam	NULL
14	3	Adams	Alex	C
15	4	Adams	Alexandra	J
16	5	Adams	Allison	L
17	6	Adams	Amanda	P
18	7	Adams	Amber	NULL

Rysunek 9.4. *Użycie ROW_NUMBER do numerowania wierszy w partycji*

■ **Uwaga** Przy korzystaniu z klauzuli `PARTITION BY` należy umieścić ją przed `ORDER BY` w klauzuli `OVER`.

Z funkcji `ROW_NUMBER` można też korzystać bez klauzuli `PARTITION BY` i wtedy cały zwracany zbiór danych jest traktowany jak jedna partycja. Traktowanie wszystkich wyników jak pojedynczej partycji może być użyteczne w niektórych przypadkach, ale najczęściej stosuje się partycjonowanie.

Stronicowanie zapytania za pomocą OFFSET/FETCH

SQL Server zawiera różne mechanizmy umożliwiające stronicowanie zwracanych danych. Tradycyjnym sposobem stronicowania jest wykorzystanie operatora `TOP` pozwalającego wybrać *n* pierwszych wierszy zwróconych przez zapytanie. SQL Server 2005 wprowadził funkcję `ROW_NUMBER`; za jej pomocą można osiągnąć podobny efekt w odrobinę inny sposób.

SQL Server 2012 wprowadził do wyrażenia `SELECT` nowe słowa kluczowe służące do obsługi stronicowania zapytania.

Słowo kluczowe `OFFSET` umożliwia dużo łatwiejsze stronicowanie. Pozwala ono dosłownie określić numer wiersza, od którego chcesz uzyskać wiersze danych. Następnie możesz użyć `FETCH`, by pobrać określoną liczbę wierszy wyniku. Jeśli połączysz `OFFSET` i `FETCH` z klauzulą `ORDER BY`, możesz zwrócić dowolnie wybraną część danych wyniku, przesuwając się przez dane wedle potrzeby.

Na listingu 9.7 pokazano stronicowane wykonane za pomocą `OFFSET` i `FETCH`. Procedura składowana korzysta z klauzul `OFFSET` i `FETCH` do pobrania wierszy z tabeli `Person.Person` z bazy danych *AdventureWorks* na podstawie wartości parametrów przekazanych przy wywołaniu procedury. Procedura określa, w jaki sposób stronicowanie ma być wykonywane za pomocą parametrów wejściowych `@RowsPerPage` i `@StartPageNum`. Parametr `@RowsPerPage` określa, jak wiele wierszy powinno znajdować się na stronie wyników. `OFFSET` określa liczbę wierszy, jakie należy opuścić, licząc od początku zbioru danych zwracanego przez zapytanie. `FETCH` określa liczbę wierszy, jaką należy zwrócić na każdej stronie wyników zapytania.

Listing 9.7. *Przykład OFFSET/FETCH*

```
CREATE PROCEDURE Person.PobierzKontakty
  @StartPageNum int,
  @RowsPerPage int
AS
  SELECT
    LastName,
    FirstName,
    MiddleName
  FROM Person.Person
  ORDER BY
    LastName,
    FirstName,
    MiddleName
  OFFSET (@StartPageNum - 1) * @RowsPerPage ROWS
  FETCH NEXT @RowsPerPage ROWS ONLY;
GO
```

Przykładowe wywołanie procedury wykorzystującej klauzulę `OFFSET/FETCH`

```
EXEC Person.PobierzKontakty 16,10
```

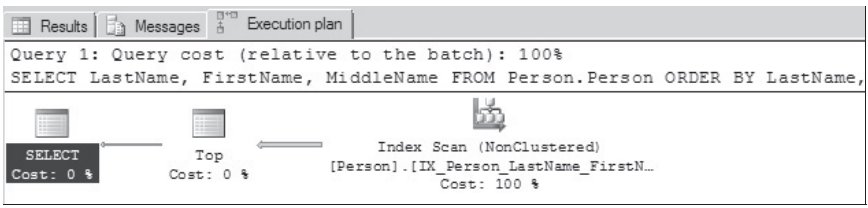
przekazuje wartość 10 jako parametr `@RowsPerPage` i wartość 16 jako parametr `@StartPageNum` oraz zwraca 10 wierszy strony 16, co widać na rysunku 9.5. Słowo kluczowe `OFFSET` w wyrażeniu `SELECT` omija wiersze przed stroną określoną wartościami parametrów `@StartPageNum` i `@RowsPerPage` — w tym przykładzie omija 150 wierszy i zaczyna zwracać wyniki od 151. wiersza. Słowo kluczowe `FETCH` zwraca tyle wierszy, ile określono w zmiennej `@RowsPerPage` (10). Plan zapytania jest pokazany na rysunku 9.6.

Zapytanie z listingu 9.7 jest dużo bardziej czytelnym i eleganckim rozwiązaniem stronicowania wyników niż wykorzystanie klauzuli `TOP` czy funkcji `ROW_NUMBER` z zapytaniami nazwanymi. Jedynym wyjątkiem będzie sytuacja, gdy korzystasz z `OFFSET/FETCH` i zechcesz uzyskać `ROW_NUMBER`; w takim przypadku będziesz musiał dodać `ROW_NUMBER` do swojego zapytania. Dlatego klauzula `OFFSET/FETCH` to dużo bardziej przejrzysty sposób implementowania stronicowania *ad hoc*.

Results | Messages | Execution plan

	LastName	FirstName	MiddleName
1	Alexander	Cassidy	NULL
2	Alexander	Chloe	NULL
3	Alexander	Christian	C
4	Alexander	Connor	G
5	Alexander	Dakota	NULL
6	Alexander	Dalton	NULL
7	Alexander	David	NULL
8	Alexander	Destiny	NULL
9	Alexander	Devin	NULL
10	Alexander	Dylan	NULL

Rysunek 9.5. *Użycie OFFSET i FETCH do implementacji stronicowania po stronie klienta*



Rysunek 9.6. *Plan zapytania dla implementacji stronicowania po stronie klienta za pomocą OFFSET i FETCH*

Korzystając z `OFFSET` i `FETCH`, pamiętaj o pewnych ograniczeniach.

- `OFFSET` i `FETCH` muszą być stosowane z klauzulą `ORDER BY`.
- `FETCH` nie można używać bez `OFFSET`, ale `OFFSET` można użyć bez `FETCH`.
- Liczba wierszy określona w klauzuli `OFFSET` musi być większa lub równa `0`.
- Liczba wierszy określona w klauzuli `FETCH` musi być większa lub równa `1`.
- W zapytaniach, w których używasz `OFFSET` i `FETCH`, nie możesz korzystać z operatora `TOP`.
- Wartości `OFFSET` i `FETCH` muszą być stałe lub muszą być parametrami o wartości całkowitej.
- `OFFSET` i `FETCH` nie są obsługiwane z klauzulą `OVER`.
- `OFFSET` i `FETCH` nie są obsługiwane dla widoków indeksowanych i widoków z `WITH CHECK OPTION`.

Generalnie w SQL Server 2012 i nowszych połączenie `OFFSET` i `FETCH` jest najbardziej przejrzystym podejściem do stronicowania wyników zapytania.

Funkcje RANK i DENSE_RANK

Funkcje `RANK` i `DENSE_RANK` to funkcje numerujące dostępne w SQL Server. Obie te funkcje przypisują wartość liczbową do każdego wiersza w partycji, ale różnią się sposobem numerowania wierszy. Oto przykład.

- Jeśli masz trzy wartości 7, 7 i 9, funkcja `RANK` przypisze im numery 1, 1 i 3. Dzieje się tak dlatego, że dwie 7 są przypisane do pierwszego miejsca, podczas gdy 9 jest trzecia na liście. `RANK` nie uwzględnia tego połączenia przy obliczaniu numeru dla wartości 9.
- Funkcja `DENSE_RANK` w takiej sytuacji przypisze numery 1, 1 i 2. Dzieje się tak, ponieważ `DENSE_RANK` łączy obie 7 na pozycji pierwszej i umieszcza 9 na drugiej pozycji.

Nie ma dobrego lub złego sposobu numerowania danych w przypadku braku wymagań biznesowych. SQL Server udostępnia dwa sposoby i możesz wybrać ten, który lepiej pasuje do wymagań biznesowych.

Załóżmy, że chcesz ustalić najlepsze dni sprzedaży firmy *AdventureWorks* w roku kalendarzowym 2012. Może to być opisane pytaniem: „W jakie dni mieliśmy najlepszą sprzedaż w 2012 roku?”. Funkcja `RANK` pozwala łatwo uzyskać takie informacje, co widać na listingu 9.8. Część wyników można zobaczyć na rysunku 9.7.

