Transact-SQL. Czarna księga

Spis Treści

Kurs Transact-SQL, część 1 (Interfejsy języka T-SQL)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Transact-SQL jest zmodyfikowaną i uzupełnioną o elementy typowe dla proceduralnych języków programowania (jak zmienne i instrukcje sterujące wykonaniem programu) wersją języka SQL (matematycznie zupełnego strukturalnego języka zapytań) ustandaryzowanego przez Amerykański Narodowy Instytut Standardów (ANSI) i Międzynarodową Organizację Standardów (ISO). Wszystkie programy klienckie, z wyjątkiem wykorzystujących technologie XML programów uruchamianych w przeglądarkach internetowych, komunikują się z bazami danych poprzez instrukcje tego języka.

Za pomocą Transact-SQL-a możemy:

- Wyszukiwać dane w bazie danych.
- Manipulować danymi — wstawiać, modyfikować i usuwać dane z baz danych.
- Definiować strukturę danych — dodawać, modyfikować oraz usuwać tabele, widoki, procedury, funkcje i indeksy.
- Zarządzać dostępem do danych — tworzyć i usuwać konta użytkowników i grup oraz nadawać uprawnienia do wykonania określonych operacji i do określonych obiektów wybranym użytkownikom i rolom.
- Zautomatyzować wykonywanie typowych zadań administracyjnych — tworzyć kopie zapasowe baz, importować i eksportować dane, czy porządkować indeksy.

Język SQL jest językiem strukturalnym, czyli opisującym wyniki, a nie procedury (metody) ich uzyskania. Programista określa, co ma być zrobione, a nie jak ma być zrobione. To serwer bazodanowy tworzy plan wykonania instrukcji SQL, wykonuje go i zwraca wyniki.

SQL Query Analyzer

Podstawowym, instalowanym razem z SQL Serwerem 2000, graficznym interfejsem języka Transact-SQL jest program SQL Query Analyzer. Program ten uważany jest za jeden z najlepszych graficznych interfejsów języka SQL. Wyposażony jest on w następujące funkcje i możliwości:

- Rozbudowany edytor tekstu umożliwiający m.in. wyróżnianie fragmentów tekstu i komentowanie poszczególnych wierszy i całych bloków tekstu.
- Wyróżnianie za pomocą kolorów czcionek słów kluczowych języka Transact-SQL.
- Możliwość jednoczesnego nawiązania wielu połączeń z bazą danych i wyświetlenia (w postaci graficznej lub tekstowej) wyniku instrukcji w osobnych okienkach.

- Możliwość wykonania jedynie zaznaczonych instrukcji wchodzących w skład skryptu.
- Wyświetlanie dodatkowych informacji prezentujących statystyki obciążenia serwera lub klienta związane z wykonaniem instrukcji Transact-SQL.
- Możliwość wyświetlenia planu wykonania instrukcji.
- Obsługa szablonów (ang. *Templates*) ułatwiających tworzenie obiektów bazodanowych.
- Wbudowana przeglądarka obiektów (ang. *Object Browser*) ułatwiająca m.in. kopiowanie obiektów.
- Analizowanie poprawności syntaktycznej instrukcji języka Transact-SQL.
- Możliwość szybkiego przeglądania i modyfikowania zawartości tabel.
- Możliwość tworzenia własnych skrótów klawiszowych do najczęściej wykonywanych instrukcji.
- Możliwość wykonywania skryptów z wiersza polecenia.

Uruchamianie programu

Program Query Analyzer może zostać uruchomiony na co najmniej trzy sposoby:

- Poprzez wybranie z menu Start.
- Z konsoli Enterprise Managera poprzez wybranie menu Tools/SQL Server Query Analyzer.
- Po wpisaniu w wierszu polecenia isqlw.

Po uruchomieniu programu należy podać informacje umożliwiające połączenie się z wybranym serwerem bazodanowym. W przypadku uruchamiania programu z wiersza polecenia można wykorzystać opcje –S nazwa serwera\nazwa instancji oraz –U użytkownik –P hasło w przypadku konta SQL Servera lub –E w przypadku uwierzytelniania na podstawie konta systemu Windows. Na przykład, aby połączyć się z nazwaną instancją (MSSQL) uruchomioną na komputerze o nazwie *LOLEK* na podstawie konta systemu Windows aktualnie zalogowanego użytkownika, napiszemy:

```
isqlw –S lolek\mssql –E
```

natomiast aby połączyć się z tą samą instancją jako użytkownik sa, napiszemy:

```
isqlw –S lolek\mssql –U sa –P hasło
```

Jeżeli wybrany użytkownik posiadał wystarczające uprawnienia, zostanie nawiązane połączenie z domyślną dla danego użytkownika bazą oraz zostanie wyświetlone główne okno programu.

Praca z programem

Głównym okienkiem programu jest okienko edytora kodu. Pisząc instrukcje języka Transact-SQL możemy skorzystać ze standardowych narzędzi edytora tekstu takich jak *Paste (Wklej)* czy *Replace (Zamień)*.

Aby oznaczyć jeden wiersz (lub jego część) jako komentarz, należy poprzedzić go znakami --. Komentarz obejmujący wiele wierszy rozpoczynamy znakami /*, a kończymy znakami */. Aby wywołać pomoc kontekstową dotyczącą zaznaczonego słowa kluczowego języka Transact-SQL, naciśnij kombinację klawiszy Shift+F1.

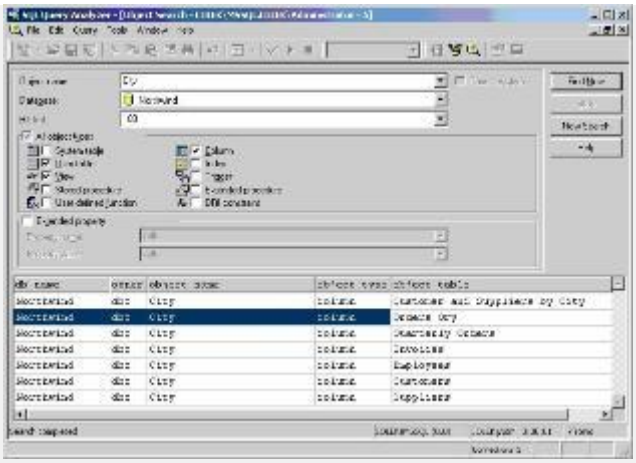
Po prawej stronie na pasku narzędzi programu Query Analyzer znajdują się ikony:

- *Show Result Pane (Pokaż okno wyniku)* — wynik instrukcji może zostać wyświetlony w postaci graficznej (Query/Results in Grid), tekstowej (Query/ Results in Text) lub może być zapisany w pliku (Query/Results to File…).
- *Current Connections Properties (Właściwości bieżącego połączenia)* — właściwości połączenia mogą zostać ustawione za pomocą zaznaczenia odpowiednich pól w oknie dialogowym wywołanym kliknięciem tej ikony lub ustawione za pomocą odpowiadającej poszczególnym opcjom instrukcji *SET*, na przykład wpisanie *SET NOCOUNT ON* spowoduje zaznaczenie pozycji Set nocount. Poszczególne opcje oznaczają:
  - *Set nocount* — wyłączenie wyświetlania informacji o liczbie zwracanych wierszy.
  - *Set noexec* — instrukcja zostanie skompilowana, ale nie wykonana. W rezultacie zostanie sprawdzona nie tylko poprawność syntaktyczna, lecz także semantyczna (np. poprawność wymienionych nazw obiektów).
  - *Set parseonly* — zostanie jedynie sprawdzona poprawność syntaktyczna instrukcji. Taki sam efekt można osiągnąć poprzez naciśnięcie kombinacji klawiszy Ctr+F5 zamiast klawisza F5.
  - *Set concat_null_yields_null* — opcja domyślnie włączona. Połączenie (konkatenacja) dowolnego ciągu z wartością NULL da w wyniku wartość NULL, a nie ciąg źródłowy.
  - *Set rowcount* — zwraca jedynie określoną liczbę wierszy. Domyślna wartość 0 oznacza zwrócenie wszystkich wierszy wyniku.
  - *Set arithabort* — opcja domyślnie włączona. Jeżeli podczas wykonywania instrukcji wystąpi błąd arytmetyczny (np. dzielenie przez zero), jej wykonanie zostanie przerwane.
  - *Set showplan_text* — wyświetlenie jedynie planu wykonania instrukcji bez jego wykonania przez SQL Server.
  - *Set statistics time* — instrukcja zostanie wykonana, a dodatkowo zostaną zwrócone informacje o liczbie milisekund potrzebnych systemowi bazodanowemu na analizę, sprawdzenie poprawności i wykonanie instrukcji.
  - *Set statistics IO* — dodatkowo zostaną wyświetlone informacje o liczbie i czasie operacji wejścia-wyjścia przeprowadzonych przez SQL Server w celu wykonania instrukcji.
  - Grupa opcji *Set ansi_defaults* wymusza zgodność ze standardem SQL-92:
    - *Set ansi_nulls* — opcja domyślnie włączona. Wynikiem jakichkolwiek porównań z wartością NULL będzie wartość nieokreślona. Ponieważ wartość NULL reprezentuje wartość nieokreśloną lub nieznaną, standard SQL-92 uniemożliwia sprawdzenie, czy jakakolwiek wartość jest równa wartości NULL (np. @adres = NULL), pozwala jedynie stwierdzić, czy wartość nie jest wartością nieokreśloną (@adres is NULL). Po wyłączeniu opcji Set ansi_nulls wynikiem porównania kilku wartości nieokreślonych będzie TRUE (prawda).
    - *Set ansi_null_dflt_on* — opcja domyślnie włączona. Nadpisuje domyślne ustawienia dla nowych kolumn, umożliwiając wstawianie w nich wartości nieokreślonej.
    - *Set ansi_padding* — opcja domyślnie włączona. Wyłącza automatyczne usuwanie wiodących zer z liczb i spacji z tekstów.
    - *Set ansi_warnings* — opcja domyślnie włączona. Wykonanie instrukcji, które naruszają zasady standardu SQL-92, ale są zgodne z językiem Transact-SQL, spowoduje wyświetlenie komunikatu ostrzeżenia.
    - *Set cursor_close_on_commit* — automatyczne zamykanie kursora w momencie zatwierdzania transakcji.

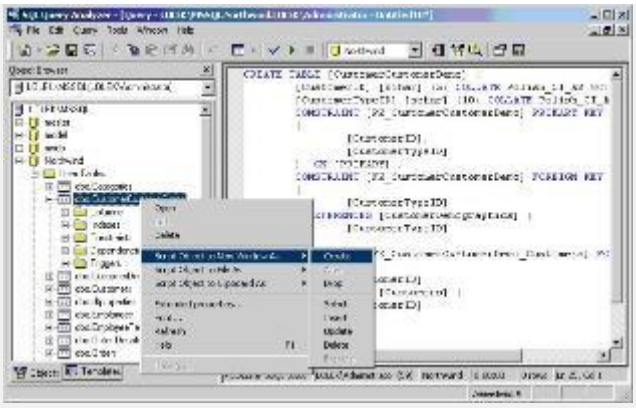
- *Set implicit_transactions* — rozpoczęcie nowej transakcji następuje automatycznie, bez konieczności wpisywania instrukcji *BEGIN TRANSACTION*.
- *Set quoted_identifier* — opcja domyślnie włączona. Identyfikatory obiektów mogą być umieszczane w cudzysłowach, a literały muszą być umieszczane w apostrofach. Po wyłączeniu identyfikatory obiektów nie mogą być umieszczane w cudzysłowach, co oznacza, że muszą spełniać wymogi konwencji nazewniczej SQL Servera.

> W wersji 2000 SQL Servera, inaczej niż w wersjach wcześniejszych, opcja quoted_ identifier jest domyślnie włączona. Jej wyłączenie spowoduje, że niektóre nowe typy obiektów, np. indeksy tworzone dla widoków, nie będą dalej dostępne.

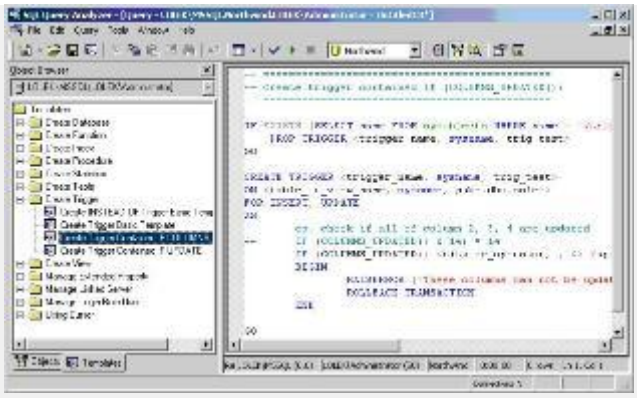
- *Object Search (Wyszukiwanie obiektów)* — po naciśnięciu klawisza F4 lub po wybraniu odpowiedniej ikony zostanie wyświetlone okno pokazane na rysunku 1.1.



- *Object Browser (Przeglądarka obiektów)* — po naciśnięciu klawisza F8 lub po wybraniu odpowiedniej ikony wyświetlone zostanie okno przeglądarki obiektów. Object Browser umożliwia:
  - Wywoływanie skryptów języka Transact-SQL związanych z danym obiektem. Lista dostępnych instrukcji zależy od typu obiektu, np. po kliknięciu prawym przyciskiem myszy nazwy tabeli i wybraniu z menu kontekstowego jednej z opcji Script Object to… będziemy mogli utworzyć następujące instrukcje:
    - tworzącą lub usuwającą wybraną tabelę,
    - odczytującą lub modyfikującą dane w tej tabeli (rysunek 1.2).



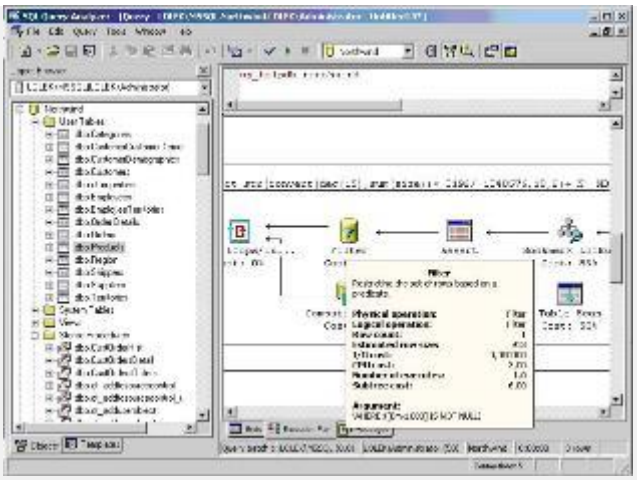
- Wykonywanie procedur składowanych, w tym procedur sparametryzowanych (przeglądarka obiektów umożliwi podanie wartości parametrów wywołania procedury).
- Wyświetlanie zawartości tabeli z możliwością modyfikacji zapisanych w niej danych.
- Zmianę definicji obiektów.
- Pracę z szablonami. Szablony są wzorami instrukcji języka Transact-SQL, za pomocą których możemy:
    - tworzyć obiekty bazodanowe (bazy danych, tabele, widoki, indeksy, procedury, wyzwalacza, statystyki i funkcje),
    - zarządzać obiektami (połączonymi serwerami, użytkownikami, rolami czy rozszerzonymi właściwościami obiektów),
    - deklarować, wykorzystywać i usuwać kursory,
    - modyfikować skrypty.
- Wykonanie skryptów wymaga zastąpienia parametrów zapisanych w nawiasach rzeczywistymi wartościami (rysunek 1.3).



Szablony są zapisane w postaci plików .tql umieszczonych w podfolderach folderu /ProgramFiles/Microsoft SQL server/80/Tools/Templates/SQL Query Analyzer, gdzie 80 oznacza numer wersji SQL Servera. Tworząc własne szablony powinniśmy zapisywać je jako pliki .tgl i umieszczać je w podfolderach wspomnianego folderu.

- *Display Estimated Execution Plan (Wyświetl obliczony plan wykonania)* — po naciśnięciu kombinacji klawiszy Ctr+L lub wybraniu odpowiedniej ikony oprócz wyników wykonania instrukcji i komunikatów serwera możemy wyświetlić plan

wykonania danej instrukcji przygotowany przez SQL Server (rysunek 1.4). Opcja ta, razem z opcjami Show Server Trace i Show Client Statistic, zostanie dokładnie opisana w części poświeconym monitorowaniu i optymalizacji wydajności SQL Servera.



- *Cancel (Przerwij), Execute (Wykonaj)* i *Parse Query (Sprawdź poprawność)* — wybranie ikony spowoduje odpowiednio przerwanie wykonywania, wykonanie lub sprawdzenie poprawności syntaktycznej instrukcji.
- *Execute Mode (Tryb wykonania)* — kliknięcie znajdującej się po prawej stronie ikony strzałki skierowanej w dół pozwoli na określenie typu i formatu wyświetlanych wyników wykonania instrukcji.

Pozostałe ikony są typowymi ikonami występującymi w wielu aplikacjach działających w systemie Microsoft Windows. Na uwagę zasługuje ikona *Clear Query Windows*, której kliknięcie (lub naciśnięcie kombinacji klawiszy Ctrl+Shift+Del) powoduje wyczyszczenie aktywnego okienka edytora kodu.

Poniżej opisano najczęściej używane skróty klawiszowe programu Query Analyzer:

- *F5* — wykonanie bieżącej instrukcji języka Transact-SQL.
- *F8* — włącza i wyłącza okienko przeglądarki obiektów.
- *F4* — wyświetla okienko wyszukiwania obiektów. Możliwe jest jednoczesne wyświetlenie kilku okien i przeszukiwanie zawartości serwera według kilku różnych kryteriów.
- *Alt+F1* — wykonuje procedurę systemową sp_help. Jeżeli nie został zaznaczony jakikolwiek fragment tekstu, naciśnięcie tej kombinacji klawiszy spowoduje wykonanie procedury sp_help dla bieżącej bazy danych. W innym przypadku wyświetlona zostanie pomoc dotycząca zaznaczonego obiektu lub słowa kluczowego języka Transact-SQL.
- *Ctr+1* — wykonuje procedurę systemową sp_who. Jeżeli nie została zaznaczona nazwa jakiegokolwiek użytkownika, wyświetlone zostaną informacje związane z bieżącą bazą danych.
- *Ctr+2* — wykonuje procedurę systemową sp_lock. Ten skrót klawiszowy nie pozwala na przekazanie parametrów do procedury.

Aby utworzyć własny skrót w programie Query Analyzer, wybierz Tools/Customize… i na zakładce Custom przypisz nazwę procedury składowanej do wybranej kombinacji klawiszy.

OSQL

OSQL jest interfejsem wiersza polecenia, który umożliwia tworzenie i wykonywanie instrukcji języka Transact-SQL oraz wykonywanie przygotowanych skryptów administracyjnych.

Program OSQL do połączenia z SQL Serverem wykorzystuje technologię ODBC.

Uruchamianie programu

Aby uruchomić program, należy w wierszu polecenia wpisać osql. Najczęściej używane parametry wywołania zostały przedstawione w tabeli 1.1.

Tabela 1.1

Parametr	Opis
-S	Nazwa serwera. Jeżeli chcemy połączyć się z nazwaną instancją SQL Servera 2000, po nazwie serwera musimy podać nazwę instancji oddzielonej znakiem \
-E	Określa, że do połączenia będziemy używać nazwy i hasła użytkownika systemu Windows
-U	Nazwa użytkownika (login) SQL Servera
-P	Hasło użytkownika. Niepodanie tego parametru spowoduje wyświetlenie pytania o hasło użytkownika SQL Servera
-i, <	Nazwa pliku z instrukcjami do wykonania.
-o, >	Nazwa pliku, do którego zostaną przekierowane wyniki wykonania instrukcji

Jeżeli komunikacja z SQL Serverem została poprawnie nawiązana i użytkownik posiada odpowiednie uprawnienia, zostaje uruchomiona sesja (lub wykonany skrypt) języka Transact-SQL. Aby zakończyć pracę z programem, należy wpisać exit i nacisnąć Enter.

Praca z programem

Program OSQL umożliwia, oprócz wykonywania wszystkich instrukcji języka Transact-SQL, wykonanie kilku poleceń dodatkowych:

Dyrektywa GO wykonuje wszystkie instrukcje wpisane od momentu wykonania ostatniej takiej dyrektywy. Dodatkowo możemy określić, ile razy dane polecenie ma być wykonane. Na przykład wykonanie skryptu:

```
USE pubs
GO
SELECT DISTINCT state
FROM authors
GO 2
```

spowoduje połączenie się z bazą *Pubs* i dwukrotne wykonanie zapytania zwracającego identyfikatory stanów przechowywane w tabeli *Authors*.

Polecenie RESET usuwa wszystkie wpisane do tej pory instrukcje z bufora programu.

Polecenie ED wywoła domyślny edytor tekst programu OSQL (znany użytkownikom pamiętającym systemu DOS program Edit). Z poziomu programu możemy również wywołać dowolny program lub skrypt powłoki systemu Windows. W tym celu należy

poprzedzić nazwę programu dwoma znakami wykrzyknika. Na przykład, aby uruchomić kalkulator, wpiszemy !!calc.

W programie OSQL możemy, za pomocą tych samych symboli co w programie Query Analyzer, komentować pojedyncze wiersze lub całe bloki instrukcji Transact-SQL.

Program OSQL umożliwia również przekazanie do systemu operacyjnego pojedynczej wartości zwróconej w wyniku wykonania instrukcji Transact-SQL. W systemach Windows NT/2000 wartość pierwszej komórki zwróconej w wyniku zapytania może zostać przekazana jako 4-bajtowa liczba całkowita. Na przykład, aby przekazać do systemu liczbę wierszy zwróconych przez ostatnie zapytanie (np. w celu sprawdzenia, czy zostały zwrócone jakiekolwiek wiersze), wpiszemy: `EXIT (SELECT @@ROWCOUNT)`

BCP

BCP (ang. *Bulk Copy Program*) jest narzędziem wiersza polecenia służącym do kopiowania danych pomiędzy SQL Serverem a plikiem tekstowym o określonym formacie. Kopiowanie tego typu nazywa się kopiowaniem masowym i jest najszybszym sposobem na zaimportowanie lub wyeksportowanie dużej liczby danych z bazy działającej w systemie SQL Server.

Program BCP do połączenia z SQL Serverem 2000 wykorzystuje technologię ODBC.

Uruchomienie programu

Aby uruchomić program, należy w wierszu polecenia wpisać bcp i podać wymagane parametry. Pełna lista parametrów programu wygląda następująco:

```
bcp [[nazwa_bazy_danych.][właściciel_obiektu].]nazwa_tabeli | nazwa_widoku} | "zapytanie"
   in | out | queryout | format nazwa_pliku_danych
   [-m liczba_błędów] [-f format_pliku] [-e plik_dziennika_błędów]
   [-F pierwszy_wiersz] [-L ostatni_wiersz] [-b wielkość_pojedynczej_transakcji]
   [-n] [-c] [-w] [-N] [-V (60 | 65 | 70)] [-6]
   [-q] [-C strona_kodowa] [-t znacznik_końca_pola] [-r znacznik_końca_wiersza]
   [-i nazwa_pliku_wejściowego] [-o nazwa_pliku_wyjściowego]
   [-a wielkość_pojedynczego_pakietu_sieciowego]
   [-S nazwa_serwera [\nazwa_instancji]] [-U nazwa_użytkownika] [-P hasło]
   [-T] [-v] [-R] [-k] [-E] [-h "wskazówki [,...n]"]
```

Praca z programem

Funkcjonalnym odpowiednikiem uruchomienia programu BCP jest wykonanie instrukcji języka Transact-SQL `BULK INSERT`. W obu przypadkach nastąpi masowe kopiowanie danych z pliku lub do pliku. Różnica polega na tym, że w pierwszym wypadku wykonanie polecenia nastąpi po stronie klienta, który wywołał program BCP, a w drugim — po stronie serwera. Na przykład, aby skopiować do pliku zawartość tabeli *Customers*, napiszemy:

```
BCP "Northwind.dbo.Customers" out "Klienci.txt" -c -S "lolek\mssql" -U"sa" -P"hasło"
```

TEXTCOPY

W folderze /ProgramFiles/Microsoft SQL Server/Nazwa serwera$Nazwa instancji/Binn znajduje się mało znane narzędzie wiersza polecenia Textcopy, za pomocą którego można kopiować dane tekstowe i pliki graficzne pomiędzy SQL Serverem a plikiem o określonym formacie.

Uruchomienie programu

Aby uruchomić program Textcopy, należy dodać do zmiennej systemowej Path wpis wskazujący na lokalizację pliku programu. Lista parametrów wywołania programu została przedstawiona poniżej:

```
TEXTCOPY [/S [nazwa_serwera]] [/U [użytkownik]] [/P [hasło]]
 [/D [baza_danych]] [/T tabela] [/C kolumna] [/W"warunek_użyty_w_klauzuli_WHERE"]
 [/F nazwa_pliku] [{/I | /O}] [/K wielkość_bufora] [/Z] [/?]
```

gdzie:

/I oznacza, że dane będą kopiowane z pliku do bazy danych,

/O oznacza, że dane będą kopiowane z bazy danych do pliku,

/Z wyświetla informacje diagnostyczne w trakcie wykonywania polecenia.

Praca z programem

Wykorzystując systemową procedurę *xp_cmdshell*, której działanie polega na wykonaniu polecenia systemu operacyjnego, możemy zautomatyzować pracę z programem Textcopy. W tym celu należy stworzyć własną procedurę przechowywaną, która wywoła z określonymi parametrami program Textcopy.

> Tworzenie własnych procedur oraz wywoływanie procedur systemowych zostanie przedstawione w części 12 kursu.

Poniższa procedura kopiuj wywołuje (poprzez procedurę systemową *xp_cmdshell*) program Textcopy:

```
CREATE PROCEDURE kopiuj (
@db      varchar(50),
@tb      varchar(50),
@kol     varchar(50),
@plik    varchar(50),
@where   varchar(50),
@io      char(1))
AS
DECLARE @instr varchar (255)
SELECT @instr =
 'textcopy /S lolek\mssql /U sa /P hasło /D' + @db +
 ' /T ' + @tb +
```

```
 ' /C ' + @kol +
 ' /F ' + @plik +
 ' /W "' + @where +
 '" /' + @io
EXEC master.dbo.xp_cmdshell @instr
```

Tworząc ciąg instr, należy zastąpić przykładowe wartości argumentów S, U i P właściwymi nazwami serwera, użytkownika oraz prawidłowym hasłem.

Po utworzeniu procedury *kopiuj* kopiowanie np. plików graficznych z bazy danych lub do bazy danych sprowadza się do wywołania jej z pięcioma niezbędnymi parametrami:

```
EXEC kopiuj @db = 'Northwind',
 @tb = 'Employees',
 @kol = 'Photo',
 @where = " WHERE EmployeeID = 1",
 @plik = 'c:\foto.jpg',
 @io = 'O'
```

Kurs Transact-SQL, część 2 (Leksykon języka T-SQL)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Język Transact-SQL jest zgodny wstecz ze standardem SQL-92 (SQL-2), co oznacza, że wszystkie poprawne instrukcje standardowego języka SQL będą właściwie wykonane przez SQL Server. Standard SQL-2 wciąż jest uznawany za zbiór podstawowych zasad, których każdy system bazodanowy musi przestrzegać. Taka interpretacja standardu spowodowała, że implementacje języka SQL w systemach zarządzania bazami danych różnych producentów, w tym systemach tworzonych w ramach otwartych projektów internetowych, różnią się od siebie nie tylko dodatkowymi instrukcjami czy funkcjami, ale także konwencjami nazewniczymi czy specyficzną obsługą typów danych, które tylko z nazwy odpowiadają typom zaimplementowanym w produktach konkurencji.

Standardy SQL i historia ich powstania

Będący logiczną podstawą języka SQL model relacyjny baz danych został formalnie zdefiniowany w 1970 r. przez dr. E. F. Codda, naukowca firmy IBM, w dokumencie zatytułowanym Relacyjny model danych dla dużych współdzielonych banków danych (ang. A Relational Model of Data for Large Shared Data Banks). Firma IBM była wtedy ogromną korporacją prowadzącą badania nad projektami, których praktycznego zastosowania spodziewano doczekać się dopiero po kilkunastu latach.

W 1974 r. IBM rozpoczął nadzorowany przez Donalda Chamberlina projekt System/R, w ramach którego został opracowany język SEQUEL (ang. Structured English Query Language). System/R został zaimplementowany w programie nazwanym SEQUEL-XRM. Dominowały wtedy systemy bazodanowe oparte na hierarchicznym lub sieciowym modelu baz danych, a dokonana przez IBM implementacja modelu relacyjnego była wielokrotnie wolniejsza od systemów konkurencji.

IBM nie był jedyną firmą prowadzącą w tym czasie badania nad modelem relacyjnym — równolegle powstawał opracowywany przez naukowców Uniwersytetu Kalifornijskiego w Berkeley projekt Ingres, a w 1979 roku pojawił się pierwszy system zarządzania bazami danych wyprodukowany przez mało wtedy znaną firmę Relational Software — Oracle. Sam IBM przygotował dwa komercyjne systemy bazodanowe, oba będące rozwinięciem projektu System/R — SQL/DS (wprowadzony w 1981 r.) oraz DB2 (rok 1983). I chociaż działający na platformie MVS (komputerach mainframe obsługujących technologię Multiple Virtual Storage) system DB2 zyskał sobie opinię wymagającego wręcz astronomicznych zasobów sprzętowych, a ponadto najwolniejszego systemu bazodanowego, to został on uznany przez firmę IBM za produkt strategiczny. W latach osiemdziesiątych określenie przez monopolistę — IBM — jakiegoś produktu jako strategicznego oznaczało, że cały świat będzie kupował i instalował to oprogramowanie.

