Microsoft

Itzik Ben-Gan
Dejan Sarka
Adam Machanic
Kevin Farlee

Zapytania w języku T-SQL

w Microsoft SQL Server 2014 i SQL Server 2012

Przekład: Natalia Chounlamany
Marek Włodarz

APN Promise, Warszawa 2015

Zapytania w języku T-SQL w Microsoft SQL Server 2014 i SQL Server 2012

Authorized Polish translation of the English language edition entitled: T-SQL Querying, ISBN: 978-0-7356-8504-8, by Itzik Ben-Gan, Dejan Sarka, Adam Machanic, and Kevin Farlee, published by Pearson Education, Inc, publishing as Microsoft Press, A Division Of Microsoft Corporation.

Polish language edition published by APN PROMISE SA Copyright © 2015

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego: T-SQL Querying, ISBN: 978-0-7356-8504-8, by Itzik Ben-Gan, Dejan Sarka, Adam Machanic, and Kevin Farlee, opublikowanego przez Pearson Education, Inc, publikującego jako Microsoft Press, oddział Microsoft Corporation.

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

ISBN: 978-83-7541-158-4

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

Dla Lilach, za to, że nadaje sens wszystkiemu, co robię.
– Itzik Ben-Gan

Spis treści

Przedmowa

Od roku 1993 pracuję w zespole Microsoft SQL Server. To kawał czasu i wspaniałe było obserwowanie od środka, jak produkt ten się rozwija i dojrzewa, aby stać się tym, czym jest obecnie. Wspaniałe było też przyglądanie się, jak coraz więcej i więcej klientów wykorzystuje SQL Server do prowadzenia swoich przedsiębiorstw i biznesów. Ale przede wszystkim miałem ten zaszczyt, że mogłem wspierać najbardziej błyskotliwą i zaangażowaną społeczność techniczną, jaką kiedykolwiek widziałem.

Społeczność Microsoft SQL Server jest pełna prawdziwie zadziwiających, inteligentnych ludzi. Są oni dumni z tego, że mogą dzielić się swoją wiedzą z innymi, wszystko po to, by społeczność była jeszcze silniejsza. Każdy na świecie może wejść do Twittera i zadać dowolne pytanie na kanale #sqlhelp, a po kilku sekundach otrzyma odpowiedź jednego z najbystrzejszych światowych ekspertów. Jeśli szukamy prawdziwej wiedzy w dziedzinie wydajności, magazynowania danych, optymalizacji zapytań, projektowania wielkoskalowego, modelowania lub dowolnego innego zagadnienia związanego z bazami danych, eksperci należący do społeczności chętnie podzielą się swoim doświadczeniem. Warto poznać ich nie tylko ze względu na wiedzę, ale również niezwykłe, przyjazne osobowości. Członkowie społeczności SQL Server lubią, aby nazywać ich rodziną - rodziną SQL.

Każdy członek społeczności zna najważniejszych udziałowców dzięki ich wiedzy w określonych dziedzinach. Jeśli ktoś zapyta, kto jest najlepszym specjalistą w dziedzinie wydajności baz danych, każdy członek społeczności poda te same nazwiska - cztery lub pięć. Jeśli pytanie będzie dotyczyło magazynowania danych, również od każdego usłyszymy tę samą odpowiedź. W każdej dziedzinie znajdziemy kilku ekspertów, którzy są najlepszymi fachowcami w wybranym obszarze domeny bazodanowej. Jest tylko jeden wyjątek, który znam, i dotyczy on języka T-SQL. Oczywiście, istnieje wielu utalentowanych programistów T-SQL, ale jeśli zapytamy kogokolwiek, kto jest najlepszy, zawsze usłyszymy tylko jedno nazwisko: Itzik Ben-Gan.

Itzik poprosił mnie o napisanie tej przedmowy do jego nowej książki i czuję się zaszczycony, że mogę to zrobić. Jego wcześniejsze książki - *Inside Microsoft SQL Server: T-SQL Querying* (Microsoft Press, 2009*), *Inside Microsoft SQL Server: T-SQL Programming*

* Wydanie polskie: *Microsoft SQL Server 2008 od środka: Zapytania w języku T-SQL*, APN Promise, Warszawa 2009.

(Microsoft Press, 2009*) oraz *Microsoft SQL Server High-Performance T-SQL Using Window Functions* (Microsoft Press, 2012**) - można znaleźć na półkach każdego administratora czy projektanta baz danych, jakiego znam. Książki te to przeszło 2000 stron najwyższej jakości wiedzy technicznej na temat języka T-SQL i określają one standard wysokiej jakości treści w dziedzinie baz danych.

Teraz dołączyła do nich nowa książka, zatytułowana po prostu *Zapytania w języku T-SQL*. Nie tylko łączy ona zagadnienia omawiane w trzech wcześniejszych pozycjach, ale również, a może przede wszystkim, dodaje informacje dotyczące SQL Server 2012 i 2014, w tym funkcje okna, nowy estymator kardynalności, sekwencje, magazyn kolumnowy, In-Memory OLTP i wiele więcej. Itzik znalazł również nowych współautorów: są to Kevin Farlee, Adam Machanic i Dejan Sarka. Kevin jest członkiem zespołu projektowego Microsoft SQL Server i kimś, z kim pracowałem przez wiele lat. Adam to jedno z tych nielicznych nazwisk, o których wspominałem wcześniej - jeden z najlepszych ekspertów w dziedzinie wydajności baz danych na świecie. Dejan wreszcie jest szeroko znany przede wszystkim dzięki swojej wiedzy na temat BI i modelowania danych.

Mogę się spodziewać, że książka ta stanie się kolejną standardową pozycją na temat T-SQL dla wszystkich członków społeczności Microsoft SQL Server.

Mark Souza
General Manager, Cloud and Enterprise Engineering
Microsoft

---

* Wydanie Polskie: *Microsoft SQL Server 2008 od środka: Programowanie w języku T-SQL*, APN Promise, Warszawa 2010.

** Wydanie polskie: *Microsoft SQL Server 2012. Optymalizacja kwerend T-SQL przy użyciu funkcji okna*, APN Promise, Warszawa 2012

Wstęp

Książka ta, będąc aktualizacją zarówno pozycji *Microsoft SQL Server 2008 od środka: Zapytania w języku T-SQL* (APN Promise, 2009), jak i części książki *Microsoft SQL Server 2008 od środka: Programowanie w języku T-SQL* (APN Promise, 2010), zapewnia projektantom i administratorom baz danych szczegółowy przegląd wewnętrznej architektury T-SQL i wyczerpujące źródło informacji. Zawiera omówienie nowych funkcjonalności wprowadzonych w wydaniach SQL Server 2012 i 2014, ale w wielu przypadkach dotyczy obszarów, które nie są zależne od wersji oprogramowania i zapewne będą aktualne również w przyszłych wydaniach SQL Server. Zaprezentowane zostały rozwiązania najtrudniejszych zagadnień dotyczących zapytańopartych na zbiorach i problemów dostrajania zapytań, wsparte głęboką wiedzą i doświadczeniem zespołu autorskiego. Czytelnik będzie mógł pogłębić swoje zrozumienie architektury i wewnętrznych mechanizmów i opanowaćpraktyczne podejścia i zaawansowane techniki optymalizowania wydajności kodu. Książka przedstawia wiele unikatowych technik, które zostały opracowane, ulepszone i doszlifowane przez autorów w ciągu wielu lat pracy, zapewniając wysoko wydajne rozwiązania dla często spotykanych wyzwań. Większość przedstawianych rozwiązań i technik koncentruje się na wydajności i skuteczności tworzonego kodu. Autorzy podkreślają również potrzebę posiadania pełnego zrozumienia języka i jego podstaw matematycznych.

Kto powinien przeczytać tę książkę

Książka ta ma na celu pomóc doświadczonym praktykom T-SQL w osiągnięciu lepszego zrozumienia i wydajności. Adresatami tej publikacji są programiści T-SQL, administratorzy baz danych, specjaliści BI, analitycy danych i oraz wszyscy, którzy poważnie zajmują się językiem T-SQL. Główny cel to przygotować Czytelnika na rzeczywiste wymagania, w których potrzebne jest posługiwanie się językiem T-SQL. Treści nie skupiają się na przygotowaniu do jakiegoś egzaminu certyfikacyjnego, ale można zauważyć, że książka wyczerpuje wiele zagadnieńsprawdzanych na egzaminach 70-461 oraz 70-464. Tak więc, choć książki tej nie można traktować jako jedynej pomocy naukowej w przygotowaniu do tych egzaminów, bez wątpienia będzie to bardzo przydatna lektura uzupełniająca.

Założenia

Autorzy zakładają, że Czytelnik ma przynajmniej rok solidnego doświadczenia w pracy w SQL Server, pisaniu i dostrajaniu kodu T-SQL. Przyjmują, że czytelnik ma już wprawę w tworzeniu kodu T-SQL, zna podstawy dostrajania zapytań i jest gotów zmierzyć się z bardziej wymagającymi wyzwaniami. Książka może być też użyteczna dla osób, które mają podobne doświadczenia z inną platformą bazodanową i dialektem języka SQL, ale preferowana jest rzeczywista wiedza i doświadczenie dotyczące SQL Server oraz T-SQL.

Kto nie powinien czytać tej książki

Książka ta raczej nie nadaje się dla nowicjuszy w dziedzinie baz danych i języka SQL.

Struktura ksiażki

Książka rozpoczyna się dwoma rozdziałami, w których przedstawiane są podstawy logicznego i fizycznego przetwarzania zapytań, niezbędne do zrozumienia większości pozostałych rozdziałów.

Rozdział pierwszy przedstawia logiczne przetwarzania zapytań. Omówione zostały szczegółowo logiczne fazy procesu przetwarzania, unikatowe aspekty zapytań SQL oraz szczególny zestaw reguł i koncepcji, które trzeba opanować, aby móc programować w relacyjnym, zorientowanym na zbiory środowisku.

Rozdział drugi zawiera omówienie dostrajania zapytań i fizyczną warstwę mechanizmu bazodanowego. Przedstawione zostały wewnętrzne struktury danych, narzędzia do pomiaru wydajności zapytań, metody dostępowe, oszacowania liczebności, indeksy, szeregowanie zapytań przy użyciu zdarzeń rozszerzonych, technologia magazynu kolumnowego, stosowanie tabel tymczasowych i zmiennych tablicowych, porównanie podejścia opartego na zbiorach do rozwiązań wykorzystujących kursory, dostrajanie zapytań oraz równoległe wykonywanie zapytań. (Fragment o równoległym przetwarzaniu zapytań został napisany przez Adama Machanica).

Kolejnych pięć rozdziałów zajmuje się różnymi zagadnieniami związanymi z manipulowaniem danymi. Oprócz przedstawienia i objaśnienia poszczególnych funkcjonalności, autorzy skupiają się głównie na wydajności kodu i wykorzystania omawianych funkcjonalności do rozwiązywania często spotykanych zadań. W rozdziale 3 przedstawione są zapytania wielotabelowe wykorzystujące podzapytania, operator APPLY, złączenia oraz operatory relacyjne UNION, INTERSECT i EXCEPT. Rozdział 4 omawia problematykę analizy danych przy użyciu grupowania, przestawiania (*pivoting*) oraz funkcji okna. Rozdział 5 zawiera omówienie filtrów TOP oraz OFFSET-FETCH i rozwiązanie problemów typu pierwszych *N* z grupy. W rozdziale 6 omówione zostały takie zagadnienia modyfikacji danych, jak minimalnie rejestrowane operacje, wydajne

stosowanie obiektów sekwencji, scalanie danych oraz klauzula OUTPUT. Wreszcie w rozdziale 7 omówione są zagadnienia dotyczące danych typu data i czas, włącznie z obsługą interwałów (przedziałów) czasowych.

Rozdział 8 omawia wykorzystanie języka T-SQL w praktyce BI; jego autorem jest Dejan Sarka. Omówione zostały techniki przygotowywania danych do analizy i wykorzystanie T-SQL do realizacji zadań statystycznej analizy. Przedstawione zostały zagadnienia związane z częstościami występowania, statystyki opisowe dla zmiennych ciągłych, zależności liniowe, średnie ruchome oraz entropia zbioru danych.

Rozdział 9 zawiera omówienie konstruktów programistycznych wspieranych przez T-SQL. Należą do nich dynamiczny kod SQL, funkcje definiowane przez użytkownika, procedury składowane, wyzwalacze, programowanie SQLCLR (ten fragment autorstwa Adama Machanica), transakcje i współbieżność oraz obsługa błędów. Wcześniej zagadnienia te zostały przedstawione w książce *Inside Microsoft SQL Server: T-SQL Programming*.