Listing 9.8. *Ranking dziennej sprzedaży AdventureWorks*

```
WITH TotalSalesBySalesDate
(
  DailySales,
  OrderDate
)
AS
(
  SELECT
    SUM(soh.SubTotal) AS DailySales,
    soh.OrderDate
  FROM Sales.SalesOrderHeader soh
  WHERE soh.OrderDate >= '20120101'
    AND soh.OrderDate < '20130101'
  GROUP BY soh.OrderDate
)
SELECT
  RANK() OVER
  (
    ORDER BY
    DailySales DESC
  ) AS Ranking,
  DailySales,
  OrderDate
FROM TotalSalesBySalesDate
ORDER BY Ranking;
```

	Ranking	DailySales	OrderDate
1	1	3617451,9232	2012-06-30 00:00:00.000
2	2	3064632,131	2012-09-30 00:00:00.000
3	3	2888213,6087	2012-07-31 00:00:00.000
4	4	2408187,2542	2012-05-30 00:00:00.000
5	5	2402984,3967	2012-12-31 00:00:00.000
6	6	2306297,6081	2012-03-30 00:00:00.000
7	7	2205251,3127	2012-10-30 00:00:00.000
8	8	1919379,3786	2012-01-01 00:00:00.000
9	9	1815297,7111	2012-08-30 00:00:00.000
10	10	1473667,5818	2012-01-29 00:00:00.000
11	11	1329986,583	2012-11-30 00:00:00.000
12	12	1016116,847	2012-04-30 00:00:00.000
13	13	895032,9488	2012-02-29 00:00:00.000
14	14	46135,95	2012-03-14 00:00:00.000
15	15	44273,4146	2012-05-22 00:00:00.000
16	16	42354,40	2012-02-21 00:00:00.000
17	17	40352,6064	2012-05-18 00:00:00.000
18	18	40174,3264	2012-03-29 00:00:00.000

Rysunek 9.7. *Ranking wartości dziennej sprzedaży AdventureWorks*

Listing 9.8 zawiera CTE zwracające dwie kolumny, `DailySales` i `OrderDate`. Kolumna `DailySales` to suma sprzedaży pogrupowana według wartości `OrderDate`. Wyniki są ograniczane przez klauzulę `WHERE` do 2012 roku.

```
WITH TotalSalesBySalesDate
(
  DailySales,
  OrderDate
)
AS
(
  SELECT
    SUM(soh.SubTotal) AS DailySales,
    soh.OrderDate
  FROM Sales.SalesOrderHeader soh
  WHERE soh.OrderDate >= '20120101'
    AND soh.OrderDate < '20130101'
  GROUP BY soh.OrderDate
)
```

Funkcja `RANK` jest wykorzystywana z klauzulą `OVER`, by numerować wiersze zwracane przez CTE w kolejności malejącej (od największych do najmniejszych) wartości kolumny `DailySales`.

```
SELECT
  RANK() OVER
  (
    ORDER BY
    DailySales DESC
  ) AS Ranking,
  DailySales,
  OrderDate
FROM TotalSalesBySalesDate
ORDER BY Ranking;
```

Tak jak funkcja `ROW_NUMBER`, funkcja `RANK` może przyjmować klauzulę `PARTITION BY` w klauzuli `OVER`. Kod z listingu 9.9 rozszerza poprzedni przykład i wykorzystuje klauzulę `PARTITION BY` do utworzenia rankingu dziennej sprzedaży dla każdego miesiąca. Tego typu zapytanie może odpowiedzieć na pytanie biznesowe: „W jakie dni każdego miesiąca 2012 roku sprzedaż firmy *AdventureWorks* była największa?”. Część wyników można zobaczyć na rysunku 9.8.

Listing 9.9. *Ustalenie rankingów dziennej sprzedaży w podziale na miesiące*

```
WITH TotalSalesBySalesDatePartitioned
(
  DailySales,
  OrderMonth,
  OrderDate
)
AS
(
  SELECT
    SUM(soh.SubTotal) AS DailySales,
    DATENAME(MONTH, soh.OrderDate) AS OrderMonth,
    soh.OrderDate
  FROM Sales.SalesOrderHeader soh
  WHERE soh.OrderDate >= '20120101'
    AND soh.OrderDate < '20130101'
  GROUP BY soh.OrderDate
)
SELECT
  RANK() OVER
  (
    PARTITION BY
    OrderMonth
```

```
    ORDER BY
    DailySales DESC
  ) AS Ranking,
  DailySales,
  OrderMonth,
  OrderDate
FROM TotalSalesBySalesDatePartitioned
ORDER BY DATEPART(mm,OrderDate),
  Ranking;
```

Results | Messages | Execution plan

	Ranking	DailySales	OrderMonth	OrderDate
1	1	1919379,3786	January	2012-01-01 00:00:00.000
2	2	1473667,5818	January	2012-01-29 00:00:00.000
3	3	40060,0682	January	2012-01-11 00:00:00.000
4	4	39856,7882	January	2012-01-26 00:00:00.000
5	5	35401,14	January	2012-01-10 00:00:00.000
6	5	35401,14	January	2012-01-22 00:00:00.000
7	7	29325,2582	January	2012-01-24 00:00:00.000
8	8	24844,61	January	2012-01-03 00:00:00.000
9	9	24666,33	January	2012-01-13 00:00:00.000
10	10	24087,7328	January	2012-01-06 00:00:00.000
11	11	23363,6346	January	2012-01-31 00:00:00.000
12	12	22867,8164	January	2012-01-14 00:00:00.000
13	13	21990,4382	January	2012-01-30 00:00:00.000
14	14	21469,62	January	2012-01-15 00:00:00.000
15	15	21088,06	January	2012-01-07 00:00:00.000
16	16	20884,78	January	2012-01-05 00:00:00.000
17	17	18590,4482	January	2012-01-02 00:00:00.000
18	17	18590,4482	January	2012-01-19 00:00:00.000

Rysunek 9.8. *Część wyników rankingów dziennej sprzedaży w podziale na miesiące*

Zapytanie z listingu 9.9, tak jak przykład z listingu 9.8, zaczyna się od CTE obliczającego dzienną sprzedaż dla wszystkich dni roku. Główna różnica między tym CTE i poprzednim przykładem jest taka, że kod z listingu 9.9 zwraca dodatkową kolumnę `OrderMonth`. Oto treść tego CTE.

```
WITH TotalSalesBySalesDatePartitioned
(
  DailySales,
  OrderMonth,
  OrderDate
)
AS
(
  SELECT
    SUM(soh.SubTotal) AS DailySales,
    DATENAME(MONTH, soh.OrderDate) AS OrderMonth,
    soh.OrderDate
  FROM Sales.SalesOrderHeader soh
  WHERE soh.OrderDate >= '20120101'
    AND soh.OrderDate < '20130101'
  GROUP BY soh.OrderDate
)
```

Zapytanie `SELECT` powiązane z CTE korzysta z funkcji `RANK`, by przypisać pozycje rankingowe wynikom. Klauzula `PARTITION BY` została wykorzystana do podzielenia wyników według `OrderMonth`, dzięki czemu pozycje rankingowe zaczynają się od 1 dla każdego miesiąca. Oto przykład.

```
SELECT
  RANK() OVER
  (
    PARTITION BY
    OrderMonth
    ORDER BY
    DailySales DESC
  ) AS Ranking,
  DailySales,
  OrderMonth,
  OrderDate
FROM TotalSalesBySalesDatePartitioned
ORDER BY DATEPART(mm,OrderDate),
  Ranking;
```

Gdy funkcja `RANK` spotyka dwie równe wartości `DailySales` w tej samej partycji, przypisuje taką samą pozycję w rankingu obu wartościom i pomija kolejną liczbę. Na rysunku 9.9 można zobaczyć, że wartość `DailySales` dla dwóch dni w październiku 2012 wynosiła `7479,3221` dolarów, co sprawiło, że funkcja `RANK` przypisała obu dniom w rankingu wartość `25`. Następnie pominęła wartość `26` i przypisała kolejnemu wierszowi w rankingu wartość `27`.