Pozostałe firmy, zachęcone wyraźnym znakiem poparcia modelu relacyjnego przez potentata, zaczęły rozwijać własne implementacje modelu. I tak w ciągu kilkunastu lat doczekaliśmy się co najmniej pięciu głównych implementacji języka SQL, które choć oparte na tym samym modelu logicznym, znacznie się od siebie różnią.

SQL oparty jest na algebrze relacji i rachunku relacyjnym krotki. Rozwinięta przez doktora Codda algebra relacji dostarczyła reguły przetwarzania składni instrukcji języka (odpowiada na pytanie „Jak wykonać instrukcje?"). Natomiast rachunek relacyjny krotki (ang. Tuple Relational Calculus — TRC), opisując deklaracje języka, odpowiedział na pytanie „Co należy zrobić?").

Oprócz właściwości wynikających z algebry lub rachunku relacyjnego, język SQL zawiera funkcje dodatkowe, takie jak:

- obsługa wstawiania, modyfikacji i usuwania danych,
- operatory arytmetyczne,
- wyświetlanie danych,
- przypisanie (aliasy),
- funkcje grupujące.

Pierwszą próba uporządkowania i ujednolicenia implementacji języka SQL w poszczególnych systemach bazodanowych był standard opublikowany w roku 1986 przez Amerykański Narodowy Instytut Standardów (ang. American National Standards Institute — ANSI). Kolejną — opublikowany rok później przez Międzynarodową Organizację Standardów dokument normalizacyjny. Na podstawie tych dokumentów w 1989 r. opublikowany został zweryfikowany standard, znany powszechnie jako SQL-89 lub SQL-1.

Niestety, głównie ze względu na sprzeczne interesy producentów systemów bazodanowych, standard ten nie określał wielu podstawowych cech języka, a wiele właściwości zdefiniowano jako zależne od implementacji. Kolejną próbą ujednolicenia była ratyfikacja (w 1992 r.) standardu SQL-92 (SQL-2). Niezbędnym kompromisem okazało się wprowadzenie trzech poziomów zgodności z nowym standardem:

1. Podstawowy poziom zgodności (ang. Entry-level conformance) był właściwie powtórzeniem standardu SQL-1 (właśnie ten poziom zgodności oferuje SQL Server).
2. Pośredni poziom zgodności (ang. Intermediate-level conformance) stanowił ogólnie osiągalny zbiór zasadniczych ujednoliceń języka.
3. Pełny poziom zgodności (ang. Full conformance) — oferował pełną zgodność z właściwościami standardu SQL-2.

W 1999 r. organizacje ANSI i ISO opracowały standard SQL-99 (nazywany również SQL-3). Dokument ten po raz pierwszy ujednolica zaawansowane funkcje i obszary zastosowań języka SQL, takie jak modele obiektowo-relacyjnych baz danych, interfejsy poziomu wywołania czy instrukcje umożliwiające zarządzanie spójnością danych. SQL-3 oferuje dwa poziomy zgodności:

1. Podstawowy poziom zgodności (ang. Core SQL-3).
2. Zaawansowany poziom zgodności (ang. Enhanced SQL-3).

Rozdział 2. zawiera opis najczęściej wykorzystywanych składników języka Transact-SQL zaimplementowanego w systemie SQL Server 2000.

Typy instrukcji języka Transact-SQL

Instrukcją nazywany jest w książce każdy poprawny syntaktycznie (składniowo) zbiór znaczników języka Transact-SQL. Instrukcje Transact-SQL zawsze zaczynają się poleceniem (znacznikiem lub grupą znaczników określających jaką operację wykona instrukcja). Instrukcje zwykle zawierają przynajmniej jedną klauzulę; klauzule są formalnymi modyfikatorami opisującymi bardziej szczegółowo funkcję instrukcji Transact-SQL.

W książce (ze względu na czytelność) przyjęto zasadę zapisywania poszczególnych klauzul w nowych wierszach. W rzeczywistości interpreter języka Transact-SQL ignoruje znak końca wiersza.

Zapytaniem jest instrukcja, która zwraca dane (na przykład instrukcja SELECT). Dane zwrócone przez zapytanie nazywane są wynikiem działania instrukcji.

SQL Server interpretuje instrukcje języka Transact-SQL jako sekwencję znaczników, zwykle rozdzielonych spacjami lub znakami nowego wiersza. Znacznik jest zastrzeżonym słowem lub znakiem języka. Może on być zidentyfikowany przez serwer bazodanowy w czasie przetwarzania lub interpretowania instrukcji Transact-SQL.

Znacznik może być słowem kluczowym, identyfikatorem, stałą, dyrektywą wsadową lub jednym z kilku symboli specjalnych:

- Słowa kluczowe to ciągi, które SQL Server rozpoznaje jako wyrazy o ściśle określonym przez standard SQL lub język Transact-SQL znaczeniu. Należą do nich:
  - instrukcje Transact-SQL,
  - klauzule,
  - nazwy funkcji systemowych,
  - terminy specjalne.
- Identyfikatory to nazwy tabel, kolumn i innych obiektów baz danych. Identyfikatorami mogą być wartości zmiennych.
- Stałe, w przeciwieństwie do słów kluczowych i identyfikatorów, opisują dane, które są interpretowane dosłownie, na przykład liczby lub ciągi znakowe.
- Dyrektywy wsadowe służą do sterowania wykonaniem wielu instrukcji języka Transact-SQL, z reguły zapisanych w skrypcie.
- Symbole specjalne są zastrzeżonymi znakami (takimi jak nawiasy czy średnik), które logicznie wpływają na znaczenie i rozmieszczenie słów kluczowych, identyfikatorów i literałów. Symbole specjalne pełnią rolę znaków interpunkcyjnych języka Transact-SQL.

Do symboli specjalnych zalicza się również operatory używane do oznaczania operacji lub wyliczeń logicznych pomiędzy wartościami danych (literałami lub wartościami reprezentowanymi przez identyfikatory).

Po ustaleniu znaczenia poszczególnych terminów możemy wśród instrukcji języka Transact- SQL wyróżnić trzy typy:

1. Instrukcje DDL (ang. Data Definition Language — język definiowania danych) tworzące i modyfikujące obiekty bazodanowe.
2. Instrukcje DML (ang. Data Manipulation Language — język modyfikowania danych), do których należą zapytania i instrukcje modyfikujące dane.
3. Instrukcje DCL (ang. Data Control Language — język sterowania danymi) umożliwiające nadawanie i odbieranie uprawnień do wykonywania instrukcji w bazie danych.

Instrukcje DDL

Do instrukcji DDL zalicza się polecenia:

CREATE typ_obiektu nazwa_obiektu

ALTER typ_obiektu nazwa_obiektu

DROP typ_obiektu nazwa_obiektu

Domyślne zasady bezpieczeństwa SQL Severa umożliwiają wykonywanie instrukcji DDL jedynie członkom ról serwera (sysadmin, dbcreator) oraz ról bazy danych (db_owner, db_ddladmin). Ponieważ użytkownik, który utworzył obiekt, a nie należał do grupy sysadmin, staje się właścicielem obiektu (obiekty utworzone przez administratorów stają się własnością specjalnego użytkownika dbo), nie należy, o ile jest to tylko możliwe, przypisywać do wymienionych ról kont użytkowników.

Instrukcje DML

Do instrukcji DML zalicza się polecenia:

SELECT

INSERT

UPDATE

DELETE

Uprawnienia do wykonywania poszczególnych instrukcji DML nadaje się rolom lub użytkownikom w odniesieniu do konkretnych obiektów bazy danych, na przykład administrator może nadać prawo wykonywania instrukcji SELECT użytkownikowi jkowalski do tabeli Region oraz zabronić temu samemu użytkownikowi wykonywania instrukcji UPDATE i INSERT względem tabeli Products.

Instrukcje DCL

Do instrukcji DCL zalicza się polecenia:

GRANT

DENY

REVOKE

Domyślne zasady bezpieczeństwa SQL Severa umożliwiają wykonywanie instrukcji DCL jedynie członkom ról serwera (sysadmin, dbcreator) oraz ról bazy danych (db_owner, db_securityadmin). Członkowie tych ról mogą nadawać oraz odbierać uprawnienia związane z wybranym obiektem lub instrukcją języka Transact-SQL.

Znaczniki języka

Znajomość poszczególnych znaczników języka jest warunkiem koniecznym poznania jego możliwości i pisania efektywnych programów w tym języku. Znaczniki języka Transact-SQL można podzielić na dziesięć grup.

Dyrektywy wsadowe

Ponieważ SQL Server ma możliwość przetwarzania wsadowego skryptów języka Transact- SQL, niezbędne okazało się dodanie dyrektyw sterujących wykonaniem skryptów.

GO

Dyrektywa GO powoduje wysłanie do SQL Servera i wykonanie wszystkich zapisanych od momentu ostatniego wystąpienia dyrektywy instrukcji języka Transact-SQL.

- Dyrektywa GO nie może występować w tym samym wierszu co jakakolwiek instrukcja języka Transact-SQL. W wierszu z dyrektywą GO można umieszczać jedynie komentarze.
- Zakres zmiennych użytkownika ograniczony jest do bloku skryptu zapisanego pomiędzy poszczególnymi dyrektywami GO. Wynika z tego, że pierwszy skrypt zostanie wykonany bez błędu, a próba wykonania skryptu drugiego z poniższych skryptów zakończy się błędem.

Przykład prawidłowo zadeklarowanej zmiennej komunikat:

```
USE Northwind
DECLARE @komunikat VARCHAR(50)
SELECT @komunikat = 'Witaj, świecie.'
GO
The command(s) completed successfully.
```

Próba odwołania się do zmiennej lokalnej spoza jej zakresu:

```
USE Northwind
DECLARE @komunikat VARCHAR(50)
GO
SELECT @komunikat = 'Witaj, świecie.'
GO
```

- Niektóre instrukcje (dotyczy to zwłaszcza instrukcji DDL) muszą być oddzielone od pozostałych instrukcji. Konieczne jest m.in. oddzielanie instrukcji: CREATE PROCEDURE, CREATE VIEW, CREATE TRIGGER, CREATE RULE AS oraz CREATE DEFAULT.
- Dyrektywa GO jest niezgodna ze standardem SQL-2 i nigdy nie jest wysyłana do serwera bazodanowego. Jedynie programy Query Analyzer, OSQL i ISQL (program instalowany z wersją 2000 SQL Servera w celu zachowania zgodności z wcześniejszymi wersjami) prawidłowo interpretują tę dyrektywę.

Dyrektywa GO musi być komentowana za pomocą znaków --, nawet w obrębie bloku komentarza. W innym wypadku kompilator zgłosi błędy: Server: Msg 113, Level 15, State 1, Line 1 Missing end comment mark '*/'. oraz Server: Msg 170, Level 15, State 1, Line 1 Line 1: Incorrect syntax near '*'.

EXEC

EXEC Dyrektywa EXEC (EXECUTE) wykorzystywana jest do wywołania procedur składowanych (systemowych i użytkownika) oraz funkcji użytkownika. Umożliwia przekazanie do wywoływanego obiektu parametrów i odczytanie wyniku ich działania. Dyrektywa EXEC pozwala również na wykonanie ciągu znaków będącego instrukcją języka Transact-SQL (wykonywanie tworzonych dynamicznie instrukcji Transact-SQL).
Składnia:

```
[[EXEC [UTE]]
```

```
  {
    [@zwrócony_stan_wykonania =]
      {nazwa_procedury [;liczba] | @nazwa_procedury
  }
    [[@parametr =] {wartość | @zmienna [OUTPUT] | [DEFAULT]]
    […n]
[WITH RECOMPILE]
```

gdzie:

- ♣ zwrócony_stan_wykonania jest opcjonalną zmienną lokalną przechowującą informacje o stanie wykonania wywoływanej procedury lub funkcji.
- ♣ nazwa_procedury jest nazwą wywoływanej procedury lub funkcji.
- ♣ @nazwa_procedury jest nazwą zmiennej lokalnej przechowującej nazwę wywoływanego obiektu.
- ♣ @parametr jest nazwą zdefiniowanego podczas tworzenia procedury lub funkcji parametru jej wywołania i musi być poprzedzony znakiem @.
- ♣ wartość jest wartością przekazywanego parametru.
- ♣ @zmienna jest nazwą zmiennej przechowującej wartość parametru wywołania.
- ♣ OUTPUT określa, że wywoływana procedura jest funkcją zwracającą jakąś wartość.
- ♣ DEFAULT przekazuje do procedury domyślne, określone podczas jej tworzenia wartości parametrów wywołania.
- ♣ WITH RECOMPILE wymusza ponowną kompilację kodu wywoływanej procedury lub funkcji, co wiąże się z obliczeniem nowego planu jej wykonania bazującego na aktualnych statystykach.

SQL Server umożliwia odwoływanie się do obiektów bazodanowych za pomocą w pełni kwalifikowanej nazwy obiektu bądź jej skrótu. Pełna nazwa składa się z nazwy serwera, nazwy bazy danych, nazwy właściciela obiektu i nazwy obiektu oddzielonych kropkami. Ponieważ optymalizator wykonania instrukcji Transact-SQL nie może obliczyć optymalnego planu ich wykonania dopóki „nie wie”, do jakiego obiektu (czy do tabeli dbo.tabela1 czy jkowlaski.tabela1) one się odwołują, przed każdym wykonaniem tych instrukcji będą one ponownie kompilowane. Stosując jednoznaczne odwołania w procedurach składowanych i funkcjach użytkownika skracamy czas ich wykonywania.

Składnia dyrektywy wywołującej ciąg znaków:

```
EXEC [UTE] ({@zmienna_tekstowa | [N] 'instrukcja_T-SQL'} [+ …n])
```

gdzie:

- ♣ zmienna_tekstowa jest nazwą lokalnej zmiennej tekstowej. Jeżeli długość ciągu znaków przekracza 4 000 znaków, należy połączyć (skonkatenować) kilka zmiennych tekstowych.
- ♣ instrukcja_T-SQL jest poprawną instrukcją języka Transact-SQL. Jeżeli długość instrukcji przekracza 4 000 znaków, należy połączyć (skonkatenować) kilka ciągów znaków składających się na tę instrukcję.

Przykład: (próba bezpośredniego użycia zmiennej w instrukcji INSERT zakończyłaby się błędem):

```
USE pubs
CREATE TABLE #t1 (id int)
DECLARE @sql varchar(60)
SET @sql = 'INSERT INTO #t1 (id)
VALUES (333)'
EXEC (@sql)
(1 row(s) affected)
```

Komentarze

Tak jak programy pisane w proceduralnych językach programowania, programy języka Transact-SQL zyskują na czytelności poprzez dodanie do kodu komentarzy opisujących role poszczególnych zmiennych czy funkcję bardziej skomplikowanych fragmentów programu. Komentarze są ignorowanymi przez kompilator ciągami znaków wyróżnionymi na jeden z dwóch sposobów:

- Za pomocą dwóch znaków myślnika (— —). Kompilator ignoruje znajdujące się po prawej stronie myślników znaki.
- Za pomocą znaków /* */. Kompilator ignoruje wszystkie znaki (z wyjątkiem dyrektywy GO) znajdujące się pomiędzy wspomnianymi znacznikami.

Przykład:

```
USE Northwind
/*
Łączymy się z bazą Northwind
-- GO
Skrypt demonstrujący stosowanie komentarzy
*/
SELECT (UnitPrice * 2), ProductName -- zafałszowane ceny produktów
FROM dbo.Products
WHERE SupplierID = 1
-- AND ProductName = 'Chang'
GO
ProductName
---------- ----------------------------------------
36.0000 Chai
38.0000 Chang
20.0000 Aniseed Syrup
(3 row(s) affected)
```

Identyfikatory

Identyfikatory określające obiekty i umożliwiające odwoływanie się do przechowywanych w nich danych lub do ich metod, zdarzeń czy właściwości muszą być

zgodne z konwencją nazewniczą SQL Servera. W przeciwnym wypadku muszą być wyróżniane za pomocą właściwych znaczników.

Konwencja nazewnicza SQL Servera określa, że:

- Identyfikatory mogą składać się z nie więcej niż 128 znaków.
- Pierwszym znakiem identyfikatora musi być litera alfabetu.
- Identyfikator nie może być słowem zastrzeżonym języka Transact-SQL.
- Identyfikator może zawierać litery, cyfry oraz symbole: @, $, #, _.Wynika z tego, że identyfikatory nie mogą zawierać spacji ani pozostałych symboli specjalnych.
- Identyfikatory rozpoczynające się od symboli @ lub # mają specjalne znaczenie:
  - Identyfikator rozpoczynający się symbolem @ oznacza lokalną zmienną.
  - Identyfikator rozpoczynający się symbolem # oznacza obiekt tymczasowy.
  - Identyfikator rozpoczynający się podwojonym symbolem ## oznacza globalny obiekt tymczasowy.

Ponieważ SQL Server nadaje obiektom tymczasowym liczbowy prefiks, ich identyfikatory nie mogą składać się z więcej niż 116 znaków.

Identyfikatory niezgodne z konwencją nazewniczą SQL Servera muszą być wyróżniane za pomocą nawiasów kwadratowych lub cudzysłowów. O ile nawiasy kwadratowe mogą zawsze zostać wykorzystane do wyróżniania identyfikatorów, o tyle użycie cudzysłowów możliwe jest tylko wtedy, gdy została ustawiona opcja *Set quoted_identifier*.

Przykład zastosowania identyfikatorów niezgodnych z konwencją nazewniczą:

```
USE Northwind
SELECT *
FROM [Order Details]
WHERE OrderID = 10262
GO
SET QUOTED_IDENTIFIER ON
GO
SELECT * FROM "Order Details"
WHERE OrderID = 10271
GO
OrderID ProductID UnitPrice Quantity Discount
----------- ----------- --------------------- -------- -----------------------
10262 5 17.0000 12 0.2
10262 7 24.0000 15 0.0
10262 56 30.4000 2 0.0
(3 row(s) affected)
OrderID ProductID UnitPrice Quantity Discount
----------- ----------- --------------------- -------- -----------------------
10271 33 2.0000 24 0.0
(1 row(s) affected)
```

- Twórz krótkie, ale mające znaczenie identyfikatory.
- Stwórz i konsekwentnie stosuj własną konwencję nazewniczą zgodną z konwencją SQL Servera, ale ułatwiającą Ci powiązanie identyfikatora ze wskazywanym przez niego obiektem.
- W swojej konwencji nazewniczej stosuj przedrostki charakteryzujące typ obiektu, np. w_ dla widoków czy p_ dla procedur.
- Nie zapisuj własnych procedur z przedrostkiem sp_. Zagadnienie to zostało przedstawione w rozdziale 12.
- Unikaj tworzenia identyfikatorów odpowiadających nazwom użytkowników lub reguł.

Typy danych

Typy danych nie są znacznikami języka Transact-SQL, jednak ich znajomość jest niezbędna do wydajnego korzystania z innego rodzaju znaczników — zmiennych.

Typ danych określa jakiego rodzaju (typu) informacje mogą być przechowywane w poszczególnych kolumnach tabeli lub w zmiennych oraz jakiego typu dane mogą być przekazywane jako parametry wywołania procedury lub funkcji. SQL Server nie wymaga określania typu danych w ramach instrukcji Transact-SQL, jednak wyniki ich wykonania bezpośrednio zależą od zdefiniowanych typów danych.

Do najważniejszych typów danych SQL Servera należą:

- Typy liczbowe:
  - *bigint* reprezentuje liczby całkowite z zakresu od -2^{63} (–9 223 372 036 854 775 808) do $2^{63}-1$ (9 223 372 036 854 775 807),
  - *int* reprezentuje liczby całkowite z zakresu od -2^{31} (–2 147 483 648) do $2^{31}-1$ (2147483647),
  - *smallint* reprezentuje liczby całkowite z zakresu od -2^{15} (–32 768) do $2^{15}-1$ (32 767),
  - *tinyinit* reprezentuje liczby całkowite z zakresu od 0 do 255,
  - *decimal* reprezentuje liczby o określonej skali i precyzji z zakresu od $-10^{38}-1$ do $10^{38}-1$. Typ numeric jest synonimem typu decimal,
  - *float* reprezentuje liczby o zmiennej precyzji z zakresu od 1,79E+308 do 1,79E+308,
  - *real* reprezentuje liczby o zmiennej precyzji z zakresu od –3,40E+38 do 3,40E+38. Typy zmiennoprzecinkowe charakteryzują się tym, że wartości tak zapisanych danych przechowywane są z pewną skończoną dokładnością.

    Poniższy przykład obrazuje to zagadnienie:

```
USE pubs
CREATE TABLE #d1 (typ_real real,typ_float float)
INSERT INTO #d1 (typ_real, typ_float)
VALUES (3.9, 3.9)
GO
SELECT *, typ_real- typ_float AS różnica
FROM #d1
WHERE typ_real=3.9 AND typ_float=3.9
```

```
GO
typ_real                typ_float                    różnica
----------------------- ---------------------- ------------------------------
3.9000001               3.8999999999999999           9.5367431729442842E-8
(1 row(s) affected)

```

- *bit* reprezentuje liczby całkowite z przedziału od 0 do 1.

- Typy daty i czasu:
  - *datetime* reprezentujący datę i czas z zakresu od 1 stycznia 1753 do 31 grudnia 9999, z dokładnością do jednej milisekundy,
  - *smalldatetime* reprezentujący datę i czas z zakresu od 1 stycznia 1900 do 6 czerwca 2079, z dokładnością do jednej minuty.
- Typy znakowe:
  - *char* reprezentujący ciąg znaków o określonej, nie większej niż 8 000 znaków długości.
  - *nchar* reprezentujący ciąg znaków o określonej, nie większej niż 4 000 znaków, długości zapisanej w postaci Unicode.
  - *varchar* reprezentujący ciąg znaków o zmiennej, nie większej niż 8 000 znaków długości.
  - *nvarchar* reprezentujący ciąg znaków o zmiennej, nie większej niż 4 000 znaków, długości zapisanej w postaci Unicode.
  - *text* reprezentujący ciąg znaków o określonej, nie większej niż $2^{31}-1$ (2 147 483 647) długości.
  - *ntext* reprezentujący ciąg znaków o zmiennej, nie większej niż $2^{30}-1$ (1 073 741 823), długości zapisanej w postaci Unicode.
- Typy binarne. Typy binarne są bardzo podobne do typów znakowych, z tą różnicą, że dane zapisane są jako poszczególne bity:
  - *binary* reprezentujący ciąg bitów o określonej, nie większej niż 8 000 bajtów, długości.
  - *varbinary* reprezentujący ciąg bitów o zmiennej, nie większej niż 8 000 bajtów, długości.
  - *rowversion* w wersji 2000 SQL Servera jest synonimem typu timestamp. W przyszłych wersjach SQL Servera typ timestamp ma zostać uzgodniony ze standardem SQL-3.
  - *image* reprezentujący dane binarne, na przykład grafikę, o wielkości od 0 do $2^{31}-1$ (2 147 483 647).
- Identyfikatory:
  - *uniqueidentifier* reprezentuje globalny identyfikator (GUID), który jest 16 bajtową liczbą heksadecymalną.
- Wartości nieznane lub nieokreślone. Każdy system zarządzania bazami danych obsługuje specjalną wartość NULL, która reprezentuje dane brakujące lub o nieokreślonej wartości. Wartość NULL jest wartością specjalną, różną od wszystkich innych wartości, w tym od 0 i pustego ciągu znaków. Obsługa tej wartości zależy od ustawienia opisanych w rozdziale 1. wartości opcji ANSI_NULLS. Jednak pomimo tego, że wartość NULL jest wartością nieokreśloną, nałożenie na kolumnę dopuszczającą wstawianie wartości NULL ograniczenia UNIQUE spowoduje, że SQL Server nie pozwoli zapisać więcej niż jeden wartości NULL w tej kolumnie. Aby umożliwić zapisanie wielu wartości NULL w tej kolumnie, należy stworzyć własny wyzwalacz sprawdzający poprawność wprowadzanych danych, zamiast korzystać z ograniczenia UNIQUE (drugi z poniższych przykładów).

Do tak zdefiniowanej tabeli wpiszemy tylko jedną wartość NULL w kolumnie n:

```
USE tempdb
CREATE TABLE t1
(n int NULL UNIQUE,
opis varchar (10) NULL)
GO
The command(s) completed successfully.
INSERT INTO t1 (opis)
VALUES ('1 rekord')
GO
INSERT INTO t1 (opis)
VALUES ('2 rekord')
GO
(1 row(s) affected)
Server: Msg 2627, Level 14, State 2, Line 1
Violation of UNIQUE KEY constraint 'UQ__t1__7A9C383C'. Cannot insert duplicate key in
ªobject 't1'.
The statement has been terminated.
```

Tak zdefiniowany wyzwalacz wycofa wszystkie transakcje, w których wartość kolumny n nie będzie unikalna, z wyjątkiem tych transakcji, które wartość kolumny n ustawia na NULL:

```
USE tempdb
CREATE TABLE t2
(n int NULL,
opis varchar (10) NULL)
GO
CREATE TRIGGER n_n on t2
FOR INSERT, UPDATE as
BEGIN
IF (SELECT MAX(licz) FROM (
SELECT COUNT(i.n) AS licz
FROM t2,
INSERTED i WHERE t2.n = i.n
GROUP BY i.n)
x) > 1
ROLLBACK TRAN
END
GO
The command(s) completed successfully.
INSERT INTO t2 (opis)
```

```
VALUES ('1 rekord')
GO
INSERT INTO t2 (opis)
VALUES ('2 rekord')
GO
SELECT * FROM t2
GO
(1 row(s) affected)
(1 row(s) affected)
n opis
----------- ----------
NULL 1 rekord
NULL 2 rekord
(2 row(s) affected)
```

SQL Server pozwala na tworzenie własnych, bazujących na typach systemowych, typów danych. Rozwiązanie to stosuje się, jeżeli w kilku kolumnach (z reguły różnych tabel) muszą być przechowywane dane dokładnie tego samego typu i długości.

Konwersja typów

Kolejne wersje SQL Servera w odrębny sposób dokonują niejawnej konwersji typów. Możemy się o tym przekonać, wykonując kolejno w wersji 2000 i 7 poniższe zapytanie:

```
USE pubs
SELECT *
FROM authors
WHERE contract = -99999
GO
```

Ponieważ kolumna contract przechowuje dane typu bit (0 lub 1) SQL Server 2000 nie zwrócił żadnych wierszy. Natomiast SQL Server 7 zwrócił 19 wierszy — wszystkie te, w których wartość atrybutu contract wynosiła 1. Aby zobaczyć, jak SQL Server wykonał zapytanie, wyświetlmy plan wykonania zapytania przez SQL Server 2000 — jak widać SQL Server dokonuje poprawnej konwersji typu bit do typu integer użytego w klauzuli WHERE:

```
|--Clustered Index Scan(OBJECT:([pubs].[dbo].[authors].[UPKCL_auidind]),
ªWHERE:(Convert([authors].[contract])=[@1]))
```

Natomiast SQL Server 7 przekształca dane typu integer na dane typu bit, w wyniku czego wartość –99999 została zamieniona na 1 i wykonał błędną klauzulę WHERE contract = 1.

Zmienne

Zmienne charakteryzują się nazwą, typem i wartością. Dzięki zmiennym możemy:

- przechowywać wartości zwrócone przez funkcje w celu ich późniejszego wykorzystania (w języku SQL funkcjonalnym odpowiednikiem zmiennych są podzapytania),
- wielokrotnie wykorzystać ten sam kod do wykonywania operacji na różnych danych wejściowych.

Język Transact-SQL pozwala na definiowanie zmiennych lokalnych, przypisywanie im wartości początkowej (inicjowanie) i wykorzystywanie w ramach instrukcji, skryptu lub procedury.

Nazwy zmiennych lokalnych rozpoczynają się pojedynczym znakiem @. Zmienne, których nazwy rozpoczynają się dwoma znakami @@ pełnią rolę funkcji systemowych i zostały opisane w podrozdziale poświęconym funkcjom.

DECLARE

Wydanie polecenia spowoduje utworzenie zmiennej o określonej nazwie i typie. Wartość zadeklarowanej zmiennej ustawiana jest na NULL.
Składnia:

```
DECLARE
  {{@zmienna typ}
    | {@kursor CURSOR}
    | {typ_tabeli}
  } [,…n]

< typ_tabeli > ::=
  TABLE ({< definicja_kolumny > | < zawężenie >} [,…]
        )

<definicja_kolumny>::=
  nazwa_kolumny typ_skalarny
  [COLLATE strona_kodowa]
  [[DEFAULT wyrażenie] | IDENTITY [(korzeń, przyrost)]]
  [ROWGUIDCOL]
  [< zawężenie_kolumny >]

< zawężenie_kolumny > ::=
  {[NULL | NOT NULL]
  | [PRIMARY KEY | UNIQUE]
  | CHECK (warunek_logiczny)
  }

< zawężenie > ::=
  {{PRIMARY KEY | UNIQUE} (nazwa_kolumny [,…])
  | CHECK (warunek_wyszukiwania)
```

```
}
```

gdzie:
CURSOR określa, że zmienna będzie lokalna zmienna kursora.
DEFAULT spowoduje wstawienie do kolumny określonej wartości, jeżeli użytkownik nie określi danych dla tej kolumny.
IDENTITY spowoduje, że dana kolumna będzie przechowywała unikalne identyfikatory rekordów.
korzeń określa wartość pierwszego wiersza tabeli odczytywanego przez klauzulę *VERYFIRSTROW*.
przyrost jest liczbą, o jaką będzie zwiększana wartość kolumny *IDENTITY*.

> W książce opisywane będą definicje jedynie wybranych znaczników. Pełna dokumentacja dostępna jest w instalowanej wraz z SQL Serverem dokumentacji Books Online (BOL).