Rozdział 10 przedstawia najważniejsze udoskonalenie dostępne w wydaniu SQL Server 2014 – silnik In-Memory OLTP. Autorem tego rozdziału jest Kevin Farlee z firmy Microsoft, który brał udział w projektowaniu tej funkcjonalności.

Rozdział 11 zawiera omówienie grafów i zapytań rekurencyjnych. Pokazuje, jak obsługiwać struktury grafów, takie jak hierarchie pracowników, zestawienia magazynowe czy mapy w SQL Server przy użyciu języka T-SQL. Pokazuje implementację takich zagadnień, jak model wyliczanej ścieżki (przy użyciu własnego rozwiązania lub za pomocą typu danych HIERARCHYID) oraz model zagnieżdżonych zbiorów. Pokazane zostało również wykorzystanie zapytań rekurencyjnych do manipulowania danymi w grafach.

Wymagania systemowe

Do wykonania przykładów kodu zawartych w książce potrzebne są następujące składniki programowe:

- Microsoft SQL Server 2014:
  - Wydanie 64-bitowe Enterprise, Developer lub Evaluation; inne wydania nie wspierają mechanizmów In-Memory OLTP ani technologii magazynu kolumnowego, omówionych w książce. Wersję próbną można pobrać ze strony *http://www.microsoft.com/sql*.
  - Aktualne wymagania sprzętowe i programowe można sprawdzić pod adresem *http://msdn.microsoft.com/en-us/library/ms143506(v=sql.120).aspx*.
  - W oknie dialogowym Feature Selection (wybór funkcjonalności) programu instalacyjnego SQL Server 2014 należy wybrać następujące komponenty: Database Engine Services, Client Tools Connectivity, Documentation Components, Management Tools – Basic, Management Tools – Complete.

- Microsoft Visual Studio 2013 z rozszerzeniem Microsoft SQL Server Data Tools (SSDT):
  - Aktualne wymagania sprzętowe i kompatybilność platformy dla Visual Studio 2013 można znaleźć pod adresem *http://www.visualstudio.com/products/visual-studio-2013-compatibility-vs*.
  - Informacje na temat instalowania SSDT zawiera strona *http://msdn.microsoft.com/en-us/data/tools.aspx*.

Zależnie od konfiguracji systemu Windows do zainstalowania i konfiguracji SQL Server 2014 oraz Visual Studio 2013 konieczne mogą być uprawnienia lokalnego administratora.

Pobieranie przykładów kodu

Książka zawiera bardzo wiele przykładów kodu. Całość kodu źródłowego można pobrać z witryny autorów *http://tsql.solidq.com/books/tq3*.

Kod źródłowy jest udostępniony jako plik skompresowany o nazwie *T-SQL Querying – YYYYMMDD.zip*, gdzie *YYYYMMDD* wskazuje ostatnią aktualizację zawartości. Po pobraniu pliku należy wykonać instrukcje zawarte w pliku Readme.txt dołączonym do archiwum, aby zainstalować przykłady kodu.

Errata, aktualizacje i wsparcie dla książki

Dołożyliśmy wszelkich starań, aby zapewnić dokładność i poprawność tej książki, jak i towarzyszącego jej kodu przykładowego. W razie wykrycia błędu można przesłać go do wydawcy pod adresem *mspinput@microsoft.com*. Pod tym samym adresem można również skontaktować się z zespołem Microsoft Press Book Support, jeśli potrzebna jest inna pomoc. Proszę zauważyć, że pod tym adresem nie jest oferowana pomoc techniczna dla oprogramowania Microsoft. Pomoc dotycząca oprogramowania i sprzętu firmy Microsoft jest dostępna poprzez witrynę *http://support.microsoft.com*.

Bezpłatne ebooki z Microsoft Press

Począwszy od wstępnych informacji technicznej, po pogłębione omówienie szczególnych zagadnień, bezpłatne ebooki wydawnictwa Microsoft Press obejmują szeroki zakres tematyki. Publikacje te są dostępne w formatach PDF, EPUB oraz Mobi for Kindle, gotowe do pobrania z witryny:

http://aka.ms/mspressfree

Warto zaglądać tam często, aby dowiedzieć się, co jest nowego!

Podziękowania

Wiele osób przyczyniło się do tego, aby książka ta ujrzała światło dzienne, zarówno bezpośrednio, jak i pośrednio, i zasługuje na podziękowania i wyróżnienie. Bardzo możliwe, że niezamierzenie pominąłem jakieś nazwiska i z góry za to przepraszam.

Lilach: jesteś tym kimś, kto sprawia, że chcę być dobry w tym, co robię. Niezależnie od tego, że inspirujesz mnie nieustannie, miałaś też nieoficjalną rolę w powstaniu tej książki – byłaś pierwszą czytelniczką! Spędziliśmy tak wiele godzin, sczytując tekst i wyszukując błędy, zanim wysłałem go do redaktorów. Mam wrażenie, że przynajmniej niektóre aspekty T-SQL znasz lepiej, niż ludzie, którzy zajmują się nim zawodowo.

Dziękuję moim rodzicom, Mila i Gabi, i moim dzieciom, Mickey i Ina, za stałe wsparcie i zaakceptowanie mojej czasowej nieobecności. Ostatnich kilka lat było bardzo intensywnych i mam nadzieję, że nadchodzące lata będą zdrowe i szczęśliwe.

Współautorzy książki, Dejan Sarka, Adam Machanic i Kevin Farlee: było prawdziwym zaszczytem stać się częścią tak doświadczonej grupy ludzi. Każdy z was jest takim ekspertem w swojej dziedzinie, że zdecydowałem, iż odpowiednie tematy najlepiej będzie powierzyć właśnie wam. Dejan stworzył rozdział o praktycznym stosowaniu języka T-SQL w BI, Adam napisał fragmenty o równoległym przetwarzaniu zapytań i programowaniu SQL CLR, zaś Kevin rozdział o mechanizmie In-Memory OLTP. Dzięki za wzięcie udziału w tworzeniu tej książki.

Do technicznego recenzenta książki, Alejandro Mesa: czytałeś i (nieoficjalnie) recenzowałeś moje wcześniejsze książki. Podszedłeś do tematu z taką pasją, że byłem szczęśliwy, że zdecydowałeś się przyjąć bardziej oficjalną rolę recenzenta. Chcę również podziękować recenzentowi wcześniejszego wydania, Steve Kass: wykonałeś tak wspaniałą i dokładną robotę, że jej echo jest ciągle słyszalne w tej najnowszej.

Mark Souza: brałeś udział w niemal całym procesie powstawania książki, od pierwszego pomysłu, biorąc odpowiedzialność za aspekty techniczne, organizacyjne i społecznościowe. Jeśli ktokolwiek naprawdę czuje puls społeczności SQL Server, to właśnie ty. Wszyscy jesteśmy wdzięczni za to, co robisz, i zaszczytem jest to, że to ty napisałeś przedmowę.

Podziękowania należą się też wielu redaktorom w wydawnictwie Microsoft Press. Devon Musgrave, który pełnił funkcję zarówno redaktora koordynującego, jak i projektowego: to dzięki tobie książka stała się realnym bytem. Zdaję sobie sprawę, że ta książka była tylko jedną z wielu, za które jesteś odpowiedzialny i chcę podziękować za czas i wysiłki, które przeznaczyłeś właśnie na nią. Jest jeszcze kilka nazwisk, które wymienię (i ponownie przepraszam, jeśli kogoś pominąłem): Carol Dillingham, redaktor projektu, Curtis Philips, menedżer projektu z Publishing.com, Roger LeBlanc, redaktor językowy, który pracował również przy moich wcześniejszych książkach, oraz Andrea Fox, korektorka. To była prawdziwa przyjemność pracować wspólnie z wami.

Co do SolidQ, mojej firmy przez ostatnią dekadę: wspaniałe jest być częścią tak świetnej firmy, która rozwinęła się do tego, czym jest obecnie. Członkowie tej firmy

są dla mnie znacznie więcej, niż kolegami; są partnerami, przyjaciółmi i rodziną. Oto niepełna z pewnością lista tych, którym chcę podziękować w tym miejscu: Fernando G. Guerrero, Douglas McDowell, Herbert Albert, Dejan Sarka, Gianluca Hotz, Antonio Soto, Jeanne Reeves, Glenn McCoin, Fritz Lechnitz, Eric Van Soldt, Berry Walker, Marilyn Templeton, Joelle Budd, Gwen White, Jan Taylor, Judy Dyess, Alberto Martin, Lorena Jimenez, Ron Talmage, Andy Kelly, Rushabh Mehta, Joe Chang, Mark Tabladillo, Eladio Rincón, Miguel Egea, Alejandro J. Rocchi, Daniel A. Seara, Javier Loria, Paco González, Enrique Catalá, Esther Nolasco Andreu, Rocío Guerrero, Javier Torrenteras, Rubén Garrigós, Victor Vale Diaz, Davide Mauri, Danilo Dominici, Erik Veerman, Jay Hackney, Grega Jerkič, Matija Lah, Richard Waymire, Carl Rabeler, Chris Randall, Tony Rogerson, Christian Rise, Raoul Illyés, Johan Åhlén, Peter Larsson, Paul Turley, Bill Haenlin, Blythe Gietz, Nigel Semmi, Paras Doshi i tak wielu innych.

Członkowie zespołu projektowego Microsoft SQL Server, w przeszłości i obecnie: Tobias Ternstrom, Lubor Kollar, Umachandar Jayachandran (UC), Boris Baryshnikov, Conor Cunningham, Kevin Farlee, Marc Friedman, Milan Stojic, Craig Freedman, Campbell Fraser, Mark Souza, T. K. Rengarajan, Dave Campbell, César Galindo-Legaria – i niewątpliwie wielu innych. Wiem, że starania o dodanie funkcji okna do SQL Server nie były łatwe ani trywialne. Dzięki za wielki wysiłek i za cały czas, który spędziliście ze mną i na odpowiadanie na moje maile, odpowiadanie na moje pytania i udzielanie wyjaśnień.

Członkowie zespołu redakcyjnego *SQL Server Pro*, byli i obecni: Megan Keller, Lavon Peters, Michele Crockett, Mike Otey, Jayleen Heft i wielu innych. Od przeszło piętnastu lat pisuję do waszego magazynu i jestem wdzięczny za możliwość dzielenia się wiedzą z waszymi czytelnikami.

MPV dla SQL Server, w przeszłości i obecnie: Paul White, Alejandro Mesa, Erland Sommarskog, Aaron Bertrand, Tibor Karaszi, Benjamin Nevarez, Simon Sabin, Darren Green, Allan Mitchell, Tony Rogerson i wielu innych – a przede wszystkim lider MVP, Simon Tien. To wspaniały program i jestem dumny z tego, że stałem się jego częścią. Poziom wiedzy i doświadczenia waszej grupy jest zadziwiający i jestem zawsze podekscytowany, gdy dochodzi do spotkania, zarówno po to, by dzielić się pomysłami, jak i by pogadać nad kuflem piwa. Szczególne podziękowania należą się Paulowi White. Nauczyłem się od ciebie tak wiele i zawsze sprawia mi radość czytanie twoich prac. Bezpiecznie mogę powiedzieć, że jesteś moim ulubionym autorem. Kto wie, może któregoś dnia siądziemy razem do pracy nad czymś wspólnym.

Na koniec chcę podziękować moim studentom: nauczanie T-SQL jest tym, co lubię najbardziej. To moja pasja. Dzięki za to, że chcecie mnie słuchać, a przede wszystkim za wszystkie celne pytania, które skłaniają mnie do szukania większej wiedzy.

Pozdrawiam, Itzik

ROZDZIAŁ 1

Logiczne przetwarzanie zapytań

Obserwując prawdziwych ekspertów z różnych dziedzin, można zauważyć, że ich wspólną cechą jest solidna znajomość podstaw. Wszyscy profesjonaliści niezależnie od specjalizacji zajmują się w gruncie rzeczy rozwiązywaniem problemów. A wszystkie problemy, niezależnie od stopnia złożoności, wiążą się ze stosowaniem kombinacji podstawowych technik. Aby zostać ekspertem w danej dziedzinie, trzeba budować wiedzę na solidnych podstawach. Opanowanie podstaw i wkładanie dużego wysiłku w doskonalenie warsztatu pozwala na zdobycie umiejętności rozwiazywania każdego problemu.