Results | Messages | Execution plan

	Ranking	DailySales	OrderMonth	OrderDate
296	22	8761,1932	October	2012-10-16 00:00:00.000
297	23	8745,4303	October	2012-10-01 00:00:00.000
298	24	7712,5325	October	2012-10-04 00:00:00.000
299	25	7479,3221	October	2012-10-21 00:00:00.000
300	25	7479,3221	October	2012-10-08 00:00:00.000
301	27	7302,5178	October	2012-10-18 00:00:00.000
302	28	7036,8471	October	2012-10-06 00:00:00.000
303	28	7036,8471	October	2012-10-22 00:00:00.000
304	30	5297,7596	October	2012-10-14 00:00:00.000
305	31	2443,35	October	2012-10-03 00:00:00.000

Rysunek 9.9. *Funkcja RANK pomija kolejną liczbę w przypadku dwóch identycznych wartości*

Funkcja `DENSE_RANK`, tak jak funkcja `RANK`, przypisuje identycznym wartościom taką samą pozycję w rankingu, ale z jedną ważną różnicą — nie pomija kolejnej pozycji rankingu. Na listingu 9.10 znajduje się kod z listingu 9.9 zmodyfikowany w taki sposób, by korzystać z funkcji `RANK` i `DENSE_RANK`. Jak możesz zobaczyć na rysunku 9.10, `DENSE_RANK` też przypisuje taką samą wartość `Ranking` do obu identycznych wartości sprzedaży, ale w odróżnieniu od funkcji `RANK` nie pomija kolejnej wartości `Ranking`.

Listing 9.10. *Użycie DENSE_RANK do uporządkowania najlepszych wyników sprzedaży w miesiącu*

```
WITH TotalSalesBySalesDatePartitioned
(
  DailySales,
  OrderMonth,
  OrderDate
)
AS
(
  SELECT
    SUM(soh.SubTotal) AS DailySales,
    DATENAME(MONTH, soh.OrderDate) AS OrderMonth,
    soh.OrderDate
  FROM Sales.SalesOrderHeader soh
  WHERE soh.OrderDate >= '20120101'
    AND soh.OrderDate < '20130101'
  GROUP BY soh.OrderDate
```

```
)
SELECT
  RANK() OVER
  (
    PARTITION BY
      OrderMonth
    ORDER BY
      DailySales DESC
  ) AS Ranking,

DENSE_RANK() OVER
(
  PARTITION BY
    OrderMonth
  ORDER BY
    DailySales DESC
) AS Dense_Ranking,
  DailySales,
  OrderMonth,
  OrderDate
FROM TotalSalesBySalesDatePartitioned
ORDER BY DATEPART(mm,OrderDate),
  Ranking;
```

Results | Messages | Execution plan

	Ranking	Dense_Ranking	DailySales	OrderMonth	OrderDate
1	1	1	1919379,3786	January	2012-01-01 00:00:00.000
2	2	2	1473667,5818	January	2012-01-29 00:00:00.000
3	3	3	40060,0682	January	2012-01-11 00:00:00.000
4	4	4	39856,7882	January	2012-01-26 00:00:00.000
5	5	5	35401,14	January	2012-01-10 00:00:00.000
6	5	5	35401,14	January	2012-01-22 00:00:00.000
7	7	6	29325,2582	January	2012-01-24 00:00:00.000
8	8	7	24844,61	January	2012-01-03 00:00:00.000
9	9	8	24666,33	January	2012-01-13 00:00:00.000
10	10	9	24087,7328	January	2012-01-06 00:00:00.000
11	11	10	23363,6346	January	2012-01-31 00:00:00.000
12	12	11	22867,8164	January	2012-01-14 00:00:00.000
13	13	12	21990,4382	January	2012-01-30 00:00:00.000
14	14	13	21469,62	January	2012-01-15 00:00:00.000
15	15	14	21088,06	January	2012-01-07 00:00:00.000
16	16	15	20884,78	January	2012-01-05 00:00:00.000

Rysunek 9.10. *DENSE_RANK nie pomija kolejnej pozycji rankingu po powieleniu*

Funkcja NTILE

`NTILE` to inna funkcja do tworzenia rankingów, która odpowiada na odrobinę inne zapotrzebowanie. Funkcja ta dzieli wyniki na przybliżone *n*-tyle. *N*-tylem może być kwartyl (podział na kawałki o wielkości 1/4 lub 25%), kwintyl (podział na kawałki o wielkości 1/5 lub 20%) czy percentyl (podział na kawałki o wielkości 1/100 lub 1%) albo każda inna część całości, jaką możesz sobie wyobrazić. `NTILE` dzieli zbiór wyników na przybliżone *n*-tyle, ponieważ liczba zwracanych wierszy nie musi być dokładnie podzielna przez określoną liczbę grup. Tabela z 27 wierszami na przykład nie da się dokładnie podzielić na kwartyle czy kwintyle. Jeśli odpytujesz za pomocą funkcji `NTILE` i liczba wierszy nie da się dokładnie podzielić na określoną liczbę grup, `NTILE` tworzy grupy o dwóch rozmiarach. Większe grupy mają o jeden wiersz więcej niż mniejsze grupy i są umieszczane na początku. Gdy przykładowo 27 wierszy jest podzielonych na kwintyle (1/5), dwie pierwsze grupy mają po 6 wierszy, a ostatnie trzy grupy mają po 5 wierszy.

Tak jak w przypadku funkcji ROW_NUMBER, do klauzuli OVER możesz dołączyć zarówno PARTITION BY, jak i ORDER BY. Funkcja NTILE wymaga dodatkowego parametru, który określa, na jak wiele grup trzeba podzielić wyniki zapytania.

Funkcja NTILE przydaje się przy odpowiadaniu na pytania biznesowe w stylu: „Którzy sprzedawcy zaliczają się do 25% najlepszych w lipcu 2013?" oraz „Jakie były wyniki ich sprzedaży?". Kod z listingu 9.11 korzysta z NTILE, by podzielić sprzedawców *AdventureWorks* na cztery grupy, z których każda obejmuje 25% pracowników. Klauzula ORDER BY sprawia, że wiersze są przypisywane do grupy na podstawie wyników sprzedaży. Wyniki zostały pokazane na rysunku 9.11.

Listing 9.11. *Użycie NTILE do grupowania i utworzenia rankingu sprzedawców*

```
WITH SalesTotalBySalesPerson
(
  SalesPersonID, SalesTotal
)
AS
(
  SELECT
    soh.SalesPersonID,
    SUM(soh.SubTotal) AS SalesTotal
  FROM Sales.SalesOrderHeader soh
  WHERE DATEPART(YEAR, soh.OrderDate) = 2013
    AND DATEPART(MONTH, soh.OrderDate) = 2
  GROUP BY soh.SalesPersonID
)
SELECT
  NTILE(4) OVER( ORDER BY st.SalesTotal DESC) AS Tile,
  p.LastName,
  p.FirstName,
  p.MiddleName,
  st.SalesPersonID,
  st.SalesTotal
FROM SalesTotalBySalesPerson st
INNER JOIN Person.Person p ON st.SalesPersonID = p.BusinessEntityID;
```

Results | Messages | Execution plan

	Tile	LastName	FirstName	MiddleName	SalesPersonID	SalesTotal
1	1	Pak	Jae	B	289	430730,8928
2	1	Carson	Jillian	NULL	277	400651,4944
3	1	Blythe	Michael	G	275	314936,4504
4	1	Reiter	Tsvi	Michael	279	172527,4835
5	2	Campbell	David	R	283	155124,1524
6	2	Mitchell	Linda	C	276	88379,2611
7	2	Ito	Shu	K	281	87934,131
8	2	Varkey Chudukatil	Ranjit	R	290	56935,0391
9	3	Ansman-Wolfe	Pamela	O	280	49152,4316
10	3	Saraiva	José	Edvaldo	282	47113,0052
11	3	Jiang	Stephen	Y	274	43254,2036
12	4	Mensa-Annan	Tete	A	284	9479,9522
13	4	Vargas	Garrett	R	278	8091,5083
14	4	Alberts	Amy	E	287	968,4284

Rysunek 9.11. *Sprzedawcy AdventureWorks pogrupowani i uszeregowani za pomocą NTILE*

Na początku kodu znajduje się proste wyrażenie nazwane, zwracające wartość SalesPersonID i sumę wartości SubTotal dla zamówień z tabeli Sales.SalesOrderHeader. CTE ogranicza wyniki do wyników sprzedaży z lutego 2013 roku. Oto treść tego CTE.

```
WITH SalesTotalBySalesPerson
(
  SalesPersonID, SalesTotal
)
AS
(
  SELECT
    soh.SalesPersonID,
    SUM(soh.SubTotal) AS SalesTotal
  FROM Sales.SalesOrderHeader soh
  WHERE DATEPART(YEAR, soh.OrderDate) = 2013
    AND DATEPART(MONTH, soh.OrderDate) = 2
  GROUP BY soh.SalesPersonID
)
```

Zapytanie `SELECT` dołączone do tego CTE korzysta z `NTILE(4)` do grupowania sprzedawców *AdventureWorks* w czterech grupach zawierających około 25%. Klauzula `OVER` wskazuje, że grupy powinny być tworzone na podstawie `SalesTotal` w kolejności malejącej. Całe zapytanie `SELECT` wygląda tak.

```
SELECT
  NTILE(4) OVER( ORDER BY st.SalesTotal DESC) AS Tile,
  p.LastName,
  p.FirstName,
  p.MiddleName,
  st.SalesPersonID,
  st.SalesTotal
FROM SalesTotalBySalesPerson st
INNER JOIN Person.Person p ON st.SalesPersonID = p.BusinessEntityID;
```

Funkcje agregacyjne, analityczne i klauzula OVER

Jak już wcześniej napisano, funkcje numerujące i tworzące rankingi (`ROW_NUMBER`, `RANK` i tak dalej) współpracują z klauzulą `OVER` przy definiowaniu kolejności i partycjonowania przekazanych do nich wierszy za pomocą klauzul `ORDER BY` i `PARTITION BY`. Klauzula `OVER` dostarcza też funkcjonalności okna do funkcji agregacyjnych, takich jak `SUM`, `COUNT` i zdefiniowane przez użytkownika agregacje SQL CLR.