SET

Nadaje wartość zdefiniowanej zmiennej lokalnej.
Składnia:

```
SET {{@zmienna = wyrażenie}
    | {@zmienna_kurosora = {@zmienna_kursora | nazwa_kursora
      | {CURSOR [FORWARD_ONLY | SCROLL]
        [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
        [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
        [TYPE_WARNING]
    FOR instrukcja_select
      [FOR {READ ONLY | UPDATE [OF nazwa_kolumny [,…n]]}
      ]
    }
  }}
}
```

gdzie:
instrukcja_select jest poprawną instrukcją, której wynik będzie zbiorem rekordów dostępnych dla kursora.
nazwa_kolumny jest nazwą kolumny przechowującą dane modyfikowane przez kursor.
Przykłady:

```
USE Northwind
DECLARE @znajdz varchar(30)
SET @znajdz = 'Fra%'
SELECT CompanyName,Address, Phone
FROM dbo.Customers
WHERE CompanyName LIKE @znajdz
```

```
GO
CompanyName Address Phone
---------- ---------- ----------
France restauration 54, rue Royale 40.32.21.21
Franchi S.p.A. Via Monte Bianco 34 011-4988260
Frankenversand Berliner Platz 43 089-0877310
(3 row(s) affected)
USE Northwind
DECLARE @kursor CURSOR
SET @kursor = CURSOR SCROLL DYNAMIC
FOR
SELECT LastName, FirstName
FROM dbo.Employees
WHERE LastName like 'D%'
OPEN @kursor
FETCH NEXT FROM @kursor
WHILE @@FETCH_STATUS = 0
BEGIN
FETCH NEXT FROM @kursor
END
CLOSE @kursor
DEALLOCATE @kursor
GO
LastName FirstName
---------- ----------
Davolio Nancy
(1 row(s) affected)
LastName FirstName
---------- ----------
Dodsworth Anne
(1 row(s) affected)
LastName FirstName
---------- ----------
(0 row(s) affected)
```

Używanie zmiennych w klauzuli FROM

SQL Server nie pozwala na bezpośrednie odwoływanie się do zmiennej typu tabelowego w klauzuli FROM (zostanie wyświetlony błąd: *Server: Msg 137, Level 15, State 2, Line 8 Must declare the variable '@tabela'*), a więc np. do połączenia zmiennej reprezentującej tabelę z tabelą istniejącą w bazie danych. Istnieje jednak nieudokumentowany w BOL sposób na odwołanie się do zmiennej w tabeli FROM — wystarczy stworzyć alias dla zmiennej. Poniższy przykład pokazuje, jak dzięki temu, że

stworzyliśmy alias tabela dla zmiennej @tabela, umożliwiliśmy odwoływanie się do niej w klauzuli FROM:

```
USE Northwind
GO
DECLARE @tabela TABLE (id int)
INSERT INTO @tabela
SELECT CategoryID
FROM dbo.Categories
WHERE CategoryName LIKE 'C%'
SELECT Categories.CategoryID, Categories.Description
FROM dbo.Categories INNER JOIN @tabela AS tabela
ON dbo.Categories.CategoryID = tabela.id
GO
(2 row(s) affected)
CategoryID Description
------------------------------------------------------------------
2 Sweet and savory sauces, relishes, spreads, and seasonings
3 Desserts, candies, and sweet breads
(2 row(s) affected)
```

Funkcje

Predefiniowane funkcje systemowe można podzielić na trzy kategorie:

- Funkcje skalarne, które zwracają pojedynczą wartość obliczoną na podstawie zera lub większej liczby argumentów skalarnych.
- Funkcje grupujące, które zwracają pojedynczą wartość dla zbioru argumentów wywołania
- Funkcje typu RowSet, które mogą być wykorzystywane jak tabele zawierające zbiory danych.

SQL Server 2000, w przeciwieństwie do swoich poprzedników pozwala na tworzenie własnych funkcji użytkownika (UDF). Tworzenie funkcji użytkownika zostało opisane w rozdziale 14.

Wśród funkcji skalarnych wyróżniamy funkcje:

- bezpieczeństwa,
- ciągu znaków,
- daty i czasu,
- konfiguracyjne,
- kursora,
- matematyczne,
- metadanych,
- statystyki systemu,
- systemowe,

- tekstu i danych binarnych.

Kolejno zostaną przedstawione funkcje poszczególnych typów.

Istniejące w systemie SQL Server 2000, a nie wymienione poniżej funkcje zostały zachowane jedynie w celu zapewnienia zgodności z wcześniejszymi wersjami SQL Servera i nie powinny być używane.

Funkcje bezpieczeństwa

IS_MEMBER

Sprawdza, czy dane konto użytkownika jest członkiem grupy systemu Windows lub roli SQL Servera.

Składnia:

```
IS_MEMBER ({'grupa' | 'rola'})
```

Funkcja zwraca wartość:

- 0 — jeżeli użytkownik nie jest członkiem grupy lub roli,
- 1 — jeżeli jest,
- NULL — jeżeli parametry wywołania funkcji były nieprawidłowe.

IS_SRVROLEMEMBER

Sprawdza, czy dane konto użytkownika jest członkiem określonej roli serwera.

Składnia:

```
IS_SRVROLEMEMBER ({'grupa' | 'rola'})
```

Funkcja zwraca wartość:

- 0 — jeżeli użytkownik nie jest członkiem roli,
- 1 — jeżeli jest,
- NULL — jeżeli parametry wywołania funkcji były nieprawidłowe.

SUSER_SID

Zwraca identyfikator (SID) konta użytkownika.

Składnia:

```
SUSER_SID (['login'])
```

SUSER_SNAME

Zwraca nazwę konta użytkownika o podanym identyfikatorze. Wywołana bez parametru zwraca informacje o bieżącym użytkowniku, nawet jeżeli login SQL Servera został powiązany z grupą użytkowników systemu Windows.

Składnia:

```
SUSER_SNAME ([sid])
```

Przykład:

```
SELECT SUSER_SNAME ()
GO
------------------------------------------------
LOLEK\Administrator
(1 row(s) affected)
```

USER_ID

Zwraca identyfikator użytkownika w bazie danych.
Składnia:

```
USER_ID ([konto_użytkownika])
```

USER

Zwraca nazwę aktualnie zalogowanego użytkownika bazy danych.
Składnia:

```
USER
```

Przykład: (wywołano funkcję USER w ramach tej samej sesji użytkownika, który wywołał funkcję SUSER_SNAME ()):

```
SELECT USER
GO
------------------------------------------------
dbo
(1 row(s) affected)
```

Funkcje ciągu znaków

ASCII

Zwraca kod ASCII pierwszego znaku argumentu wywołania.
Składnia:

```
ASCII ('znak' | 'ciąg_znaków')
```

CHAR

Zwraca znak o podanym kodzie ASCII.
Składnia:

```
CHAR (kod_znaku)
```

Lista najczęściej używanych kodów znaków:

- 9 — tabulacja,
- 10 — znak końca wiersza,

- 13 — znak powrotu karetki (naciśnięcie klawisza Enter odpowiada wysłaniu znaków o kodach 10 i 13),
- 32 — spacja.

CHARINDEX

Zwraca pozycję początkową wyrażenia w określonym ciągu znaków lub zero, jeżeli wyrażenie nie zostanie znalezione.

Składnia:

```
CHARINDEX (wyrażenie, 'ciąg_znaków' [, pozycja_początkowa])
```

Przykład (zwracający pozycję, od której rozpoczyna się słowo Menager zapisane w kolumnie ContactTitle tabeli Customers):

```
USE Northwind
GO
SELECT CHARINDEX('Manager',ContactTitle) As pozycja
FROM dbo.Customers
GO
pozycja
-----------
0
.
.
.
11
(91 row(s) affected)
```

DIFFERENCE

Zwraca różnicę w wynikach funkcji SOUNDEX dla dwóch ciągów znaków.

Składnia:

```
DIFFERENCE ('ciąg_znaków_1', 'ciąg_znaków_2')
```

LEFT

Zwraca określoną liczbę początkowych znaków ciągu.

Składnia:

```
LEFT ('ciąg_znaków', liczba)
```

LEN

Zwraca całkowitą długość ciągu znaków, włącznie z wiodącymi spacjami (bez spacji kończących!).

Składnia:

```
LEN ('ciąg_znaków')
```

LOWER

Konwertuje wszystkie duże litery ciągu znaków na małe.
Składnia:

```
LOWER ('ciąg_znaków')
```

LTRIM

Usuwa wiodące spacje z ciągu znaków.
Składnia:

```
LTRIM ('ciąg_znaków')
```

NCHAR

Zwraca odpowiadający wartości kodu Unicode (liczby całkowitej z przedziału od 0 do 65 535).
Składnia:

```
NCHAR (liczba)
```

PATINDEX

Zwraca początkową pozycję wzorca w ciągu znaków lub zero, jeżeli wzorzec nie zostanie znaleziony.
Składnia:

```
PATINDEX ('%wzorzec%', 'ciąg_znaków')
```

REPLACE

Zastępuje wszystkie wystąpienia ciągu znaków, przekazanego jako drugi parametr, na ciąg znaków będący pierwszym parametrem w ciągu znaków, przekazanym jako trzeci parametr wywołania.
Składnia:

```
REPLACE ('ciąg_znaków_1', 'ciąg_znaków_2', 'ciąg_znaków_3')
```

Przykład: (jeżeli w pewnej kolumnie przechowywane są dane tekstowe, które były wprowadzane z klawiatury i zatwierdzane klawiszem Enter, raport wyświetlający te dane będzie nieuporządkowany — wyświetlane dane będą dzielone pomiędzy wiersze po każdym napotkaniu sekwencji znaków CHAR(10) i CHAR(13)):

```
SELECT id, REPLACE (REPLACE(tekst, CHAR(10), ''), CHAR(13), '') AS poprawiony_tekst
FROM tabela1
GO
```

QUOTENAME

Zwraca ciąg znaków Unicode z dodanymi znacznikami zgodnymi z konwencją nazewniczą SQL Servera. W rezultacie identyfikator obiektu, nawet jeśli nie jest zgodny z tą konwencją, zostanie prawidłowo zinterpretowany przez SQL Server.

Składnia:

```
QUOTENAME ('ciąg_znaków' [, 'znacznik'])
```

gdzie:

znacznik jest opcjonalnie podawanym znakiem wyróżniającym identyfikatory. Jeżeli nie zostanie podany, przyjmowany jest nawias kwadratowy.

REPLICATE

Powtarza określoną liczbę razy ciąg znaków. Funkcja jest często wykorzystywana do wypełniania tabel danymi.

Składnia:

```
REPLICATE (ciąg_znaków, liczba)
```

REVERSE

Zwraca odwrócony ciąg znaków.

Składnia:

```
REVERSE (ciąg_znaków)
```

Przykład (wyświetlający nazwy firm będące palindromami):

```
USE NORTHWIND
GO
SELECT CompanyName
FROM dbo.Customers
WHERE CompanyName = REVERSE (CompanyName)
GO
CompanyName
----------------------------------------
(0 row(s) affected)
```

RIGHT

Zwraca określoną liczbę końcowych znaków ciągu.

Składnia:

```
RIGHT (ciąg_znaków, liczba)
```

RTRIM

Usuwa wolne spacje z ciągu znaków.

Składnia:

```
RTRIM (ciąg_znaków)
```

SOUNDEX

Zwraca czteroznakowy kod ciągu znaków umożliwiający porównanie dwóch ciągów.

Składnia:

```
SOUNDEX (ciąg_znaków)
```

SPACE

Zwraca ciąg znaków składający się z określonej liczby spacji.
Składnia:

```
SPACE (liczba)
```

STR

Dokonuje konwersji liczby na ciąg znaków.
Składnia:

```
STR (liczba [, długość [, precyzja]])
```

gdzie:
długość jest całkowitą długością (razem ze znakiem części dziesiętnej czy spacjami) wynikowego ciągu znaków. Domyślną wartością parametru jest 10,
precyzja określa liczbę miejsc po przecinku konwertowanej liczby.

STUFF

Zastępuje daną liczbę znaków podanym ciągiem znaków, rozpoczynając od określonej pozycji.
Składnia:

```
STUFF (ciąg_znaków_1, początek, długość, ciąg_znaków_2)
```

gdzie:
ciąg_znaków_1 reprezentuje źródłowy ciąg znaków.
ciąg_znaków_2 reprezentuje ciąg znaków wstawiony do ciągu źródłowego.

SUBSTRING

Zwraca fragment ciągu znaków o określonej długości i pozycji początkowej.
Składnia:

```
SUBSTRING (ciąg_znaków, początek, długość)
```

UNICODE

Zwraca kod Unicode pierwszego znaku argumentu wywołania.
Składnia:

```
UNICODE (Nznak | Nciąg_znaków)
```

UPPER

Konwertuje wszystkie małe litery ciągu znaków na duże.
Składnia:

```
UPPER (ciąg_znaków)
```

Funkcje daty i czasu

DATEADD

Zwraca informację o dacie i czasie określoną na podstawie dodania interwału do parametru wywołania.

Składnia:

```
DATEADD (jednostka, liczba, data)
```

gdzie:

jednostka może przyjąć jedną z wartości wymienionych w tabeli 2.1 (dotyczy to również pozostałych funkcji daty i czasu),
liczba określa liczbę wybranych jednostek, które należy dodać do określonej daty.

DATEDIFF

Zwraca liczbę jednostek dzielących daty przekazane jako parametry wywołania.

Składnia:

```
DATEDIFF (jednostka, data_początkowa, data_końcowa)
```

Jednostka	Skrót	Znaczene
year	yy, yyyy	Rok
quarter	qq, q	Kwartał
month	mm, m	Miesiąc
dayofyear	dy, y	Dzień roku
day	dd, d	Dzień tygodnia
week	wk, ww	Tydzień
hour	Hh	Godzina
minute	mi, n	Minuta
second	ss, s	Sekunda
millisecond	Ms	Milisekunda

DATENAME

Zwraca ciąg znaków będący wartością wybranej jednostki daty lub czasu.

Składnia:

```
DATENAME (jednostka, data)
```

Przykład:

```
SELECT DATENAME(mm, getdate())
GO
-----------------------------
November
(1 row(s) affected)
```

DATEPART

Zwraca liczbę będącą wartością wybranej jednostki daty lub czasu.
Składnia:

```
DATEPART (jednostka, data)
```

DAY

Zwraca liczbę dni obliczoną na podstawie przekazanej daty.
Składnia:

```
DAY (data)
```

GETDATE

Zwraca (w formacie datetime) aktualną datę i godzinę systemową.
Składnia:

```
GETDATE ()
```

MONTH

Zwraca liczbę miesięcy obliczoną na podstawie przekazanej daty.
Składnia:

```
MONTH (data)
```

YEAR

Zwraca liczbę miesięcy obliczoną na podstawie przekazanej daty.
Składnia:

```
YEAR (data)
```

Funkcje konfiguracyjne

@@CONNECTIONS

Zwraca liczbę połączeń (udanych lub nie) nawiązanych od momentu uruchomienia SQL Servera.
Składnia:

```
@@CONNECTIONS
```

@@DATEFIRST

Zwraca wartość parametru *DATEFIRST* (ustawianego za pomocą instrukcji *SET DATEFIRST* liczba) określającego, który dzień (1 oznacza poniedziałek, 2 — wtorek itd.) będzie traktowany jako pierwszy dzień tygodnia.
Składnia:

```
@@DATEFIRST
```

@@DBTS

Zwraca aktualną wartość unikalnego w obrębie bazy danych znacznika timestamp.
Składnia:

```
@@DBTS
```

@@LANGUAGE

Zwraca nazwę aktualnie używanego języka (język można wybrać za pomocą instrukcji *SET LANGUAGE 'nazwa języka'*).
Składnia:

```
@@LANGUAGE
```

@@LANGID

Zwraca identyfikator aktualnie używanego języka.
Składnia:

```
@@LANGID
```

@@LOCK_TIMEOUT

Zwraca aktualną wartość parametru *LOCK_TIMEOUT* określającego, ile milisekund instrukcja będzie oczekiwała na wykonanie przed zgłoszeniem błędu. Domyślna wartość –1 oznacza, że instrukcja będzie oczekiwać w nieskończoność.
Składnia:

```
@@LOCK_TIMEOUT
```

@@MAX_CONNECTIONS

Zwraca liczbę jednoczesnych połączeń obsługiwanych przez SQL Server. Wartość tego parametru (ustawianego za pomocą procedury systemowej sp_configure) zależy m.in. od wersji SQL Servera i wydajności komputera.
Składnia:

```
@@MAX_CONNECTIONS
```

@@MAX_PRECISION

Zwraca aktualną precyzję typów decimal oraz numeric (precyzję typów zmiennoprzecinkowych ustala się na poziomie serwera).
Składnia:

```
@@MAX_PRECISION
```

@@OPTIONS

Zwraca liczbową reprezentację (tabela 2.2) ustawionych opcji sesji użytkownika.
Składnia:

```
@@OPTIONS
```

@@NESTLEVEL

Zwraca poziom zagnieżdżenia bieżącej procedury składowanej. Każde wywołanie procedury przez inną procedurę zwiększa o jeden wartość parametru *NESTLEVEL*. Po osiągnięciu wartości 32 wykonanie procedury jest przerywane.

Tabela 2.2. Lista opcji sesji użytkownika i ich liczbowych reprezentacji

Wartość	Parametr	Opis
1	DISABLE_DEF_CNST_CHK	Sprawdzanie spójności referencyjnej danych poprzez wyzwalacze
2	IMPLICIT_TRANSACTIONS	Niejawne rozpoczynanie transakcji poprzez wykonanie dowolnej instrukcji języka Transact-SQL
4	CURSOR_CLOSE_ON_COMMIT	Automatyczne zamykanie kursora w momencie zatwierdzania transakcji
8	ANSI_WARNINGS	Ostrzeżenia przed wykonaniem instrukcji Transact-SQL niezgodnych ze standardem SQL-2
16	ANSI_PADDING	Usuwanie nieznaczącej pozycji dla typów o stałej długości
32	ANSI_NULLS	Obsługa wartości specjalnej NULL
64	ARITHABORT	Przerywanie wykonania instrukcji po napotkaniu błędu arytmetycznego
128	ARITHIGNORE	Zwracanie wartości NULL w przypadku napotkania błędu arytmetycznego
256	QUOTED_IDENTIFIER	Wymuszanie stosowania identyfikatorów zgodnych z konwencją nazewniczą SQL Servera
512	NOCOUNT	Brak informacji o liczbie wierszy wyniku
1024	ANSI_NULL_DFLT_ON	Zmiana domyślnej definicji kolumn — kolumny, dla których nie podano jawnie zawężenia NOT NULL będą mogły zawierać wartości nieokreślone
2048	ANSI_NULL_DFLT_OFF	Kolumny tworzone w ramach sesji użytkownika domyślnie nie będą dopuszczać wartości NULL
4096	CONCAT_NULL_YIELDS_NULL	Połączenie dowolnego ciągu z wartością NULL zwróci wartość NULL
8192	NUMERIC_ROUNDABORT	Zgłoszenie błędu, jeżeli w wyniku wykonania instrukcji dane zmiennoprzecinkowe zostaną zaokrąglone

Składnia:

```
@@NESTLEVEL
```

@@REMSERVER

Zwraca nazwę SQL Servera, z którego wywołano procedurę.

Składnia:

```
@@REMSERVER
```

@@SPID

Zwraca identyfikator serwera bieżącej sesji użytkownika.

Składnia:

```
@@SPID
```

@@SERVERNAME

Zwraca nazwę lokalnego serwera (w przypadku nazwanej instancji uzupełniona o nazwę bieżącej instancji).

Składnia:

```
@@SERVERNAME
```

@@SERVICENAME

Zwraca nazwę instancji SQL Servera. W przypadku instancji domyślnej zwraca ciąg MSSQLServer.

Składnia:

```
@@SERVICENAME
```

@@TEXTSIZE

Zwraca aktualną (ustawioną za pomocą instrukcji *SET TEXTSIZE liczba*) maksymalną długość danych typu text oraz image.

Składnia:

```
@@TEXTSIZE
```

@@VERSION

Zwraca datę wydania i numer wersji SQL Servera oraz typ procesora komputera, na którym SQL Server został zainstalowany.

Składnia:

```
@@VERSION
```

Funkcje kursora

@@CURSOR_ROWS

Zwraca liczbę wierszy ostatnio pobranych przez kursor. Wartość ujemna oznacza liczbę wierszy zwróconą za pomocą kursora otwartego asynchronicznie, –1 oznacza kursor dynamiczny, 0 — brak zwróconych wierszy lub brak otwartego kursora.

Składnia:

```
@@CURSOR_ROWS
```

@@FETCH_STATUS

Zwraca status ostatnio wykonanej operacji *FETCH* (pobrania danych). 0 oznacza poprawnie wykonaną operację, –1 błąd pobrania danych, –2 próbę pobrania nieistniejących danych.

Składnia:

```
@@FETCH_STATUS
```

CURSOR_STATUS

Zwraca informacje o otwartym kursorze i jego parametrach.

Składnia:

```
CURSOR_STATUS
  (
    {'local', 'nazwa_kursora'}
    | {'global', 'nazwa_kursora'}
    | {'variable', 'zmienna_kursora'}
  )
```

Listę zwracanych przez funkcję wartości zawiera tabela 2.3.

Tabela 2.3. Wartości zwracane przez funkcję CURSOR_STATUS

Wynik	Nazwa kursora	Zmienna kursora
1	Kursor zwrócił co najmniej jeden wiersz lub kursor został zdefiniowany jako dynamiczny	Kursor przyporządkowany do zmiennej został otwarty i zwrócił co najmniej jeden wiersz lub jest to kursor dynamiczny
0	Kursor nie zwrócił danych	Kursor przyporządkowany do zmiennej został otwarty, ale nie zwrócił żadnych danych
-1	Kursor został zamknięty	Kursor przyporządkowany do zmiennej został zamknięty
-2	Nie dotyczy	Kursor nie został przyporządkowany do zmiennej lub został przyporządkowany i zamknięty przez poprzednie wywołanie procedury
-3	Kursor nie istnieje	Kursor przyporządkowany do zmiennej nie istnieje

Funkcje matematyczne

ABS

Zwraca wartość absolutną liczby.

Składnia:

```
ABS (liczba)
```

ACOS

Zwraca wyrażoną w radianach wartość funkcji trygonometrycznej arccos. Jeżeli parametr wywołania nie należy do przedziału od –1 do 1, funkcja zgłasza błąd i zwraca wartość NULL.

Składnia:

```
ACOS (liczba)
```

ASIN

Zwraca wyrażoną w radianach wartość funkcji trygonometrycznej arcsin. Jeżeli parametr wywołania nie należy do przedziału od –1 do 1, funkcja zgłasza błąd i zwraca wartość NULL.

Składnia:

```
ASIN (liczba)
```

ATAN

Zwraca (wyrażoną w radianach) wartość funkcji trygonometrycznej arcustangens.

Składnia:

```
ATAN (liczba)
```

ATN2

Zwraca (wyrażoną w radianach) wartość kąta skierowanego, którego arcustangens został obliczony na podstawie przekazanych argumentów.

Składnia:

```
ATN2 (liczba, liczba)
```

CEILING

Zwraca najmniejszą liczbę całkowitą większą lub równą liczbie przekazanej jako parametr wywołania.

Składnia:

```
CEILING (liczba)
```

COS

Zwraca (wyrażoną w radianach) wartość funkcji trygonometrycznej cosinus.

Składnia:

```
COS (liczba)
```

COT

Zwraca (wyrażoną w radianach) wartość funkcji trygonometrycznej cotangens.

Składnia:

```
COT (liczba)
```

DEGREES

Przekształca wartość kąta skierowanego wyrażoną w radianach na stopnie.
Składnia:

```
DEGREES (liczba)
```

EXP

Zwraca wartość stałej e (2,7182818284590451) podniesionej do podanej jako parametr wywołania potęgi.
Składnia:

```
EXP (liczba)
```

FLOOR

Zwraca największą liczbę całkowitą mniejszą lub równą liczbie przekazanej jako parametr wywołania.
Składnia:

```
FLOOR (liczba)
```

LOG

Zwraca wartość logarytmu naturalnego określonej liczby.
Składnia:

```
LOG (liczba)
```

LOG10

Zwraca wartość logarytmu dziesiętnego określonej liczby.
Składnia:

```
LOG10 (liczba)
```

PI

Zwraca wartość stałej pi.
Składnia:

```
PI ()
```

POWER

Zwraca wartość liczby podniesionej do określonej potęgi.
Składnia:

```
POWER (liczba, potęga)
```

Przykład (obliczający kolejne potęgi liczby 2):

```
DECLARE @liczba int, @potęga int
SET @liczba = 2
SET @potęga = 1
```

```
SET NOCOUNT ON
WHILE @potęga < 8
BEGIN
PRINT POWER (@liczba, @potęga)
SET @potęga = @potęga + 1
END
SET NOCOUNT OFF
GO
2
4
8
16
32
64
128
```

RADIANS

Przelicza miarę konta podaną w stopniach na radiany.
Składnia:

```
RADIANS (liczba)
```

RAND

Zwraca losową liczbę typu float z zakresu od 0 do 1.
Składnia:

```
RAND ([wzorzec])
```

> Wielokrotne wywołanie w ramach jednej sesji użytkownika funkcji RAND z tą samą wartością parametru wzorzec spowoduje wygenerowanie takich samych wartości.

ROUND

Zwraca zaokrągloną do określonej liczby miejsc po przecinku liczbę przekazaną jako parametr wywołania. SQL Server 2000 zawsze zwraca wartość funkcji ROUND — jeżeli wartość parametru precyzja będzie mniejsza od 0, funkcja zwróci wartość 0.
Składnia:

```
ROUND (liczba, precyzja [, operacja])
```

gdzie:
operacja określa, czy liczba będzie zaokrąglana (operacja domyślna), czy obcinana (wartość parametru różna od 0).

SIGN

Zwraca wartość +1 dla liczb dodatnich, 0 dla zera i –1 dla liczb ujemnych przekazanych jako parametr wywołania.

Składnia:

```
SIGN (liczba)
```

SIN

Zwraca (wyrażoną w radianach) wartość funkcji trygonometrycznej sinus.
Składnia:

```
SIN (liczba)
```

SQUARE

Zwraca kwadrat liczby przekazanej jako parametr wywołania.
Składnia:

```
SQUARE (liczba)
```

SQRT

Zwraca pierwiastek kwadratowy liczby przekazanej jako parametr wywołania.
Składnia:

```
SQRT (liczba)
```

TAN

Zwraca (wyrażoną w radianach) wartość funkcji trygonometrycznej tangens.
Składnia:

```
TAN (liczba)
```

Funkcje metadanych

COL_LENGTH

Zwraca zdefiniowaną długość kolumny.
Składnia:

```
COL_LENGTH ('tabela', 'kolumna')
```

COL_NAME

Zwraca (jako ciąg typu sysname) nazwę kolumny.
Składnia:

```
COL_NAME (tabela_id, kolumna_id)
```

gdzie:
tabela_id i *kolumna_id* są identyfikatorami obiektu (identyfikator obiektu zwraca funkcja OBJECT_ID).

COLUMNPROPERTY

Zwraca informację o właściwościach kolumny lub procedury.

Składnia:

```
COLUMNPROPERTY (id, 'kolumna', 'właściwość')
```

gdzie:

id jest identyfikatorem obiektu,
kolumna jest nazwą kolumny lub procedury,
właściwość jest jedną z wymienionych w tabeli 2.4 nazw właściwości.

Tabela 2.4. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję columnproperty

Właściwości	Opis	Wynik
AllowsNull	Dopuszcza wartości NULL	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsComputed	Kolumna przechowuje obliczane wartości	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsCursorType	Parametr procedury jest parametrem typu CURSOR	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsDeterministic	Kolumna przechowuje dane deterministyczne (których wartość obliczona na podstawie tych samych danych jest zawsze taka sama)	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsFulltextIndexed	Kolumna jest powiązana z indeksem pełnotekstowym	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsIdentity	Na kolumnę zostało nałożone zawężenie IDENTITY	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsIdNotForRepl	Dla kolumny sprawdzany jest warunek IDENTITY_INSERT	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsIndexable	Kolumna jest powiązana z indeksem	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsOutParam	Parametr jest wyjściowym parametrem procedury	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsPrecise	Określono precyzję dla przechowywanych w kolumnie danych	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsRowGuidCol	Kolumna przechowuje identyfikatory obiektów i	1 = PRAWDA

	został jej nadany atrybut ROWGUIDCOL	0 = FAŁSZ NULL = Błędne dane
Precision	Precyzja danych przechowywanych w kolumnie lub przekazanych przez parametr	Określona dla kolumny precyzja NULL = Błędne dane
Scale	Precyzja danych przechowywanych w kolumnie lub przekazanych przez parametr	Określona dla kolumny skala NULL = Błędne dane
UsesAnsiTrim	Ustawienie opcji ansi_padding dla kolumny	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane

DATABASEPROPERTYEX

Zwraca informacje o bieżących ustawieniach bazy danych.

Funkcja DATABASEPROPERTYEX zastępuje występującą we wcześniejszych wersjach SQL Servera funkcję DATABASEPROPERTY.

Składnia:

```
DATABASEPROPERTYEX('baza', 'właściwość')
```

gdzie:

właściwość jest jedną z wymienionych w tabeli 2.5 nazw właściwości.

DB_ID

Zwraca identyfikator bazy danych.