Ta książka poświęcona jest zapytaniom Transact-SQL (T-SQL) – podstawowym technikom i ich zastosowaniu do rozwiązywania problemów. Moim zdaniem, najlepiej rozpocząć ją od rozdziału przedstawiającego podstawy logicznego przetwarzania zapytań. Rozdział ten ma kluczowe znaczenie – nie tylko dlatego, że wprowadza najważniejsze zagadnienia związane z przetwarzaniem zapytań, ale również dlatego, że programowanie w języku SQL zasadniczo różni się od programowania w innych językach.

Uwaga To trzecia edycja książki. Poprzednie wydanie składało się z dwóch tomów, które zostały połączone w jeden tom. Szczegółowe informacje o zmianach oraz instrukcję pobrania kodu źródłowego i przykładowych danych znaleźć można we Wstępie.

T-SQL to stosowany w systemie Microsoft SQL Server dialekt (lub rozszerzenie) standardów ANSI oraz ISO języka SQL. Na łamach tej książki będę zamiennie używał terminów *SQL* oraz *T-SQL*. Omawiając aspekty języka wywodzące się ze standardu ANSI/ISO SQL i odnoszące się do większości dialektów, będę zazwyczaj używać terminu *SQL* lub *standardowy język SQL*. Natomiast przy omawianiu aspektów języka związanych z implementacją systemu SQL Server, zwykle będę korzystać z terminu *T-SQL*. Warto pamiętać, że formalna nazwa języka to *Transact-SQL*, choć zwykle używa się skrótu *T-SQL*. Większość programistów uważa skrót T-SQL za wygodniejszy w użyciu, w związku z tym zdecydowałem się na stosowanie tego terminu.

Źródła wymowy skrótu SQL

Wielu angielskojęzycznych profesjonalistów zajmujących się bazami danych wymawia skrót *SQL* jako *sequel* („s-ikłel"), pomimo tego że poprawna wymowa to *S-Q-L* („es kju el"). Można postawić pewną hipotezę co do źródeł tej nieprawidłowej wymowy. Podejrzewam, że wynika ona zarówno z aspektów historycznych, jak i językowych.

Odnośnie aspektów historycznych, w latach 70-tych firma IBM opracowała język o nazwie SEQUEL, która była akronimem słów Structured English QUEry Language (Strukturalny język zapytań w języku angielskim). Język ten został zaprojektowany z myślą o manipulowaniu danymi zapisanymi w systemie bazodanowym o nazwie System R, który był oparty na opracowanym przez dr. Edgara F. Coddiego modelu RDBMS (Relational Database Management Systems – Systemy zarządzania relacyjnymi bazami danych). W późniejszym okresie z powodu sporu o zastrzeżony znak towarowy, akronim SEQUEL został skrócony do SQL. Instytut ANSI przyjął język SQL jako standard w roku 1986, a organizacja ISO w roku 1987. Instytut ANSI zadeklarował, że oficjalna wymowa skrótu SQL to „es kju el", ale ewidentnie nie wszyscy zdają sobie z tego sprawę.

Ze względów językowych – słowo *sequel* („sikłel") jest po prostu wygodniejsze w wymowie, w szczególności dla osób angielskojęzycznych i to wpływa na powszechne użycie tej formy.

Dodatkowe informacje Zamieszczone w tym rozdziale informacje o historii języka SQL zostały zaczerpnięte z artykułu wolnej encyklopedii Wikipedia, który dostępny jest (w języku angielskim) pod adresem *http://en.wikipedia.org/wiki/SQL*.

Programowanie w języku SQL składa się w wielu różnych aspektów, takich jak myślenie w kategoriach zbiorów, kolejność logicznego przetwarzania elementów zapytania oraz logika trójwartościowa. Próby programowania w języku SQL bez tej wiedzy prowadzą do powstawania kodu, który jest rozwlekły, mało efektywny i trudny w utrzymywaniu. Celem tego rozdziału jest zaprezentowanie języka SQL w sposób zgodny z intencjami jego twórców. Rozpocznę od zbudowania solidnych podstaw, na których opierać się będą pozostałe umiejętności. Omawiając elementy charakterystyczne dla języka T-SQL, podkreślę ten fakt.

Na łamach tej książki omawiać będę złożone problemy i zaawansowane techniki. Jednak w tym rozdziale ograniczę się do podstaw tworzenia zapytań. Wydajnością zajmę się w dalszej części książki, na razie skoncentruję się na logicznych aspektach przetwarzania zapytań. W związku z tym proszę o kompletne zignorowanie problemu wydajności podczas lektury tego rozdziału. Problematyka ta zostanie szczegółowo omówiona w dalszej

części książki. Niektóre z faz przetwarzania logicznego zapytań mogą wydawać się bardzo nieefektywne. Jednak warto mieć świadomość, że rzeczywisty sposób fizycznego przetwarzania zapytania może bardzo różnić się od przetwarzania logicznego.

W systemie SQL Server za generowanie fizycznego planu wykonania zapytania odpowiada *optymalizator zapytań*. W ramach planu optymalizator definiuje m.in. kolejność uzyskiwania dostępu do tabel i wykorzystywane do tego celu metody, zastosowane indeksy czy algorytmy złączania. Optymalizator generuje wiele poprawnych planów wykonania i wybiera spośród nich ten o najniższym koszcie. Fazy logicznego przetwarzania zapytania mają ściśle określoną kolejność. Natomiast optymalizator może stosować pewne skróty w generowanym fizycznym planie wykonywania. Oczywiście tego rodzaju skróty stosowane są tylko wówczas, gdy istnieje gwarancja, że otrzymany zbiór wynikowy będzie zbiorem poprawnym – innymi słowy, że będzie to taki sam zbiór, jaki zostałby uzyskany w wyniku ścisłego realizowania kolejnych faz logicznego przetwarzania zapytania. Na przykład, aby umożliwić wykorzystanie określonego indeksu, optymalizator może zadecydować o zastosowaniu filtra znacznie wcześniej, niż to wynika z przetwarzania logicznego. Kolejność przetwarzania elementów zgodnie z planem fizycznego wykonania zapytania nazywać będziemy *kolejnością fizycznego przetwarzania*.

Z wspomnianych powodów należy dokonać wyraźnego rozgraniczenia pomiędzy logicznym i fizycznym przetwarzaniem zapytań

Nadeszła pora, aby zająć się szczegółowym omówieniem faz logicznego przetwarzania zapytań.

Fazy logicznego przetwarzania zapytań

W tym podrozdziale przedstawię kolejne fazy logicznego przetwarzania zapytań. Na początku pokrótce opiszę wszystkie kroki tego procesu, a następnie w kolejnych podrozdziałach omówię je bardziej szczegółowo, demonstrując ich przebieg na przykładzie konkretnego zapytania. Ten podrozdział stanowi w pewnym sensie streszczenie, do którego można się odwołać, aby szybko sprawdzić kolejność i ogólne znaczenie poszczególnych faz.

Listing 1-1 zawiera ogólną postać zapytania, wraz z numerami wskazującymi kolejność logicznego przetwarzania poszczególnych klauzul.

Listing 1-1 Ponumerowane kroki procesu logicznego przetwarzania zapytania

```
(5) SELECT (5-2) DISTINCT (7) TOP(<specyfikacja_top>) (5-1) <lista_select>
(1) FROM (1-J) <lewa_tabela> <typ_złączenia> JOIN <prawa_tabela> ON <predykat_ON>
       | (1-A) <lewa_tabela> <typ_operatora_apply> APPLY <prawa_tabela_wejściowa>
                                                    AS <alias>
       | (1-P) <lewa_tabela> PIVOT(<specyfikacja_pivot>) AS <alias>
       | (1-U) <lewa_tabela> UNPIVOT(<specyfikacja_unpivot >) AS <alias>
(2) WHERE <predykat_where >
```

```
(3) GROUP BY <specyfikacja_group_by>
(4) HAVING <predykat_having >
(6) ORDER BY <lista_order_by>
(7) OFFSET <specyfikacja_offset> ROWS FETCH NEXT <specyfikacja_fetch> ROWS ONLY;
```

Pierwszym zauważalnym aspektem języka SQL, odróżniającym go od innych języków programowania, jest kolejność przetwarzania kodu. W większości języków programowania przetwarzanie instrukcji odbywa się w kolejności zgodnej z kierunkiem tekstu (nazywanej *kolejnością wpisania*). W języku SQL proces przetwarzania rozpoczyna się od klauzuli FROM, natomiast klauzula SELECT, która została wpisana na początku, zostaje przetworzona prawie na samym końcu. Kolejność tę nazywać będę *kolejnością logicznego przetwarzania* i nie należy mylić jej z kolejnością wpisania lub kolejnością fizycznego przetwarzania.

Nie bez powodu kolejność logicznego przetwarzania różni się od kolejności wpisania. Jak wspomniałem wcześniej, SQL wywodzi się z języka SEQUEL, a pierwsza litera *E* w nazwie SEQUEL reprezentuje słowo *English* (Angielski). Projektanci języka SQL dążyli do uzyskania języka deklaratywnego, w którym instrukcje mogą być wpisywane w sposób przypominający język naturalny. Analizując sposób wydawania poleceń, np. „Przynieś mi książkę z biura", można zauważyć, że obiekt (książka) został wskazany przed określeniem jego lokalizacji (biuro). Mimo iż osoba realizująca polecenie będzie musiała najpierw dotrzeć do biura, a dopiero później wybrać książkę. Analogicznie wpisując polecenia w języku SQL, rozpoczynamy od klauzuli SELECT z wybranymi kolumnami, a dopiero później dodajemy klauzulę FROM z tabelami źródłowymi. Jednak proces logicznego przetwarzania zapytania rozpoczyna się od klauzuli FROM. Świadomość tego faktu pomaga w zrozumieniu wielu aspektów języka SQL, które wcześniej mogły wydawać się niejasne.

W każdym kroku logicznego przetwarzania zapytania generowana jest tabela wirtualna, która pełni rolę danych wejściowych kolejnego kroku. Te tabele wirtualne nie są dostępne dla obiektu wywołującego (aplikacji klienckiej lub zapytania zewnętrznego). Zwrócona zostaje tylko tabela wygenerowana w ostatnim kroku procesu. Jeśli pewna klauzula nie została użyta w zapytaniu, odpowiadający jej krok jest po prostu pomijany. W kolejnej części rozdziału pokrótce omówię poszczególne fazy przetwarzania logicznego.

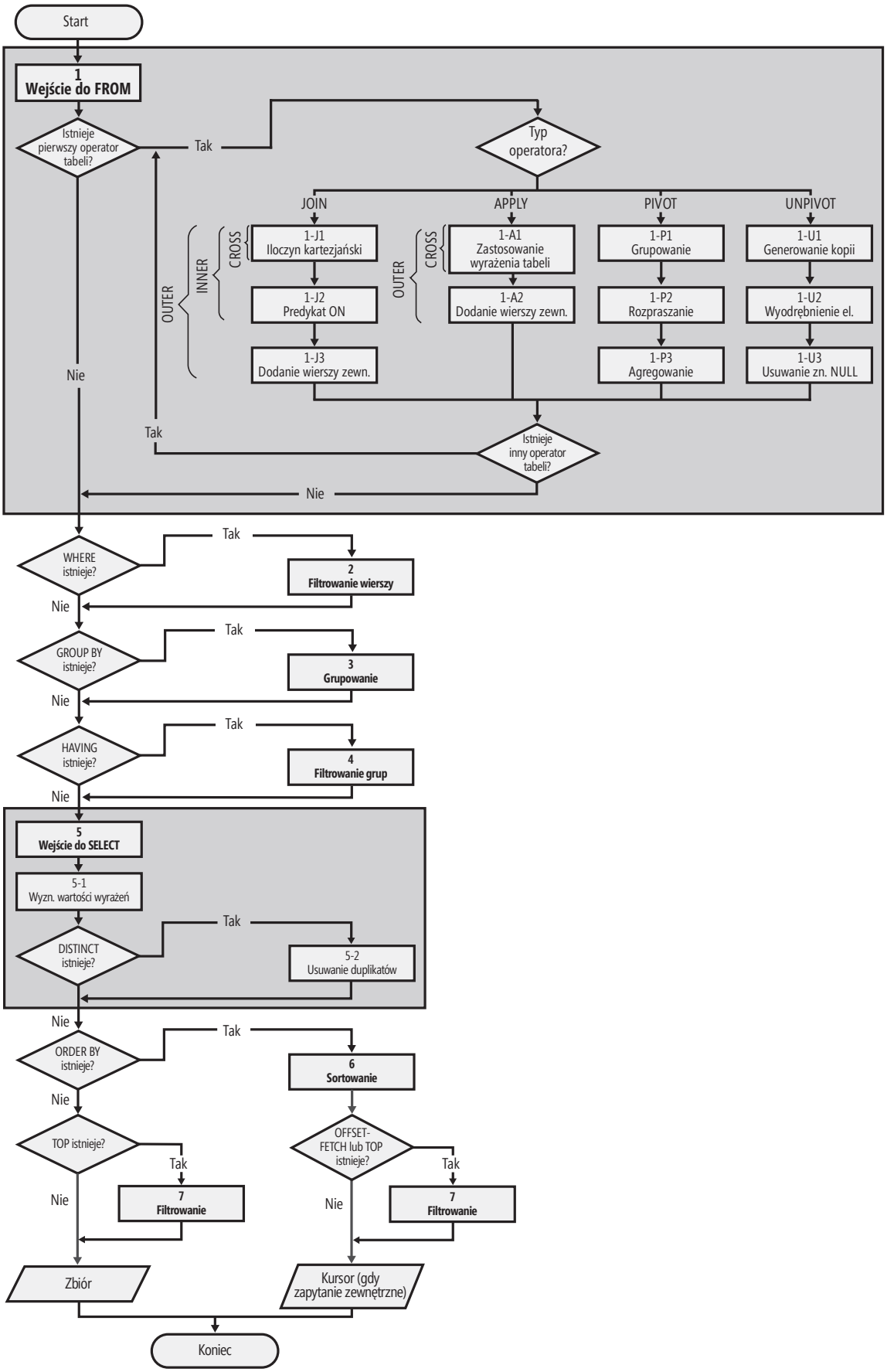
Krótkie omówienie faz logicznego przetwarzania zapytania