Funkcje okna mogą pomóc przy odpowiadaniu na typowe pytania biznesowe obejmujące tworzenie sum częściowych i średnich kroczących. Możesz na przykład użyć klauzuli `OVER` na tabeli `Purchasing.PurchaseOrderDetail` z bazy danych *AdventureWorks*, aby uzyskać sumę wartości zamówionych produktów w postaci sumy kroczącej. Możesz dodatkowo ograniczyć zakres zwracanych danych, na których zechcesz użyć funkcji agregującej, partycjonując zbiór wyników według `PurchaseOrderId`, co w efekcie wygeneruje podsumę dla każdego zamówienia. Przykładowe zapytanie pokazane zostało na listingu 9.12. Część jego wyników można zobaczyć na rysunku 9.12.

Listing 9.12. *Użycie klauzuli OVER z SUM*

```
SELECT
  PurchaseOrderID,
  ProductID,
  OrderQty,
  UnitPrice,
  LineTotal,
  SUM(LineTotal)
    OVER (PARTITION BY PurchaseOrderID ORDER BY ProductId
      RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
    AS CumulativeOrderQty
FROM Purchasing.PurchaseOrderDetail;
```

Results | Messages

	PurchaseOrderID	ProductID	OrderQty	UnitPrice	LineTotal	CumulativeOrderQty
1	1	1	4	50,26	201,04	201,04
2	2	359	3	45,12	135,36	135,36
3	2	360	3	45,5805	136,7415	272,1015
4	3	530	550	16,086	8847,30	8847,30
5	4	4	3	57,0255	171,0765	171,0765
6	5	512	550	37,086	20397,30	20397,30
7	6	513	550	26,5965	14628,075	14628,075
8	7	317	550	27,0585	14882,175	14882,175
9	7	318	550	33,579	18468,45	33350,625
10	7	319	550	46,0635	25334,925	58685,55
11	8	403	3	47,4705	142,4115	142,4115
12	8	404	3	45,3705	136,1115	278,523
13	8	405	3	49,644	148,932	427,455
14	8	406	3	45,3705	136,1115	563,5665
15	8	407	3	43,2705	129,8115	693,378
16	9	422	3	47,523	142,569	142,569
17	9	423	3	45,423	136,269	278,838
18	9	424	3	49,6965	149,0895	427,9275

Rysunek 9.12. *Część wyników z zapytania generującego sumę kroczącą*

Na listingu 9.12 znalazła się nowa klauzula.

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Nazywana jest **klauzulą ramki** (ang. *framing clause*) i w tym przypadku określa, że każda suma zawiera wszystkie wartości od pierwszego wiersza partycji do bieżącego wiersza. Klauzula taka ma sens tylko wtedy, kiedy wiersze są uporządkowane, a aby to uzyskać, korzystamy z klauzuli `ORDER BY ProductId`. Klauzula ramki w połączeniu z klauzulą `ORDER BY` generuje sumę kroczącą widoczną na rysunku 9.12.

■ **Wskazówka** Istnieją też inne klauzule ramki. Zapisana na listingu 9.12 klauzula `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` jest wartością domyślną przyjmowaną w sytuacji, gdy klauzula ramki nie zostanie jawnie określona. Pamiętaj, że często piszący zapytania są zdumieni niespodziewanymi wynikami, ponieważ nie wiedzą, iż używana jest domyślna klauzula ramki.

Przeanalizujmy przykład, by zobaczyć, jak domyślna klauzula ramki może wpłynąć na wyniki zapytania. Powiedzmy, że zechcesz obliczyć i zwrócić sumę sprzedaży w podziale według `PurchaseOrder` dla każdego wiersza. W zależności od sposobu zdefiniowania ramki, możesz uzyskać bardzo różne wyniki, ponieważ suma może oznaczać całkowitą sumę lub sumę kroczącą. Zmodyfikujmy zapytanie z listingu 9.12, określmy klauzulę ramki `RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` obok domyślnej klauzuli ramki i sprawdźmy wyniki. Zmodyfikowane zapytanie pokazane jest na listingu 9.13, a wyniki jego działania widać na rysunku 9.13.

Listing 9.13. *Sumowanie z różnymi ramkami*

```
SELECT
  PurchaseOrderID,
  ProductID,
  OrderQty,
  UnitPrice,
  LineTotal,
```

```
  SUM(LineTotal)
    OVER (PARTITION BY PurchaseOrderID ORDER BY ProductId)
  AS TotalSalesDefaultFraming,
  SUM(LineTotal)
    OVER (PARTITION BY PurchaseOrderID ORDER BY ProductId
    RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
  AS TotalSalesDefinedFraming
FROM Purchasing.PurchaseOrderDetail
ORDER BY PurchaseOrderID;
```

Results | Messages | Execution plan

	PurchaseOrderID	ProductID	OrderQty	UnitPrice	LineTotal	TotalSalesDefaultFraming	TotalSalesDefinedFraming
1	1	1	4	50,26	201,04	201,04	201,04
2	2	359	3	45,12	135,36	135,36	272,1015
3	2	360	3	45,5805	136,7415	272,1015	272,1015
4	3	530	550	16,086	8847,30	8847,30	8847,30
5	4	4	3	57,0255	171,0765	171,0765	171,0765
6	5	512	550	37,086	20397,30	20397,30	20397,30
7	6	513	550	26,5965	14628,075	14628,075	14628,075
8	7	317	550	27,0585	14882,175	14882,175	58685,55
9	7	318	550	33,579	18468,45	33350,625	58685,55
10	7	319	550	46,0635	25334,925	58685,55	58685,55
11	8	403	3	47,4705	142,4115	142,4115	693,378
12	8	404	3	45,3705	136,1115	278,523	693,378
13	8	405	3	49,644	148,932	427,455	693,378
14	8	406	3	45,3705	136,1115	563,5665	693,378
15	8	407	3	43,2705	129,8115	693,378	693,378
16	9	422	3	47,523	142,569	142,569	694,1655
17	9	423	3	45,423	136,269	278,838	694,1655
18	9	424	3	49,6965	149,0895	427,9275	694,1655

Rysunek 9.13. *Część wyników zapytania z różnymi ramkami*

Na rysunku 9.13 możesz zobaczyć, że suma sprzedaży w ostatnich dwóch kolumnach znacząco się różni. W kolumnie 6., `TotalSalesDefaultFraming`, znajduje się suma krocząca wyników sprzedaży — ponieważ nie została określona ramka dla tej kolumny, użyta została domyślna ramka `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, co oznacza, że funkcja agregująca jest obliczana tylko do bieżącego wiersza. Jednak w przypadku kolumny 7., `TotalSalesDefinedFraming`, określona jest klauzula ramki `RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`, co oznacza, że ramka jest rozciągana na wszystkie wiersze partycji i dlatego suma jest obliczana ze wszystkich pozycji o tej samej wartości `PurchaseOrderID`. Celem jest obliczenie i zwrócenie sumy sprzedaży dla zamówienia w każdym wierszu, ale brak doprecyzowania rodzaju ramki powoduje zwrócenie sumy kroczącej. Ten przykład pokazuje, jak ważne jest określenie odpowiedniej ramki, aby uzyskać pożądany wynik.

Przyjrzymy się teraz innemu przykładowi. Na listingu 9.14 znajduje się zmodyfikowany kod z listingu 9.13. Dalej używamy w nim klauzuli `OVER` na tabeli `Purchasing.PurchaseOrderDetail` z bazy danych *AdventureWorks*, ale tym razem chcemy pobrać dwudniową średnią wartości zamówionych produktów. Wyniki są posortowane według `DueDate`. Zauważ inną klauzulę ramki w tym zapytaniu; jest to `ROWS BETWEEN 1 PRECEDING AND CURRENT ROW`. Wiersze są sortowane według dat. Dla każdego wiersza dla dwudniowej średniej pobierany jest bieżący wiersz oraz wiersz opisujący poprzedni dzień. Część wyników tego zapytania można zobaczyć na rysunku 9.14.