Składnia:

```
DB_ID (['baza_danych'])
```

Tabela 2.5. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję databaseproperty

Właściwość	Opis	Wynik
IsAnsiNullDefault	Baza danych jest zgodna z regułami standardu SQL-2 dopuszczającymi użycie wartości NULL	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsAnsiNullsEnabled	Porównania z wartością NULL zwracają w wyniku NULL	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsAnsiWarningsEnabled	Wykonanie instrukcji niezgodnych ze standardem spowoduje wygenerowanie ostrzeżenia	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsAutoClose	Baza danych zostanie automatycznie	1 = PRAWDA

	zamknięta po zakończeniu wszystkich sesji klienckich	0 = FAŁSZ NULL = Błędne dane
IsAutoCreateStatistics	Statystyki są automatycznie tworzone na podstawie zmieniających się danych	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsAutoShrink	Pliki bazy danych są automatycznie zmniejszane, gdy ilość wolnego miejsca przekroczy 25%	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsAutoUpdateStatistics	Opcja auto update statistics została włączona	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsBulkCopy	Dopuszczalne są operacje nielogowane (niezapisane w dzienniku transakcyjnym)	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsCloseCursorsOnCommitEnabled	Automatyczne zamykanie kursora w momencie zatwierdzania transakcji	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsDboOnly	Baza danych działa w trybie wyłącznego dostępu użytkownika dbo	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsDetached	Baza danych została odłączona	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsEmergencyMode	Baza danych działa w trybie awaryjnym	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsFulltextEnabled	Usługa wyszukiwania pełnotekstowego została skonfigurowana dla bazy danych	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsInLoad	Baza danych jest uruchamiana	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsInRecovery	Baza danych jest odtwarzana	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsInStandBy	Baza danych jest w trybie „tylko do odczytu” z zapisywaniem operacji	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane

IsLocalCursorsDefault	Domyślnym typem kursora jest LOCAL	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsNotRecovered	Baz danych nie została odtworzona do spójnego stanu — wystąpił błąd procesu recovery	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsNullConcat	Łączenie (konkatenacja) wartości z wartością NULL zwraca wartość NULL	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsOffline	Baza danych jest w trybie off-line	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsQuotedIdentifiersEnabled	Identyfikatory mogą być wyróżniane za pomocą cudzysłowów	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsReadOnly	Baza danych jest w trybie „tylko do odczytu”	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsRecursiveTriggersEnabled	Wyzwalacze mogą wywoływać inne wyzwalacze	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsShutDown	Baza danych nie została uruchomiona	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsSingleUser	Baza danych działa w trybie dostępu jednego użytkownika	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsSuspect	Spójność danych nie mogła zostać sprawdzona.	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsTruncLog	Nieaktywna część dziennika transakcyjnego jest usuwana podczas procesu checkpoit	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
Version	Wewnętrzny numer wersji bazy danych	Numer wersji NULL = brak dostępu do bazy

DB_NAME

Zwraca nazwę bazy danych.
Składnia:

```
DB_NAME ([baza_id])
```

FILE_ID

Zwraca identyfikator pliku bazy danych.
Składnia:

```
FILE_ID (['plik_bazy_danych'])
```

FILE_NAME

Zwraca nazwę pliku bazy danych.
Składnia:

```
FILE_NAME ([plik_id])
```

FILEGROUP_ID

Zwraca identyfikator grupy plików bazy danych.
Składnia:

```
FILEGROUP_ID (['grupa_plików_bazy_danych'])
```

FILEGROUP_NAME

Zwraca nazwę grupy plików bazy danych.
Składnia:

```
FILE_NAME ([plik_id])
```

Wszystkie informacje o plikach bazy danych należących do poszczególnych grup plików możemy wyświetlić odczytując zawartość dwóch tabel systemowych istniejących w każdej bazie danych:

```
SELECT sysfiles.fileid, sysfiles.size, sysfiles.name, sysfilegroups.groupname
FROM sysfiles INNER JOIN sysfilegroups ON
sysfiles.groupid = sysfilegroups.groupid
ORDER BY sysfilegroups.groupname
GO
fileid size name groupname
------ ----------- ------------------------------------ --------------------
1 688 Northwind PRIMARY
(1 row(s) affected)
```

FILEGROUPPROPERTY

Zwraca informacje o grupie plików.
Składnia:

```
FILEGROUPPROPERTY ('grupa_plików_bazy_danych', 'właściwość')
```

gdzie:

właściwość jest jedną z wymienionych w tabeli 2.6 nazw właściwości.

Tabela 2.6. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję FILEGROUPPROPERTY

Właściwość	Opis	Wynik
IsReadOnly	Grupa plików jest w trybie „tylko do odczytu”	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsUserDefinedFG	Grupa plików została utworzona przez użytkownika	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsDefault	Grupa plików jest domyślną grupą	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane

FILEPROPERTY

Zwraca informacje o pliku.

Składnia:

```
FILEPROPERTY ('plik_bazy_danych', 'właściwość')
```

gdzie:

właściwość jest jedną z wymienionych w tabeli 2.7 nazw właściwości.

Tabela 2.7. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję fileproperty

Właściwość	Opis	Wynik
IsReadOnly	Plik jest w trybie „tylko do odczytu”	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsPrimaryFile	Plik jest głównym plikiem bazy danych	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsLogFile	Plik przechowuje dziennik transakcyjny	1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
SpaceUsed	Wielkość pliku	Liczba stron danych zapisanych w pliku

FULLTEXTCATALOGPROPERTY

Zwraca informacje o katalogu wyszukiwania pełnotekstowego.

Składnia:

```
FULLTEXTCATALOGPROPERTY ('katalog', 'właściwość')
```

gdzie:

właściwość jest jedną z wymienionych w tabeli 2.8 nazw właściwości.

FULLTEXTSERVICEPROPERTY

Zwraca informacje o usłudze wyszukiwania pełnotekstowego.
Składnia:

```
FULLTEXTSERVICEPROPERTY ('właściwość')
```

gdzie:
właściwość jest jedną z wymienionych w tabeli 2.9 nazw właściwości.

INDEX_COL

Zwraca nazwę kolumny powiązanej z indeksem.
Składnia:

```
INDEX_COL ('tabela', indeks_id, klucz_id)
```

Tabela 2.8. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję fulltextcatalogproperty

Właściwość	Wynik
PopulateStatus	0 = bezczynny, 1 = trwa wypełnianie, 2 = zatrzymany z powodu niewystarczających zasobów systemowych, 3 = wstrzymany, 4 = wznowiono wstrzymane wypełnianie, 5 = katalog jest niedostępny, 6 = trwa wypełnianie przyrostowe, 7 = trwa aktualizacja indeksu (ostatniego etapu wypełniania katalogu), 8 = zatrzymany z powodu braku miejsca na dysku, 9 = trwa ogłaszanie
ItemCount	Liczba indeksowanych obiektów
IndexSize	Wyrażona w Mb wielkość katalogu
UniqueKeyCount	Liczba niepowtarzalnych słów (kluczy) w indeksie katalogu
LogSize	Wyrażona w bajtach wielkość pliku dziennika usługi MS Search
PopulateCompletionAge	Liczba sekund, jakie upłynęły pomiędzy ostatnim wypełnieniem katalogu a datą 01-01-1990 00:00:00

Tabela 2.9. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję fulltextserviceproperty

Właściwość	Wynik
ResourceUsage	Liczba z zakresu od 1 do 5 określająca priorytet, z którym uruchomiono usługę. Wartością domyślną jest 3
ConnectTimeout	Liczba sekund określająca czas oczekiwania usługi MS Search Service na połączenie z SQL Serverem
IsFulltextInstalled	1 = Zainstalowano usługę indeksowania pełnotekstowego 0 = Nie zainstalowano usługi indeksowania pełnotekstowego NULL = Błędne dane

INDEXPROPERTY

Zwraca informacje o wybranym indeksie.
Składnia:

```
INDEXPROPERTY (tablela_id, 'indeks', 'właściwość')
```

gdzie:
właściwość jest jedną z wymienionych w tabeli 2.10 nazw właściwości.

Tabela 2.10. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję indexproperty

Właściwość	Wynik
IndexDepth	Liczba poziomów indeksu
IndexFillFactor	Procent wypełnienia bloku indeksu danymi
IndexID	Identyfikator indeksu
IsAutoStatistics	Indeks został utworzony automatycznie 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsClustered	Indeks jest indeksem grupującym 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsFulltextKey	Indeks jest indeksem pełnotekstowym. 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsHypothetical	Indeks jest indeksem hipotetycznym (przechowującym statystyki związane z kolumną) 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsPadIndex	Indeks może być wykorzystywany przez procesy wewnętrzne 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsPageLockDisallowed	Strona indeksu może zostać zablokowana 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsRowLockDisallowed	Wiersz indeksu może zostać zablokowany 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsStatistics	Indeks został utworzony automatycznie lub poprzez instrukcję CREATE STATISTICS 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsUnique	Indeks przechowuje wartości niepowtarzalne 1 = PRAWDA

0 = FAŁSZ NULL = Błędne dane

OBJECT_ID

Zwraca identyfikator obiektu o podanej nazwie.
Składnia:

```
OBJECT_ID ('nazwa_obiektu')
```

OBJECT_NAME

Zwraca nazwę obiektu o podanym identyfikatorze.
Składnia:

```
OBJECT_NAME (obiekt_id)
```

OBJECTPROPERTY

Zwraca informacje o wybranym obiekcie bazy danych.
Składnia:

```
OBJECTPROPERTY (obiekt_id, 'właściwość')
```

gdzie:
właściwość jest jedną z wymienionych w tabeli 2.11 nazw właściwości.

Tabela 2.11. Lista dopuszczalnych dla obiektów różnych typów wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję objectproperty

Właściwość	Typ obiektu	Wynik
CnstIsClustKey	Zawężenie	Klucz podstawowy z domyślnym indeksem grupującym 1 = PRAWDA 0 = FAŁSZ
CnstIsColumn	Zawężenie	Zawężenie zdefiniowane dla kolumny 1 = PRAWDA 0 = FAŁSZ
CnstIsDeleteCascade	Zawężenie	Klucz obcy zdefiniowany z klauzulą ON DELETE CASCADE
CnstIsDisabled	Zawężenie	Zablokowane zawężenie 1 = PRAWDA 0 = FAŁSZ
CnstIsNonclustKey	Zawężenie	Klucz podstawowy z indeksem 1 = PRAWDA 0 = FAŁSZ
CnstIsNotTrusted	Zawężenie	Zawężenie zostało włączone bez sprawdzania poprawności danych 1 = PRAWDA

		0 = FAŁSZ
CnstIsNotRepl	Zawężenie	Zawężenie zdefiniowane z klauzulą NOT FOR REPLICATION 1 = PRAWDA 0 = FAŁSZ
CnstIsUpdateCascade	Zawężenie	Klucz obcy zdefiniowany z klauzulą ON UPDATE CASCADE 1 = PRAWDA 0 = FAŁSZ
ExecIsAfterTrigger	Wyzwalacz	Wyzwalacz wywoływany po wykonaniu instrukcji 1 = PRAWDA 0 = FAŁSZ
ExecIsAnsiNullsOn	Procedura składowana, Wyzwalacz, Widok	Ustawiono opcję ANSI_NULLS podczas tworzenia obiektu 1 = PRAWDA 0 = FAŁSZ
ExecIsDeleteTrigger	Wyzwalacz	Wyzwalacz wywoływany poleceniem DELETE 1 = PRAWDA 0 = FAŁSZ
ExecIsFirstDeleteTrigger	Wyzwalacz	Wyzwalacz wywoływany jako pierwszy po wykonaniu polecenia DELETE 1 = PRAWDA 0 = FAŁSZ
ExecIsFirstInsertTrigger	Wyzwalacz	Wyzwalacz wywoływany jako pierwszy po wykonaniu polecenia INSERT 1 = PRAWDA 0 = FAŁSZ
ExecIsFirstUpdateTrigger	Wyzwalacz	Wyzwalacz wywoływany jako pierwszy po wykonaniu polecenia UPDATE 1 = PRAWDA 0 = FAŁSZ
ExecIsInsertTrigger	Wyzwalacz	Wyzwalacz wywoływany poleceniem DELETE 1 = PRAWDA 0 = FAŁSZ
ExecIsInsteadOfTrigger	Wyzwalacz	Wyzwalacz wywoływany zamiast instrukcji Transact-SQL 1 = PRAWDA 0 = FAŁSZ
ExecIsLastDeleteTrigger	Wyzwalacz	Wyzwalacz wywoływany jako ostatni po wykonaniu polecenia DELETE 1 = PRAWDA

		0 = FAŁSZ
ExecIsLastInsertTrigger	Wyzwalacz	Wyzwalacz wywoływany jako ostatni po wykonaniu polecenia INSERT 1 = PRAWDA 0 = FAŁSZ
ExecIsLastUpdateTrigger	Wyzwalacz	Wyzwalacz wywoływany jako ostatni po wykonaniu polecenia UPDATE 1 = PRAWDA 0 = FAŁSZ
ExecIsQuotedIdentOn	Procedura składowana. Wyzwalacz. Widok	Ustawiono opcję QUOTED_IDENTIFIER podczas tworzenia obiektu 1 = PRAWDA 0 = FAŁSZ
ExecIsStartup	Procedura składowana	Procedura wykonywana automatycznie podczas uruchamiania SQL Servera 1 = PRAWDA 0 = FAŁSZ
ExecIsTriggerDisabled	Wyzwalacz	Zablokowany wyzwalacz 1 = PRAWDA 0 = FAŁSZ
ExecIsUpdateTrigger	Wyzwalacz	Wyzwalacz wywoływany poleceniem UPDATE 1 = PRAWDA 0 = FAŁSZ
HasAfterTrigger	Tabela, Widok	Obiekt, dla którego zdefiniowany został wyzwalacz wywoływany po wykonaniu instrukcji Transact-SQL 1 = PRAWDA 0 = FAŁSZ
HasInsertTrigger	Tabela, Widok	Obiekt, dla którego zdefiniowany został wyzwalacz wywoływany poleceniem INSERT 1 = PRAWDA 0 = FAŁSZ
HasInsteadOfTrigger	Tabela, Widok	Obiekt, dla którego zdefiniowany został wyzwalacz wywoływany zamiast instrukcji Transact-SQL 1 = PRAWDA 0 = FAŁSZ
HasUpdateTrigger	Tabela, Widok	Obiekt, dla którego zdefiniowany został wyzwalacz wywoływany poleceniem UPDATE

		1 = PRAWDA 0 = FAŁSZ
IsCheckCnst	Każdy	Obiekt jest zawężeniem typu CHECK 1 = PRAWDA 0 = FAŁSZ
IsConstraint	Każdy	Obiekt jest zawężeniem 1 = PRAWDA 0 = FAŁSZ
IsDefault	Każdy	Zdefiniowano wartość domyślną obiektu 1 = PRAWDA 0 = FAŁSZ
IsDefaultCnst	Każdy	Obiekt jest zawężeniem typu DEFAULT 1 = PRAWDA 0 = FAŁSZ
IsDeterministic	Funkcja, Widok	Wartości obliczane na podstawie tych samych danych są zawsze takie same 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsExecuted	Każdy	Obiekt może zostać wywołany 1 = PRAWDA 0 = FAŁSZ
IsExtendedProc	Każdy	Obiekt jest procedurą dodatkową 1 = PRAWDA 0 = FAŁSZ
IsForeignKey	Każdy	Obiekt jest kluczem obcym 1 = PRAWDA 0 = FAŁSZ
IsIndexed	Tabela, Widok	Obiekt posiada indeks
IsIndexable	Tabela, Widok	Obiekt może posiadać indeks
IsInlineFunction	Funkcja	Obiekt jest funkcją zwracającą zbiór rekordów zamiast wartości zmiennej 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsMSShipped	Każdy	Obiekt został utworzony podczas instalacji SQL Servera 1 = PRAWDA 0 = FAŁSZ
IsPrimaryKey	Każdy	Obiekt jest kluczem podstawowym 1 = PRAWDA 0 = FAŁSZ

IsProcedure	Każdy	Obiekt jest procedurą składowaną 1 = PRAWDA 0 = FAŁSZ
IsQuotedIdentOn	Funkcja, Procedura składowana, Tabela, Wyzwalacz, Widok	Dla obiektu została ustawiona opcja QUOTED_IDENTIFIER 1 = PRAWDA 0 = FAŁSZ
IsReplProc	Każdy	Obiekt jest procedurą replikacji 1 = PRAWDA 0 = FAŁSZ
IsRule	Każdy	Dla obiektu zostały zdefiniowane zasady 1 = PRAWDA 0 = FAŁSZ
IsScalarFunction	Funkcja	Obiekt jest funkcją skalarną 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsSchemaBound	Funkcja, Widok	Obiekt jest powiązany z obiektami bazodanowymi i jego definicja nie może zostać zmieniona 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsSystemTable	Tabela	Obiekt jest tabelą systemową 1 = PRAWDA 0 = FAŁSZ
IsTable	Tabela	Obiekt jest tabelą 1 = PRAWDA 0 = FAŁSZ
IsTableFunction	Funkcja	Obiekt jest funkcją zwracającą wartości w postaci tabeli i zdefiniowaną jako zbiór instrukcji Transact-SQL 1 = PRAWDA 0 = FAŁSZ NULL = Błędne dane
IsTrigger	Każdy	Obiekt jest wyzwalaczem 1 = PRAWDA 0 = FAŁSZ
IsUniqueCnst	Każdy	Obiekt jest zawężeniem UNIQUE 1 = PRAWDA 0 = FAŁSZ
IsUserTable	Tabela	Obiekt jest tabelą użytkownika 1 = PRAWDA

		0 = FAŁSZ
IsView	Widok	Obiekt jest widokiem 1 = PRAWDA 0 = FAŁSZ
OwnerId	Każdy	Identyfikator właściciela obiektu NULL = Błędne dane
TableDeleteTrigger	Tabela	Tabela ma zdefiniowany wyzwalacz wywoływany poleceniem DELETE >1 identyfikator pierwszego wyzwalacza 0 = FAŁSZ NULL = Błędne dane
TableFullTextBackground-UpdateIndexOn	Tabela	Z tabelą jest powiązany aktywny indeks wyszukiwania pełnotekstowego 1 = PRAWDA 0 = FAŁSZ
TableFulltextCatalogId	Tabela	Identyfikator katalogu przechowującego powiązany z tabelą indeks wyszukiwania pełnotekstowego >1 identyfikator katalogu 0 = Brak powiązanego katalogu
TableFullTextChangeTrackingOn	Tabela	Zmiany w tabeli są śledzone i odnotowane w indeksie wyszukiwania pełnotekstowego 1 = PRAWDA 0 = FAŁSZ
TableFulltextKeyColumn	Tabela	Identyfikator kolumny zawierającej unikalne identyfikatory rekordów na potrzeby usługi MS Search 0 = Z tabelą nie jest powiązany indeks wyszukiwania pełnotekstowego
TableFullTextPopulateStatus	Tabela	Stan wypełniania powiązanego z tabelą indeksu wyszukiwania pełnotekstowego 0 = nie wypełniony 1 = wypełniony 2 = wypełniony przyrostowo
TableHasActiveFulltextIndex	Tabela	Istnieje powiązany z tabelą aktywny indeks wyszukiwania pełnotekstowego 1 = PRAWDA

		0 = FAŁSZ
TableHasCheckCnst	Tabela	Istnieje zdefiniowane dla tabeli zawężenie CHECK 1 = PRAWDA 0 = FAŁSZ
TableHasClustIndex	Tabela	Istnieje powiązany z tabelą indeks grupujący 1 = PRAWDA 0 = FAŁSZ
TableHasDefaultCnst	Tabela	Istnieje zdefiniowane dla tabeli zawężenie DEFAULT 1 = PRAWDA 0 = FAŁSZ
TableHasDeleteTrigger	Tabela	Istnieje zdefiniowany dla tabeli wyzwalacz wywołany poleceniem DELETE 1 = PRAWDA 0 = FAŁSZ
TableHasForeignKey	Tabela	Istnieje zdefiniowane dla tabeli zawężenie FOREIGN KEY 1 = PRAWDA 0 = FAŁSZ
TableHasForeignRef	Tabela	Tabela jest powiązana z innymi tabelami poprzez zawężenie FOREIGN KEY 1 = PRAWDA 0 = FAŁSZ
TableHasIdentity	Tabela	Istnieje zdefiniowany dla tabeli identyfikator rekordów 1 = PRAWDA 0 = FAŁSZ
TableHasIndex	Tabela	Istnieje indeks powiązany z tabelą 1 = PRAWDA 0 = FAŁSZ
TableHasInsertTrigger	Tabela	Istnieje zdefiniowany dla tabeli wyzwalacz wywołany poleceniem INSERT 1 = PRAWDA 0 = FAŁSZ
TableHasNonclustIndex	Tabela	Istnieje niegrupujący indeks powiązany z tabelą 1 = PRAWDA 0 = FAŁSZ
TableHasPrimaryKey	Tabela	Istnieje zdefiniowane dla tabeli zawężenie PRIMARY KEY 1 = PRAWDA 0 = FAŁSZ
TableHasRowGuidCol	Tabela	Istnieje kolumna typu

		ROWGUIDCOL 1 = PRAWDA 0 = FAŁSZ
TableHasTextImage	Tabela	Istnieje kolumna typu TEXT 1 = PRAWDA 0 = FAŁSZ
TableHasTimestamp	Tabela	Istnieje kolumna typu TIMESTAMP 1 = PRAWDA 0 = FAŁSZ
TableHasUniqueCnst	Tabela	Istnieje zdefiniowane dla tabeli zawężenie UNIQUE 1 = PRAWDA 0 = FAŁSZ
TableHasUpdateTrigger	Tabela	Istnieje zdefiniowany dla tabeli wyzwalacz wywołany poleceniem UPDATE 1 = PRAWDA 0 = FAŁSZ
TableInsertTrigger	Tabela	Tabela ma zdefiniowany wyzwalacz wywoływany poleceniem INSERT >1 identyfikator pierwszego wyzwalacza 0 = FAŁSZ NULL = Błędne dane
TableIsFake	Tabela	Tabela nie istnieje i jest tworzona na żądanie przez SQL Server 1 = PRAWDA 0 = FAŁSZ
TableIsPinned	Tabela	Tabela jest buforowana w tabeli podręcznej 1 = PRAWDA 0 = FAŁSZ
TableTextInRowLimit	Tabela	>1 Maksymalna długość tekstu w wierszu 0 = opcja nie została ustawiona
TableUpdateTrigger	TableUpdateTrigger	Tabela ma zdefiniowany wyzwalacz wywoływany poleceniem UPDATE >1 identyfikator pierwszego wyzwalacza 0 = FAŁSZ

@@PROCID

Zwraca identyfikator aktualnie wykonywanej procedury składowanej (0 oznacza, że żadna procedura nie jest wykonywana).

Składnia:

```
@@PROCID
```

TYPEPROPERTY

Zwraca informacje o typie danych.
Składnia:

```
TYPEPROPERTY ('typ', 'właściwość')
```

gdzie:
właściwość jest jedną z wymienionych w tabeli 2.12 nazw właściwości.

Tabela 2.12. Lista dopuszczalnych wartości parametru właściwość i odpowiadające im wartości zwracane przez funkcję typeproperty

Właściwość	Opis	Wynik
Precision	Precyzja typu	Liczba cyfr lub znaków NULL = Błąd danych
Scale	Skala typu	Liczba miejsc po przecinku NULL = Błąd danych
AllowsNull	Typ zezwala na zapisywanie wartości NULL	1 = PRAWDA 0 = FAŁSZ NULL = Błąd danych
UsesAnsiTrim	Opcja ansi_padding była włączona podczas tworzenia typu	1 = PRAWDA 0 = FAŁSZ NULL = Błąd danych

Funkcje statystyki systemu

@@CPU_BUSY

Zwraca liczbę milisekund, podczas których, od czasu uruchomienia SQL Servera, procesor obsługiwał jego żądania.
Składnia:

```
@@CPU_BUSY
```

Tabela master..sysprocesses zawiera informacje o każdym połączeniu z SQL Serverem, w tym informacje o czasie jego nawiązania. Ponieważ połączenie o identyfikatorze równym 1 jest połączeniem systemowym nawiązywanym w momencie uruchamiania instancji, wykonując poniższe zapytanie poznamy dokładny czas jej uruchomienia:

```
USE master
SELECT login_time
FROM sysprocesses
WHERE spid=1
GO
login_time
--------------------------
```

```
2002-11-02 15:34:36.010
(1 row(s) affected)
```

@@IDLE

Zwraca liczbę milisekund, podczas których od czasu uruchomienia SQL Server był bezczynny.
Składnia:

```
@@IDLE
```

@@IO_BUSY

Zwraca liczbę milisekund, podczas których od czasu uruchomienia SQL Server przeprowadzał operacje wejścia-wyjścia.
Składnia:

```
@@IO_BUSY
```

@@PACK_RECEIVED

Zwraca liczbę pakietów odebranych przez SQL Server od czasu jego uruchomienia.
Składnia:

```
@@PACK_RECEIVED
```

@@PACK_SENT

Zwraca liczbę pakietów wysłanych przez SQL Server od czasu jego uruchomienia.
Składnia:

```
@@PACK_SENT
```

@@PACKET_ERRORS

Zwraca liczbę utraconych pakietów od czasu uruchomienia SQL Servera.
Składnia:

```
@@PACKET_ERRORS
```

@@TIMETICKS

Zwraca liczbę mikrosekund składających się na jeden cykl.
Składnia:

```
@@TIMETICKS
```

@@TOTAL_ERRORS

Zwraca liczbę błędów operacji zapisu lub odczytu danych z dysku, które wystąpiły od czasu uruchomienia SQL Servera.
Składnia:

```
@@TOTAL_ERRORS
```

@@TOTAL_READ

Zwraca liczbę przeprowadzonych od czasu uruchomienia SQL Servera operacji fizycznych odczytu (odczytu bloków danych z dysku, nie z pamięci podręcznej).

Składnia:

```
@@TOTAL_READ
```

@@TOTAL_WRITE

Zwraca liczbę przeprowadzonych od czasu uruchomienia SQL Servera operacji fizycznych zapisów (zapisu bloków danych na dysku, nie do pamięci podręcznej).

Składnia:

```
@@TOTAL_WRITE
```

Funkcje systemowe

APP_NAME

Zwraca nazwę aplikacji, za pomocą której została nawiązana bieżąca sesja użytkownika.

Składnia:

```
APP_NAME ()
```

Przykład:

```
DECLARE @CurrentApp varchar(35)
SET @CurrentApp = APP_NAME()
IF @CurrentApp <> 'SQL Query Analyzer'
RAISERROR ('Sesja nieautoryzowana',1,1)
ELSE PRINT 'Połączenie nawiązane przez Query Analyzer'
GO
Połączenie nawiązane przez Query Analyzer
```

CASE

Sprawdza wartość argumentu wywołania i zwraca jedną z wielu predefiniowanych wyników.

Składnia:

```
CASE argument
  WHEN warunek THEN wynik
    [...n]
  [
    ELSE wynik_n
```

```
  ]
END
```

Przykład (sortujący w porządku rosnącym lub malejącym dane według wybranej kolumny):

```
USE master
DECLARE @sort char(10), @porzadek char(1)
SET @sort = 'name'
SET @porzadek = 'a'
SELECT *
FROM sysobjects
ORDER BY
CASE WHEN @porzadek = 'a' AND @sort = 'id' THEN id
  ELSE NULL END,
CASE WHEN @porzadek = 'd' AND @sort = 'id' THEN id
  ELSE NULL END DESC,
CASE WHEN @porzadek = 'a' AND @sort = 'name' THEN name
  ELSE NULL END,
CASE WHEN @porzadek = 'd' AND @sort = 'name' THEN name
  ELSE NULL END
DESC
GO
```

CAST

Dokonuje konwersji danych do określonego typu.
Składnia:

```
CAST (wyrażenie AS typ)
```

gdzie:
typ jest poprawną nawą typu danych SQL Servera.

Funkcja CONVERT jest odpowiednikiem funkcji CAST.

COALESCE

Zwraca pierwszą określoną wartość przekazaną jako argument wywołania.
Składnia:

```
COALESCE (wyrażenie [,…n])
```

Funkcja COALESCE jest funkcjonalnym odpowiednikiem poniższego wywołania funkcji CASE:

```
CASE
WHEN (wyrażenie_1 IS NOT NULL) THEN wyrażenie_1
...
WHEN (wyrażenie_n IS NOT NULL) THEN wyrażenie_n
ELSE NULL
```

CURRENT_TIMESTAMP

Zwraca bieżącą datę i czas systemowy.
Składnia:

```
CURRENT_TIMESTAMP
```

Funkcjonalnym odpowiednikiem funkcji CURRENT_TIMESTAMP jest funkcja GETDATE().

CURRENT_USER

Zwraca nazwę aktualnie zalogowanego użytkownika.
Składnia:

```
CURRENT_USER
```

Funkcjonalnym odpowiednikiem funkcji CURRENT_USER jest funkcja USER_NAME().

DATALENGTH

Zwraca liczbę bajtów zarezerwowanych do przechowywania określonych wartości.
Składnia:

```
DATALENGTH (wartość)
```

Przykład:

```
USE pubs
SELECT au_lname, DATALENGTH (au_lname) AS długość
FROM dbo.authors
WHERE city = 'Oakland'
GO
au_lname długość
---------------------------------------- -----------
Green 5
Straight 8
Stringer 8
MacFeather 10
Karsen 6
(5 row(s) affected)
```

@@ERROR

Zwraca numer błędu wywołanego przez ostatnią instrukcję Transact-SQL. Ponieważ po wykonaniu instrukcji wartość funkcji jest kasowana, należy wpisać ją np. do specjalnej tabeli użytkownika.

Składnia:

```
@@ERROR
```

FORMATMESSAGE

Odczytuje odpowiedni wiersz z tabeli master..sysmessages przechowującej komunikaty błędów, ale w przeciwieństwie do polecenia RAISERROR, nie wyświetla komunikatu błędu.