Opis niektórych faz może wydać się na razie mało zrozumiały, lecz nie należy się tym zbytnio przejmować. Poniższe opisy mają jednie charakter referencyjny. Poszczególne fazy zostaną omówione dużo bardziej szczegółowo po przedstawieniu przykładowego scenariusza. Opiszę jedynie operator tabel JOIN, ponieważ jest on najczęściej stosowany i powszechnie znany. Pozostałe operatory tabel (APPLY, PIVOT oraz UNPIVOT)

zostaną omówione w dalszej części rozdziału zatytułowanej „Pozostałe aspekty logicznego przetwarzania zapytań".

Na rysunku 1-1 zaprezentowany został diagram przepływu ilustrujący poszczególne fazy logicznego przetwarzania zapytania. W tym rozdziale będę wielokrotnie odwoływać się do numerów przypisanych krokom procesu na tym diagramie.

- **(1) FROM** W tej fazie następuje zidentyfikowanie źródłowych tabel zapytania i przetworzenie operatorów tabel. Realizacja każdego z operatorów tabel przebiega w serii faz podrzędnych. Np. operacja złączania JOIN składa się z faz: (1-J1) Iloczyn kartezjański, (1-J2) Predykat ON oraz (1-J3) Dodanie wierszy zewnętrznych. W tej fazie generowana jest tabela wirtualna VT1.
- **(1-J1) Iloczyn kartezjański** Ta faza polega na wykonaniu iloczynu kartezjańskiego (CROSS JOIN) pomiędzy dwiema tabelami połączonymi operatorem tabel, co skutkuje wygenerowaniem tabeli wirtualnej VT1-J1.
- **(1-J2) Predykat ON** Ta faza polega na odfiltrowaniu wierszy z tabeli VT1-J1 na podstawie predykatu zastosowanego w klauzuli ON (<*predykat_ON*>). Do generowanej tabeli VT1-J2 wstawione zostają tylko te wiersze, dla których podany predykat ma wartość TRUE.
- **(1-J3) Dodanie wierszy zewnętrznych** Jeśli użyta została klauzula OUTER JOIN (zamiast CROSS JOIN lub INNER JOIN), wiersze z zachowanej tabeli lub tabel, dla których nie został spełniony warunek dopasowania, zostaną dodane jako wiersze zewnętrzne do tabeli VT1-J2. W ten sposób wygenerowana zostanie tabela wirtualna VT1-J3.
- **(2) WHERE** W tej fazie następuje odfiltrowanie wierszy z tabeli VT1 na podstawie predykatu zastosowanego w klauzuli WHERE (<*predykat_where*>). Do tabeli VT2 wstawione zostają tylko te wiersze, dla których podany predykat ma wartość TRUE.
- **(3) GROUP BY** Ta faza polega na zorganizowaniu wierszy tabeli VT2 w grupy, na podstawie zbioru wyrażeń zdefiniowanego w klauzuli GROUP BY i nazywanego *zbiorem grupowania*. W ten sposób wygenerowana zostaje tabela VT3. Ostateczny efekt polega na utworzeniu po jednym wierszu wynikowym dla każdej grupy spełniającej określone warunki.
- **(4) HAVING** Ta faza polega na odfiltrowaniu grup z tabeli VT3 na podstawie predykatu zastosowanego w klauzuli HAVING (<*predykat_having*>). Do tabeli VT4 wstawione zostają tylko te grupy, dla których podany predykat ma wartość TRUE.
- **(5) SELECT** W tej fazie następuje przetworzenie elementów klauzuli SELECT, co skutkuje wygenerowaniem tabeli VT5.
- **(5-1) Wyznaczenie wartości wyrażeń** Ta faza polega na przetworzeniu wyrażeń zdefiniowanych na liście klauzuli SELECT, czego rezultatem jest wygenerowanie tabeli VT5-1.



Rysunek 1-1 Diagram przepływu ilustrujący logiczne przetwarzanie zapytania.

- **(5-2) DISTINCT** W tej fazie następuje usunięcie powtarzających się wierszy z tabeli VT5-1 i wygenerowanie tabeli VT5-2.
- **(6) ORDER BY** W tej fazie wiersze tabeli wirtualnej VT5-2 zostają posortowane według listy określonej w klauzuli ORDER BY, co skutkuje wygenerowaniem kursora wirtualnego VC6. W przypadku braku klauzuli ORDER BY tabela VT5-2 staje się tabelą wirtualną VT6.
- **(7) TOP | OFFSET-FETCH** Ta faza polega na odfiltrowaniu wierszy z kursora VC6 lub tabeli VT6 na podstawie specyfikacji TOP lub OFFSET-FETCH, co skutkuje wygenerowaniem odpowiednio kursora VC7 lub tabeli VT7. Jeśli zastosowana została specyfikacja TOP, w tej fazie odfiltrowana zostaje określona liczba wierszy w kolejności zdefiniowanej w klauzuli ORDER BY lub w przypadku jej braku w kolejności przypadkowej. Jeśli zastosowana została specyfikacja OFFSET-FETCH, w tej fazie pominięta zostaje określona liczba wierszy, a następnie wybrana zostaje wskazana liczba wierszy w kolejności zdefiniowanej w klauzuli ORDER BY. Filtr OFFSET-FETCH został wprowadzony w wersji SQL Server 2012.

Przykładowe zapytanie oparte na scenariuszu z użyciem tabeli klientów i zamówień

Przeanalizujemy teraz przykładowe zapytanie, które pomoże nam w zilustrowaniu poszczególnych faz procesu logicznego przetwarzania zapytań Zaczniemy od uruchomienia pokazanego poniżej fragmentu kodu, który tworzy tabele dbo.Customers (dane klientów) i dbo.Orders (dane zamówień), wypełnia je przykładowymi danymi, a następnie wykona zapytania wyświetlające zawartość obu tabel:

```
SET NOCOUNT ON;
USE tempdb;

IF OBJECT_ID(N'dbo.Orders', N'U') IS NOT NULL DROP TABLE dbo.Orders;
IF OBJECT_ID(N'dbo.Customers', N'U') IS NOT NULL DROP TABLE dbo.Customers;

CREATE TABLE dbo.Customers
(
  custid  CHAR(5)     NOT NULL,
  city    VARCHAR(10) NOT NULL,
  CONSTRAINT PK_Customers PRIMARY KEY(custid)
);

CREATE TABLE dbo.Orders
(
  orderid INT     NOT NULL,
  custid  CHAR(5)     NULL,
  CONSTRAINT PK_Orders PRIMARY KEY(orderid),
  CONSTRAINT FK_Orders_Customers FOREIGN KEY(custid)
    REFERENCES dbo.Customers(custid)
);
GO
```

```
INSERT INTO dbo.Customers(custid, city) VALUES
  ('FISSA', 'Madrid'),
  ('FRNDO', 'Madrid'),
  ('KRLOS', 'Madrid'),
  ('MRPHS', 'Zion'  );

INSERT INTO dbo.Orders(orderid, custid) VALUES
  (1, 'FRNDO'),
  (2, 'FRNDO'),
  (3, 'KRLOS'),
  (4, 'KRLOS'),
  (5, 'KRLOS'),
  (6, 'MRPHS'),
  (7, NULL   );

SELECT * FROM dbo.Customers;
SELECT * FROM dbo.Orders;
```

W wyniku wykonania kodu wygenerowane zostaną następujące dane wyjściowe:

```
custid city
------ ----------
FISSA  Madrid
FRNDO  Madrid
KRLOS  Madrid
MRPHS  Zion

orderid     custid
----------- ------
1           FRNDO
2           FRNDO
3           KRLOS
4           KRLOS
5           KRLOS
6           MRPHS
7           NULL
```

Za przykład niech posłuży zapytanie przedstawione na Listingu 1-2. Zapytanie to zwraca listę tych klientów z miasta Madrid, którzy złożyli mniej niż trzy zamówienia (w tym również zero zamówień), wraz z informacją o liczbie złożonych przez nich zamówień. Wynik zostanie posortowany rosnąco według liczby złożonych zamówień.

Listing 1-2 Zapytanie: Klienci z miasta Madrid, którzy złożyli mniej niż trzy zamówienia

```
SELECT C.custid, COUNT(O.orderid) AS numorders
FROM dbo.Customers AS C
  LEFT OUTER JOIN dbo.Orders AS O
    ON C.custid = O.custid
WHERE C.city = 'Madrid'
GROUP BY C.custid
HAVING COUNT(O.orderid) < 3
ORDER BY numorders;
```

Powyższe zapytanie zwróci następujące wiersze:

```
custid numorders
------ -----------
FISSA  0
FRNDO  2
```

FISSA i FRNDO są klientami z miasta Madrid i złożyli mniej niż trzy zamówienia. Warto przestudiować tekst przykładowego zapytania i spróbować odczytać go zgodnie z fazami przedstawionymi na Listingu 1-1, na Rysunku 1-1 oraz w podrozdziale „Krótkie omówienie faz logicznego przetwarzania zapytania". Osoby, które po raz pierwszy przeprowadzają tego typu analizę zapytania, mogą mieć z nią pewne problemy. Kolejny podrozdział powinien jednak pomóc w zrozumieniu tajników tego procesu.

Szczegółowe omówienie faz logicznego przetwarzania zapytania

Ten podrozdział zawiera szczegółowy opis faz procesu logicznego przetwarzania zapytania zilustrowany na przykładzie przedstawionego uprzednio zapytania.

Krok 1: Faza FROM

W fazie FROM identyfikowana jest tabela lub tabele, z których muszą zostać pobrane dane. Jeśli zostały zastosowane operatory tabel, są one przetwarzane w tej fazie w kolejności od lewej do prawej. Każdy operator tabeli działa na jednej lub dwóch tabelach wejściowych i zwraca jedną tabelę wyjściową. Wynik zastosowania operatora tabeli zostaje użyty w roli lewej tabeli wejściowej następnego operatora, jeśli takowy istnieje, lub jako tabela wejściowa dla następnej fazy przetwarzania logicznego. Każdy operator tabeli charakteryzuje się własnym zestawem podrzędnych faz przetwarzania. Np. operacja JOIN składa się z faz: (1-J1) Iloczyn kartezjański, (1-J2) Predykat ON oraz (1-J3) Dodanie wierszy zewnętrznych. Jak wspomniałem wcześniej, pozostałe operatory tabel opiszę w dalszej części tego rozdziału. Faza FROM prowadzi do wygenerowania wirtualnej tabeli VT1.