Listing 9.14. *Użycie klauzuli OVER do zdefiniowania ramki zwracającej dwudniową średnią kroczącą*

```
SELECT
  PurchaseOrderID,
  ProductID,
  DueDate,
  LineTotal,
  Avg(LineTotal)
    OVER (ORDER BY Duedate
    ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS [2DayAvg]
FROM Purchasing.PurchaseOrderDetail
ORDER BY Duedate;
```

Results | Messages

	PurchaseOrderID	ProductID	DueDate	LineTotal	2DayAvg
1	1	1	2011-04-30 00:00:00.000	201,04	201,04
2	2	359	2011-04-30 00:00:00.000	135,36	168,20
3	2	360	2011-04-30 00:00:00.000	136,7415	136,0507
4	3	530	2011-04-30 00:00:00.000	8847,30	4492,0207
5	4	4	2011-04-30 00:00:00.000	171,0765	4509,1882
6	5	512	2011-05-14 00:00:00.000	20397,30	10284,1882
7	6	513	2011-05-14 00:00:00.000	14628,075	17512,6875
8	7	317	2011-05-14 00:00:00.000	14882,175	14755,125
9	7	318	2011-05-14 00:00:00.000	18468,45	16675,3125
10	7	319	2011-05-14 00:00:00.000	25334,925	21901,6875
11	8	403	2011-05-14 00:00:00.000	142,4115	12738,6682
12	8	404	2011-05-14 00:00:00.000	136,1115	139,2615
13	8	405	2011-05-14 00:00:00.000	148,932	142,5217
14	8	406	2011-05-14 00:00:00.000	136,1115	142,5217
15	8	407	2011-05-14 00:00:00.000	129,8115	132,9615
16	9	422	2011-12-28 00:00:00.000	142,569	136,1902
17	9	423	2011-12-28 00:00:00.000	136,269	139,419
18	9	424	2011-12-28 00:00:00.000	149,0895	142,6792

Rysunek 9.14. *Część wyników zapytania zwracającego dwudniową średnią kroczącą*

Przyjrzyjmy się jeszcze jednemu scenariuszowi, w którym obliczamy sumę kroczącą sprzedaży z grupowaniem po `ProductId`, by dostarczyć zarządowi informacje o tym, które produkty szybko się sprzedają. Na listingu 9.15 znajduje się zmodyfikowane zapytanie z listingu 9.14 ze zdefiniowanymi dodatkowymi oknami partycjonującymi wyniki według `ProductID`. Możesz zobaczyć, jak zwiększa się rozmiar ramki podczas wykonywania obliczeń w ramce. Po zmianie `ProductID` ramka jest tworzona od nowa i obliczenia również zaczynane są od nowa. Na rysunku 9.15 pokazana jest część wyników.

Listing 9.15. *Definiowanie ramek wewnątrz klauzuli OVER do obliczania sumy kroczącej*

```
SELECT
  PurchaseOrderID,
  ProductID,
  OrderQty,
  UnitPrice,
  LineTotal,
  SUM(LineTotal)
    OVER (PARTITION BY ProductId ORDER BY DueDate
    RANGE BETWEEN UNBOUNDED PRECEDING AND
    CURRENT ROW) AS CumulativeTotal,
  ROW_NUMBER()
    OVER (PARTITION BY ProductId ORDER BY DueDate ) AS No
FROM Purchasing.PurchaseOrderDetail
ORDER BY ProductId, DueDate;
```

Results | Messages | Execution plan

	PurchaseOrderID	ProductID	OrderQty	UnitPrice	LineTotal	CumulativeTotal	No
44	3378	1	3	50,2635	150,7905	6685,0315	44
45	3457	1	3	50,2635	150,7905	6835,822	45
46	3536	1	3	50,2635	150,7905	6986,6125	46
47	3615	1	3	50,2635	150,7905	7137,403	47
48	3694	1	3	50,2635	150,7905	7288,1935	48
49	3773	1	3	50,2635	150,7905	7438,984	49
50	3852	1	3	50,2635	150,7905	7589,7745	50
51	3931	1	3	50,2635	150,7905	7740,565	51
52	79	2	3	41,916	125,748	125,748	1
53	158	2	3	41,916	125,748	251,496	2
54	237	2	3	41,916	125,748	377,244	3
55	316	2	3	41,916	125,748	502,992	4
56	395	2	3	41,916	125,748	628,74	5
57	425	2	3	41,916	125,748	754,488	6
58	504	2	3	41,916	125,748	880,236	7
59	587	2	3	41,916	125,748	1005,984	8
60	674	2	3	41,916	125,748	1131,732	9
61	757	2	3	41,916	125,748	1257,48	10

Rysunek 9.15. *Część wyników zawierających sumę kroczącą liczoną dla kolejnych ProductID*

Możesz też zobaczyć w zapytaniu z listingu 9.15, że nie jesteś ograniczony do korzystania z jednej funkcji agregującej w wyrażeniu `SELECT`. Możesz użyć wielu funkcji agregujących w tym samym zapytaniu.

Ramki można definiować, używając wierszy (`ROWS`) lub zakresów (`RANGE`) z dolnym i górnym ograniczeniem. Jeśli zdefiniujesz tylko dolne ograniczenie, górnym ograniczeniem będzie bieżący wiersz. Jeśli zdefiniujesz ramkę, korzystając z `ROWS`, możesz określić granicę za pomocą liczby lub wyrażenia skalarnego zwracającego liczbę całkowitą. Jeśli nie zdefiniujesz granicy ramki, przyjmowana jest wartość domyślna `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.

Przykłady funkcji analitycznych

W SQL Server 2012 wprowadzono kilka pomocnych funkcji analitycznych. Niektóre bardziej użyteczne z nich zostały opisane w kolejnych punktach. Niektóre są bardziej statystyczne, inne są przydatne przy tworzeniu raportów, w których musisz odwoływać się do wartości z wielu zwracanych wierszy.

CUME_DIST i PERCENT_RANK

`CUME_DIST` i `PERCENT_RANK` to dwie funkcje analityczne, które zostały wprowadzone w SQL Server 2012. Przyjmijmy, że chcesz ustalić, jakie wyniki mają najlepsi, średni i najsłabsi sprzedawcy *AdventureWorks*, i porównać je. Szczególnie zainteresowany jesteś danymi niejakiej Jillian Carson, na której nazwisko natknąłeś się przy wstępnym przeglądaniu danych. W języku biznesowym może być to wyrażone pytaniem: „Jakie są wyniki Jillian Carson w porównaniu z innymi sprzedawcami?”. `CUME_DIST` pozwala w łatwy sposób uzyskać taką informację, co widać na listingu 9.16. Wyniki zapytania pokazane zostały na rysunku 9.16.

Listing 9.16. *Użycie funkcji CUME_DIST*

```
SELECT
  round(SUM(TotalDue),1) AS Sales,
  LastName,
  FirstName,
  SalesPersonId,
  CUME_DIST() OVER (ORDER BY round(SUM(TotalDue),1)) as CUME_DIST
FROM
  Sales.SalesOrderHeader soh
  JOIN Sales.vSalesPerson sp
    ON soh.SalesPersonID = sp.BusinessEntityID
GROUP BY SalesPersonID,LastName,FirstName;
```

Results | Messages | Execution plan

	Sales	LastName	FirstName	SalesPersonId	CUME_DIST
1	195528,80	Abbas	Syed	285	0,0588235294117647
2	826417,50	Alberts	Amy	287	0,117647058823529
3	1235934,40	Jiang	Stephen	274	0,176470588235294
4	1606441,40	Tsoflias	Lynn	286	0,235294117647059
5	2062393,10	Valdez	Rachel	288	0,294117647058824
6	2608116,40	Mensa-Annan	Tete	284	0,352941176470588
7	3748246,10	Ansman-Wolfe	Pamela	280	0,411764705882353
8	4069422,20	Vargas	Garrett	278	0,470588235294118
9	4207894,60	Campbell	David	283	0,529411764705882
10	5087977,20	Varkey Chudukatil	Ranjit	290	0,588235294117647
11	6683536,70	Saraiva	José	282	0,647058823529412
12	7259567,90	Ito	Shu	281	0,705882352941177
13	8086073,70	Reiter	Tsvi	279	0,764705882352941
14	9585124,90	Pak	Jae	289	0,823529411764706
15	10475367,10	Blythe	Michael	275	0,882352941176471
16	11342385,90	Carson	Jillian	277	0,941176470588235
17	11695019,10	Mitchell	Linda	276	1

Rysunek 9.16. *Wyniki obliczeń CUME_DIST*

Zapytanie z listingu 9.16 zaokrągla wartość `TotalDue` i zwraca wartość `Sales` w postaci bardziej czytelnej. Ponieważ `CUME_DIST` zwraca pozycję wiersza, wartości w kolumnie są zwracane w postaci ułamka dziesiętnego. Procentowe wyniki można uzyskać, mnożąc tą wartość przez 100. Wyniki z rysunku 9.16 pokazują, że 94,11% wszystkich sprzedawców ma sprzedaż gorszą od sprzedaży Jillian Carson lub jej równą, o czym mówi wartość dystrybuanty 0,9411.