Skladnia:

```
FORMATMESSAGE (numer_komunikatu, parametr [,…n])
```

GETANSINULL

Zwraca 1, jeżeli ustawienia określonej bazy danych pozwalają na zapisywanie wartości NULL (ustawienia domyślne), w przeciwnym wypadku funkcja zwraca 0.

Składnia:

```
GETANSINULL (['baza_danych'])
```

HOST_ID

Zwraca identyfikator komputera, z którego funkcja została wywołana.

Składnia:

```
HOST_ID ()
```

HOST_NAME

Zwraca nazwę komputera, z którego funkcja została wywołana.

Składnia:

```
HOST_ NAME ()
```

IDENT_INCR

Zwraca wartość kolumny typu IDENTITY ustawioną podczas tworzenia tabeli lub widoku.

Składnia:

```
IDENT_INCR ('nazwa_obiektu')
```

@@IDENTITY

Zwraca ostatnią wartość identyfikatora wiersza. Jeżeli w ramach sesji żaden identyfikator nie został wygenerowany, funkcja zwraca wartość NULL.

Składnia:

```
@@IDENTITY
```

ISDATE

Zwraca 1, jeżeli parametrem wywołania była poprawna data, w przeciwnym wypadku funkcja zwraca 0.

Składnia:

```
ISDATE (wyrażenie)
```

ISNULL

Zastępuje wartość nieokreśloną podaną wartością.

Składnia:

```
ISNULL (wyrażenie, wartość)
```

Przykład:

```
USE pubs
SELECT discounttype, highqty, ISNULL (highqty, 5555)
FROM dbo.discounts
GO
discounttype highqty
---------------------------------------- ------- ------
Initial Customer NULL 5555
Volume Discount 1000 1000
Customer Discount NULL 5555
(3 row(s) affected)
```

ISNUMERIC

Zwraca 1, jeżeli parametrem wywołania była poprawna liczba, w przeciwnym wypadku funkcja zwraca 0.

Składnia:

```
ISNUMERIC (wyrażenie)
```

NEWID

Generuje nowy unikalny identyfikator (wartość typu uniqueidentifier).

Składnia:

```
NEWID()
```

NULLIF

Zwraca wartość nieokreśloną, jeżeli oba parametry wywołania maja taką samą wartość. W przeciwnym wypadku zwraca wartość pierwszego parametru wywołania.

Składnia:

```
NULLIF (wyrażenie_1, wyrażenie_2)
```

PARSENAME

Zwraca wybrany element pełnej nazwy obiektu.
Składnia:

```
PARSENAME ('nazwa_obiektu', element)
```

gdzie:
element może przyjąć jedną z wymienionych w tabeli 2.13 wartości.

Tabela 2.13. Lista wartości parametru elementu i ich opis

Wartość	Opis
1	Nazwa obiektu
2	Właściciel obiektu
3	Nazwa bazy danych
4	Nazwa serwera

Przykład:

```
DECLARE @element int
SET @element=1
WHILE @element<=4
BEGIN
PRINT PARSENAME ('LOLEK\MSSQL.Northwind.dbo.Categories',@element)
SET @element = @element+1
END
GO
Categories
dbo
Northwind
LOLEK\MSSQL
```

PERMISSIONS

Zwraca zapisaną w postaci ciągu bitów informację o uprawnieniach bieżącego użytkownika.
Składnia:

```
PERMISSIONS ([obiekt_id [, 'kolumna']])
```

gdzie:
obiekt_id jeżeli został podany jest identyfikatorem obiektu, dla którego będą sprawdzane uprawnienia użytkownika. Jeżeli parametr nie zostanie podany, sprawdzone zostaną uprawnienia użytkownika do wykonywania instrukcji (tabele 2.14, 2.15, 2.16).

Tabela 2.14. Lista uprawnień systemowych

Reprezentacja bitowa	Uprawnienie
1	CREATE DATABASE
2	CREATE TABLE

4	CREATE PROCEDURE
8	CREATE VIEW
16	CREATE RULE
32	CREATE DEFAULT
64	BACKUP DATABASE
128	BACKUP LOG

Tabela 2.15. Lista uprawnień obiektowych

Reprezentacja bitowa	Wartość
1	SELECT ALL
2	UPDATE ALL
4	REFERENCES ALL
8	INSERT
16	DELETE
32	EXECUTE
4096	SELECT ANY
8192	UPDATE ANY
16 384	REFERENCES ANY

Tabela 2.16. Lista uprawnień obiektowych nadawanych na poziomie poszczególnych kolumn

Reprezentacja bitowa	Uprawnienie
1	SELECT
2	UPDATE
4	REFERENCES

@@ROWCOUNT

Zwraca liczbę wierszy ostatnio zwróconego wyniku.
Składnia:

```
@@ROWCOUNT
```

STATS_DATE

Zwraca informacje o dacie ostatniej aktualizacji statystyk dla wybranego obiektu.
Składnia:

```
STATS_DATE (tabela_id, indeks_id)
```

@@TRANCOUNT

Zwraca liczbę aktywnych transakcji dla bieżącej sesji użytkownika.
Składnia:

```
@@TRANCOUNT
```

USER_NAME

Zwraca nazwę konta użytkownika o podanym identyfikatorze.

Składnia:

```
USER_NAME ([użytkownik_id])
```

Funkcje tekstu i danych binarnych

PATINDEX

Zwraca początkową pozycję pierwszego wystąpienia wzorca w podanym wyrażeniu lub 0, jeżeli wzorzec nie został znaleziony.

Składnia:

```
PATINDEX ('%wzorzec%', wyrażenie)
```

TEXTPTR

Zwraca wskaźnik do danych typu text, ntext lub image przechowywanych w kolumnie.

Składnia:

```
TEXTPTR (kolumna)
```

TEXTVALID

Zwraca 1, jeżeli podany wskaźnik do danych typu text, ntext czy image jest prawidłowy lub 0 w przeciwnym wypadku.

Składnia:

```
TEXTVALID ('tabela.kolumna', wskaźnik)
```

Funkcje grupujące dane

AVG

Zwraca wartość średnią zbioru liczb przekazanych jako parametry wywołania. Wartości NULL są ignorowane.

Składnia:

```
AVG ([ALL | DISTINCT] wyrażenie)
```

gdzie:

ALL (wartość domyślna) oznacza, że wynik zostanie obliczony na podstawie wszystkich danych.
LDISTINCT oznacza, że wynik zostanie obliczony na podstawie pojedynczych wystąpień poszczególnych wartości.

COUNT

Zwraca liczbę wyrażeń przekazanych jako parametr wywołania. Funkcja wywołana z argumentem * jako jedyna z funkcji grupujących uwzględnia wartości NULL.

Składnia:

```
COUNT ({[ALL | DISTINCT] wyrażenie] | *})
```

MAX

Zwraca największą wartość spośród przekazanych parametrów wywołania.
Składnia:

```
MAX ([ALL | DISTINCT] wyrażenie)
```

MIN

Zwraca najmniejszą wartość spośród przekazanych parametrów wywołania.
Składnia:

```
MIN ([ALL | DISTINCT] wyrażenie)
```

SUM

Zwraca sumę wartości przekazanych jako parametry wywołania. Parametrami funkcji SUM mogą być jedynie dane liczbowe, a wartości NULL są ignorowane.
Składnia:

```
SUM ([ALL | DISTINCT] wyrażenie)
```

STDEV

Zwraca standardową dewiację statystyczną obliczoną dla przekazanych parametrów wywołania.
Składnia:

```
STDEV (wyrażenie)
```

STDEVP

Zwraca standartową dewiację statystyczną obliczoną dla populacji przekazanych parametrów wywołania.
Składnia:

```
STDEVP (wyrażenie)
```

VAR

Zwraca wariancję obliczoną dla przekazanych parametrów wywołania.
Składnia:

```
VAR (wyrażenie)
```

VARP

Zwraca wariancję obliczoną dla populacji przekazanych parametrów wywołania.
Składnia:

```
VARP (wyrażenie)
```

Funkcje typu RowSet

CONTAINSTABLE

Zwraca tabelę składającą się z dowolnej liczby wierszy spełniających określone kryteria wyszukiwania.

Składnia:

```
CONTAINSTABLE (tabela, {kolumna | *},'<warunek_wyszukiwania>' [, pierwsze_n_wierszy])
```

Tabela, z której wybierane są wiersze, musi być powiązana z indeksem wyszukiwania pełnotekstowego. Wyszukiwanie pełnotekstowe zostało opisane w rozdziale 5.

FREETEXTTABLE

Zwraca tabelę składającą się z dowolnej liczby wierszy zawierających ciąg znaków przekazany jako kryterium wyszukiwania.

Składnia:

```
FREETEXTTABLE (tabela, {kolumna | *},'tekst_wzorca' [, pierwsze_n_wierszy])
```

Tabele z której wybierane są wiersze musi być powiązana z indeksem wyszukiwania pełnotekstowego.

OPENDATASOURCE

Otwiera połączenie (korzystając z dostawcy danych OLE DB) z serwerem bazodanowym.

Składnia:

```
OPENDATASOURCE (dostawca, ciąg_połączenia)
```

Przykład:

```
SELECT *
FROM OPENDATASOURCE(
'SQLOLEDB',
'Data Source=LOLEK\MSSQL;User ID=sa;Password=hasło'
).Northwind.dbo.Categories
--poniżej znajduje się plan wykonania zapytania
GO
StmtText
-------------------------------------------------------------------------------
|--Remote Query(SOURCE:(<UNKNOWN>), QUERY:(SELECT Remote1001."CategoryID"
Col1003,Remote1001."CategoryName" Col1004,Remote1001."Description"
Col1005,Remote1001."Picture" Col1006,Remote1001."uzytkownik"
Col1007,Remote1001."czas" Col1008 FROM "Northwind"."
(1 row(s) affected)
```

OPENQUERY

Wykonuje zapytanie na zdalnym serwerze bazodanowym. Serwer musi zostać wcześniej połączony (ang. Linked Serverz) z serwerem lokalnym.

Składnia:

```
OPENQUERY ('serwer_połaczony', 'zapytanie')
```

OPENROWSET

Pobiera dane znajdujące się na zdalnym serwerze bazodanowym.
Składnia:

```
OPENROWSET ('dostawca',
  {'źródło_danych'; 'użytkownik'; 'hasło'
    | 'ciąg_połaczenia'},
  {[baza_danych.] [właściciel.] obiekt
    | 'zapytanie'}
  )
```

Przykład:

```
USE pubs
GO
SELECT *
FROM OPENROWSET('MSDASQL',
'DRIVER={SQL Server};SERVER=LOLEK\MSSQL;UID=sa;PWD=hasło', Northwind.dbo.Categories)
GO
```

Operatory

Operatory pełnią rolę spójników języka Transact-SQL i chociaż mogą zostać zastąpione odpowiednią funkcją, poprawiają czytelność kodu programu. Funkcja każdego operatora zależy od kontekstu jego wystąpienia. Operatory mogą być łączone w rozbudowane funkcje. W takim przypadku istotna jest znajomość kolejności, w jakiej poszczególne operatory zostaną wykonane przez SQL Server. Wyróżnia się cztery typy operatorów:

- arytmetyczne,
- ciągów znaków,
- logiczne,
- porównania.

Operatory działają na pojedynczej wartości albo na parze wartości. Większość operatorów działa na dwóch wartościach; operator jest wtedy umieszczony pomiędzy nimi (np. a + b). Operatory oczekujące pojedynczej wartości nazywane są operatorami jednoargumentowymi.

Operatory arytmetyczne

SQL Server obsługuje następujące operatory arytmetyczne:

- iloczyn (*),
- iloraz (/),
- modulo (%),
- sumę (+),
- różnicę (-).

Operator ciągów znaków

Jedynym operatorem ciągów znaków jest operator konkatenacji (+). Pozostałe operacje na ciągach znaków przeprowadzane są przez odpowiednie funkcje ciągu znaków.

Operatory logiczne

Operatory logiczne wykorzystywane są najczęściej do łączenia warunków umieszczonych w klauzuli WHERE. SQL Server obsługuje trzy operatory logiczne:

- koniunkcji (AND) zwracający wartość 1 (prawda), jeżeli wszystkie wyrażenia są prawdziwe,
- alternatywy (OR) zwracający wartość 1 (prawda), jeżeli którekolwiek z wyrażeń jest prawdziwe,
- negacji (NOT) zwracający wartość 1 (prawda), jeżeli wyrażenie było nieprawdziwe i 0 w przeciwnym wypadku.

Operatory porównania

Do operatorów porównania obsługiwanych przez SQL Server należą:

- równy (=),
- mniejszy niż (<),
- większy niż (>),
- mniejszy lub równy (<=),
- większy lub równy (>=),
- różny (<>),
- należy do zbioru (IN),
- należy do przedziału (BETWEEN … AND),
- jest zgodny ze wzorcem (LIKE).

Hierarchia operatorów

Jeżeli w jednym wyrażeniu posłużono się kilkoma operatorami, SQL Server obliczy jego wartość w następującej (mogącej wpłynąć na wynik) kolejności:

1. Operatory umieszczone w nawiasach okrągłych.
2. Mnożenie, dzielenie, modulo.
3. Dodawanie, odejmowanie.
4. Konkatenacja.
5. Negacja.
6. Koniunkcja.
7. Alternatywa.

Operatory o tym samym priorytecie zostaną wykonane od strony lewej do prawej.

Wyrażenia

Wyrażenia są połączeniem symboli i operatorów reprezentującym pojedynczą wartość. Typ reprezentowanych danych zależy od typu elementów wyrażenia.

Znaczniki sterujące wykonaniem programu

W przeciwieństwie do standardowego języka SQL, Transact-SQL posiada kilka wzorowanych na językach proceduralnych znaczników sterujących wykonaniem programu.

BEGIN ... END

Dyrektywy BEGIN … END wydzielają zbiór instrukcji Transact-SQL traktowanych jako funkcjonalna całość (blok instrukcji).
Składnia:

```
BEGIN
  {
    instrukcja_Transact-SQL | Blok_instrukcji
  }
END
```

GOTO

Powoduje zmianę sposobu wykonania programu i przejście do oznaczonej etykietą instrukcji. Instrukcje znajdujące się za poleceniem GOTO nie zostaną wykonane.
Składnia:
Definicja etykiety:

```
etykieta :
```

zmiana sposobu wykonania programu: `GOTO etykieta`

IF ... ELSE

Znaczniki IF … ELSE pozwalają na warunkowe wykonanie instrukcji Transact-SQL. Instrukcja zostanie wykonana, jeżeli warunek zostanie spełniony, w przeciwnym wypadku może zostać wykonana instrukcja umieszczona w opcjonalnej klauzuli ELSE.
Składnia:

```
IF warunek_logiczny
{instrukcja_Transact-SQL | Blok_instrukcji}
[ELSE
{instrukcja_Transact-SQL | Blok_instrukcji}]
```

Przykład:

```
USE Northwind
DECLARE @klient varchar(10)
SET @klient = 'FRANK'
IF EXISTS (SELECT * FROM orders
WHERE customerid = @klient)
  BEGIN
    PRINT 'Istnieją powiązane rekordy w tabeli Orders'
    PRINT 'Klient nie może zostać usunięty'
  END
ELSE
  BEGIN
    DELETE customers WHERE customerid = @klient
    PRINT 'Usunięto klienta ' + @klient
  END
GO
Istnieją powiązane rekordy w tabeli Orders
Klient nie może zostać usunięty
```

Jeżeli chcemy wykonywać warunkowe instrukcje na poziomie poszczególnych wierszy, musimy skorzystać z funkcji CASE.

RETURN

Powoduje natychmiastowe przerwanie wykonywania procedury lub bloku instrukcji Transact-SQL.

Składnia:

```
RETURN [liczba]
```

gdzie:

liczba jest opcjonalną wartością zwracaną przez przerwaną procedurę.

WAITFOR

Określa czas, po którym określona instrukcja zostanie wykonana.

Składnia:

```
WAITFOR {DELAY 'czas' | TIME 'czas'}
```

gdzie:

DELAY powoduje, że instrukcja zostanie wykonana po upływie określonego czasu (maksymalnie po upływie 24 godzin),
TIME powoduje, że instrukcja zostanie wykonana w określonym czasie.

WHILE

Polecenie WHILE pozwala na wielokrotne (aż do momentu spełnienia określonego warunku logicznego) wykonanie instrukcji Transact-SQL. Dodatkowo możemy sterować wykonaniem instrukcji za pomocą słów kluczowych BREAK i CONTINUE.
Składnia:

```
WHILE warunek_logiczny
{instrukcja_Transact-SQL | Blok_instrukcji}
[BREAK]
{instrukcja_Transact-SQL | Blok_instrukcji}
[CONTINUE]
```

gdzie:
BREAK powoduje natychmiastowe przerwanie wykonywania instrukcji Transact-SQL, *CONTINUE* powoduje restart pętli WHILE. Instrukcje znajdujące się za poleceniem CONTINUE nie zostaną wykonane.

Słowa kluczowe

Słowa kluczowe są ciągami znaków mających ściśle określone znaczenie i interpretowanych przez SQL Server w sposób charakterystyczny dla słowa kluczowego. Niedopuszczalne jest używanie słów kluczowych niezgodnie z określonymi dla nich zasadami syntaktycznymi (na przykład w niewłaściwej dla słowa klauzuli) oraz z określonymi dla nich zasadami semantycznymi (ich znaczeniem dla SQL Servera). Możliwe jest używanie słów kluczowych jako identyfikatorów obiektów, o ile stosowane są odpowiednie symbole wyróżniające identyfikatory obiektów (cudzysłowy lub nawiasy kwadratowe).

Kurs Transact-SQL, część 3 (Pobieranie danych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Informacje przechowywane w bazach danych mogą zostać pobrane za pomocą instrukcji języka Transact-SQL SELECT. Instrukcja SELECT (zapytanie) określa, jakie dane mają zostać zwrócone w wyniku jej wykonania, natomiast to, w jaki sposób instrukcja zostanie wykonana, zależy od optymalizatora zapytań SQL Servera.

Przetwarzanie zapytań przez SQL Server

Wszystkie zapytania przetwarzane są w ten sam sposób. Z tego powodu SQL Server może odwołać się do wcześniej obliczonego planu w wypadku kilkukrotnego wykonywania tego samego zapytania. Możliwość ta jest podstawą podziału zapytań na zapytania ad hoc oraz zapytania składowane.

Przetwarzanie zapytań ad hoc składa się z pięciu etapów:

1. Sprawdzenie poprawności syntaktycznej (ang. Parse). Na tym etapie sprawdzana jest zgodność użytych znaczników z regułami języka Transact-SQL.
2. Sprawdzenie poprawności semantycznej (ang. Resolve) — sprawdzenie poprawności użytych identyfikatorów oraz uprawnień użytkownika do wykonania określonych operacji na wskazanych obiektach.
3. Optymalizacja (ang. Optimize) — wybór tych indeksów oraz metod łączenia obiektów, które umożliwią najszybsze wykonanie zapytania.
4. Kompilacja (ang. Compile). Na tym etapie instrukcja zostaje przekształcona do postaci wykonywalnej.
5. Wykonanie (ang. Execute) — wykonanie skompilowanej instrukcji.

W celu poprawy wydajności SQL Server buforuje w buforze procedury (ang. Procedure Cache) plan wykonania danego zapytania. Kolejne wywołania tego zapytania wymagają jedynie dwóch etapów:

W celu poprawy wydajności SQL Server buforuje w buforze procedury (ang. Procedure Cache) plan wykonania danego zapytania. Kolejne wywołania tego zapytania wymagają jedynie dwóch etapów:

1. Pobranie skompilowanej instrukcji z pamięci podręcznej procedury.
2. Wykonania skompilowanej instrukcji.

> Mechanizm buforowania instrukcji jest w pełni automatyczny i nie może być konfigurowany z poziomu języka Transact-SQL.

Odwołanie do zapytania składowanego następuje w trzech przypadkach:

1. Jeżeli wystąpi żądanie wykonania dokładnie takiego samego zapytania (składającego się z takich samych znaczników). W przypadku poniższych zapytań pierwsze i trzecie zapytanie zostaną wykonane w oparciu o ten sam plan.

   Przykład zapytań ad hoc:

```
2.
3.  USE Northwind
4.  SELECT * FROM dbo.Orders WHERE Freight = 10.22
5.  SELECT * FROM dbo.Orders WHERE Freight = $10.22
6.  SELECT * FROM dbo.Orders WHERE Freight = 10.22
7.
```

8. Jeżeli SQL Server prawidłowo zamieni stałą użytą w instrukcji na parametr, to przygotuje plan wykonania instrukcji sparametryzowanej i wykorzysta go do wykonania wszystkich zapytań, tak jak w przypadku poniższych:

```
9.
10. USE Northwind
11. SELECT * FROM dbo.Orders WHERE OrderID = 10250
12. SELECT * FROM dbo.Orders WHERE OrderID = 10255
13. SELECT * FROM dbo.Orders WHERE OrderID = 10257
14.
```

15. Jeżeli wywołano procedurę systemową sp_executesql. Poniższe zapytania zostaną wykonane w oparciu o ten sam plan.

```
16.
17. USE Northwind
18. EXEC sp_executesql
19. N'SELECT * FROM dbo.Orders WHERE OrderID = @l', N'@l int', @l=10255
20. EXEC sp_executesql
21. N'SELECT * FROM dbo.Orders WHERE OrderID = @l', N'@l int', @l=10257
22. EXEC sp_executesql
23. N'SELECT * FROM dbo.Orders WHERE OrderID = @l', N'@l int', @l=10259
24. GO
25.
```

Pierwsza część książki opisuje dwa pierwsze etapy wykonania instrukcji Transact-SQL. Etapowi trzeciemu poświecone są rozdziały 4 i 10, a opis dwóch ostatnich etapów wykracza poza zakres książki.

Instrukcja SELECT

Instrukcja SELECT służy do pobierania danych z bazy. Instrukcja musi zwierać (z wyjątkiem polecenia SELECT odwołującego się wyłącznie do stałych, zmiennych lub wyrażeń arytmetycznych) co najmniej jedną klauzulę. Za pomocą polecenia SELECT określamy interesujące nas kolumny (dokonujemy operacji selekcji pionowej,

projekcji), za pomocą klauzuli FROM wskazujemy tabelę, z której pobieramy dane. Z reguły ogranicza się również, za pomocą klauzuli WHERE, liczbę zwracanych wierszy do rekordów spełniających określone kryteria (operacja selekcji poziomej, selekcji).
Składnia:

```
SELECT [ALL | DISTINCT]
[TOP n [PERCENT] [WITH TIES]]
< lista_wyboru >
< lista_wyboru > ::=
{ *
| {nazwa_tabeli | nazwa_widoku | alias}.*
| {nazwa_kolumny | wyrażenie | IDENTITYCOL | ROWGUIDCOL}
[[AS] alias_kolumny]
| alias_kolumny = wyrażenie
} [,...n]
```

gdzie:
ALL (wartość domyślna) oznacza, że zwrócone zostaną wszystkie wybrane wiersze,
DISTINCT oznacza, że z wyniku zostaną usunięte duplikaty wybranych wierszy,
TOP n oznacza, że zwrócona zostanie tylko określona liczba wierszy.

> Słowo kluczowe TOP jest prawie zawsze używane razem z klauzulą ORDER BY, ponieważ w przeciwnym wypadku SQL Server zwróci pierwsze n wierszy z uporządkowanego według losowego kryterium wyniku.

PRECENT pozwala na względne określenie liczby zwracanych wierszy. Jeżeli użyto znacznika PRECENT, wartość n musi należeć do zakresu od 0 do 100.
WITH TIES oznacza, że zostaną zwrócone wszystkie wiersze, których wartość kolumny wymienionej w klauzuli ORDER BY będzie równa najmniejszej wartości wiersza wyniku. W rezultacie, jeżeli wybieramy 5 towarów o najwyższej marży, a w tabeli znajduje się 7 towarów o tej samej, najwyższej marży, zapytanie zwróci 7, nie 5 wierszy.
IDENTITYCOL zwraca wartość kolumny IDENTITY.
ROWGUIDCOL zwraca wartość globalnego identyfikatora.
Przykłady:

```
USE Northwind
SELECT OrderDate,Freight/2 AS wyrażenie
FROM dbo.Orders
WHERE OrderDate BETWEEN '1996-07-11' AND '1996-07-18'
GO
OrderDate wyrażenie
--------------------------- ---------------------
1996-07-11 00:00:00.000 11.4900
1996-07-12 00:00:00.000 74.1650
1996-07-15 00:00:00.000 6.9850
1996-07-16 00:00:00.000 40.9550
1996-07-17 00:00:00.000 70.2550
```

```
1996-07-18 00:00:00.000 1.6250
(6 row(s) affected)
USE pubs
SELECT TOP 3 'autor: ' AS literał, au_fname + ' '+ au_lname AS konkatencja
FROM dbo.authors
ORDER BY au_lname
GO
literał konkatencja
------- ------------------------------------------------------------
autor: Abraham Bennet
autor: Reginald Blotchet-Halls
autor: Cheryl Carson
(3 row(s) affected)
```

1. Wartości są zwracane w kolejności określonej w poleceniu SELECT. Jeżeli użyto symbolu *, dane będą zwrócone w kolejności odpowiadającej kolejności kolumn w tabeli.
2. Nazwy kolumn należy oddzielać przecinkami.
3. Należy unikać stosowania symbolu * w zapytaniach.

Zwracanie określonej liczby wierszy

Ograniczyć liczbę wierszy wyniku zapytania możemy albo poprzez określenie warunku, który musi zostać spełniony, aby dany wiersz został dołączony do wyniku, albo przez podanie liczbę wierszy wyniku. W drugim przypadku możemy ustawić opcję ROWCOUNT dla połączenia lub użyć słowa kluczowego TOP. Kolejne zapytanie zwraca imię i nazwisko tych samych trzech autorów, których otrzymaliśmy w wyniku zapytania poprzedniego.

Ograniczenie liczby wierszy wyniku poprzez ustawienie opcji ROWCOUNT:

```
USE pubs
SET ROWCOUNT 3
SELECT 'autor: ' AS literał, au_fname + ' '+ au_lname AS konkatencja
FROM dbo.authors
ORDER BY au_lname
SET ROWCOUNT 0
GO
literał konkatenacja
------- ------------------------------------------------------------
autor: Abraham Bennet
autor: Reginald Blotchet-Halls
autor: Cheryl Carson
(3 row(s) affected)
```

Optymalizator SQL Servera tworzy wydajniejszy plan wykonania instrukcji dla zapytań, w których występuje słowo kluczowe TOP, od zapytań wydanych w ramach sesji z ustawioną opcją ROWCOUNT. Wynika z tego, że opcję ROWCOUNT należy ustawiać tylko wtedy, gdy uzyskanie podobnej funkcjonalności za pomocą słowa kluczowego TOP jest niemożliwe, na przykład jeśli liczba wierszy wyniku jest wartością zmiennej.

Jeżeli liczba wierszy wyniku jest przekazana jako wartość zmiennej lokalnej, musimy użyć opcji ROWCOUNT zamiast słowa kluczowego TOP. Poniższy przykład zawiera poprawne ograniczenie liczby wierszy wyniku do wartości zmiennej @liczba.

Próba użycia słowa kluczowego TOP ze zmienną skończyłaby się błędem Server: Msg 170, Level 15, State 1, Line 5: Incorrect syntax near '@liczba'.

```
USE pubs
DECLARE @liczba int
SET @liczba=3
SET ROWCOUNT @liczba
SELECT 'autor: ' AS literał, au_fname + ' '+ au_lname AS konkatencja
FROM dbo.authors
ORDER BY au_lname
SET ROWCOUNT 0
GO
literał konkatencja
------- ------------------------------------------------------------
autor: Abraham Bennet
autor: Reginald Blotchet-Halls
autor: Cheryl Carson
(3 row(s) affected)
```

Opcja WITH TIES pozwala na dołączenie do wyniku wszystkich wierszy, dla których wartość wybranej kolumny była równa wartości ostatniego wiersza dodanego do wyniku. Kolejne zapytanie zwraca informacje o wszystkich towarach zamówionych w największej liczbie w pojedynczym zamówieniu:

```
USE Northwind
SELECT TOP 1 WITH TIES OrderID,Quantity
FROM dbo.[Order Details]
ORDER BY Quantity DESC
GO
OrderID Quantity
----------- --------
10764 130
11072 130
(2 row(s) affected)
```

Klauzula FROM...