Krok 1-J1: Wyznaczenie iloczynu kartezjańskiego (złączenie krzyżowe)

Jest to pierwsza z trzech faz podrzędnych, występujących w przypadku użycia operatora tabeli JOIN. W tej fazie podrzędnej wyznaczony zostaje iloczyn kartezjański (złączenie krzyżowe) dwóch tabel, na których wykonywana jest operacja złączania. W rezultacie wygenerowana zostaje tabela wirtualna VT1-J1. Ta tabela zawiera po jednym wierszu dla każdej możliwej kombinacji wiersza z lewej tabeli i wiersza z prawej tabeli. Jeśli lewa tabela zawiera *n* wierszy, a prawa *m* wierszy, tabela VT1-J1 zawierać

będzie *n*×*m* wierszy. Kolumny tabeli VT1-J1 zostają zakwalifikowane (opatrzone przedrostkiem) przy użyciu nazw tabel źródłowych (lub aliasów tabel, jeśli zostały one zdefiniowane w zapytaniu). W kolejnych krokach (począwszy od kroku 1-J2) odwołania do nazwy kolumny, która nie jest jednoznaczna (tj. występuje w więcej niż jednej tabeli wejściowej), muszą zawierać nazwę tabeli (np. *C.custid*). Dodawanie nazw tabel do nazw kolumn, które pojawiają się w tylko jednej tabeli wejściowej, nie jest obowiązkowe (np. można użyć wyrażenia *O.orderid* lub po prostu *orderid*).

A teraz zastosujemy krok 1-J1 na przykładowym zapytaniu (przedstawionym na Listingu 1-2):

```
FROM dbo.Customers AS C ... JOIN dbo.Orders AS O
```

W rezultacie otrzymamy tabelę wirtualną VT1-J1 zawierającą 28 wierszy (4×7) (przedstawioną poniżej jako Tabela 1-1).

Tabela 1-1 Tabela wirtualna VT1-J1 wygenerowana w kroku 1-J1

C.custid	C.city	O.orderid	O.custid
FISSA	Madrid	1	FRNDO
FISSA	Madrid	2	FRNDO
FISSA	Madrid	3	KRLOS
FISSA	Madrid	4	KRLOS
FISSA	Madrid	5	KRLOS
FISSA	Madrid	6	MRPHS
FISSA	Madrid	7	NULL
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
FRNDO	Madrid	3	KRLOS
FRNDO	Madrid	4	KRLOS
FRNDO	Madrid	5	KRLOS
FRNDO	Madrid	6	MRPHS
FRNDO	Madrid	7	NULL
KRLOS	Madrid	1	FRNDO
KRLOS	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
KRLOS	Madrid	6	MRPHS
KRLOS	Madrid	7	NULL
MRPHS	Zion	1	FRNDO

Tabela 1-1 Tabela wirtualna VT1-J1 wygenerowana w kroku 1-J1

C.custid	C.city	O.orderid	O.custid
MRPHS	Zion	2	FRNDO
MRPHS	Zion	3	KRLOS
MRPHS	Zion	4	KRLOS
MRPHS	Zion	5	KRLOS
MRPHS	Zion	6	MRPHS
MRPHS	Zion	7	NULL

Krok 1-J2: Zastosowanie predykatu ON (warunku złączania)

Klauzula ON stanowi pierwszą z trzech klauzul zapytania (ON, WHERE lub HAVING) służących do filtrowania wierszy w oparciu o predykat. Predykat w klauzuli ON zostaje zastosowany na wszystkich wierszach tabeli wirtualnej zwróconej w poprzednim kroku (VT1-J1). Tylko wiersze, dla których wyrażenie *<predykat_ON>* ma wartość TRUE, zostają dodane do tabeli wirtualnej generowanej w tym kroku (VT1-J2).

A teraz wykonajmy krok 1-J2 dla przykładowego zapytania:

```
ON C.custid = O.custid
```

Pierwsza kolumna Tabeli 1-2 zawiera wartości wyrażenia logicznego zawartego w klauzuli ON dla wierszy tabeli wirtualnej VT1-J1.

Tabela 1-2 Wartości logiczne predykatu ON dla wierszy tabeli VT1-J1

Wartość logiczna	C.custid	C.city	O.orderid	O.custid
FALSE	FISSA	Madrid	1	FRNDO
FALSE	FISSA	Madrid	2	FRNDO
FALSE	FISSA	Madrid	3	KRLOS
FALSE	FISSA	Madrid	4	KRLOS
FALSE	FISSA	Madrid	5	KRLOS
FALSE	FISSA	Madrid	6	MRPHS
UNKNOWN	FISSA	Madrid	7	NULL
TRUE	FRNDO	Madrid	1	FRNDO
TRUE	FRNDO	Madrid	2	FRNDO
FALSE	FRNDO	Madrid	3	KRLOS
FALSE	FRNDO	Madrid	4	KRLOS
FALSE	FRNDO	Madrid	5	KRLOS
FALSE	FRNDO	Madrid	6	MRPHS

Tabela 1-2 Wartości logiczne predykatu ON dla wierszy tabeli VT1-J1

Wartość logiczna	C.custid	C.city	O.orderid	O.custid
UNKNOWN	FRNDO	Madrid	7	NULL
FALSE	KRLOS	Madrid	1	FRNDO
FALSE	KRLOS	Madrid	2	FRNDO
TRUE	KRLOS	Madrid	3	KRLOS
TRUE	KRLOS	Madrid	4	KRLOS
TRUE	KRLOS	Madrid	5	KRLOS
FALSE	KRLOS	Madrid	6	MRPHS
UNKNOWN	KRLOS	Madrid	7	NULL
FALSE	MRPHS	Zion	1	FRNDO
FALSE	MRPHS	Zion	2	FRNDO
FALSE	MRPHS	Zion	3	KRLOS
FALSE	MRPHS	Zion	4	KRLOS
FALSE	MRPHS	Zion	5	KRLOS
TRUE	MRPHS	Zion	6	MRPHS
UNKNOWN	MRPHS	Zion	7	NULL

Do tabeli wirtualnej VT1-J2 wstawione zostają tylko te wiersze, dla których <*predykat_ON*> ma wartość TRUE, jak pokazano w tabeli 1-3.

Tabela 1-3 Tabela wirtualna VT1-J2 otrzymana w kroku 1-J2

Wartość logiczna	C.custid	C.city	O.orderid	O.custid
TRUE	FRNDO	Madrid	1	FRNDO
TRUE	FRNDO	Madrid	2	FRNDO
TRUE	KRLOS	Madrid	3	KRLOS
TRUE	KRLOS	Madrid	4	KRLOS
TRUE	KRLOS	Madrid	5	KRLOS
TRUE	MRPHS	Zion	6	MRPHS

Krok 1-J3: Dodanie wierszy zewnętrznych

Ten krok wykonywany jest tylko w przypadku zastosowania złączeń zewnętrznych. Przeprowadzając złączenie zewnętrzne, można oznaczyć jedną lub obie tabele jako *zachowane*, wybierając typ złączenia zewnętrznego (LEFT, RIGHT lub FULL). Oznaczenie tabeli jako zachowanej oznacza, że zwrócone zostają wszystkie jej wiersze, nawet te niespełniające warunku <*predykat_ON*>.

Znaczniki NULL oraz logika trójwartościowa

Pozwolę sobie teraz na małą dygresję, aby zwrócić uwagę na pewne istotne aspekty języka SQL związane z tą fazą. Klasyczny model relacyjny definiuje dwa znaczniki reprezentujące brakującą wartość: znacznik A (brakująca i odnosząca się) oraz znacznik I (brakująca i nieodnosząca się). Znacznik A reprezentuje sytuację, gdy wartość odnosi się do danej relacji, ale z jakiegoś powodu jej brakuje. Weźmy pod uwagę na przykład atrybut *dataurodzenia* relacji Osoba. Mimo iż informacja ta jest znana dla większości osób, istnieją rzadkie wyjątki. Przykład może stanowić dziadek mojej żony, który oczywiście urodził się określonego dnia, ale nikt nie zna dokładnej daty. Natomiast znacznik I reprezentuje sytuację, gdy wartość nie odnosi się do danej relacji. Za przykład może posłużyć atrybut *idklienta* relacji Zamówienia. Załóżmy, że jeśli w wyniku przeprowadzenia inwentaryzacji odkryjemy niezgodność między oczekiwanym stanem magazynu a stanem faktycznym, dodamy fikcyjne zamówienie uzupełniające wykryty brak. Identyfikator klienta nie odnosi się do tego typu transakcji.

W odróżnieniu od modelu relacyjnego, standardowy język SQL nie wprowadza rozróżnienia między wspomnianymi typami brakujących wartości i oferuje tylko jeden znacznik ogólnego użytku – znacznik NULL. Znaczniki NULL bywają źródłem wielu niejasności i problemów w języku SQL, a zatem warto dobrze je zrozumieć w celu uniknięcia pomyłek. Jednym z najczęściej popełnianych błędów jest używanie określenia „wartość NULL" – NULL nie jest wartością, lecz znacznikiem reprezentującym brakującą wartość. W związku z tym prawidłowym określeniem jest „znacznik NULL" lub po prostu „NULL". Ponieważ w języku SQL brakujące wartości są reprezentowane przy pomocy tylko jednego znacznika NULL, system SQL nie wie, czy ma do czynienia z brakującą i odnoszącą się, czy też brakującą i nieodnoszącą się wartością. Na przykład w wykorzystywanej w przykładzie tabeli jedno z zamówień zawiera znacznik NULL w kolumnie *custid*. Załóżmy, że jest to fikcyjne zamówienie niezwiązane z żadnym klientem (jak w opisanym poprzednio scenariuszu). Przypisany znacznik NULL sygnalizuje brak wartości, ale nie informuje o tym, że dany atrybut nie odnosi się do tego typu zamówienia.

Stosowanie znaczników NULL w języku SQL prowadzi często do nieporozumień związanych z predykatami definiowanymi m.in. w filtrach zapytań czy ograniczeniach CHECK. W języku SQL predykat może przyjąć wartość TRUE (prawda), FALSE (fałsz) lub UNKNOWN (nieznana). Jest to tak zwana *logika trójwartościowa*, która jest charakterystyczna dla języka SQL. Wyrażenia logiczne w większości języków programowania mogą jedynie przyjmować wartość TRUE lub FALSE. Wartość logiczna UNKNOWN w języku SQL pojawia się zwykle w wyrażeniach powiązanych ze znacznikiem *NULL*. Wartość logiczną UNKNOWN

mają na przykład następujące trzy wyrażenia: *NULL* > 1759; *NULL* = *NULL*; *X* + *NULL* > *Y*. Należy pamiętać, że znacznik NULL reprezentuje brak wartości. Zgodnie ze standardem języka SQL, logiczny wynik porównania brakującej wartości z inną wartością (nawet innym znacznikiem NULL) zawsze jest nieznany (UNKNOWN).

Występowanie wyników logicznych o wartości UNKNOWN oraz znaczników NULL może prowadzić do nieporozumień Choć wyrażenie *NOT* TRUE ma wartość FALSE, a wyrażenie NOT FALSE ma wartość TRUE, to zaprzeczenie wartości UNKNOWN (NOT UNKNOWN) jest nadal wartością UNKNOWN.

Wartości logiczne UNKNOWN oraz znaczniki NULL są traktowane w różny sposób w różnych elementach języka. Na przykład, wszystkie filtry zapytania (ON, WHERE oraz HAVING) traktują wartość UNKNOWN tak samo jak wartość FALSE. Wiersz, dla którego filtr przyjmuje wartość UNKNOWN, zostaje wyeliminowany ze zbioru wyników. Innymi słowy, filtry zapytania zwracają jedynie wystąpienia TRUE. Natomiast w ograniczeniu CHECK wartość UNKNOWN jest traktowana tak samo jak wartość TRUE. Załóżmy, że dla określonej tabeli zdefiniowaliśmy ograniczenie CHECK, zgodnie z którym wartość kolumny reprezentującej wynagrodzenie musi być większa niż zero. Wiersz ze znacznikiem NULL będzie mógł zostać wstawiony do tego typu tabeli, ponieważ wyrażenie (*NULL* > *0*) ma wartość UNKNOWN, która w ograniczeniu CHECK jest traktowana jak wartość TRUE. Innymi słowy, ograniczenie CHECK odrzuca wystąpienia FALSE.

Porównanie w filtrze dwóch znaczników NULL prowadzi do uzyskania wartości UNKNOWN, która, jak wspomnieliśmy wcześniej, jest traktowana jak wartość FALSE, jak gdyby porównywane znaczniki NULL różniły się od siebie.