Jeśli odrobinę zmodyfikujesz pytanie do postaci: „W jakim percentylu znajduje się sprzedaż Jillian Carson?", odpowiedź możesz uzyskać za pomocą `PERCENT_RANK`. Na listingu 9.17 znajduje się zmodyfikowana wersja zapytania z listingu 9.16, zawierającą wywołanie `PERCENT_RANK`. Część wyników można zobaczyć na rysunku 9.17.

Listing 9.17. *Użycie funkcji PERCENT_RANK*

```
SELECT
  round(SUM(TotalDue),1) AS Sales,
  LastName,
  FirstName,
  SalesPersonId,
  CUME_DIST() OVER (ORDER BY round(SUM(TotalDue),1)) as CUME_DIST,
  PERCENT_RANK() OVER (ORDER BY round(SUM(TotalDue),1)) as PERCENT_RANK
FROM Sales.SalesOrderHeader soh
  JOIN Sales.vSalesPerson sp
    ON soh.SalesPersonID = sp.BusinessEntityID
GROUP BY SalesPersonID,LastName,FirstName;
```

Results | Messages | Execution plan

	Sales	LastName	FirstName	SalesPersonId	CUME_DIST	PERCENT_RANK
1	195528,80	Abbas	Syed	285	0,0588235294117647	0
2	826417,50	Alberts	Amy	287	0,117647058823529	0,0625
3	1235934,40	Jiang	Stephen	274	0,176470588235294	0,125
4	1606441,40	Tsoflias	Lynn	286	0,235294117647059	0,1875
5	2062393,10	Valdez	Rachel	288	0,294117647058824	0,25
6	2608116,40	Mensa-Annan	Tete	284	0,352941176470588	0,3125
7	3748246,10	Ansman-Wolfe	Pamela	280	0,411764705882353	0,375
8	4069422,20	Vargas	Garrett	278	0,470588235294118	0,4375
9	4207894,60	Campbell	David	283	0,529411764705882	0,5
10	5087977,20	Varkey Chudukatil	Ranjit	290	0,588235294117647	0,5625
11	6683536,70	Saraiva	José	282	0,647058823529412	0,625
12	7259567,90	Ito	Shu	281	0,705882352941177	0,6875
13	8086073,70	Reiter	Tsvi	279	0,764705882352941	0,75
14	9585124,90	Pak	Jae	289	0,823529411764706	0,8125
15	10475367,10	Blythe	Michael	275	0,882352941176471	0,875
16	11342385,90	Carson	Jillian	277	0,941176470588235	0,9375
17	11695019,10	Mitchell	Linda	276	1	1

Rysunek 9.17. *Wyniki obliczeń CUME_DIST i PERCENT_RANK dla sprzedawców*

Funkcja `PERCENT_RANK` zwraca pozycję wśród wszystkich sprzedawców w *AdventureWorks*. Jak możesz zobaczyć, pojawia się tu 17 unikalnych wartości: pierwszą jest 0, a ostatnią 1. Inne wiersze mają wartości mniejsze od 1. W tym przykładzie Jillian Carson ma percentyl 93,75% wśród sprzedawców *AdventureWorks*, co opisuje wartość `PERCENT_RANK` równa `0,9375`.

■ **Uwaga** Możesz użyć klauzuli `PARTITION BY` na funkcjach `CUME_DIST` i `PERCENT_RANK`, aby zdefiniować okno, na którym będziesz przeprowadzał te obliczenia.

PERCENTILE_CONT i PERCENTILE_DISC

`PERCENTILE_CONT` i `PERCENTILE_DISC` to nowe funkcje, które są odwrotnościami funkcji `CUME_DIST` i `PERCENT_RANK`. Przyjmijmy, że zechcesz ustalić 40 percentyl sprzedaży dla wszystkich kont *AdventureWorks*. Można to wyrazić w języku biznesowym pytaniem: „Jaki jest 40 percentyl całej sprzedaży dla wszystkich kont?”. `PERCENTILE_CONT` i `PERCENTILE_DISC` wymagają klauzuli `WITHIN GROUP` określającej kolejność i kolumny do obliczeń. `PERCENTILE_CONT` interpoluje wszystkie wartości okna i na podstawie tego oblicza wynik. `PERCENTILE_DISC` zwraca istniejącą wartość. Zarówno `PERCENTILE_CONT`, jak i `PERCENTILE_DISC` wymagają podania percentyla jako argumentu o wartości z zakresu od `0,0` do `1,0`. Przykład z listingu 9.18 oblicza sumę sprzedaży dla 40 percentyla w podziale na konta. W przykładzie wykorzystane są `PERCENTILE_CONT` i `PERCENTILE_DISC` z wartością `0,4` jako percentyl do obliczenia oznaczającą 40 percentyl. Wyniki zapytania pokazane są na rysunku 9.18.

Listing 9.18. *Użycie PERCENTILE_CONT i PERCENTILE_DISC*

```
SELECT
  round(SUM(TotalDue),1) AS Sales,
  LastName,
  FirstName,
  SalesPersonId,
  AccountNumber,
  PERCENTILE_CONT(0.4) WITHIN GROUP (ORDER BY round(SUM(TotalDue),1))
    OVER(PARTITION BY AccountNumber ) AS PERCENTILE_CONT,
  PERCENTILE_DISC(0.4) WITHIN GROUP(ORDER BY round(SUM(TotalDue),1))
```

```
    OVER(PARTITION BY AccountNumber ) AS PERCENTILE_DISC
FROM
  Sales.SalesOrderHeader soh
  JOIN Sales.vSalesPerson sp
    ON soh.SalesPersonID = sp.BusinessEntityID
GROUP BY AccountNumber,SalesPersonID,LastName,FirstName
```

Results | Messages | Execution plan

	Sales	LastName	FirstName	SalesPersonId	AccountNumber	PERCENTILE_CONT	PERCENTILE_DISC
1	95924,00	Ansman-Wolfe	Pamela	280	10-4020-000001	95924	95924,00
2	28310,00	Campbell	David	283	10-4020-000002	28310	28310,00
3	176830,40	Carson	Jillian	277	10-4020-000003	198391,28	176830,40
4	230732,60	Blythe	Michael	275	10-4020-000003	198391,28	176830,40
5	222309,60	Carson	Jillian	277	10-4020-000004	308720,28	222309,60
6	438336,30	Blythe	Michael	275	10-4020-000004	308720,28	222309,60
7	97031,20	Ito	Shu	281	10-4020-000005	97031,2	97031,20
8	3023,30	Mitchell	Linda	276	10-4020-000006	3023,3	3023,30
9	3302,50	Jiang	Stephen	274	10-4020-000007	4166,78	3302,50
10	5463,20	Mitchell	Linda	276	10-4020-000007	4166,78	3302,50
11	25064,60	Reiter	Tsvi	279	10-4020-000008	25064,6	25064,60
12	14250,80	Reiter	Tsvi	279	10-4020-000009	14250,8	14250,80
13	500181,70	Pak	Jae	289	10-4020-000010	500181,7	500181,70
14	54905,20	Saraiva	José	282	10-4020-000011	54905,2	54905,20
15	281414,80	Vargas	Garrett	278	10-4020-000012	281414,8	281414,80
16	8393,50	Valdez	Rachel	288	10-4020-000014	8393,5	8393,50
17	163865,50	Tsoflias	Lynn	286	10-4020-000015	163865,5	163865,50
18	56147,10	Alberts	Amy	287	10-4020-000016	224876,46	56147,10

Rysunek 9.18. *Wyniki z funkcji PERCENTILE_CONT i PERCENTILE_DISC*

Na rysunku 9.18 możesz zobaczyć, że wartości `PERCENTILE_CONT` i `PERCENTILE_DISC` różnią się w zależności od numeru konta. Dla konta o numerze `10-4020-000003`, niezależnie od sprzedawcy, `PERCENTILE_CONT` ma wartość `198391,28`, która jest wartością interpolowaną i może nie występować w zbiorze danych. `PERCENTILE_DISC` ma wartość `176830,40`, co jest równe wartości istniejącej w zbiorze danych. Dla konta `10-4020-000004` `PERCENTILE_CONT` ma wartość `308720,28`, a `PERCENTILE_DISC` ma wartość `222309,60`.