Określa, z których obiektów będą pobierane dane źródłowe.
Składnia:

```
[FROM {< tabela >} [,...n]]
< tabela > ::=
nazwa_tabeli [[AS] alias_tabeli] [WITH (< optymalizacja > [,...n])]
| nazwa_widoku [[AS] alias_widoku]
| funkcja_rowset [[AS] alias_tabeli]
| OPENXML
| fragment_tabeli [AS] alias_tabeli [(alias_kolumny [,...n])]
| < złączona_tabela >
< złączona_tabela > ::=
< tabela > < typ_złączenia > < tabela > ON < warunek >
| < tabela > CROSS JOIN < tabela >
| < złączona_tabela >
< typ_złączenia > ::=
[INNER | {{LEFT | RIGHT | FULL} [OUTER]}]
[< optymalizacja_złączenia >]
JOIN
```

gdzie:
optymalizacja wymusza użycie przez optymalizator konkretnej metody skanowania tabeli, indeksu czy trybu blokowania. Ponieważ optymalizator SQL Servera 2000 wybiera najlepszy plan wykonania instrukcji, opcja powinna być wykorzystywana tylko podczas optymalizacji (strojenia) bazy danych.
OPENXML spowoduje zapisanie wyniku w formacie XML.
fragment_tabeli jest wynikiem podzapytania.
CROSS JOIN określa, że wynik będzie obliczony jako iloczyn kartezjański złączonych tabel.
Minimalną liczbę warunków użytych do łączenia tabel można wyliczyć ze wzoru: minimalna ilość warunków = liczba złączonych tabel –1. Metody optymalizacji łączenia tabel zostały opisane w rozdziale 4.
INNER JOIN określa, że wynik będzie obliczony jako złączenie naturalnie złączonych tabel. Wynikiem złączenia naturalnego są tylko te wiersze, w których w powiązanych kolumnach zapisane były te same wartości.
LEFT OUTER JOIN określa, że wynik będzie zawierał wszystkie wiersze z lewej tabeli i wyłącznie te wiersze z prawej tabeli, w których w powiązanych kolumnach zapisane były te same wartości.
RIGHT OUTER JOIN określa, że wynik będzie zawierał wszystkie wiersze z prawej tabeli i wyłącznie te wiersze z lewej tabeli, w których w powiązanych kolumnach zapisane były te same wartości. Złączenie prawostronne jest rzadko stosowane przez osoby przyzwyczajone do czytania od strony lewej do prawej.
FULL OUTER JOIN określa, że wynik będzie zawierał wszystkie wiersze z obu złączonych tabel.
optymalizacja_złączenia wymusza użycie przez optymalizator konkretnej metody

złączenia tabel (złączenie poprzez połączenie, złączenie na podstawie zgodności funkcji skrótu lub złączenie poprzez pętlę wyszukania).
Przykłady:

```
USE pubs
SELECT au_lname, title
FROM dbo.authors INNER JOIN dbo.titleauthor ON dbo.authors.au_id =
dbo.titleauthor.au_id
INNER JOIN dbo.titles ON dbo.titleauthor.title_id = dbo.titles.title_id
GO
au_lname title
---------- ----------------------------------------------
Carson But Is It User Friendly?
MacFeather Computer Phobic AND Non-Phobic Individuals: Behavior Variations
Karsen Computer Phobic AND Non-Phobic Individuals: Behavior Variations
.
.
.
Ringer The Gourmet Microwave
Green You Can Combat Computer Stress!
(25 row(s) affected)
USE pubs
SELECT au_lname, au_fname, titleauthor.title_id
FROM authors LEFT OUTER JOIN titleauthor
ON authors.au_id = titleauthor.au_id
GO
au_lname au_fname title_id
---------- ---------- ---------- --------
Bennet Abraham BU1032
Blotchet-Halls Reginald TC4203
...
Smith Meander NULL
Straight Dean BU7832
Stringer Dirk NULL
(29 row(s) affected)
```

Łączenie wielu obiektów

Z reguły łączy się obiekty na podstawie wartości wspólnego atrybutu, na przykład wartości pary klucz podstawowy — klucz obcy. W takim przypadku musimy użyć jednoznacznego identyfikatora obiektu (kolumny). Ponieważ nazwy kolumn zawierających klucz podstawowy i obcy najczęściej są takie same, musimy poprzedzać nazwy kolumn nazwami tabel. Jeżeli dodatkowo stosujemy konwencję, w myśl której

poprzedzamy nazwy obiektów nazwą ich właściciela, zapytanie wybierające dane z kilku obiektów staje się mało czytelne. Rozwiązaniem tego problemu są aliasy.

W przypadku niektórych typów złączeń (np. złączeń tabeli z nią samą) lub podzapytań konieczne jest stosowanie aliasów.

Kolejne zapytanie zbudowano z wykorzystaniem aliasów (alias może, ale nie musi zostać poprzedzony słowem kluczowym AS):

```
USE pubs
SELECT au_lname, title
FROM dbo.authors AS a INNER JOIN dbo.titleauthor ta ON a.au_id = ta.au_id
INNER JOIN dbo.titles t ON ta.title_id = t.title_id
GO
```

Łącząc obiekty, stosuj się do następujących wskazówek:

1. Staraj się łączyć tabele za pomocą kolumn przechowujących parę kluczy podstawowy-obcy.
2. Do złączenia wykorzystuj całe klucze podstawowe tabel. Jeżeli dla jakiejś tabeli zdefiniowano złożony (składający się z kilku atrybutów) klucz podstawowy, łącząc taką tabelę, odwołuj się do całego klucza;
3. Łącz obiekty za pomocą kolumn tego samego typu.
4. Poprzedzaj nazwy kolumn aliasem nazwy obiektu źródłowego, nawet jeżeli ich nazwy są unikalne — w ten sposób poprawisz czytelność zapytania.
5. Ograniczaj liczbę łączonych obiektów do niezbędnego minimum.

Złączenie naturalne

Wynikiem złączenia naturalnego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same. Ponieważ w relacyjnych bazach danych informacje są podzielone pomiędzy tabele zawierające dane o obiektach jednego typu, złączenie naturalne jest najczęściej wykorzystywanym (i domyślnym) złączeniem obiektów. Łącząc w ten sposób obiekty należy pamiętać o następujących zasadach:

1. Ponieważ złączenie naturalne jest domyślnym typem złączenia, możesz pominąć słowo kluczowe INNER.
2. Musisz podać pełną nazwę atrybutu, za pomocą którego łączysz obiekty.
3. Łączenie obiektów za pomocą atrybutów, które przyjmują wartość NULL zakończy się niespodziewanymi wynikami.
4. SQL Server nie sortuje wyniku złączenia naturalnego — aby uzyskać posortowane dane, należy użyć klauzuli ORDER BY.

Przykład:

```
USE pubs
SELECT au_lname, COUNT(title_id) AS [liczba tytułów]
FROM dbo.authors AS a JOIN dbo.titleauthor ta
ON a.au_id = ta.au_id
```

```
GROUP BY a.au_lname
ORDER BY [liczba tytułów] DESC
GO
au_lname liczba tytułów
---------------------------------------- --------------
Ringer 4
O'Leary 2
Locksley 2
MacFeather 2
Green 2
Gringlesby 1
.
.
.
(18 row(s) affected)
```

Złączenia zewnętrzne

Wynikiem lewo- lub prawostronnego złączenia zewnętrznego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same; zbiór ten uzupełniony jest o pozostałe wiersze z lewej lub prawej łączonej tabeli. Nieistniejące wartości reprezentowane są w wyniku złączenia poprzez wartość NULL.

Wynikiem pełnego złączenia zewnętrznego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same, zbiór ten uzupełniony jest o pozostałe wiersze z obu łączonych tabel.

Złączenia zewnętrzne wykorzystywane są do wyświetlania kompletnych informacji o wszystkich obiektach danego typu, nawet jeżeli nie istnieją powiązane z nimi obiekty innego typu.

Przykład:

```
USE pubs
SELECT au_lname, COUNT(title_id) AS [liczba tytułów]
FROM dbo.authors AS a LEFT JOIN dbo.titleauthor ta
ON a.au_id = ta.au_id
GROUP BY a.au_lname
ORDER BY [liczba tytułów] DESC
GO
au_lname liczba tytułów
---------------------------------------- --------------
Ringer 4
Green 2
.
```

```
.
.
Greene 0
McBadden 0
(22 row(s) affected)
Warning: Null value eliminated by aggregate or other set operation.
```

Złączenie krzyżowe

Wynikiem złączenia krzyżowego jest iloczyn kartezjański łącznych obiektów. W przeciwieństwie do innych typów złączeń, w tym wypadku łączone obiekty nie muszą mieć wspólnych atrybutów. Złączenia tego typu są rzadko stosowane w znormalizowanych bazach danych i służą raczej do generowania danych testowych niż do wybierania danych.

Przykład:

```
USE pubs
SELECT au_lname, title
FROM dbo.authors AS a CROSS JOIN dbo.titles ta
GO
au_lname title
------------------------------------------------------------------------
.
.
.
Ringer The Psychology of Computer Cooking
Smith The Psychology of Computer Cooking
Straight The Psychology of Computer Cooking
Stringer The Psychology of Computer Cooking
White The Psychology of Computer Cooking
Yokomoto The Psychology of Computer Cooking
Bennet You Can Combat Computer Stress!
Blotchet-Halls You Can Combat Computer Stress!
Carson You Can Combat Computer Stress!
.
.
.
(437 row(s) affected)
```

Złączenie tabeli z nią samą

Złączenie tabeli z nią samą stosujemy, kiedy chcemy wybrać rekordy z tabeli na podstawie wspólnych wartości atrybutów rekordów tej samej tabeli.

Złączenie tabeli z nią samą jest techniką odpowiadającą użyciu zmiennych w proceduralnych językach programowania.

Łącząc w ten sposób, należy pamiętać o następujących zasadach:

1. Trzeba utworzyć różne aliasy dla łączonej tabeli i konsekwentnie odwoływać się do aliasów, a nie do nazwy tabeli w ramach zapytania.
2. Każdy rekord, w którym wartości atrybutu złączenia będą sobie równe, zostanie dodany do wyniku złączenia, co spowoduje powstanie duplikatów rekordów.

Przykłady:

Pierwsze zapytanie zawiera powtórzenia, drugie — nie:

```
USE Northwind
SELECT a.employeeid, a.lastname, b.employeeid, b.lastname
FROM dbo.employees AS a INNER JOIN dbo.employees AS b
ON a.title = b.title
GO
employeeid lastname employeeid lastname
----------- -------------------- ----------- --------------------
1 Davolio 1 Davolio
3 Leverling 1 Davolio
4 Peacock 1 Davolio
6 Suyama 1 Davolio
7 King 1 Davolio
9 Dodsworth 1 Davolio
2 Fuller 2 Fuller
.
.
.
(39 row(s) affected)
USE Northwind
SELECT a.employeeid, a.lastname, b.employeeid, b.lastname
FROM dbo.employees AS a INNER JOIN dbo.employees AS b
ON a.title = b.title
WHERE a.employeeid > b.employeeid
GO
employeeid lastname employeeid lastname
----------- -------------------- ----------- --------------------
3 Leverling 1 Davolio
4 Peacock 1 Davolio
4 Peacock 3 Leverling
6 Suyama 1 Davolio
6 Suyama 3 Leverling
6 Suyama 4 Peacock
```

```
7 King 1 Davolio
7 King 3 Leverling
7 King 4 Peacock
7 King 6 Suyama
9 Dodsworth 1 Davolio
9 Dodsworth 3 Leverling
9 Dodsworth 4 Peacock
9 Dodsworth 6 Suyama
9 Dodsworth 7 King
(15 row(s) affected)
```

Złączenie wyników (operator UNION)

Za pomocą operatora teoriomnogościowego UNION możemy dodać wyniki poszczególnych zapytań. Łączone wyniki muszą składać się z takiej samej liczby kolumn, a poszczególne kolumny muszą być tego samego typu, poza tym konieczne jest, aby występowały one w tej samej kolejności w obu wynikach.

Domyślnie duplikaty są usuwane ze zbioru powstałego poprzez złączenie wyników zapytań. Użycie słowa kluczowego ALL po operatorze UNION spowoduje, że nie będą one usuwane. Dzielenie skomplikowanych zapytań i łączenie ich wyników jest jednym ze sposobów poprawy wydajności SQL Servera.

Przykład:

```
USE Northwind
SELECT (firstname + ' ' + lastname) AS nazwa, city
FROM dbo.employees
WHERE firstname LIKE 'A%'
UNION
SELECT companyname, city
FROM dbo.customers
WHERE companyname LIKE 'W%'
ORDER BY nazwa DESC
GO
nazwa city
---------------------------------------- ---------------
Wolski Zajazd Warszawa
Wilman Kala Helsinki
White Clover Markets Seattle
Wellington Importadora Resende
Wartian Herkku Oulu
Anne Dodsworth London
Andrew Fuller Tacoma
(7 row(s) affected)
```

Klauzula WHERE

Za pomocą klauzuli WHERE możemy ograniczyć (dokonać operacji selekcji) wynik do wierszy spełniających podany warunek wyszukiwania.
Składnia:

```
[WHERE < warunek_wyszukiwania > | < złączenie_starego_typu >]
< złączenie_starego_typu > ::=
nazwa_kolumny {* = | = *} nazwa_kolumny
< warunek_wyszukiwania > ::=
{ [NOT] < predykat > | (< warunek_wyszukiwania >)}
[{AND | OR} [NOT] {< predykat > | (< warunek_wyszukiwania >)}]
} [,...n]
< predykat > ::=
{ wyrażenie {= | < > | ! = | > | > = | ! > | < | < = | ! <} wyrażenie
| wyrażenie_znakowe [NOT] LIKE wyrażenie_znakowe
[ESCAPE 'wzorzec']
| wyrażenie [NOT] BETWEEN wyrażenie AND wyrażenie
| wyrażenie IS [NOT] NULL
| CONTAINS
({kolumna | *} , '< warunek_wyszukiwania_typu_contains >')
| FREETEXT ({kolumna | *}, 'szukany_ciąg_znaków')
| wyrażenie [NOT] IN (podzapytanie | wyrażenie [,...n])
| wyrażenie {= | < > | ! = | > | > = | ! > | < | < = | ! <}
{ALL | SOME | ANY} (podzapytanie)
| EXISTS (podzapytanie)
}
```

gdzie:
złączenie_starego_typu jest niestandardowym określeniem warunków złączenia tabel. Symbol * = odpowiada złączeniu lewostronnemu, symbol = * — prawostronnemu,
predykat jest wyrażeniem o wartości 1 (prawda), 0 (fałsz) lub NULL,
ESCAPE 'wzorzec' określa znak eliminujący z wyniku wiersze, które go zawierają,
warunek_wyszukiwania_typu_contains powoduje wybranie wierszy zgodnych z podanym wzorcem na podstawie wyszukiwania pełnotekstowego. Wyszukiwane mogą być: wyrazy lub frazy, prefiksy oraz słowa zbliżone do wzorca na podstawie ich wymowy lub budowy,
szukany_ciąg_znaków określa wzorzec wyszukiwania pełnotekstowego, z którego znaczeniem będą porównywane wiersze wyniku (wzorzec zostaje podzielony na wyrazy, a następnie wyszukiwane są wiersze zawierające wyrazy o budowie zbliżonej do wyrazów wzorca).
Przykład:

```
USE Northwind
SELECT lastname, firstname, title
FROM employees
WHERE employeeid = 1
GO
lastname firstname title
-------------------- ---------- ------------------------------
Davolio Nancy Sales Representative
(1 row(s) affected)
```

Wybieranie danych na podstawie warunków różnego rodzaju

Język Transact-SQL nie nakłada żadnych ograniczeń na liczbę argumentów wyszukiwania wymienionych w klauzuli WHERE. W zależności od rodzaju kryteriów i typu wyszukiwanych danych możemy skorzystać z:

1. Operatorów porównania — jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli odpowiada wartości warunku wyszukiwania. Na przykład, aby wybrać nazwiska i nazwy miast pracowników ze Stanów Zjednoczonych urodzonych przed pierwszym sierpnia 1955 roku, napiszemy:

```
2.
3.  USE Northwind
4.  SELECT lastname, city
5.  FROM dbo.employees
6.  WHERE country = 'USA' AND BirthDate <'8/1/55'
7.  GO
8.  lastname city
9.  -------------------- ---------------
10. Davolio Seattle
11. Fuller Tacoma
12. Peacock Redmond
13. (3 row(s) affected)
14.
```

15. Operatora LIKE — jeżeli chcemy sprawdzić zgodność tekstu z podanym wzorcem. Operator LIKE może zostać użyty dla danych typu: char, nchar, varchar, nvarchar, binary, varbinary, smalldatetime, datetime oraz (z pewnymi ograniczeniami) dla danych typów text, ntext oraz image. Operator LIKE dopuszcza stosowanie następujących symboli wieloznacznych:
    1. % — odpowiada dowolnemu ciągowi znaków o długości równej 0 lub większej,
    2. _ — odpowiada dowolnemu pojedynczemu znakowi,
    3. *[]* — odpowiada pojedynczemu znakowi z określonego zbioru lub zakresu,
    4. *[^]* — odpowiada pojedynczemu znakowi nienależącemu do określonego zbioru lub zakresu.

Na przykład, aby wyświetlić nazwy firm zawierających w nazwie słowo Restaurant, których nazwiska przedstawicieli zaczynają się na literę M, napiszemy:

```
USE Northwind
SELECT companyname
FROM dbo.customers
WHERE companyname LIKE '%Restaurant%' AND ContactName LIKE 'M%'
GO
companyname
----------------------------------------
GROSELLA-Restaurante
Tortuga Restaurante
(2 row(s) affected)
```

16. Operatorów logicznych — jeżeli chcemy połączyć kilka warunków wyszukiwania.

    Za pomocą nawiasów możemy nie tylko poprawić czytelność złożonego warunku wyszukiwania, ale także zmienić kolejność, w której będą sprawdzane poszczególne argumenty, a tym samym wpłynąć na wynik wyszukiwania.

    Na przykład, aby wyświetlić informacje o tych towarach, których nazwy nie zaczynają się na literę T lub R lub zostały dostarczone przez dostawcę o identyfikatorze równym 21 i jednocześnie kosztowały mniej niż 15 $, napiszemy:

```
17.
18. USE Northwind
19. SELECT productname, supplierid, unitprice
20. FROM dbo.products
21. WHERE (productname NOT LIKE 'T%' AND productname NOT LIKE 'R%')
22. OR (supplierid = 21 AND unitprice < 15.00)
23. GO
24. productname supplierid unitprice
25. ---------------------------------------- ----------- ---------------------
26. Chang 1 19.0000
27. Aniseed Syrup 1 10.0000
28. .
29. .
30. .
31. (65 row(s) affected)
32.
```

33. Operatora BETWEEN … AND — jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli należy (lub nie) do określonego przedziału zamkniętego. Na

przykład, aby wyświetlić nazwy towarów, których cena nie należy do przedziału od 5 do 55, napiszemy:

```
34.
35. USE Northwind
36. SELECT productname
37. FROM dbo.products
38. WHERE unitprice NOT BETWEEN 5 AND 55
39. GO
40. productname
41. ----------------------------------------
42. Mishi Kobe Niku
43. Carnarvon Tigers
44. Sir Rodney's Marmalade
45. Guaraná Fantástica
46. Thüringer Rostbratwurst
47. Geitost
48. Côte de Blaye
49. (7 row(s) affected)
50.
```

Jeżeli chcemy sprawdzić, czy dana wartość należy do przedziału otwartego, musimy użyć koniunkcji dwóch warunków: > = x AND < = y. Z uwagi na wydajność należy unikać stosowania w klauzuli WHERE operatora NOT (np. NOT IN czy NOT BETWEEN). Metody optymalizacji zapytań zostały opisane w rozdziale 4.

51. Operatora IN — jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli należy do określonego zbioru. Na przykład, aby wyświetlić nazwy firm ze Stanów lub Włoch, napiszemy:

```
52.
53. USE Northwind
54. SELECT companyname, country
55. FROM dbo.suppliers
56. WHERE country IN ('USA', 'Italy')
57. GO
58. companyname country
59. ---------------------------------------- ---------------
60. New Orleans Cajun Delights USA
61. Grandma Kelly's Homestead USA
62. Formaggi Fortini s.r.l. Italy
63. Bigfoot Breweries USA
64. New England Seafood Cannery USA
65. Pasta Buttini s.r.l. Italy
66. (6 row(s) affected)
67.
```

SQL Server wykonuje zapytania z warunkiem IN w ten sam sposób co zapytania z ciągiem warunków OR. Plan wykonania obu zapytań będzie identyczny.

Plan wykonania zapytania zwracającego nazwy firm ze Stanów Zjednoczonych lub Włoch wykorzystującego operator IN:

```
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Suppliers].
[PK_Suppliers]), WHERE:([Suppliers].[Country]='Italy' OR
[Suppliers].[Country]='USA'))
```

Nie używaj wartości NULL z operatorem IN — wynikiem porównania z wartością NULL jest wartość nieznana, co może spowodować zwracanie niespodziewanych wyników przez zapytanie.

68. Operatora IS NULL — jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli jest wartością nieokreśloną. Na przykład, aby wyświetlić nazwy i numery telefonów tych firm, dla których nie znamy numeru faksu, napiszemy:

```
69.
70. USE Northwind
71. SELECT Companyname, Phone
72. FROM dbo.suppliers
73. WHERE fax IS NULL
74. GO
75. Companyname Phone
76. ---------------------------------------- ------------------------
77. Exotic Liquids (171) 555-2222
78. New Orleans Cajun Delights (100) 555-4822
79. Tokyo Traders (03) 3555-5011
80. .
81. .
82. .
83. (16 row(s) affected)
84.
```

Klauzula ORDER BY

Wynik zapytania może zostać sformatowany zgodnie z wymaganiami użytkownika:

- ♣ Za pomocą słowa kluczowego DISTINCT możemy wyeliminować duplikaty.

- Za pomocą aliasów możemy zmienić nazwy kolumn.
- Za pomocą literałów możemy dodać do wyniku dowolne wyrażenia.
- Za pomocą klauzuli ORDER BY możemy posortować wiersze wyniku.

Składnia:

```
[ORDER BY {wyrażenie [ASC | DESC]} [,...n]]
```

gdzie:

wyrażenie określa kolumnę, według której będą sortowane dane,
ASC (wartość domyślna) określa, że dane będą sortowane w porządku rosnącym (od wartości najmniejszej do największej),
DESC określa, że dane będą sortowane w porządku malejącym.

Porządek sortowania ustawiany jest (w wersji 2000 SQL Servera) dla całej bazy danych podczas jej tworzenia. Odczytać ustawiony porządek sortowania możemy poprzez wywołanie procedury składowanej sp_helpsort. W wersji 2000, inaczej niż we wcześniejszych wersjach SQL Servera, aby uzyskać uporządkowane dane, musimy użyć klauzuli ORDER BY. Dane nie mogą być sortowane według kolumn typu text, ntext lub image.

Przykłady:

SQL Server 2000 nie umożliwia użycia klauzuli ORDER BY jako elementu definicji widoku, chyba że użyte zostało słowo kluczowe TOP. Poniższy przykład pokazuje, jak utworzyć uporządkowany widok zawierający wszystkie wiersze tabeli:

```
USE Northwind
GO
CREATE VIEW w_orders
AS
SELECT TOP 100 PERCENT OrderID,CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
ORDER BY OrderDate DESC
GO
SELECT *
FROM w_orders
GO
OrderID CustomerID EmployeeID OrderDate
----------- ---------- ----------- --------------------------
11074 SIMOB 7 1998-05-06 00:00:00.000
.
.
.
10249 TOMSP 6 1996-07-05 00:00:00.000
10248 VINET 5 1996-07-04 00:00:00.000
(830 row(s) affected)
```

Podczas sortowania wartościom nieokreślonym przypisana jest najmniejsza wartość, czyli wiersze zawierające wartość NULL zostaną zwrócone jako pierwsze, o ile nie został

zmieniony domyślny porządek sortowania. Poniższy przykład pokazuje w jaki sposób, wykorzystując opisaną w rozdziale 1 funkcję CASE, przypisać wartościom nieokreślonym największą wartość:

```
USE pubs
SELECT *
FROM Publishers
ORDER BY CASE
WHEN state IS NULL THEN 1
ELSE 0
END, state
GO
pub_id pub_name city state country
------ ---------------------------------------- -------------------- ----- ----------
1389 Algodata Infosystems Berkeley CA USA
0877 Binnet & Hardley Washington DC USA
1622 Five Lakes Publishing Chicago IL USA
0736 New Moon Books Boston MA USA
9952 Scootney Books New York NY USA
1756 Ramona Publishers Dallas TX USA
9901 GGG&G München NULL Germany
9999 Lucerne Publishing Paris NULL France
(8 row(s) affected)
```

Zagadnienie sortowania danych zostało dodatkowo opisane w rozdziałach 4. i 10., przy okazji omawiania optymalizacji baz danych.

Kolejny przykład pokazuje, jak uporządkować wynik zapytania poprzez posortowanie wierszy, eliminacje duplikatów, oraz wykorzystanie aliasów i literałów:

```
USE Northwind
SELECT DISTINCT 'Nazwa kraju: ', Country AS 'Kraj'
FROM dbo.suppliers
WHERE SupplierID <5
ORDER BY country
GO
Kraj
------------- ---------------
Nazwa kraju: Japan
Nazwa kraju: UK
Nazwa kraju: USA
(3 row(s) affected)
```

Ponieważ użycie słowa kluczowego DISTINCT spowoduje posortowanie danych według losowych kryteriów, z reguły używa się go razem z klauzulą ORDER BY. W takim przypadku kolumna wymieniona w klauzuli ORDER BY musi zostać wymieniona w poleceniu SELECT (standard SQL-2 wymagał, aby kolumny wymienione w klauzuli ORDER BY zawsze znajdowały się na liście polecenia SELECT, ale w niewielu SZBD zaimplementowano to ograniczenie).

Klauzula GROUP BY

Określa grupy wierszy, dla których będą obliczane wyniki funkcji grupujących wymienionych w poleceniu SELECT. Jeżeli została użyta, na liście argumentów klauzuli GROUP BY muszą znaleźć się wszystkie niebędące wynikiem funkcji grupującej wyrażenia wymienione w poleceniu SELECT.

Składnia:

```
[GROUP BY [ALL] wyrażenie [,...n]
[WITH {CUBE | ROLLUP}]
]
```

gdzie:

ALL powoduje, że do wyniku zostaną dodane wiersze niespełniające warunku określonego w klauzuli WHERE. Wynik funkcji grupowych dla tych wierszy jest wartością NULL. Słowo kluczowe ALL nie może być użyte razem ze słowami CUBE i ROLLUP.
CUBE powoduje zwrócenie dodatkowego wiersza zawierającego sumę wyników obliczonych dla wszystkich możliwych kombinacji poszczególnych grup.
ROLLUP powoduje zwrócenie dodatkowego wiersza zawierającego sumę wyników obliczonych grup wartości.

1. SQL Server zwróci jeden wiersz dla każdej grupy wartości.
2. Kolumny wymienione w klauzuli GROUP BY muszą zostać wymienione w poleceniu SELECT.
3. Maksymalna liczba kolumn ograniczona jest typem poszczególnych danych zwróconych przez funkcje grupujące. Domyślnie suma długości tych danych nie może przekraczać 8 KB.

Przykłady:

Wyświetlenie sumy zamówień klientów o identyfikatorach rozpoczynających się od litery T wraz z listą wszystkich identyfikatorów klientów:

```
USE Northwind
SELECT CustomerID, SUM(Freight)
FROM dbo.orders
WHERE CustomerID Like 'T%'
GROUP BY ALL CustomerID
GO
CustomerID
```

```
---------- --------------------
TOMSP 125.9700
LILAS NULL
GREAL NULL
.
.
.
(89 row(s) affected)
Warning: Null value eliminated by aggregate or other set operation.
```

Większość funkcji grupujących ignoruje wartości NULL. Wyjątkiem jest funkcja COUNT wywołana z argumentem *:

```
USE Northwind
SELECT COUNT(*), COUNT (Region)
FROM dbo.employees
GO
----------- -----------
9 5
(1 row(s) affected)
Warning: Null value eliminated by aggregate or other set operation.
```

Obliczenie sum wybranych zamówień złożonych przez klientów o identyfikatorach zaczynających się od liter TO. Do wyniku dodano kolumnę opisującą wynik zapisany w danym wierszu. Tylko dlatego, że w bazie Northwind dane zamówienie może być złożone przez jednego klienta, podsumowania zawierają takie same wartości:

```
USE Northwind
SELECT orderid, CustomerID, SUM(Freight) as Suma
FROM dbo.orders
WHERE CustomerID LIKE 'TO%' AND orderid IN (10249,10276)
GROUP BY orderid, CustomerID
WITH ROLLUP
ORDER BY orderid, CustomerID
GO
orderid CustomerID Suma OPIS
----------- ---------- --------- ------------------------------------------------------
NULL NULL 46.6300 suma dla wszystkich zamówień wybranych klientów
10249 NULL 11.6100 suma obliczona dla zamówienia o id=10249
10249 TOMSP 11.6100 suma obliczona dla zamówienia o id=10249 i klienta TOMSP
10276 NULL 13.8400 suma obliczona dla zamówienia o id=10276
10276 TORTU 13.8400 suma obliczona dla zamówienia o id=10276 i klienta TORTU
(5 row(s) affected)
```

Obliczenie sum wybranych zamówień złożonych przez klientów o identyfikatorach zaczynających się od liter TO uzupełnionych o podsumowania dla poszczególnych klientów i zamówień. Do wyniku dodano kolumnę opisującą wynik zapisany w danym wierszu. Tylko dlatego, że w bazie Northwind dane zamówienie może być złożone przez jednego klienta, podsumowania zawierają takie same wartości:

```
USE Northwind
SELECT orderid, CustomerID, SUM(Freight) as Suma
FROM dbo.orders
WHERE CustomerID LIKE 'TO%' AND orderid IN (10249,10276)
GROUP BY orderid, CustomerID
WITH CUBE
ORDER BY orderid, CustomerID
GO
orderid CustomerID Suma OPIS
----------- ---------- -------- ------------------------------------------------------------
NULL NULL 25.4500 suma dla wszystkich zamówień wybranych klientów
NULL TOMSP 11.6100 suma wszystkich zamówień klienta TOMSP
NULL TORTU 13.8400 suma wszystkich zamówień klienta TORTU
10249 NULL 11.6100 suma obliczona dla zamówienia o id=10249
10249 TOMSP 11.6100 suma obliczona dla zamówienia o id=10249 i klienta TOMSP
10276 NULL 13.8400 suma obliczona dla zamówienia o id=102276
10276 TORTU 13.8400 suma obliczona dla zamówienia o id=10276 i klienta TORTU
(7 row(s) affected)
```

Za pomocą funkcji GROUPING możemy odróżnić wyliczone przez operatory CUBE i ROLLUP sumy jednostkowe od sum wyliczonych dla zbiorów grup. Wartość 1 zwrócona przez funkcję oznacza obliczoną sumę częściową, 0 — sumę obliczoną dla pojedynczej grupy.