Natomiast w ograniczeniach UNIQUE, niektórych operatorach relacyjnych oraz operacjach sortowania i grupowania znaczniki NULL są traktowane jak równe wartości:

- Nie można wstawić do tabeli dwóch wierszy ze znacznikiem NULL w kolumnie, dla której zdefiniowane zostało ograniczenie UNIQUE. Pod tym względem język T-SQL wyłamuje się ze standardu.
- Klauzula GROUP BY umieszcza wszystkie znaczniki NULL w jednej grupie.
- Klauzula ORDER BY sortuje wszystkie znaczniki NULL razem.
- Operatory UNION, EXCEPT oraz INTERSECT traktują znaczniki NULL jak równe wartości, porównując wiersze obu zbiorów.

Co ciekawe, w standardzie SQL proces porównywania wartości przy pomocy operatorów relacyjnych UNION, EXCEPT oraz INTERSECT został opisany nie na zasadzie porównania (= oraz <>), lecz przy pomocy relacji unikatowości (odpowiednio *NOT DISTINCT* oraz *IS DISTINCT*). Co więcej, standard definiuje jawny *predykat unikatowości* postaci: IS [NOT] DISTINCT FROM. Natomiast

wspomniane operatory relacyjne wykorzystują operator unikatowości w sposób niejawny, który różni się od predykatów bazujących na operatorach = oraz <> tym, że realizuje logikę dwuwartościową. Następujące wyrażenia są prawdziwe: *NULL IS NOT DISTINCT FROM NULL*, *NULL IS DISTINCT FROM 1759*. W wersji SQL Server 2014 język T-SQL nie wspiera jawnego predykatu unikatowości, choć mamy nadzieję, że taka możliwość zostanie dodana w przyszłości (prośba o zaimplementowanie tej funkcji została przedstawiona na stronie: *http://connect.microsoft.com/SQLServer/feedback/details/286422/*).

Podsumowując, aby uniknąć przyszłych komplikacji, warto wiedzieć, jak wartości logiczne UNKNOWN oraz znaczniki NULL są traktowane w różnych elementach języka.

Lewe złączenie zewnętrzne (LEFT OUTER JOIN) oznacza lewą tabelę jako zachowaną, prawe złączenie zewnętrzne (RIGHT OUTER JOIN) oznacza prawą tabelę, natomiast (LEFT OUTER JOIN) oznacza obie tabele. W kroku 1-J3 otrzymujemy wiersze z tabeli wirtualnej VT1-J2 wraz z wierszami zachowanej tabeli lub tabel, dla których nie odnaleziono dopasowania w kroku 1-J2. Te dodane wiersze są nazywane *wierszami zewnętrznymi*. Atrybutom niezachowanej tabeli w wierszach zewnętrznych przypisane zostają znaczniki NULL. W efekcie wygenerowana zostaje tabela wirtualna VT1-J3.

W naszym przykładzie tabela Customers została oznaczona jako zachowana:

```
Customers AS C LEFT OUTER JOIN Orders AS O
```

Tylko dla wiersza klienta FISSA nie odnaleziono żadnego dopasowania w tabeli Orders i dlatego nie został on umieszczony w tabeli wirtualnej VT1-J2. W związku z tym wiersz odpowiadający klientowi FISSA zostaje dodany do zawartości tabeli VT1-J2 ze znacznikami NULL przypisanymi atrybutom z tabeli Orders. Efekt stanowi tabela wirtualna VT1-J3 (zilustrowana przy użyciu Tabeli 1-4). Ponieważ klauzula FROM przykładowego zapytania nie zawiera więcej operatorów tabel, tabela wirtualna VT1-J3 stanowi równocześnie tabelę wirtualną VT1 zwracaną w fazie FROM.

Tabela 1-4 Tabela wirtualna VT1-J3 (a także VT1) otrzymana w kroku 1-J3

C.custid	C.city	O.orderid	O.custid
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
MRPHS	Zion	6	MRPHS
FISSA	Madrid	NULL	NULL

Uwaga Jeśli klauzula FROM zawiera więcej niż jeden operator tabeli, proces przetwarzania logicznego rozpoczyna się od lewej strony. Wynik zastosowania pierwszego operatora tabeli zostaje przekazany w postaci lewej tabeli wejściowej do kolejnego operatora tabeli. Ostatnia otrzymana w ten sposób tabela wirtualna staje się tabelą wejściową kolejnego kroku.

Krok 2: Faza WHERE

Filtr WHERE zostaje zastosowany na wszystkich wierszach tabeli wirtualnej otrzymanej w poprzednim kroku. Te wiersze, dla których <*predykat_where*> ma wartość TRUE, wchodzą w skład zwracanej w tym kroku tabeli wirtualnej (VT2).

Ostrzeżenie W tym miejscu nie można odwoływaćsię do aliasów kolumn zdefiniowanych na liście SELECT, ponieważlista ta nie została jeszcze przetworzona. W związku z tym nie można napisać np. *SELECT YEAR(orderdate) AS orderyear ... WHERE orderyear > 2014*. To ograniczenie może z pozoru wydawać się usterką programu SQL Server, ale staje się jasne po zrozumieniu procesu logicznego przetwarzania zapytania. Co więcej, w tej fazie nie można również użyć agregacji, ponieważ dane nie zostały jeszcze pogrupowane – nie można napisać np. *WHERE orderdate = MAX(orderdate)*.

Zastosujemy teraz filtr z przykładowego zapytania:

```
WHERE C.city = 'Madrid'
```

Wiersz klienta o identyfikatorze MRPHS zostanie usunięty z tabeli wirtualnej VT1, ponieważ jego atrybut *city* nie ma wartości Madrid. W efekcie wygenerowana zostaje tabela wirtualna VT2 przedstawiona w tabeli 1-5.

Tabela 1-5 Tabela wirtualna VT2 otrzymana w kroku 2

C.custid	C.city	O.orderid	O.custid
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
FISSA	Madrid	NULL	NULL

Przy definiowaniu złączenia OUTER JOIN w zapytaniu często pojawia się wątpliwość, czy predykat powinien zostać umieszczony w klauzuli ON, czy też w klauzuli WHERE. Główna różnica między tymi dwoma rozwiązaniami polega na tym,

że klauzula ON zostaje zastosowana przed dodaniem wierszy zewnętrznych (krok 1-J3), natomiast klauzula WHERE po ich dodaniu. Przeprowadzona przez klauzulę ON eliminacja wiersza z zachowanej tabeli nie jest ostateczna, ponieważ zostanie on dodany ponownie w kroku 1-J3, natomiast eliminacja wiersza dokonana przez klauzulę WHERE jest ostateczna. Można w uproszczeniu stwierdzić, że predykat ON służy do dopasowywania wierszy, natomiast predykat WHERE do ich odfiltrowywania. Wiersze pochodzące z zachowanej tabeli w operacji złączania nie mogą zostać usunięte przez klauzulę ON, ponieważ służy ona jedynie do dopasowywania wierszy z niezachowanej tabeli. Natomiast klauzula WHERE może eliminować wiersze również z zachowanej tabeli. Zapamiętanie tej głównej różnicy może pomóc w dokonywaniu trafnego wyboru.

Załóżmy na przykład, że chcemy zwrócić informacje o wybranych klientach i ich zamówieniach z tabel Customers (Klienci) oraz Orders (Zamówienia). Chcemy zwrócić tylko dane klientów powiązanych z miastem Madrid – niezależnie od tego, czy złożyli oni jakiekolwiek zamówienia. Właśnie z myślą o tego typu sytuacjach zaprojektowana została operacja złączenia zewnętrznego. Zastosujemy lewe złączenie zewnętrzne (LEFT OUTER JOIN) między tabelą Customers, która zostanie oznaczona jako zachowana oraz tabelą Orders. Aby zwrócone zostały dane także tych klientów, którzy nie złożyli żadnych zamówień, korelacja między tabelami Customers oraz Orders musi zostać zdefiniowana w klauzuli ON (*ON C.custid = O.custid*). Wiersze klientów bez zamówień zostaną wyeliminowane w kroku 1-J2, ale dodane z powrotem w postaci wierszy zewnętrznych w kroku 1-J3. Jednocześnie, ponieważ interesują nas jedynie klienci z miasta Madrid, predykat określający miasto umieszczamy w klauzuli WHERE (*WHERE C.city = 'Madrid'*). Umieszczenie go w klauzuli ON spowodowałoby, że dane klientów z miast innych niż Madrid zostałyby z powrotem dodane do zbioru wynikowego w kroku 1-J3.

Wskazówka Ta różnica logiczna między klauzulami ON oraz WHERE występuje tylko wtedy, gdy używamy złączenia zewnętrznego. W przypadku zastosowania złączenia wewnętrznego miejsce zdefiniowania predykatu nie ma znaczenia, ponieważ krok 1-J3 zostaje pominięty. Predykaty zostają wykonane bezpośrednio po sobie i oba służą do filtrowania.

Krok 3: Faza GROUP BY

W fazie GROUP BY wiersze tabeli zwróconej w poprzednim kroku zostają powiązane z grupami zgodnie z zestawem lub zestawami grupowania zdefiniowanymi w klauzuli GROUP BY. Dla uproszczenia przyjmijmy założenie, że mamy do czynienia z jednym zestawem wyrażeń określających sposób grupowania danych. Zestaw ten nazywać będziemy *zestawem grupowania*.

W tej fazie wiersze zwróconej w poprzednim kroku tabeli zostają poukładane w grupy. Dla każdej unikatowej kombinacji wartości wyrażeń z zestawu grupowania utworzona zostaje osobna grupa. Każdy zwrócony wiersz zostaje powiązany z tylko jedną grupą. Tabela wirtualna VT3 składa się z wierszy tabeli VT2 poukładanych w grupy (nazywanych *surowymi* danymi) wraz z identyfikatorami grup (nazywanymi *informacjami o grupie*).

Wykonajmy teraz krok 3 dla przykładowego zapytania:

```
GROUP BY C.custid
```

Otrzymamy tabelę wirtualną VT3 przedstawioną w tabeli 1-6.

Tabela 1-6 Tabela wirtualna VT3 otrzymana w kroku 3

Informacje o grupach	Surowe dane			
C.custid	C.custid	C.city	O.orderid	O.custid
FRNDO	FRNDO	Madrid	1	FRNDO
	FRNDO	Madrid	2	FRNDO
KRLOS	KRLOS	Madrid	3	KRLOS
	KRLOS	Madrid	4	KRLOS
	KRLOS	Madrid	5	KRLOS
FISSA	FISSA	Madrid	NULL	NULL

Ostatecznie zapytanie grupujące wygeneruje po jednym wierszu dla każdej grupy (o ile niektóre grupy nie zostaną odfiltrowane). W związku z tym, gdy zapytanie zawiera klauzulę GROUP BY, wszystkie kolejne kroki (HAVING, SELECT itd.) mogą zawierać tylko wyrażenia, które zwracają dla każdej z grup wartość skalarną (pojedynczą). Te wyrażenia mogą odwoływać się do kolumn lub wyrażeń z listy GROUP BY (jak *C.custid* w przedstawionym przykładowym zapytaniu) bądź do funkcji agregujących np. *COUNT(O.orderid)*.

Przyjrzyjmy się wirtualnej tabeli VT3 przedstawionej w postaci Tabeli 1-6 i zastanówmy się, co powinno zwrócić zapytanie dla grupy klienta FRNDO, gdyby klauzula SELECT miała postać *SELECT C.custid, O.orderid.* Grupa zawiera dwie różne wartości *orderid*, a zatem zwracane dane nie miałyby charakteru skalarnego. System SQL nie zezwala na budowanie tego typu zapytań Jednak gdybyśmy zastosowali wyrażenie *SELECT C.custid, COUNT(O.orderid) AS numorders,* odpowiedź dla klienta FRNDO byłaby skalarna i byłaby nią liczba 2.

W tej fazie znaczniki NULL są traktowane jak równe sobie, w związku z tym wszystkie znaczniki NULL zostają umieszczone w tej samej grupie, podobnie jak znane wartości.

Krok 4: Faza HAVING

Filtr HAVING zostaje zastosowany na grupach w tabeli zwróconej w poprzednim kroku. W generowanej w tym kroku tabeli wirtualnej (VT4) umieszczone zostają tylko te grupy, dla których *<predykat_having>* przyjmuje wartość TRUE. Filtr HAVING stanowi jedyny filtr stosowany na pogrupowanych danych.

Wykonajmy ten krok dla przykładowego zapytania:

```
HAVING COUNT(O.orderid) < 3
```

Grupa związana z identyfikatorem klienta KRLOS zostaje usunięta, ponieważ zawiera trzy zamówienia. Wygenerowana zostaje tabela wirtualna VT4, przedstawiona w tabeli 1-7.