LAG i LEAD

`LAG` i `LEAD` to nowe funkcje przesunięcia, które umożliwiają wykonywanie obliczeń dla określonego wiersza znajdującego się przed bieżącym wierszem lub po nim. Funkcje te umożliwiają dostęp do więcej niż jednego wiersza bez konieczności tworzenia złączenia typu *self-join*. Funkcja `LAG` udostępnia wiersze poprzedzające bieżący wiersz, a `LEAD` daje dostęp do wierszy znajdujących się za bieżącym wierszem.

`LAG` pomaga odpowiedzieć na pytania biznesowe: „Jakie są bieżące i historyczne koszty produkcji dla wszystkich produktów obecnie produkowanych?”. Na listingu 9.19. pokazany jest przykład zapytania liczącego bieżące koszty produkcji i wcześniejsze koszty produkcji dla wszystkich aktywnych produktów za pomocą funkcji `LAG`. Część wyników znajduje się na rysunku 9.19.

Listing 9.19. *Użycie funkcji LAG*

```
WITH ProductCostHistory AS
(SELECT
  ProductID,
  LAG(StandardCost)
    OVER (PARTITION BY ProductID ORDER BY ProductID) AS PreviousProductCost,
```

```
  StandardCost AS CurrentProductCost,
  Startdate,Enddate
FROM Production.ProductCostHistory
)
SELECT
  ProductID,
  PreviousProductCost,
  CurrentProductCost,
  StartDate,
  EndDate
FROM ProductCostHistory
WHERE Enddate IS NULL
```

Results | Messages | Execution plan

	ProductID	PreviousProductCost	CurrentProductCost	StartDate	EndDate
1	707	13,8782	13,0863	2013-05-30 00:00:00.000	NULL
2	708	13,8782	13,0863	2013-05-30 00:00:00.000	NULL
3	711	13,8782	13,0863	2013-05-30 00:00:00.000	NULL
4	712	5,2297	6,9223	2013-05-30 00:00:00.000	NULL
5	713	29,0807	38,4923	2013-05-30 00:00:00.000	NULL
6	714	29,0807	38,4923	2013-05-30 00:00:00.000	NULL
7	715	29,0807	38,4923	2013-05-30 00:00:00.000	NULL
8	716	29,0807	38,4923	2013-05-30 00:00:00.000	NULL
9	717	722,2568	868,6342	2013-05-30 00:00:00.000	NULL
10	718	722,2568	868,6342	2013-05-30 00:00:00.000	NULL
11	719	722,2568	868,6342	2013-05-30 00:00:00.000	NULL
12	720	722,2568	868,6342	2013-05-30 00:00:00.000	NULL
13	721	722,2568	868,6342	2013-05-30 00:00:00.000	NULL
14	722	170,1428	204,6251	2013-05-30 00:00:00.000	NULL
15	723	170,1428	204,6251	2013-05-30 00:00:00.000	NULL
16	724	170,1428	204,6251	2013-05-30 00:00:00.000	NULL
17	736	170,1428	204,6251	2013-05-30 00:00:00.000	NULL
18	737	170,1428	204,6251	2013-05-30 00:00:00.000	NULL

Rysunek 9.19. *Wyniki pozwalające na porównanie historycznych kosztów produkcji za pomocą funkcji LAG*

W tym przykładzie w listingu 9.19 skorzystano z funkcji `LAG` w CTE do obliczenia różnicy pomiędzy bieżącymi kosztami produkcji i wcześniejszymi poprzez partycjonowanie zbioru danych względem `ProductID`.

```
SELECT
  ProductID,
  LAG(StandardCost)
    OVER (PARTITION BY ProductID ORDER BY ProductID) AS PreviousProductCost,
  StandardCost AS CurrentProductCost,
  Startdate,Enddate
FROM Production.ProductCostHistory
```

Wyrażenie `SELECT` związane z CTE zwraca wiersze zawierające najnowsze koszty produkcji ze zbioru danych przy użyciu wartości `NULL` w kolumnie `EndDate` za pomocą poniższego wywołania.

```
SELECT
  ProductID,
  PreviousProductCost,
  CurrentProductCost,
  StartDate,
  EndDate
FROM ProductCostHistory
WHERE Enddate IS NULL
```

Funkcja LEAD, która jest przeciwieństwem LAG, pomaga odpowiadać na pytania biznesowe, na przykład takie: „Jak miesięczne wyniki sprzedaży wypadają w porównaniu ze sprzedażą w kolejnym miesiącu dla wszystkich sprzedawców *AdventureWorks* w roku 2012?". Listing 9.20 zawiera przykładowe zapytanie, które zawiera sprzedaż z kolejnego miesiąca pobraną za pomocą funkcji LEAD dla porównania ze sprzedażą w bieżącym miesiącu dla danych z roku 2012. Częściowe wyniki są pokazane na rysunku 9.20.

Listing 9.20. *Użycie funkcji LEAD*

```
SELECT
  LastName,
  SalesPersonID,
  Sum(SubTotal) CurrentMonthSales,
  DateNAME(Month,OrderDate) Month,
  DateName(Year,OrderDate) Year,
  LEAD(Sum(SubTotal),1)
    OVER (ORDER BY SalesPersonID, OrderDate) TotalSalesNextMonth
FROM Sales.SalesOrderHeader soh
  JOIN Sales.vSalesPerson sp
    ON soh.SalesPersonID = sp.BusinessEntityID
WHERE DateName(Year,OrderDate) = 2012
GROUP BY FirstName, LastName, SalesPersonID,OrderDate
ORDER BY SalesPersonID,OrderDate;
```

Results | Messages | Execution plan

	LastName	SalesPersonID	CurrentMonthSales	Month	Year	TotalSalesNextMonth
132	Alberts	287	12648,1326	September	2012	29649,0509
133	Alberts	287	29649,0509	December	2012	631704,2589
134	Pak	289	631704,2589	May	2012	331640,849
135	Pak	289	331640,849	June	2012	474009,9807
136	Pak	289	474009,9807	July	2012	393554,9573
137	Pak	289	393554,9573	August	2012	283398,3465
138	Pak	289	283398,3465	September	2012	408527,5444
139	Pak	289	408527,5444	October	2012	257637,1408
140	Pak	289	257637,1408	November	2012	233804,9696
141	Pak	289	233804,9696	December	2012	97496,2916
142	Varkey ...	290	97496,2916	May	2012	262749,5275
143	Varkey ...	290	262749,5275	June	2012	163687,8125
144	Varkey ...	290	163687,8125	July	2012	55789,7317
145	Varkey ...	290	55789,7317	August	2012	180614,7458
146	Varkey ...	290	180614,7458	September	2012	87995,2931
147	Varkey ...	290	87995,2931	October	2012	34516,2132
148	Varkey ...	290	34516,2132	November	2012	113442,2926
149	Varkey ...	290	113442,2926	December	2012	NULL

Rysunek 9.20. *Wyniki porównania sprzedaży handlowców w 2012 roku za pomocą funkcji LEAD*

Na rysunku 9.20 możesz zobaczyć, że ostatni wiersz zwraca NULL dla sprzedaży w kolejnym miesiącu, ponieważ nie ma wartości LEAD dla ostatniego wiersza.

FIRST_VALUE i LAST_VALUE

FIRST_VALUE i LAST_VALUE to funkcje przesunięcia zwracające pierwszą i ostatnią wartość w oknie zdefiniowanym za pomocą klauzuli OVER. Funkcja FIRST_VALUE zwraca pierwszą wartość z okna, a LAST_VALUE zwraca ostatnią wartość z okna.

Funkcje te pomagają odpowiedzieć na przykład na pytania: „Jaka jest kwota pierwszej i ostatniej transakcji w miesiącu dla danego sprzedawcy?". Na listingu 9.21 znajduje się przykładowe zapytanie, a na rysunku 9.21 pokazano część wyników zwracanych przez to zapytanie.