Wykorzystanie funkcji GROUPING:

```
USE Northwind
SELECT orderid, GROUPING (orderid), CustomerID, GROUPING (CustomerID),
SUM(Freight) as Suma
FROM dbo.orders
WHERE CustomerID LIKE 'TO%' AND orderid IN (10249,10276)
GROUP BY orderid, CustomerID
WITH CUBE
ORDER BY orderid, CustomerID
GO
orderid CustomerID Suma
----------- ---- ---------- ---- ---------------------
```

```
NULL 1 NULL 1 25.4500
NULL 1 TOMSP 0 11.6100
NULL 1 TORTU 0 13.8400
10249 0 NULL 1 11.6100
10249 0 TOMSP 0 11.6100
10276 0 NULL 1 13.8400
10276 0 TORTU 0 13.8400
(7 row(s) affected)
```

Klauzula HAVING

Określa warunki wyszukiwania dla grup lub funkcji grupujących. Za pomocą klauzuli HAVING określa się warunki, które muszą zostać spełnione przez grupy wymienione w klauzuli GROUP BY, tak jak za pomocą klauzuli WHERE określa się warunki, które muszą spełnić wiersze wymienione w klauzuli SELECT.

Składnia:

```
[HAVING < warunek_wyszukiwania >]
```

1. Używanie klauzuli HAVING w zapytaniu, w którym nie występuje klauzula GROUP BY, choć dopuszczalne, jest semantycznie nieuzasadnione.
2. Warunek wyszukiwania może dotyczyć dowolnych danych wymienionych w poleceniu SELECT.
3. Klauzula HAVING ma wyższy priorytet niż słowo kluczowe ALL i jeżeli wystąpi w zapytaniu zawierającym to słowo, zapytanie zwróci wyłącznie grupy spełniające podany warunek wyszukiwania.

Przykład:

```
USE Northwind
SELECT CustomerID, SUM(Freight)
FROM dbo.orders
WHERE CustomerID Like 'T%'
GROUP BY CustomerID
HAVING SUM(Freight) >200
GO
CustomerID
--------------------------------
THEBI 262.0900
TORTU 475.6300
TRADH 274.5600
(3 row(s) affected)
```

Klauzule COMPUTE i COMPUTE BY

Obie klauzule zwracają dodatkowy wiersz zawierający obliczone dla wyniku podsumowania. Klauzula COMPUTE BY dodatkowo zwraca obliczone sumy częściowe. Obie klauzule mogą być użyte w tym samym zapytaniu.

Składnia:

```
[COMPUTE
{{AVG | COUNT | MAX | MIN | STDEV | STDEVP
| VAR | VARP | SUM}
(wyrażenie)} [,...n]
[BY wyrażenie [,...n]]
]
```

1. Niemożliwe jest sformatowanie danych zwracanych przez obie klauzule. Brak możliwości sformatowania danych wynikowych jest powodem dla którego nie należy używać klauzul COMPUTE i COMPUTE BY jak elementów instrukcji SELECT INTO.
2. Niemożliwe jest obliczenie wyniku na podstawie danych typu text, ntext lub image.
3. Dane, na podstawie których będzie obliczony wynik, muszą zostać wymienione w oleceniu SELECT.
4. Należy używać klauzuli COMPUTE BY łącznie z klauzulą ORDER BY — w innym wypadku obliczone sumy częściowe będą nieuporządkowane i nieczytelne.
5. Lista i kolejność kolumn wymieniona w klauzuli COMPUTE BY musi dokładnie odpowiadać liście kolumn wymienionych w klauzuli ORDER BY.

Przykłady:

Wyświetlenie wartości poszczególnych zamówień użytkownika TOMSP uzupełnionych o liczbę zamówień:

```
USE Northwind
SELECT CustomerID, Freight
FROM dbo.orders
WHERE CustomerID = 'TOMSP'
COMPUTE COUNT (Freight)
GO
CustomerID Freight
---------- ---------------------
TOMSP 11.6100
TOMSP 8.2400
TOMSP 14.6800
TOMSP 1.4300
TOMSP 27.7900
TOMSP 62.2200
cnt
==========
```

```
6
(7 row(s) affected)
```

Wyświetlenie wartości towarów o określonym ID z podziałem na grupy towarowe:

```
USE Northwind
SELECT CategoryID, ProductName, UnitPrice
FROM dbo.Products
WHERE ProductID BETWEEN 6 AND 9
ORDER BY CategoryID
COMPUTE SUM(UnitPrice) BY CategoryID
COMPUTE SUM(UnitPrice)
GO
CategoryID ProductName UnitPrice
---------- ---------------------------------------- ---------------------
2 Grandma's Boysenberry Spread 25.0000
2 Northwoods Cranberry Sauce 40.0000
sum
=====================
65.0000
CategoryID ProductName UnitPrice
---------- ---------------------------------------- ---------------------
6 Mishi Kobe Niku 97.0000
sum
=====================
97.0000
CategoryID ProductName UnitPrice
---------- ---------------------------------------- ---------------------
7 Uncle Bob's Organic Dried Pears 30.0000
sum
=====================
30.0000
sum
=====================
192.0000
(8 row(s) affected)
```

Klauzula OPTION

Określa techniki optymalizacji wykorzystane przez SQL Server przy wykonaniu zapytania. Każda z opcji (technik optymalizacji) może być wymieniona tylko raz, ale klauzula może zawierać dowolną liczbę opcji.

Ponieważ optymalizator SQL Servera 2000 wybiera najlepszy plan wykonania instrukcji, klauzula powinna być wykorzystywana tylko podczas optymalizacji (strojenia) bazy danych.

Składnia:

```
[OPTION (< optymalizacja > [,...n)]
< optymalizacja > ::=
{ {HASH | ORDER} GROUP
| {CONCAT | HASH | MERGE} UNION
| {LOOP | MERGE | HASH} JOIN
| FAST liczba_wierszy
| FORCE ORDER
| MAXDOP liczba
| ROBUST PLAN
| KEEP PLAN
| KEEPFIXED PLAN
| EXPAND VIEWS
}
```

gdzie:
{HASH | ORDER} GROUP określa, czy wartości grupowe (wyniki funkcji grupowych, parametry klauzul GROUP BY, DISTINCT lub COMPUTE) będą obliczone na podstawie funkcji skrótu, czy uporządkowania wierszy,
{CONCAT | HASH | MERGE} UNION określa, czy operacja sumowania wyników będzie przeprowadzona na podstawie funkcji skrótu, scalania, czy konkatenacji wyników,
{LOOP | MERGE | HASH} JOIN określa, czy operacja łączenia będzie przeprowadzona na podstawie scalania, zgodności funkcji skrótu, czy skanowania indeksów.
FAST liczba_wierszy określa liczbę wierszy wyniku, dla której zapytanie zostanie zoptymalizowane,
FORCE ORDER wymusza łączenie tabel w takiej kolejności, jaka została podana w klauzuli FROM,
MAXDOP liczba określa maksymalną liczbę procesów, za pomocą których zapytanie może być równocześnie przetwarzane (zastępuje ustawienie opcji max degree of parallelism dla wybranego zapytania),
ROBUST PLAN wymusza tworzenie planu wykonania uwzględniającego największą z możliwych liczbę wierszy wyniku,
KEEP PLAN wymusza wykorzystywanie skompilowanej postaci zapytania (zapytanie będzie rekompilowane rzadko lub wcale pomimo zmiany danych w tabelach źródłowych),
KEEPFIXED PLAN wymusza wykorzystywanie skompilowanej postaci zapytania (zapytanie będzie rekompilowane rzadko lub wcale pomimo zmiany statystyk lub indeksów związanych z tabelami źródłowymi),
EXPAND VIEWS określa, że do wykonania zapytania nie będą wykorzystywane ewentualnie istniejące indeksy stworzone dla widoków.

Przykład:

Wymuszenie złączenia tabel poprzez ich scalenie, pomimo tego, że złączenie odbywa się poprzez kolumny, dla których istnieje indeks grupujący:

```
USE Northwind
SELECT LastName, FirstName, OrderID, OrderDate
FROM dbo.Employees e JOIN dbo.Orders o
ON e.EmployeeID = o.EmployeeID
OPTION (MERGE JOIN)
GO
```

Tematowi optymalizacji zapytań w całości poświęcony jest rozdział 4.

Klauzula FOR

Przyjmuje dwa niezwiązane ze sobą parametry: BROWSE oraz XML. Pierwszy określa, że wynik zapytania może być aktualizowany przez klientów połączonych z serwerem za pomocą biblioteki OLE DB. Drugi parametr powoduje zwrócenie danych w postaci dokumentu XML.

Składnia:

```
[FOR {BROWSE | XML {RAW | AUTO | EXPLICIT}
[, XMLDATA]
[, ELEMENTS]
[, BINARY BASE64]
}
]
```

gdzie:

RAW powoduje wydzielenie każdego wiersza wyniku za pomocą znacznika <row />>,
AUTO powoduje zapisanie wyniku w postaci prostego drzewa XML,
EXPLICIT powoduje zapasanie wyników niektórych zapytań w określonej przez użytkownika postaci,
XMLDATA zwraca wyłącznie schemat, bez korzenia drzewa XML,
ELEMENTS powoduje, że kolumny są zwracane jako odrębne elementy,
BINARY BASE64 powoduje zapisanie wyniku w postaci binarnej, zgodnej ze standardem kodowania base 64.

Przykład:

```
USE Northwind
SELECT Orders.OrderID, Shippers.CompanyName, Orders.CustomerID
FROM dbo.Orders JOIN dbo.Shippers
ON Orders.shipvia = Shippers.ShipperID
WHERE OrderID < 10250
FOR XML AUTO
GO
XML_F52E2B61-18A1-11d1-B105-00805F49916B
```

```
----------------------------------------
<Orders OrderID="10248" CustomerID="VINET">
<Shippers CompanyName="Federal Shipping"/>
</Orders>
<Orders OrderID="10249" CustomerID="TOMSP">
<Shippers CompanyName="Speedy Express"/>
</Orders>
```

Klauzula INTO

Tworzy nową tabelę i wypełnia ją danymi będącymi wynikiem zapytania.

> Do wykonania zapytania zawierającego klauzulę INTO niezbędne jest posiadanie uprawnienia do tworzenia tabel w bazie danych. Klauzula INTO nie może być użyta łącznie z klauzulami COMPUTE i COMPUTE BY oraz wykonana w obrębie jawnie rozpoczętej transakcji.

Składnia:

```
[INTO nowa_tabela]
```

Przykład:

```
USE pubs
SELECT (au_fname + 'au_lname ') AS autor, phone, address
INTO autor_2
FROM dbo.authors
WHERE city IN ('Oakland', 'Berkeley')
GO
(7 row(s) affected)
```

Podzapytania

Podzapytaniem nazywamy instrukcję SELECT umieszczoną wewnątrz innej instrukcji języka Transact-SQL (instrukcji SELECT, INSERT, UPDATE lub DELETE). Podzapytania z reguły mogą zostać zastąpione złączeniami lub wyrażeniami. Najczęstszymi powodami tworzenia podzapytań są:

1. Chęć uproszczenia zapytania poprzez podzielenie go na kilka prostszych zapytań.
2. Wykorzystanie wyniku jednego zapytania jako danych źródłowych dla innego zapytania.

> Jeżeli zadanie może zostać rozwiązane zarówno za pomocą złączenia, jak i podzapytania, wybieraj złączenia — optymalizator SQL Servera zawsze potrafi znaleźć optymalny plan wykonania złączenia i zapisać go w buforze procedury, natomiast podzapytania są trudniejsze do optymalizacji i z reguły wykonywane jako zapytania ad hoc.

Stosując podzapytania pamiętaj o następujących zasadach:

1. Podzapytanie musi zostać zapisane w nawiasie.
2. Wynik podzapytania nie może zawierać danych typów text, ntext oraz image.
3. SQL Server pozwala na zagnieżdżanie zapytań do 32 poziomu, ale w przypadku skomplikowanych lub zwracających duże liczby danych podzapytań maksymalny poziom zagnieżdżenia może ulec zmniejszeniu.

Podzapytania jako źródła danych

Wynik podzapytania może być zbiorem danych źródłowych dla innego zapytania. W takim przypadku podzapytanie znajduje się w klauzuli FROM zapytania nadrzędnego.
Przykład:

```
USE Northwind
SELECT wynik.*
FROM (SELECT CategoryID, CategoryName
FROM dbo.Categories
WHERE CategoryID<3) AS wynik
GO
CategoryID CategoryName
----------- ---------------
1 Beverages
2 Condiments
(2 row(s) affected)
```

Podzapytania jako wyrażenia

Wynik podzapytania może zostać potraktowany jako lista wartości wyrażenia, pod warunkiem, że podzapytanie zwróciło wartość skalarną. W takim przypadku podzapytanie-wyrażenie jest wykonywane tylko raz podczas wykonywania całej instrukcji SELECT.
Przykład:

```
USE Northwind
SELECT ProductName, UnitPrice,
(SELECT AVG(UnitPrice) FROM dbo.Products) AS [średnia],
UnitPrice-(SELECT AVG(UnitPrice) FROM dbo.Products) AS [różnica]
FROM dbo.Products
WHERE CategoryID=4
GO
ProductName UnitPrice średnia różnica
---------------------------------------------------------- ---------------------
Queso Cabrales 21.0000 28.8663 -7.8663
```

```
Queso Manchego La Pastora 38.0000 28.8663 9.1337
Gorgonzola Telino 12.5000 28.8663 -16.3663
Mascarpone Fabioli 32.0000 28.8663 3.1337
Geitost 2.5000 28.8663 -26.3663
Raclette Courdavault 55.0000 28.8663 26.1337
Camembert Pierrot 34.0000 28.8663 5.1337
Gudbrandsdalsost 36.0000 28.8663 7.1337
Flotemysost 21.5000 28.8663 -7.3663
Mozzarella di Giovanni 34.8000 28.8663 5.9337
(10 row(s) affected)
```

Podzapytania powiązane

Poprzednio opisane typy podzapytań określa się nazwą podzapytań niepowiązanych. Ich wykonanie przez zewnętrzne zapytanie sprowadza się do wykonania wewnętrznej instrukcji SELECT i zwrócenia obliczonego wyniku do zapytania zewnętrznego.

Podzapytania drugiego typu, podzapytania powiązane, wykonywane są według innego schematu. W tym wypadku podzapytanie wykonywane jest dla każdego wiersza wyniku zapytania zewnętrznego i może być z nim porównywane. Podzapytanie powiązane jest przykładem dynamicznego złączenia wyniku zapytania z każdym kolejnym wierszem wyniku zapytania zewnętrznego.

Podzapytanie powiązane, jako przykład dynamicznego złączenia można łatwo rozpoznać po tym, że kolumna (kolumny) wyniku podzapytania jest porównywana z kolumną (kolumnami) wyniku zapytania zewnętrznego.

Przykład:

Za pomocą podzapytania powiązanego możemy rozszerzyć funkcjonalność opcji TOP i zwrócić w wyniku pozycje np. od 5 do 10 zamiast pięciu pierwszych:

```
USE Northwind
SELECT TOP 5 o.OrderID, SUM (UnitPrice * Quantity) AS [łączna wartość]
FROM dbo.Orders o INNER JOIN dbo.[Order Details] od
ON o.OrderID = od.OrderID
WHERE o.OrderID NOT IN
(SELECT TOP 5 o.OrderID
FROM dbo.Orders o INNER JOIN dbo.[Order Details] od
ON o.OrderID = od.OrderID
GROUP BY o.OrderID
ORDER BY SUM (UnitPrice * Quantity))
GROUP BY o.OrderID
ORDER BY [łączna wartość]
GO
OrderID łączna wartość
----------- ---------------------
```

```
10883 36.0000
10815 40.0000
10900 45.0000
11051 45.0000
11057 45.0000
(5 row(s) affected)
```

SQL Server wykonuje podzapytanie powiązane według następującego schematu:

1. Zapytanie zewnętrzne zwraca wiersz, który jest porównywany (w powyższym przykładzie za pomocą operatora NOT IN) z wynikiem podzapytania.
2. Podzapytanie jest wykonywane, a jego wynik zostaje przekazany do zapytania zewnętrznego. W tym przypadku, jeżeli zapytanie zewnętrzne zwróciło identyfikator jednego z pięciu najtańszych produktów (wynik podzapytania), to wynikiem porównania będzie fałsz (wartość warunku OrderID NOT IN OrderID dla tych produktów).
3. Zapytanie zewnętrzne zwraca kolejny wiersz i proces się powtarza.

> Jeżeli podzapytanie powiązane zwraca więcej niż jeden wiersz, użycie operatora = jest błędem semantycznym.

Kolejny przykład pokazuje, w jaki sposób możemy wybrać nazwy tych towarów, których dostępna liczba jest równa liczbie pojedynczego zamówienia złożonego u dostawcy o identyfikatorze równym 7:

```
SELECT ProductName
FROM dbo.Products
WHERE SupplierID=7 AND ProductID IN
(SELECT ProductID
FROM dbo.[Order Details]
WHERE UnitsInStock = Quantity)
GO
ProductName
----------------------------------------
Vegie-spread
Outback Lager
(2 row(s) affected)
```

Podzapytanie jako złączenie

Podzapytanie powiązane, choć jest funkcjonalnym odpowiednikiem złączenia tabel, może być wykonane przez SQL Server według innego planu. Chociaż wyniki wykonania dwóch kolejnych instrukcji są identyczne (oba zapytania zwracają gatunki książek wydawanych przez kilku wydawców), to plany ich wykonania różnią się od siebie. W pierwszym przypadku najpierw sortowane są według kolumny złączenia wartości zwrócone przez zapytanie zewnętrzne, po czym następuje złączenie wyników i

wyszukanie wierszy spełniających podane kryteria. W drugim optymalizator zadecydował o tym, że tabela t1 będzie zewnętrzną tabelą złączenia, pogrupował ją według wartości kolumny złączenia, złączył tabele (w obu przypadkach za pomocą prostego porównania wszystkich wartości z łączonych kolumn), posortował tabelę wewnętrzną złączenia według wartości kolumn i sprawdził, które wiersze wyniku spełniają podane warunki (w obu przypadkach użyty zastał indeks grupujący powiązany z kolumną title_id).

Przykładowe podzapytanie i plan jego wykonania:

```
USE pubs
SELECT DISTINCT t1.type
FROM titles AS t1
WHERE t1.type IN
(SELECT t2.type
FROM titles AS t2
WHERE t1.pub_id <> t2.pub_id)
GO
(2 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Sort(DISTINCT ORDER BY:([t1].[type] ASC))
|--Nested Loops(Left Semi Join, WHERE:([t1].[pub_id]<>[t2].[pub_id] AND
[t1].[type]=[t2].[type]))
|--Clustered Index Scan(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind] AS [t1]))
|--Clustered Index Scan(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind] AS [t2]))
(4 row(s) affected)
```

Zwracające ten sam wynik złączenie i plan jego wykonania:

```
USE pubs
SELECT DISTINCT t1.type
FROM titles AS t1
INNER JOIN titles AS t2
ON t1.type = t2.type
WHERE t1.pub_id <> t2.pub_id
GO
(2 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Stream Aggregate(GROUP BY:([t1].[type]))
|--Nested Loops(Inner Join, WHERE:([t2].[type]=[t1].[type] AND
[t1].[pub_id]<>[t2].[pub_id]))
|--Sort(ORDER BY:([t2].[type] ASC))
| |--Clustered Index Scan(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind] AS [t2]))
```

```
|--Clustered Index Scan(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind] AS [t1]))
(5 row(s) affected)
```

Podzapytania z operatorem EXISTS

Za pomocą operatorów EXISTS i NOT EXISTS możemy ograniczyć wynik zapytania zewnętrznego do wierszy odpowiednio zwróconych (lub nie) przez podzapytanie powiązane. Operator EXIST zwraca wartość logiczną 1 (prawda), jeżeli określony wiersz istnieje w wyniku podzapytania, operator NOT EXISTS — jeżeli wynik podzapytania nie zawiera określonego wiersza.

> Podzapytania z operatorem EXISTS lub NOT EXISTS nie zwracają żadnych danych, a jedynie wartości 1 lub 0.

Kolejne podzapytanie powiązane wyświetla wszystkie informacje o tych autorach, którzy mają to samo imię i nazwisko, a więc dane, które pomyłkowo mogły zostać kilkakrotnie wprowadzone do bazy:

```
USE pubs
SELECT au_id, au_fname,au_lname,phone
FROM dbo.authors as t1
WHERE EXISTS (SELECT au_id
FROM dbo.authors AS t2
WHERE T2.au_lname = T1.au_lname
AND T2.au_fname = T1.au_fname
AND T1.au_id <> T2.au_id)
GO
au_id au_fname au_lname phone
----------- -------------------- ---------------------------------------- ------------
(0 row(s) affected)
```

Kursory

Język Transact-SQL jest językiem strukturalnym, operującym na zbiorach danych. Jednak ci programiści, którzy mają doświadczenie w pisaniu programów w językach proceduralnych, takich jak C, wykorzystują swoje umiejętności, pisząc instrukcje, które za pomocą kursorów odwołują się do kolejnych rekordów. Chociaż wynik programu odwołującego się do nieuporządkowanego zbioru rekordów i programu pobierającego rekord po rekordzie w określonej kolejności może być taki sam, optymalizator SQL Servera nie znajdzie optymalnego planu wykonania instrukcji korzystających z kursorów i czas ich wykonania może być wielokrotnie dłuższy. Ponadto programy wykorzystujące kursory są z reguły dłuższe (choć dla programistów początkowo czytelniejsze) niż programy odwołujące się do zbiorów danych.

> Programy wykorzystujące kursory po stronie serwera powinny zostać zastąpione programami operującymi na zbiorach danych. Korzystając z podzapytań, złączeń lub

zmiennych można zastąpić program odwołujący się do danych rekord po rekordzie programem odwołującym się do zbioru danych.

Stosowanie kursorów pozwala na:

- wybieranie określonego wiersza ze zbioru wynikowego,
- pobieranie i modyfikowanie wybranych wierszy zbioru wynikowego.

Kursory mogą zostać utworzone za pomocą instrukcji języka Transact-SQL albo procedury API. W książce opisana została tylko pierwsza metoda.

Deklarowanie kursora

Kursor rezerwuje dla siebie zasoby systemowe, takie jak: pamięć operacyjna, tymczasowa przestrzeń na dysku, przestrzeń po stronie klienta (kursory strony klienta) czy przestrzeń po stronie serwera bazodanowego (kursory strony serwera).

Zadeklarować kursor i określić jego właściwości można za pomocą polecenia *DECLARE CURSOR*.

Składnia:

```
DECLARE nazwa CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR instrukcja_SELECT
[FOR UPDATE [OF kolumna [,...n]]]
```

gdzie:

LOCAL oznacza, że do kursora będzie można odwołać się wyłącznie z bieżącej procedury wsadowej, procedury składowanej lub wyzwalacza,
GLOBAL oznacza zakres kursora jako równy sesji użytkownika,
FORWARD_ONLY określa, że kursor będzie mógł być „przesuwany” jedynie od pierwszego do ostatniego rekordu. Jedynym sposobem pobrania danych poprzez tak zadeklarowany kursor jest wykonanie polecenia FETCH NEXT,
SCROLL umożliwia odwoływanie się do dowolnych rekordów zwróconych przez kursor,
STATIC powoduje utworzenie tymczasowej tabeli (w bazie tempdb) przechowującej rekordy kursora. Zmiany w tabeli źródłowej, które nastąpiły po jego zadeklarowaniu nie będą widoczne poprzez kursor,
KEYSET powoduje utworzenie tymczasowej tabeli keyset (w bazie tempdb) przechowującej identyfikatory tabeli źródłowej kursora. Aktualizacje rekordów, których identyfikatory zostały zapisane w tabeli keyset, będą widoczne poprzez kursor, natomiast dodanie nowych wierszy do tabeli źródłowej — nie,
DYNAMIC powoduje, że wszystkie zmiany w tabeli źródłowej będą widoczne poprzez kursor,
FAST_FORWARD powoduje utworzenie kursora typu FORWARD_ONLY umożliwiającego jedynie odczytywanie danych i zoptymalizowanego pod kątem odczytywania kolejnych rekordów.

Przykład:

```
USE pubs
DECLARE autorzy CURSOR FAST_FORWARD
FOR SELECT au_lname, au_fname, address
FROM dbo.authors
WHERE au_lname = 'Ringer'
GO
The command(s) completed successfully.
```

Typy kursorów

Typ kursora określa zarówno jego funkcjonalność, jak i sposób wykorzystania zasobów systemowych. SQL Server umożliwia zdefiniowanie kursorów czterech typów:

1. *FORWARD_ONLY* — umożliwiający jedynie pobieranie (poprzez instrukcję FETCH) kolejnych wierszy zbioru wynikowego. Innymi słowy, za pomocą kursora tego typu niemożliwe jest odczytanie danych, dopóki odpowiedni wiersz nie zostanie pobrany. Wszystkie zmodyfikowane lub dodane informacje są dostępne poprzez raz otwarty kursor typu FORWARD_ONLY.
2. *STATIC* — charakterystyczną cechą kursorów tego typu jest możliwość odczytywania tylko tych danych, które istniały w bazie danych w momencie otwarcia kursora. Kursory tego typu nie umożliwiają modyfikowania danych. Zbiór danych pobranych przez kursor przechowywany jest w bazie tempdb.
3. *DYNAMIC* — kursory tego typu, w przeciwieństwie do kursorów statycznych, uwzględniają wszystkie modyfikacje danych przeprowadzone przez użytkowników na danych źródłowych już po otwarciu kursora. Kursory tego typu otwierane są szybciej niż kursory statyczne.
4. *KEYSET* — kursory tego typu identyfikują poszczególne wiersze zbioru wynikowego na podstawie ich unikalnych identyfikatorów, które tworzą zbiór kluczy (ang. Keys). Zbiór danych kursora przechowywany jest w bazie tempdb. Kursory typu STATIC oraz KEYSET zapewniają największą wydajność przy łączeniu danych przechowywanych w różnych tabelach.

Blokady

SQL Server utrzymuje integralność danych między innymi poprzez blokowanie danych pobieranych za pomocą kursorów. Po utworzeniu kursora na pobraną daną wiersza zakładana jest blokada współdzielona (ang. Shared Lock). Jeżeli wystąpi żądanie zmodyfikowania określonych wierszy poprzez kursor, SQL Server blokuje te wiersze za pomocą blokady wyłącznej (ang. Exclusive Lock).

Tematowi blokad poświecony został rozdział 20.

Jeżeli podczas deklarowania kursora użyto opcji SCROLL_LOCKS, SQL Server zawsze zmodyfikuje dane poprzez kursor, nawet jeżeli inny użytkownik po otworzeniu kursora próbował je zmodyfikować.

Opcja OPTIMISTIC umożliwia innym użytkownikom zmodyfikowanie danych pobranych przez kursor. W tym wypadku, jeżeli wystąpi żądanie zmodyfikowania danych przez kursor, SQL Server na podstawie czasu przeprowadzania i kosztu wycofania

poszczególnych modyfikacji zadecyduje, które zmiany zostaną wprowadzone na stałe do bazy, a które zostaną wycofane.

Pobieranie danych

Zadeklarowany kursor musi zostać otwarty zanim będzie możliwe odwoływanie się do zwracanych przez niego danych. Otwarcie kursora powoduje wykonanie instrukcji języka Transact-SQL określonej w klauzuli FOR podczas deklarowania kursora.

Składnia:

```
OPEN {{[GLOBAL] nazwa} | @zmienna_kursora}
```

Przykład:

```
OPEN autorzy
GO
The command(s) completed successfully.
```

Dane zwracane prze kursor pobierane są za pomocą polecenie FETCH.