Tabela 1-7 Tabela wirtualna VT4 otrzymana w kroku 4

C.custid	C.custid	C.city	O.orderid	O.custid
FRNDO	FRNDO	Madrid	1	FRNDO
	FRNDO	Madrid	2	FRNDO
FISSA	FISSA	Madrid	NULL	NULL

Uwaga W tym przypadku musieliśmy użyć wyrażenia *COUNT(O.orderid)* zamiast *COUNT(*)*. Ponieważ zastosowaliśmy złączenie zewnętrzne, dodane zostały wiersze zewnętrzne reprezentujące klientów bez zamówień. Wyrażenie *COUNT(*)* skutkowałoby zliczeniem również tych wierszy zewnętrznych i wygenerowaniem niepożądanej wartości 1 dla klienta o identyfikatorze FISSA. Wyrażenie *COUNT(O.orderid)* prawidłowo zlicza liczbę zamówień dla każdego z klientów i skutkuje wygenerowaniem pożądanej wartości 0 dla klienta o identyfikatorze FISSA. Warto pamiętać, że wyrażenie *COUNT(<wyrażenie>)* podobnie jak inne funkcje agregujące ignoruje znaczniki NULL.

Uwaga Parametrem wejściowym funkcji agregującej nie może być zapytanie podrzędne np. *HAVING SUM((SELECT ...)) > 10*.

Krok 5: Faza SELECT

Mimo iż klauzula SELECT znajduje się na samym początku zapytania, zostaje przetworzona dopiero w piątym kroku. W fazie SELECT konstruowana jest tabela, która finalnie zostanie zwrócona do procesu wywołującego. Faza ta składa się z dwóch faz podrzędnych: (5-1) Wyznaczenie wartości wyrażeń oraz (5-2) Zastosowanie klauzuli DISTINCT.

Krok 5-1: Wyznaczenie wartości wyrażeń

Wyrażenia znajdujące się na liście SELECT mogą zwracać kolumny lub wynik przetwarzania kolumn tabeli wirtualnej, która została wygenerowana w poprzednim kroku. Należy pamiętać, że w zapytaniu agregującym po kroku 3 można odwoływać się do kolumn z poprzedniego kroku tylko wtedy, gdy zostały one umieszczone w zbiorze informacji o grupie (na liście GROUP BY). Do kolumn należących do surowych danych można odwołać się tylko wtedy, gdy zostały one zagregowane. Kolumny pobrane z tabeli wygenerowanej w poprzednim kroku zachowują oryginalne nazwy, o ile nie użyto aliasu (np. *col1 AS c1*). Wyrażenia przetwarzające kolumny powinny zostać opatrzone aliasem decydującym o tym, jaka nazwa kolumny zostanie zastosowana w tabeli wynikowej np. *YEAR(orderdate) AS orderyear*.

Ważne Jak wspomniałem wcześniej, aliasów utworzonych przez listę SELECT nie można wykorzystywać we wcześniejszych krokach np. w fazie WHERE. Powody stają się całkiem oczywiste po zrozumieniu kolejności logicznego przetwarzania klauzul zapytania. Co więcej, aliasy wyrażeń nie mogą być stosowane nawet w innych wyrażeniach z tej samej listy SELECT. To ograniczenie wynika z innej charakterystycznej cechy języka SQL, a mianowicie równoczesnego wykonywania wielu operacji. Na przykład, kolejność logicznego przetwarzania wyrażeń z następującej listy SELECT nie powinna mieć znaczenia i nie jest gwarantowana: *SELECT c1 + 1 AS e1, c2 + 1 AS e2*. W związku z tym nie można użyć następującego wyrażenia: *SELECT c1 + 1 AS e1, e1 + 1 AS e2*. Aliasy kolumn mogą być stosowane wyłącznie w krokach realizowanych po kroku, w którym zostały one utworzone. Jeśli zdefiniujemy alias kolumny w fazie SELECT, możemy odwołać się do niego w fazie ORDER BY np. *SELECT YEAR(orderdate) AS orderyear ... ORDER BY orderyear*.

Zrozumienie koncepcji równoczesnego wykonania operacji może być trudne. W większości języków programowania do zamiany wartości między zmiennymi wykorzystywana jest zmienna tymczasowa. Natomiast do zamiany wartości kolumn w języku SQL wystarczy poniższe zapytanie:

```
UPDATE dbo.T1 SET c1 = c2, c2 = c1;
```

Z logicznego punktu widzenia należy przyjąć założenie, że cała operacja odbywa się w jednej chwili, tak jakby tabela źródłowa nie była modyfikowana do czasu zakończenia całej operacji, kiedy to wynik zastępuje dane źródłowe. Analogiczny mechanizm powoduje, że następujące zapytanie UPDATE służy do modyfikacji wszystkich wierszy tabeli T1 w taki sposób, że do wartości kolumny c1 dodana zostaje maksymalna wartość kolumny c1 tabeli T1 z momentu rozpoczęcia procesu modyfikacji:

```
UPDATE dbo.T1 SET c1 = c1 + (SELECT MAX(c1) FROM dbo.T1);
```

Nie trzeba się obawiać, że maksymalna wartość c1 ulegnie zmianie w czasie wykonywania operacji. Nie ma takiego ryzyka, ponieważ cała operacja zostaje wykonana w jednej chwili.

Zrealizujmy ten krok dla przykładowego zapytania:

```
SELECT C.custid, COUNT(O.orderid) AS numorders
```

Otrzymamy tabelę wirtualną VT5-1 przedstawioną w tabeli 1-8. Ponieważ przykładowe zapytanie nie wymaga wykonania dodatkowej fazy podrzędnej (DISTINCT) fazy SELECT, tabela wirtualna VT5-1 wygenerowana w tej fazie podrzędnej stanowi jednocześnie tabelę wirtualną VT5 zwróconą z fazy SELECT.

Tabela 1-8 Tabela wirtualna VT5-1 (a także VT5) otrzymana w kroku 5

C.custid	numorders
FRNDO	2
FISSA	0

Krok 5-2: Zastosowanie klauzuli DISTINCT

Jeśli w zapytaniu określona została klauzula DISTINCT, z wygenerowanej w poprzednim kroku tabeli wirtualnej usunięte zostają duplikaty wierszy i w ten sposób utworzona zostaje tabela wirtualna VT5-2.

Uwaga Język SQL, w odróżnieniu od modelu relacyjnego, zezwala na występowanie powtarzających się wierszy w tabeli (o ile nie zdefiniowano klucza głównego bądź ograniczenia UNIQUE) lub zbiorze wyników zapytania. Natomiast w modelu relacyjnym treścią relacji jest zbiór *krotek* (w języku SQL nazywanych *wierszami*), a zgodnie z teorią mnogości *zbiór* (w odróżnieniu od *wielozbioru*) nie może zawierać dwóch (lub więcej) identycznych elementów. Zastosowanie klauzuli DISTINCT gwarantuje, że zapytanie zwróci jedynie unikatowe wiersze i pod tym względem spełni założenia modelu relacyjnego.

Warto mieć świadomość, że faza SELECT rozpoczyna się od wyznaczenia wartości wyrażeń (faza podrzędna 5-1) i dopiero później usuwa zduplikowane wiersze (faza podrzędna 5-2). Brak tej wiedzy utrudnia przewidywanie, jaki wynik zwrócą pewnego rodzaju zapytania. Na przykład poniższe zapytanie służy do pobrania unikatowych identyfikatorów klientów, którzy złożyli przynajmniej jedno zamówienie:

```
SELECT DISTINCT custid
FROM dbo.Orders
WHERE custid IS NOT NULL;
```

Jak można się spodziewać, zapytanie zwróci trzy wiersze:

```
custid
------
FRNDO
KRLOS
MRPHS
```

A teraz załóżmy, że chcemy dodać do wynikowych wierszy numerację bazującą na kolejności wartości w kolumnie *custid*. Nie zdając sobie sprawy z problemu, dodajemy do zapytania wyrażenie bazujące na funkcji ROW_NUMBER:

```
SELECT DISTINCT custid, ROW_NUMBER() OVER(ORDER BY custid) AS rownum
FROM dbo.Orders
WHERE custid IS NOT NULL;
```

Zachęcam czytelników, aby przed uruchomieniem zapytania spróbowali zgadnąć, ile wierszy zawierać będzie zbiór wyników. Z pozoru mogłoby się wydawać, że prawidłowa odpowiedź to 3, jednak w praktyce jest inaczej. Faza 5-1 wyznacza wartość wszystkich wyrażeń, także tego bazującego na funkcji ROW_NUMBER, jeszcze przed usunięciem duplikatów. W tym przykładzie w fazie SELECT przetworzona zostaje tabela VT4 zawierająca sześć wierszy (po zastosowaniu filtra WHERE). Natomiast w fazie 5-1 wygenerowana zostaje tabela VT5-1 z sześcioma wierszami o numerach od 1 do 6. Dodanie numerów wierszy sprawia, że wszystkie wiersze są unikatowe. W związku z tym w fazie 5-2 nie zostają znalezione żadne duplikaty do usunięcia. Oto wynik wykonania tego zapytania:

```
custid rownum
------ --------------------
FRNDO  1
FRNDO  2
KRLOS  3
KRLOS  4
KRLOS  5
MRPHS  6
```

Aby zastosować numerację na wierszach o unikatowym identyfikatorze klienta, trzeba pozbyć się duplikatów przed przypisaniem numeru wiersza. Cel ten można osiągnąć przy pomocy wyrażenia tabeli, takiego jak wspólne wyrażenie tabeli (Common Table Expression – CTE), na przykład:

```
WITH C AS
(
  SELECT DISTINCT custid
  FROM dbo.Orders
  WHERE custid IS NOT NULL
)
SELECT custid, ROW_NUMBER() OVER(ORDER BY custid) AS rownum
FROM C;
```

Tym razem uzyskujemy pożądany efekt:

```
custid rownum
------ --------------------
FRNDO  1
KRLOS  2
MRPHS  3
```

W procesie przetwarzania pierwotnego przykładowego zapytania (przedstawionego na listingu 1-2) krok 5-2 został pominięty ze względu na brak klauzuli DISTINCT. Nawet gdyby została ona zastosowana, w tym konkretnym przykładzie i tak nie spowodowałaby usunięcia żadnych wierszy.

Krok 6: Faza ORDER BY

Wiersze otrzymane w poprzednim kroku zostają posortowane zgodnie z listą kolumn określoną w klauzuli ORDER BY, co skutkuje zwróceniem kursora VC6. Jest to jedyna faza, w której można skorzystać z aliasów kolumn utworzonych w fazie SELECT.

Jeśli zdefiniowana została klauzula DISTINCT, wyrażenia w klauzuli ORDER BY mają dostęp jedynie do tabeli wirtualnej wygenerowanej w fazie SELECT (VT5). W przeciwnym przypadku wyrażenia w klauzuli ORDER BY mogą uzyskać dostęp zarówno do wyjściowej, jak i wejściowej tabeli wirtualnej fazy SELECT (VT4 oraz VT5). Co więcej, brak klauzuli DISTINCT oznacza, że można umieścić w klauzuli ORDER BY dowolne wyrażenie, które mogłoby wystąpić w klauzuli SELECT. Innymi słowy, można przeprowadzać sortowanie według wyrażeń, które nie zostają zwrócone w finalnym zbiorze wyników.

Nie bez powodu dostęp do wyrażeń, które nie są zwracane, nie jest dozwolony w przypadku określenia klauzuli DISTINCT. W przypadku dodania wyrażeń do listy SELECT klauzula DISTINCT może potencjalnie zmienić liczbę zwracanych wierszy. Natomiast brak klauzuli DISTINCT sprawia, że lista SELECT nie wpływa na liczbę zwracanych wierszy.

W naszym przykładzie klauzula DISTINCT nie została zastosowana, w związku z tym klauzula ORDER BY ma dostęp zarówno do tabeli VT4 (przedstawionej w tabeli 1-7), jak i tabeli VT5 (przedstawionej w tabeli 1-8).