Listing 9.21. *Użycie FIRST_VALUE i LAST_VALUE*

```
SELECT DISTINCT
  LastName,
  SalesPersonID,
  datename(year,OrderDate) OrderYear,
  datename(month, OrderDate) OrderMonth,
  FIRST_VALUE(SubTotal)
    OVER (PARTITION BY SalesPersonID, OrderDate ORDER BY SalesPersonID )
  FirstSalesAmount,
  LAST_VALUE(SubTotal)
    OVER (PARTITION BY SalesPersonID, OrderDate ORDER BY SalesPersonID)
  LastSalesAmount,
  OrderDate
FROM Sales.SalesOrderHeader soh
  JOIN Sales.vSalesPerson sp
    ON soh.SalesPersonID = sp.BusinessEntityID
ORDER BY OrderDate;
```

Results | Messages | Execution plan

	LastName	SalesPersonID	OrderYear	OrderMonth	FirstSalesAmount	LastSalesAmount	OrderDate
1	Blythe	275	2011	May	6122,082	20541,4072	2011-05-31 00:00:00.000
2	Mitchell	276	2011	May	419,4589	5056,4896	2011-05-31 00:00:00.000
3	Carson	277	2011	May	6107,082	874,794	2011-05-31 00:00:00.000
4	Vargas	278	2011	May	7793,1108	1316,0575	2011-05-31 00:00:00.000
5	Reiter	279	2011	May	20565,6206	419,4589	2011-05-31 00:00:00.000
6	Ansman-Wolfe	280	2011	May	24432,6088	24432,6088	2011-05-31 00:00:00.000
7	Ito	281	2011	May	9799,9243	38510,8973	2011-05-31 00:00:00.000
8	Saraiva	282	2011	May	32726,4786	2624,382	2011-05-31 00:00:00.000
9	Campbell	283	2011	May	14352,7713	3463,2998	2011-05-31 00:00:00.000
10	Jiang	274	2011	July	20544,7015	20544,7015	2011-07-01 00:00:00.000
11	Blythe	275	2011	July	32940,1556	4049,988	2011-07-01 00:00:00.000
12	Mitchell	276	2011	July	3911,5991	68,40	2011-07-01 00:00:00.000
13	Carson	277	2011	July	1718,8983	32971,323	2011-07-01 00:00:00.000
14	Vargas	278	2011	July	919,5201	50948,9161	2011-07-01 00:00:00.000
15	Reiter	279	2011	July	8580,0739	52803,4477	2011-07-01 00:00:00.000
16	Ansman-Wolfe	280	2011	July	10993,3942	32459,9278	2011-07-01 00:00:00.000

Rysunek 9.21. *Wyniki pokazujące wartość pierwszej i ostatniej transakcji*

Przykład ten zwraca wielkość pierwszej i ostatniej transakcji dla każdej osoby w podziale na miesiące i lata. Na rysunku 9.21 możesz zobaczyć, że w niektórych przypadkach wartości `FirstSalesAmount` i `LastSalesAmount` są takie same, co oznacza, że w danym miesiącu odbyła się tylko jedna transakcja. W miesiącach z więcej niż jedną transakcją wymienione są wartości pierwszej i ostatniej transakcji.

Podsumowanie

Wyrażenia nazwane to użyteczne narzędzie SQL Server istniejące w dwóch odmianach, rekurencyjnej i nierekurencyjnej. Nierekurencyjne CTE pozwalają na pisanie bardziej przejrzystego kodu T-SQL łatwiejszego w tworzeniu, debugowaniu i zarządzaniu od złożonych zapytań korzystających z tabel pośrednich. Rekurencyjne CTE upraszczają pisanie zapytań do pobierania danych hierarchicznych i umożliwiają proste generowanie przydatnych w wielu zastosowaniach zbiorów danych, zawierających kolejne liczby całkowite.

SQL Server obsługuje funkcje okna, a klauzula OVER upraszcza korzystanie z funkcji agregujących z uwzględnieniem okien i porządkowania. SQL Server obsługuje kilka poniższych funkcji okna.

- ROW_NUMBER numeruje zwracane wiersze, rozpoczynając od 1.
- RANK i DENSE_RANK tworzą ranking zwracanych wyników, dopisując przy numerowaniu taką samą liczbę do wierszy zawierających pola o takiej samej wartości.
- NTILE dzieli zbiór wyników na określoną przez użytkownika liczbę grup.
- CUME_DIST, PERCENTILE_CONT, PERCENT_RANK i PERCENTILE_DISC to funkcje analityczne T-SQL, umożliwiające między innymi obliczanie dystrybuanty.
- LAG i LEAD umożliwiają odwoływanie się do sąsiednich wierszy oddalonych o określoną liczbę wierszy.
- FIRST_VALUE i LAST_VALUE zwracają pierwszy i ostatni wiersz okna zdefiniowanego przez klauzulę partycjonującą dane.

Można też wykorzystać klauzulę OVER do utworzenia okna przy korzystaniu z wbudowanych funkcji agregujących oraz funkcji agregujących SQL CLR utworzonych przez użytkownika.

Zarówno wyrażenia nazwane, jak i funkcje okna są bardzo użyteczne i rozszerzają składnię T-SQL, pozwalając na pisanie kodu potężniejszego niż kiedykolwiek za pomocą dużo prostszych konstrukcji.

Ćwiczenia

1. Czy jeśli CTE nie jest pierwszym wyrażeniem w ciągu, poprzedzające je wyrażenie musi być zakończone średnikiem?
2. Co jest wymagane w rekurencyjnym CTE:
   a. słowo kluczowe WITH,
   b. zapytanie kotwiczące,
   c. słowo kluczowe EXPRESSION,
   d. zapytanie rekurencyjne.
3. Opcja MAXRECURSION przyjmuje wartości pomiędzy 0 i ..... .
4. Które z poniższych funkcji okna są obsługiwane przez SQL Server:
   a. ROW_NUMBER,
   b. RANK,
   c. DENSE_RANK,
   d. NTILE,
   e. wszystkie powyższe.
5. Czy można użyć ORDER BY w klauzuli OVER wykorzystanej z funkcjami agregującymi?
6. Czy jeśli PARTITION BY i ORDER BY są wykorzystane w klauzuli OVER, klauzula PARTITION BY musi pojawić się pierwsza?
7. Nazwy wszystkich kolumn zwracanych przez CTE muszą być ..... .
8. Domyślną klauzulą ramki jest ..... .
9. Czy jeśli nie jest określona klauzula ORDER BY dla funkcji niewymagającej stosowania klauzuli OVER, ramka okna obejmuje całą partycję?

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Z, Ź

T-SQL dla zaawansowanych. Przewodnik programisty

W wielu środowiskach IT serwer bazodanowy jest szczególnie ważnym elementem infrastruktury. Microsoft SQL Server to jedno z częściej spotykanych rozwiązań. Aby w pełni wykorzystać jego możliwości, trzeba dogłębnie poznać język Transact-SQL — potężne narzędzie, które wyewoluowało z języka zapytań SQL i wciąż jest rozwijane przez firmę Microsoft.

Niniejsza książka, przeznaczona dla programistów T-SQL, jest uaktualnionym wydaniem świetnego przewodnika po SQL Server. Omówiono w niej zaawansowane mechanizmy dostępne w T-SQL, w tym nowości, takie jak mechanizmy pamięciowe będące częścią SQL Server 2014. Istotne kwestie są przystępnie wyjaśnione za pomocą praktycznych przykładów i obszernych fragmentów kodu źródłowego. Autorzy położyli nacisk na demonstrację opcji T-SQL, ich możliwe zastosowania, a sam podręcznik zorganizowali w sposób umożliwiający szybkie wyszukanie potrzebnych informacji.

Najważniejsze zagadnienia omówione w książce:

- podstawy T-SQL i przegląd dobrych praktyk programowania w tym języku
- narzędzia dołączone do SQL Server 2014 (w tym SSMS, SQLCMD, SSDT oraz SQL Profiler)
- procedury składowane wykonywane po stronie serwera oraz tabele OLTP
- obsługa XML, XQuery oraz XPath w SQL Server 2014
- szyfrowanie baz danych, usługi ADO.NET oraz usługi IIS
- obsługa błędów, testowanie i zabezpieczenie kodu SQL

Miguel Cebollero — od ponad 16 lat zajmuje się SQL Server i innymi systemami bazodanowymi, włączając w to projektowanie, programowanie i administrację. Często wypowiada się na różnego rodzaju konferencjach związanych z bazami danych.

Jay Natarajan — od ponad 15 lat projektuje i implementuje rozwiązania oparte na SQL Server. Ma na koncie wiele skomplikowanych rozwiązań dla dużych klientów. Dołączyła do Microsoft Consulting Services w 2008 roku.

Michael Coles — ma za sobą 10-letnie doświadczenie w przy projektowaniu i administrowaniu bazami danych SQL Server. Jest autorem licznych artykułów na temat SQL Server, szczególnie o specjalnych zastosowaniach T-SQL.

Helion
44767 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

Apress

ISBN 978-83-283-2247-9
9 788328 322479
cena: 99,00 zł