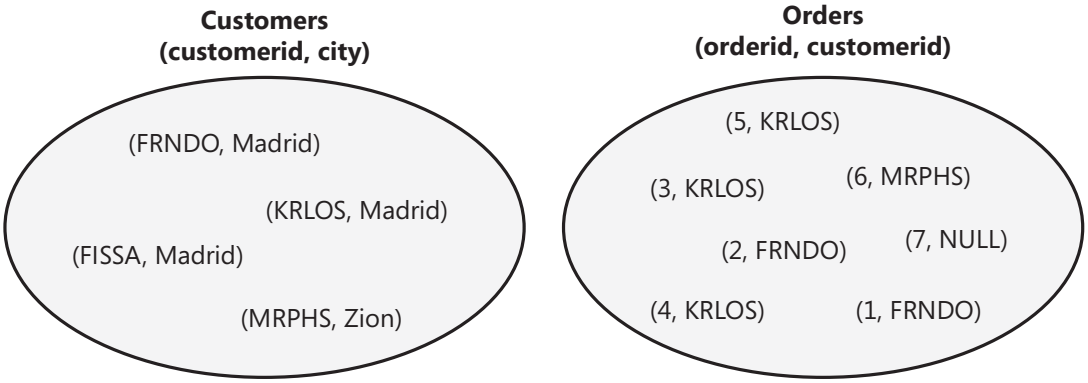
Ważne Ten krok różni się od pozostałych tym, że zamiast relacyjnego zbioru wyników zwrócony zostaje kursor. Należy pamiętać, że model relacyjny bazuje na teorii zbiorów, a treścią relacji (nazywanej w języku SQL *tabelą*) jest zbiór krotek (wierszy). Kolejność elementów zbioru nie jest określona. Natomiast wynik zapytania z klauzulą prezentacji ORDER BY zawiera wiersze uporządkowane w określonej kolejności. W standardzie języka SQL tego typu wynik nazywany jest *kursorem*. Zrozumienie tej różnicy stanowi klucz do właściwego zrozumienia języka SQL.

W klauzuli ORDER BY można również użyć numeru porządkowego kolumny wynikowej na liście SELECT. Na przykład, następujące zapytanie sortuje wiersze tabeli Orders według kolumny *custid* i dodatkowo *orderid*:

```
SELECT orderid, custid FROM dbo.Orders ORDER BY 2, 1;
```

Jednak praktyka ta nie jest zalecana ze względu na ryzyko, że osoba modyfikująca listę SELECT zapomni o odpowiednim dostosowaniu listy ORDER BY. Ponadto długi ciąg zapytania utrudnia sprawdzanie, który element na liście ORDER BY odpowiada któremu elementowi na liście SELECT.

Opisując zawartość tabeli, zwykle przedstawiamy listę jej wierszy. Jednak taki sposób prezentacji może być mylący, ponieważ sygnalizuje istnienie pewnego porządku, podczas gdy tabela stanowi zbiór wierszy o nieokreślonej kolejności. Rysunek 1-2 demonstruje przykład bardziej prawidłowego sposobu prezentowania zawartości tabel bez sugerowania kolejności.



Rysunek 1-2 Zbiory Customers oraz Orders.

Uwaga Język SQL nie określa kolejności *wierszy* tabeli, aczkolwiek definiuje pozycje porządkowe *kolumn* wynikające z kolejności ich utworzenia. Użycie wyrażenia *SELECT ** (niezalecanego ze względów, które zostaną wyjaśnione w dalszej części książki) sprawia, że kolumny zostają zwrócone w kolejności ich utworzenia. Pod tym względem standard SQL odbiega od modelu relacyjnego.

Ponieważ w tym kroku zamiast tabeli zostaje zwrócony kursor, zapytanie, które zawiera klauzulę ORDER BY służącą wyłącznie do określania porządku prezentacji, nie może zostać wykorzystane do definiowania wyrażenia tzn. widoku, wbudowanej funkcji zwracającej tabelę, tabeli pochodnej lub wspólnego wyrażenia tabeli. Wynik zapytania musi zostać przekazany do procesu wywołującego, który wspiera przetwarzanie kolejnych rekordów po jednym na raz. Tego typu procesem może być aplikacja kliencka lub kod SQL wykorzystujący obiekt CURSOR. Gdy spróbujemy użyć kursora w procesie, który spodziewa się relacyjnych danych wejściowych, wykonanie zakończy się niepowodzeniem. Na przykład poniższa próba zdefiniowania tabeli pochodnej skutkować będzie wygenerowaniem błędu:

```
SELECT orderid, custid
FROM ( SELECT orderid, custid
       FROM dbo.Orders
       ORDER BY orderid DESC  ) AS D;
```

Nieprawidłowy jest również poniższy kod SQL stanowiący próbę zdefiniowania widoku:

```
CREATE VIEW dbo.MyOrders
AS

SELECT orderid, custid
FROM dbo.Orders
ORDER BY orderid DESC;
GO
```

W języku T-SQL wyjątek od reguły, która zabrania definiowania wyrażeń tabeli bazujących na zapytaniu z klauzulą ORDER BY, stanowią zapytania z filtrem TOP lub OFFSET-FETCH. Omówieniem tego wyjątku zajmę się po przedstawieniu wspomnianych filtrów.

Klauzula ORDER BY traktuje znaczniki NULL jak równe sobie, co oznacza, że zajmują one tę samą pozycję w porządku sortowania. Standard SQL pozostawia swobodę w decydowaniu, czy mechanizm sortowania umieszcza znaczniki NULL po lub przed znanymi wartościami, ale działanie to musi być spójne. W języku T-SQL znaczniki NULL są uznawane za mniejsze od znanych wartości (czyli poprzedzają je).

Wykonajmy ten krok dla przykładowego zapytania z listingu 1-2:

```
ORDER BY numorders
```

Otrzymamy kursor VC6 zaprezentowany w tabeli 1-9.

Tabela 1-9 Kursor VC6 otrzymany w kroku 6

C.custid	numorders
FISSA	0
FRNDO	2

Krok 7: Zastosowanie filtra TOP lub OFFSET-FETCH

TOP oraz OFFSET-FETCH stanowią filtry zapytania oparte na liczbie wierszy i kolejności. To odróżnia je od bardziej tradycyjnych filtrów zapytania (ON, WHERE oraz HAVING), które bazują na predykacie. Filtr TOP jest charakterystyczny dla języka T-SQL, natomiast filtr OFFSET-FETCH został zdefiniowany w standardzie. Jak wspomniano wcześniej, filtr OFFSET-FETCH został wprowadzony w wersji SQL Server 2012. W tej części rozdziału filtry te zostaną omówione w kontekście logicznego przetwarzania zapytań

Specyfikacja filtra TOP składa się z dwóch elementów. Pierwszy z nich (wymagany) to liczba lub procent zwracanych wierszy (zaokrąglany w górę). Drugi z nich (opcjonalny) definiuje kolejność decydującą o tym, które wiersze zostaną zwrócone. Niestety filtr TOP (a także OFFSET-FETCH) bazuje na kolejności prezentacji określonej przy

pomocy klauzuli ORDER BY zapytania, zamiast na niezależnej definicji kolejności. Za chwilę wyjaśnię, dlaczego takie rozwiązanie projektowe przysparza problemów i prowadzi do nieporozumień Jeśli porządek nie został zdefiniowany, należy przyjąć założenie, że jest on przypadkowy, co prowadzi do uzyskiwania niedeterministycznego wyniku. W związku z tym, uruchamiając zapytanie ponownie, nie mamy gwarancji otrzymania tego samego wyniku, nawet jeśli dane nie zostały zmodyfikowane w międzyczasie. W tym kroku wygenerowana zostaje tabela wirtualna VT7. Jeśli zdefiniowany został porządek, wygenerowany zostaje kursor VC7. Na przykład następujące zapytanie zwraca trzy zamówienia o najwyższych wartościach *orderid*:

```
SELECT TOP (3) orderid, custid
FROM dbo.Orders
ORDER BY orderid DESC;
```

Zapytanie spowoduje wygenerowanie poniższego wyniku:

```
orderid     custid
----------- ------
7           NULL
6           MRPHS
5           KRLOS
```

Nawet zastosowanie klauzuli ORDER BY nie gwarantuje determinizmu. Jeśli sortowane wartości nie są unikatowe (np. gdybyśmy w poprzednim zapytaniu zastąpili atrybut *orderid* atrybutem *custid*), kolejność wierszy z taką samą wartością sortowania powinna być traktowana jako przypadkowa. Istnieją dwie techniki zapewniania deterministycznego działania filtra TOP. Pierwsza z nich polega na zastosowaniu listy ORDER BY gwarantującej unikatowość (np. *custid, orderid*). Natomiast druga polega na dodaniu do nieunikatowej listy ORDER BY opcji WITH TIES. Ta opcja sprawia, że filtr dołącza wszystkie wiersze z wygenerowanego zbioru wyników zapytania, w których wartości sortowania są takie same jak w ostatnim wierszu zwracanym bez tej opcji. Użycie tej opcji sprawia, że wybór zwracanych wierszy jest deterministyczny, w odróżnieniu od kolejności ich prezentowania, która nadal może się zmieniać w przypadku istnienia powtarzających się wartości sortowania.

Filtr OFFSET-FETCH przypomina filtr TOP, lecz zawiera dodatkową możliwość określenia, ile wierszy ma zostać pominiętych (OFFSET), poprzedzającą opcję definiującą liczbę wierszy do pobrania (FETCH). Klauzule OFFSET oraz FETCH muszą zostać umieszczone po klauzuli ORDER BY. Na przykład, następujące zapytanie definiuje sortowanie w kolejności malejącej według atrybutu *orderid*, pomija cztery pierwsze wiersze i zwraca dwa kolejne wiersze:

```
SELECT orderid, custid
FROM dbo.Orders
ORDER BY orderid DESC
OFFSET 4 ROWS FETCH NEXT 2 ROWS ONLY;
```

Powyższe zapytanie zwróci następujący wynik:

```
orderid     custid
----------- ------
3           KRLOS
2           FRNDO
```

W wersji SQL Server 2014 implementacja filtra OFFSET-FETCH w języku T-SQL nie wspiera jeszcze opcji PERCENT oraz WITH TIES, mimo iż zostały one udokumentowane w standardzie języka SQL. W tym kroku generowany jest kursor VC7.

Krok 7 został w naszym przykładzie pominięty, ponieważ kwerenda nie zawiera filtra TOP ani OFFSET-FETCH.

Jak wspomniałem wcześniej, do definiowania porządku filtra TOP oraz OFFSET-FETCH służy ta sama klauzula ORDER BY, która zwykle jest wykorzystywana do określania porządku prezentacji, co może prowadzić do nieporozumień. Przeanalizujmy następujące przykładowe zapytanie:

```
SELECT TOP (3) orderid, custid
FROM dbo.Orders
ORDER BY orderid DESC;
```

W tym przypadku klauzula ORDER BY służy do dwóch różnych celów. Po pierwsze, do definiowania porządku prezentacji (czyli prezentowania zwracanych wierszy w kolejności malejącej według atrybutu *orderid*). Po drugie, do określenia, które trzy wiersze mają zostać wybrane przez filtr TOP (trzy wiersze o najwyższej wartości *orderid*). Problem pojawia się, gdy chcemy zdefiniować wyrażenie tabeli bazujące na tego typu zapytaniu. Jak wspomniałem w części zatytułowanej „Krok 6: Faza ORDER BY", zazwyczaj nie można definiować wyrażenia tabeli bazującego na zapytaniu z klauzulą ORDER BY, ponieważ dla tego typu zapytania generowany jest kursor. Pokazałem, że próby zdefiniowania tabeli pochodnej lub widoku zakończą się niepowodzeniem, gdy wewnętrzne zapytanie zawiera klauzulę ORDER BY.

Istnieje wyjątek od tej reguły, ale, aby uniknąć przyszłych rozczarowań, trzeba go dobrze zrozumieć. Wyjątek zezwala na zastosowanie w wewnętrznym zapytaniu klauzuli ORDER BY, wspierającej filtr TOP lub OFFSET-FETCH. Należy pamiętać, że jeśli w takiej sytuacji zapytanie zewnętrzne nie zawiera klauzuli ORDER BY, kolejność prezentowania wierszy w zbiorze wyników nie jest gwarantowana. Zdefiniowana kolejność odnosi się jedynie do zapytania zawierającego klauzulę ORDER BY. Aby określić porządek prezentacji wierszy w zbiorze wyników, trzeba umieścić klauzulę ORDER BY w najbardziej zewnętrznym zapytaniu. Oto odnoszący się do tego faktu cytat z dokumentacji standardu ISO/IEC 9075-2:2011(E), sekcja 4.15.3 zatytułowana Derived Tables:

> *<Wyrażenie zapytania> może zawierać opcjonalną <klauzulę order by>. Kolejność wierszy tabeli określona przez <wyrażenie zapytania> jest gwarantowana tylko dla <wyrażenia zapytania>, które bezpośrednio zawiera <klauzulę order by>.*