Składnia:

```
FETCH
[[NEXT | PRIOR | FIRST | LAST
| ABSOLUTE {n | @n}
| RELATIVE {n | @n}
]
FROM
]
{{[GLOBAL] nazwa} | @zmienna_kursora}
[INTO @zmienna [,...n]]
Przykład:
WHILE @@FETCH_STATUS <> -1
BEGIN
FETCH NEXT FROM autorzy
END
GO
au_lname au_fname address
---------------------------------------- -------------------- ------------------------
Ringer Anne 67 Seventh Av.
(1 row(s) affected)
au_lname au_fname address
---------------------------------------- -------------------- ------------------------
(0 row(s) affected)
```

Opcje kursora

Listę otwartych kursorów możemy poznać wywołując procedurę systemową sp_cursor_list:

```
DECLARE @kursor CURSOR
EXEC master.dbo.sp_cursor_list @cursor_return = @kursor OUTPUT,
@cursor_scope = 3
FETCH NEXT from @kursor
WHILE (@@FETCH_STATUS <> -1)
BEGIN
FETCH NEXT from @kursor
END
GO
reference_name cursor_name cursor_scope status model concurrency scrollable
open_status cursor_rows fetch_status column_count row_count last_operation
cursor_handle
------------- ------------ ------------ ------- ----- ------------- ---------- --------
---------- ------------ ---------- ------------ -------------- ---------------
autorzy autorzy 2 1 4 1 0 1
-1 -9 9 0 1 180150001
(1 row(s) affected)
```

Informacje o wybranym kursorze zwraca procedura systemowa sp_describe_cursor:

```
DECLARE @kursor CURSOR
EXEC sp_describe_cursor @cursor_return = @kursor OUTPUT,
@cursor_source = N'global', @cursor_identity = N'autorzy'
FETCH NEXT from @kursor
WHILE (@@FETCH_STATUS <> -1)
BEGIN
FETCH NEXT from @kursor
END
GO
reference_name cursor_name cursor_scope status model concurrency scrollable open_status
cursor_rows fetch_status column_count row_count last_operation cursor_handle
------------- ------------ ------------ ------- ----- ------------- ---------- --------
---------- ------------ ---------- ------------ -------------- ---------------
autorzy autorzy 2 1 4 1 0 1
-1 -9 9 0 1 180150001
(1 row(s) affected)
```

Listę wszystkich kolumn źródłowych i atrybutów kursora poznamy, wywołując procedurę systemową sp_describe_cursor_columns:

```
DECLARE @kursor CURSOR
EXEC sp_describe_cursor_columns @cursor_return = @kursor OUTPUT,
@cursor_source = N'global', @cursor_identity = N'autorzy'
FETCH NEXT from @kursor
WHILE (@@FETCH_STATUS <> -1)
BEGIN
FETCH NEXT from @kursor
END
GO
column_name ordinal_position column_characteristics_flags column_size data_type_sql
column_precision column_scale order_position order_direction hidden_column columnid
objectid dbid dbname
----------- ---------------- ---------------------------- ----------- ------------------
--------------- --------------- -------------- -------------- ----------- --------------
---------- ------- -----
au_id 0 0 11 257 0
0 0 NULL 0 1 1977058079 5 pubs
(1 row(s) affected)
...
```

Informacje o tabeli przechowującej dane pobierane przez kursor uzyskamy, wywołując procedurę systemową sp_describe_cursor_tables:

```
DECLARE @tabela CURSOR
EXEC sp_describe_cursor_tables @cursor_return = @tabela OUTPUT,
@cursor_source = N'global', @cursor_identity = N'autorzy'
FETCH NEXT from @tabela
WHILE (@@FETCH_STATUS <> -1)
BEGIN
FETCH NEXT from @tabela
END
GO
table_owner table_name optimizer_hint lock_type server_name objectid dbid dbname
------------------------------------------------------------ --------- ---- -------
dbo authors 0 0 LOLEK\MSSQL 1977058079 5 pubs
(1 row(s) affected)
```

Informacje o liczbie ostatnio pobranych wierszy oraz o wyniku ostatniej próby pobrania danych (próbie wykonania instrukcji FETCH) uzyskamy, wywołując odpowiednio jedną z dwóch funkcji systemowych: @@cursor_rows lub @@fetch_status:

```
SELECT @@cursor_rows, @@fetch_status
GO
```

```
----------- -----------
0 -1
(1 row(s) affected)
```

gdzie wartości zwrócone przez funkcję @@fetch_status oznaczają: -1 — niepowodzenie wykonania instrukcji FETCH, 0 — prawidłowe wykonanie instrukcji, -2 — nieznalezienie żądanego wiersza w zbiorze danych kursora.

Usuwanie kursora

Ponieważ otwarty kursor powoduje utrzymywanie blokady na danych źródłowych i zajmuje zasoby komputera, po pobraniu danych kursor powinien zostać zamknięty za pomocą polecenia CLOSE.

Składnia:

```
CLOSE {{[GLOBAL] nazwa} | @zmienna_kursora}
```

Przykład:

```
CLOSE autorzy
GO
The command(s) completed successfully.
```

Zamknięty i niewykorzystywany dłużej kursor powinien zostać usunięty za pomocą polecenia DEALLOCATE. W ten sposób tymczasowe obiekty powiązane z kursorem stworzone przez SQL Server zostaną usunięte, a powiązania kursora z jego nazwą czy zmienną kursora zostaną zerwane .

Przykład:

```
DEALLOCATE autorzy
GO
The command(s) completed successfully.
```

Poniższy program za pomocą kursorów strony serwera odczytuje nazwy wszystkich tabel użytkownika istniejących w bieżącej bazie danych:

```
DECLARE @t1 varchar(100)
DECLARE opis CURSOR FAST_FORWARD
FOR SELECT name
FROM dbo.sysobjects
WHERE type = 'U'
ORDER BY name
OPEN opis
FETCH NEXT FROM opis INTO @t1
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
PRINT @t1 + CHAR(13) + CHAR(10)
FETCH NEXT FROM opis INTO @t1
END
CLOSE opis
DEALLOCATE opis
GO
Categories
CustomerCustomerDemo
CustomerDemographics
Customers
dtproperties
Employees
EmployeeTerritories
Order Details
Orders
Products
Region
Shippers
Suppliers
Territories
```

Kurs Transact-SQL, część 4 (Optymalizacja zapytań)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Zapytania, tak jak pozostałe instrukcje języka Transact-SQL, przed skompilowaniem i wykonaniem są optymalizowane przez wewnętrzny proces SQL Servera o nazwie Query Optimizer. Jego zdaniem jest znalezienie najtańszego sposobu wykonania instrukcji. Query Optimizer bazuje na liczbie operacji wejścia-wyjścia oraz na liczbie obliczeń dokonanych przez procesior niezbędnych do wykonania instrukcji.

Query Optimizer szacuje koszt operacji wejścia-wyjścia na podstawie:

1. Struktury tabel przechowujących dane, do których odwołuje się instrukcja i istniejących, związanych z nimi indeksów.
2. Kosztu operacji złączenia różnego typu danych zapisanych w kilku tabelach źródłowych.
3. Istniejących lub tworzonych dynamicznie przez Query Optimizera statystyk opisujących dane źródłowe.
4. Metainformacji opisujących fizyczną strukturę plików, w których zapisane są żądane dane.

> Ponieważ Query Optimizer oblicza koszt wykonania instrukcji na podstawie statystyk, nieaktualne lub nieadekwatne statystki spowodują utworzenie nieefektywnego planu jej wykonania. Mechanizm tworzenia i aktualizacji statystyk został opisany w rozdziale 10.

Wynikiem optymalizacji jest znalezienie takiego sposobu wykonania instrukcji, który zwraca najmniejszą z możliwych liczbę wierszy i w związku z tym wymaga odczytania najmniejszej liczby stron (8 KB spójnych bloków danych). Celem optymalizacji jest także znalezienie takiego wykonywania, którego całkowity czas przeprowadzania jest najkrótszy.

> Fizyczna struktura baz danych została opisana w znajdującym się w. części poświęconej projektowaniu i tworzeniu baz danych rozdziale 8. W rozdziale 10. znajdują się szczegółowe wskazówki dotyczące tworzenia i wykorzystywania indeksów w celu poprawy wydajności zapytań.

Query Optimizer

Optymalizacja instrukcji języka Transact-SQL przebiega według następującego schematu:

1. Po sprawdzeniu poprawności syntaktycznej sprawdzana jest poprawność semantyczna (ang. Parse). Na tym etapie instrukcja zostanie „podzielona” na znaczniki interpretowane przez SQL Server.
2. Następuje standaryzacja — zapisanie znaczników instrukcji Transact-SQL w jednoznacznej postaci (np. ujednolicenie definiowania aliasów raz definiowanych za pomocą słowa kluczowego AS, raz bez niego). Na tym etapie wszystkie redundantne znaczniki instrukcji zostają usunięte.
3. Kolejnym etapem jest optymalizacja — wybór jednego z przygotowanych planów wykonania. Na tym etapie następuje analiza indeksów i statystyk oraz metod złączania danych. Etap można podzielić na trzy fazy:
    1. Analiza zapytania — wybór metod wyszukiwania i łączenia danych źródłowych mający na celu zminimalizowanie liczby danych, które muszą zostać odczytane do wykonania instrukcji.
    2. Wybór indeksów — na podstawie statystyk tabel i indeksów następuje wybór tych indeksów, do których odwołanie spowoduje zwrócenie najmniejszej liczby danych (minimalizacja liczby logicznych i fizycznych odczytów bloków danych).
    3. Wybór metody złączania tabel — na podstawie struktury i liczby danych oraz liczby pamięci operacyjnej niezbędnej do wykonania złączenia wybierany jest sposób złączania tabel. W tym momencie określone zostaje również, która tabela będzie traktowana jako wewnętrzna, a która jako zewnętrzna, w przypadku złączenia poprzez pętlę wyszukiwania.
4. Następnie instrukcja wykonana według opracowanego optymalnego planu zostaje skompilowana.
5. Query Optimizer wybiera optymalny sposób pobrania wybranych (wynikowych) danych. Na przykład: czy odwołać się do indeksu, czy odczytać tabelę — wbrew pozorom dla małych tabel odczytanie całej ich zawartości okazuje się tańszym sposobem na odczytanie danych niż odwoływanie się do nich poprzez indeksy.

Skompilowana według optymalnego planu instrukcja języka Transact-SQL zostaje zapisana w buforze procedury — przydzielonej przez SQL Server części pamięci operacyjnej wykorzystywanej wyłącznie do przechowywania skompilowanych procedur.

W buforze procedury mogą być przechowywane maksymalnie dwie wersje skompilowanej instrukcji — jedna wykonywana równolegle, druga szeregowo.

Ponieważ w pamięci procedury zapisana zostaje wyłącznie instrukcja języka Transact-SQL, bez informacji o tym, który użytkownik ją wykonał, odwołanie się do niej powoduje, że SQL Server musi wyznaczyć bieżący kontekst jej wykonania. Dlatego, jeżeli instrukcja zawiera wyłącznie identyfikatory obiektów, bez identyfikatora ich właściciela, SQL Server przyjmie, że właścicielem obiektów jest użytkownik wykonujący instrukcję, a dopiero potem sprawdzi, czy obiekty nie należą do użytkownika dbo. Wynika z tego, że jawne odwoływanie się do obiektów za pomocą nazwy użytkownika i nazwy obiektu przyspiesza wykonanie skompilowanych instrukcji.

Zoptymalizowana na podstawie nieaktualnych statystyk instrukcja nie będzie dłużej wykonywana według najlepszego planu. Również, jeżeli zmieniła się struktura obiektów, do których odwoływała się instrukcja, próba wykonania tej instrukcji zakończy się błędem. Z tych powodów skompilowana instrukcja jest w zależności od potrzeb rekompilowana. Ponowna kompilacja instrukcji jest przeprowadzana, jeżeli:

1. Zmieniona została definicja obiektu, do którego instrukcja się odwołuje (wykonano polecenie ALTER).
2. Wymuszono aktualizacje statystyk, na podstawie których przygotowany został plan wykonania instrukcji (wykonano instrukcję UPDATE STATISTIC).

3. Usunięto indeks wykorzystywany przez instrukcję (wykonano instrukcję DROP INDEX).
4. Z tabeli źródłowej usunięto lub dodano do niej dużą (względem stanu w momencie optymalizacji instrukcji) liczbę wierszy.
5. Wymuszono rekompilację instrukcji (wywołano procedurę systemową sp_recompile).

Plan wykonania zapytania

Plan wykonania instrukcji można poznać odczytując zawartość tabeli systemowej sysindexes. Jednak bezpośrednie odwoływanie się do tabel systemowych nie jest zalecane, a w tym przypadku istnieją inne sposoby uzyskania żądanych danych.

Query Analyzer pozwala na wyświetlenie statystyk związanych z czasem (włączenie opcji SET STATISTIC TIME) i liczbą operacji wejścia-wyjścia (opcja SET STATISTIC IO). Ponadto możemy wyświetlić opracowany przez Query Optimizera plan wykonania instrukcji (opcja SET SHOWPLAN TEXT lub SET SHOWPLAN_ALL). Na przykład po włączeniu dwóch pierwszych opcji wykonanie poniższego zapytania spowoduje wyświetlenie poniższych informacji:

```
USE pubs
SELECT DISTINCT t1.type
FROM titles AS t1
INNER JOIN titles AS t2
ON t1.type = t2.type
WHERE t1.pub_id <> t2.pub_id
GO
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
type
------------
business
psychology
(2 row(s) affected)
Table 'titles'. Scan count 19, logical reads 38, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

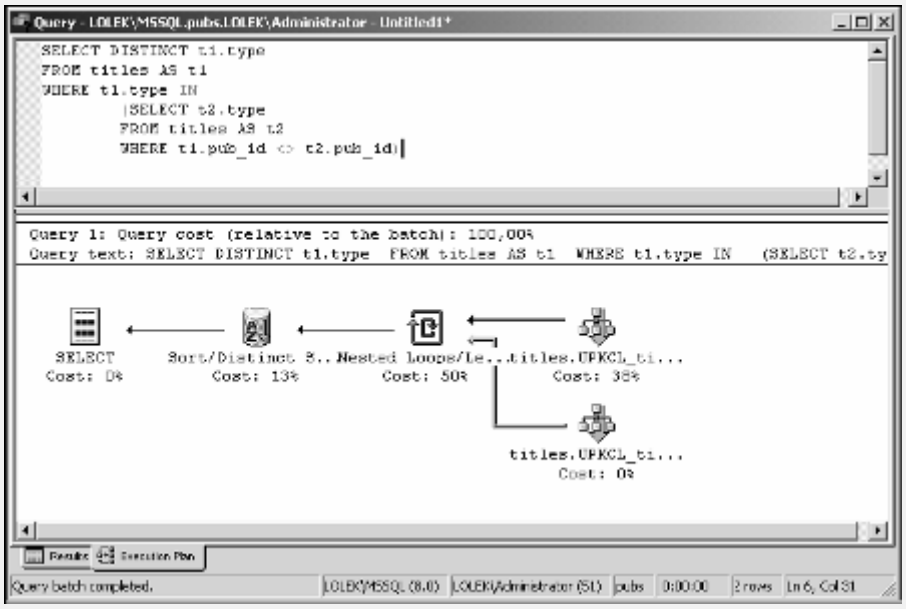
Natomiast włączenie opcji SET SHOWPLAN TEXT spowoduje wyświetlenie pokazanego w rozdziale 3. planu wykonania instrukcji.

Informacje na temat czasu wykonania poszczególnych opisanych wcześniej faz przetwarzania przez SQL Server instrukcji języka Transact-SQL nie wymagają komentarza. Przyjrzyjmy się informacjom związanymi z liczbą operacji wejścia-wyjścia:

1. *scan count* — określa liczbę odwołań do tabeli źródłowej.
2. *logical reads* — określa liczbę stron danych odczytanych z pamięci podręcznej.
3. *physical reads* — określa liczbę stron danych odczytanych z dysku. Ta liczba nigdy nie jest większa od liczby stron odczytanych z pamięci podręcznej. Na podstawie tych dwóch wartości można obliczyć współczynnik trafień: współczynnik trafień = (logical reads – physical reads) / logical reads
4. *read-ahead reads* — określa liczbę stron umieszczoną w pamięci podręcznej.

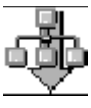
Sam plan wykonania instrukcji może również zostać przedstawiony w postaci graficznej (rysunek 4.1). Zawarte są w nim informacje dotyczące następujących zagadnień:

1. Kroki wykonywania instrukcji i ich kolejność.
2. Logiczne operatory algebry zbiorów użyte podczas wykonywania instrukcji.
3. Fizyczna implementacja tych operatorów wykorzystana do wykonania zapytania (tabela 4.1).



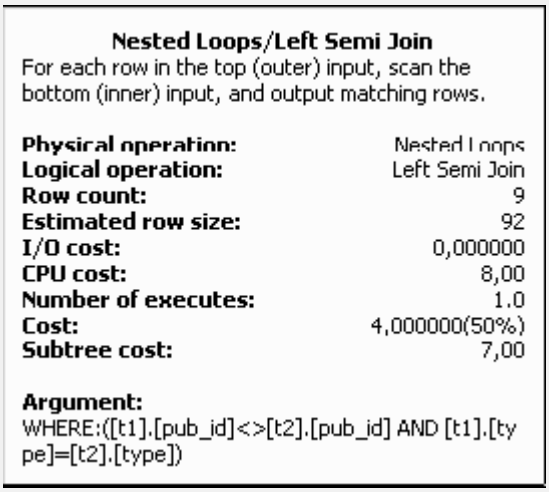
Rysunek 4.1. Plan wykonania zapytania

Tabela 4.1. Lista operatorów fizycznych

Symbol	Znaczenie	Opis
	Odczytanie indeksu	Wyszukanie (na podstawie podanych warunków) danych z indeksu niegrupującego
	Pętla wyszukiwania	Wyszukanie (z reguły w oparciu o indeks) w wewnętrznej tabeli złączenia wszystkich wierszy zgodnych z kolejnym wierszem zewnętrznej tabeli złączenia
	Porządkowanie danych	Uporządkowanie wszystkich danych źródłowych

	Przeszukanie indeksu	Wyszukanie w indeksie niegrupującym wierszy danych
	Przeszukanie tabeli	Wyszukanie w tabeli danych spełniających podane kryteria
	Przeszukanie wskaźników	Wyszukanie danych na podstawie identyfikatora wiersza lub klucza indeksu grupującego rekordy w tabeli lub w indeksie grupującym
	Wybieranie danych	Wyszukanie w zbiorze danych źródłowych rekordów spełniających podane kryteria
	Zgodność funkcji skrótu	Złączenie tabel na podstawie wyliczonej dla każdego wiersza wartości funkcji skrótu
	Złączenie przez połączenie	Dowolnego typu (z wyjątkiem złączenia tabeli z nią samą oraz złączenia krzyżowego) złączenie tabel lub widoków

Poszczególne kroki związane z wykonaniem instrukcji zapisywane są od strony prawej do lewej. W ramach każdego kroku możliwe jest wykonanie dowolnej liczby operacji — w tym przypadku odczytanie indeksów obu kolumn zostało wykonane jako jeden krok składający się z dwóch operacji. Związek pomiędzy poszczególnymi operacjami reprezentują strzałki. Ustawienie kursora na symbolu operatora fizycznego spowoduje wyświetlenie dodatkowych informacji o danej operacji (rysunek 4.2).



Rysunek 4.2. Informacje o złączeniu tabel t1 i t2

Wyświetlając dodatkowe informacje o każdej operacji poznamy:

- argumenty wywołania operacji (ang. Argument),
- koszt wykonania operacji i jego szacunkowy udział w koszcie wykonania instrukcji (ang. Cost),
- koszt wykonania operacji i operacji przez nią wywołanych (ang. Subtree cost),
- liczbę wykonania operacji w ramach instrukcji (ang. Number of executes),
- liczbę zwróconych przez operację wierszy (ang. Row count),
- szacunkową wielkość zwróconych przez operację wierszy (ang. Estimated row size),
- szacunkowy koszt operacji wejścia-wyjścia przeprowadzonych przez operację (ang. I/O cost),
- szacunkowy koszt wykorzystania zasobów procesora przez operację (ang. CPU cost).

Indeksy zawierające zapytania

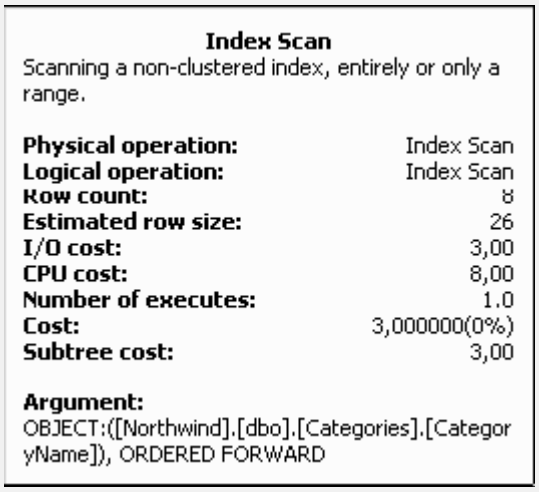
Indeksy różnego typu i sposoby ich tworzenia zostały opisane w rozdziale 10.

Optymalną pod względem szybkości odczytu danych jest sytuacja, w której wszystkie żądane dane (wyrażenia wymienione w instrukcji SELECT) mogą zostać odczytane z indeksu. Mówimy wtedy, że indeks zawiera zapytanie. Aby indeks zawierał zapytanie, wszystkie dane źródłowe muszą być poindeksowane. Dotyczy to kolumn wymienionych w poleceniu SELECT, w klauzuli WHERE, GROUP BY i ORDER BY. W takim przypadku pobranie danych sprowadza się do znalezienia i odczytania odpowiednich liści indeksu, bez konieczności odczytywania stron zawierających dane.

SQL Server pozwala na tworzenie indeksów dla danych będących wynikiem funkcji grupującej. Jeżeli zapytania dotyczą wyliczanych wartości utworzenie indeksów tego typu spowoduje wielokrotny wzrost wydajności zapytania, w ramach którego obliczane są te wartości.

Sprawdzić, czy istnieje indeks zawierający zapytanie, możemy wyświetlając graficzny plan jego wykonania, a następnie wyświetlając szczegóły operacji odczytania indeksu. Jeżeli znajduje się tam informacja: Scanning a non-clustered index entirely or only a range oznacza to, że do wykonania zapytania wykorzystano wyłącznie dane przechowywane w niegrupującym indeksie (rysunek 4.3).

Nie oznacza to, że najlepszym rozwiązaniem jest stworzenie indeksu zawierającego wszystkie kolumny wybranej tabeli — w takim przypadku utworzymy po prostu kopię tabeli i zamiast spodziewanego zysku wydajności uzyskamy jej spadek. Tworząc indeksy zawierające zapytania należy wybierać wyłącznie kolumny często występujące w zapytaniach i o podobnej wielkości (dodanie do indeksu zawierającego dane z trzech kolumn typu smallint danych z kolumny typu varchar (255) jest ekstremalnym przykładem źle zaprojektowanego indeksu).



Rysunek 4.3. Odczytanie nazw poszczególnych kategorii z tabeli categories

Niegrupujący indeks zawierający zapytanie jest funkcjonalnym odpowiednikiem indeksu grupującego i korzystanie z niego wiąże się z tymi samymi korzyściami — po znalezieniu pierwszej spełniającej podany warunek wartości nie potrzebne jest tworzenie wskaźników do zewnętrznych danych (tabeli), a ponieważ dane indeksu zapisane są w określonym porządku, wystarczy odczytać określoną liczbę stron przechowujących żądane dane.

Analiza zapytań

W zależności od typu zapytania, wykorzystanych operatorów logicznych czy metod łączenia tabel, Query Optimizer posłuży się odmiennymi szablonami umożliwiającymi znalezienie optymalnego planu wykonania instrukcji. Znajomość tych szablonów jest niezbędna do tworzenia wydajnych zapytań.

Aby wyniki uzyskane przez Czytelników nie różniły się od przedstawianych w książce przed wykonaniem opisywanych programów należy przywrócić oryginalną postać baz Northwind i pubs. Można to osiągnąć zatrzymując SQL Server i nadpisując pliki .mdf i .ldf tych baz plikami znajdującymi się na płycie instalacyjnej serwera.

SARG

Akronim SARG (ang. Search ARGuments) określa pewien specjalny podzbiór argumentów wyszukiwania, czyli wyrażeń wymienionych w klauzuli WHERE instrukcji SELECT. Argumenty SARG charakteryzuje:

- Obecność stałej, której wartość jest porównywana z polami wybranej kolumny tabeli źródłowej.
- Wyszukiwanie wartości równych wzorcowi, należących do zakresu wyznaczonego przez wzorzec lub przez połączenie kilku argumentów SARG za pomocą operatora koniunkcji.

Wynika z tego, że dla argumentów SARG dopuszczalnymi operatorami są: =, <, <=, >, >=, BETWEEN oraz, po spełnieniu dodatkowych warunków, LIKE. To, czy argument zawierający operator LIKE może zostać uznany za argument SARG, zależy od pozycji symbolu wieloznacznego (%). Jeżeli występuje on jako ostatni znak wzorca, czyli możliwe będzie ograniczenie liczby stron, które SQL Server będzie musiał odczytać, aby znaleźć żądane wartości, to taki argument będzie argumentem SARG.

W przypadku użycia operatora <>, NOT, !=, !>, !<, NOT EXISTS, NOT IN czy NOT LIKE konieczne okazuje się sprawdzenie wartości wszystkich wierszy tabeli źródłowej. Chociaż nie oznacza to, że SQL Server nie potrafi skorzystać z indeksów przy tworzeniu planu zapytania zawierającego wyżej wymienione operatory, to należy dążyć do zastąpienia takich operatorów argumentami SARG.

Wykonanie zapytania zawierającego argument SARG przebiega według następującego schematu:

1. Optymalizator sprawdza, czy istnieją przydatne do wykonania zapytania indeksy.
2. Jeżeli taki indeks zostanie znaleziony, rozpoczyna się wyszukiwanie (za pomocą operatora >=) stron indeksu przechowujących dane zgodne z żądanym wzorcem.

   Domyślnie dane indeksów zapisane są na dysku w porządku rosnącym.
3. Wszystkie wartości spełniające zadane kryteria są odczytywane, a jeśli jest to konieczne, odczytywane są z tabeli wartości przechowywane w pozostałych polach danego wiersza.

Porównajmy koszt i plan wykonania zapytania wykorzystującego argument SARG z zapytaniem zwracającym ten sam wynik, ale niezawierającym argumentu SARG:

```
USE Northwind
SELECT *
FROM dbo.Orders
```

```
WHERE CustomerID = 'GROSR'
GO
StmtText
------------------------------------------------------------------------------
|--Bookmark Lookup(BOOKMARK:([Bmk1000]), OBJECT:([Northwind].[dbo].[Orders]))
|--Index Seek(OBJECT:([Northwind].[dbo].[Orders].[CustomerID]),
SEEK:([Orders].[CustomerID]=Convert([@1])) ORDERED FORWARD)
(2 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 6, physical reads 0, read-ahead reads 0.
```

Ponieważ pobieramy wszystkie dane z wszystkich kolumn tabeli, optymalizator musi użyć wskaźnika do tabeli — utworzenie indeksu kompozytowego dla wszystkich kolumn tabeli spowodowałby jedynie pogorszenie wydajności. Jednak wybór wierszy spełniających zadane kryteria odbywa się poprzez odczytanie kolejnych liści indeksu, dzięki czemu SQL Server musi odczytać jedynie wybrane strony tabeli.

Gdyby liczba odczytanych stron była stosunkowo duża, tak jak na przykład w przypadku klienta o identyfikatorze FRANK, optymalizator zdecydowałby się na wykorzystanie indeksu grupującego i sprawdzanie warunku bezpośrednio na danych tabeli, bez wykorzystywania indeksu powiązanego z kolumną CustomerID:

```
USE Northwind
SELECT *
FROM dbo.Orders
WHERE CustomerID = 'FRANK'
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Orders].[PK_Orders]),
WHERE:([Orders].[CustomerID]=Convert([@1])))
(1 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
```

Wykonanie zapytania według wybranego przez optymalizator planu wymagało odczytania 21 stron danych. Przekonajmy się, czy próba wymuszenia użycia indeksu powiązanego z kolumną CustomerID poprawi wydajność:

```
USE Northwind
SELECT *
FROM dbo.Orders (INDEX = CustomerID)
WHERE CustomerID = 'FRANK'
GO
StmtText
------------------------------------------------------------------------------
|--Bookmark Lookup(BOOKMARK:([Bmk1000]), OBJECT:([Northwind].[dbo].[Orders]))
```

```
|--Index Seek(OBJECT:([Northwind].[dbo].[Orders].[CustomerID]),
SEEK:([Orders].[CustomerID]='FRANK ') ORDERED FORWARD)
(2 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 40, physical reads 0, read-ahead reads 0.
```

Jak widać, optymalizator właściwie oszacował liczbę operacji wejścia-wyjścia i wybrał optymalny plan wykonania zapytania. Jeżeli któryś z Czytelników udokumentuje i prześle do firmy Microsoft informację o tym, że wystąpił przypadek, w którym optymalizator podjął błędną decyzję, będzie to podstawą do poprawienia kodu programu i (czasami) do nagrodzenia użytkownika.

Wróćmy do porównania planu wykonania zapytania z operatorem SARG z planem wykonania zapytania zwracającego te same dane, ale niewykorzystującego operatora SARG:

```
USE Northwind
SELECT *
FROM dbo.Orders
WHERE CustomerID LIKE '%ROSR'
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Orders].[PK_Orders]),
WHERE:(like([Orders].[CustomerID], '%ROSR')))
(1 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 2 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Jeżeli czas wykonania zapytania jest tak krótki, że Query Optimizer szacuje go na 0 milisekund, informacja ta nie jest zamieszczana w książce.

Ponieważ na podstawie warunku CustomerID LIKE '%ROSR' niemożliwe jest oszacowanie liczby spełniających go stron (argument nie jest argumentem SARG), SQL Server zdecydował się na „pewniejszą” opcję i wybrał odczytanie indeksu grupującego.

Zapytania z operatorem AND

Zapytania wykorzystujące koniunkcję kilku warunków logicznych przetwarzane są następująco:

1. W pierwszej kolejności Query Optimizer zwraca wszystkie wiersze spełniające poszczególne kryteria wymienione w klauzuli WHERE;
2. Z otrzymanego zbioru kolejno usuwane są wiersze niespełniające kolejnych warunków.

Podczas przetwarzania zapytań tego typu Query Optimizer:

1. Wykorzysta wszelkie dostępne indeksy zawierające dane wymienione w klauzuli WHERE. W przypadku ich braku przeszukana zostanie cała tabela zwierająca odpowiednie dane.
2. Może wykorzystać różne indeksy, o ile każdy z nich zawiera fragment danych wymienionych w klauzuli WHERE.

Największy wzrost wydajności dla zapytań tego typu uzyskamy, tworząc co najmniej jeden indeks zawierający dane o silnie zróżnicowanych (najlepiej unikalnych) wartościach (a więc np. wielkość zamówienia, a nie identyfikatory dostawców, których w przykładowej bazie jest kilkunastu, a każdy z nich składa wiele zamówień różnej wielkości), do których odwołują się warunki z klauzuli WHERE.

Przykład:

Utworzymy kopię tabeli Order Details i wybierzemy z niej dane spełniające koniunkcje dwóch warunków logicznych. Następnie utworzymy dla naszej tabeli indeks zawierający dane z obu kolumn wymienionych w klauzuli WHERE i ponownie wykonamy zapytanie.

```
USE Northwind
SELECT *
INTO dbo.od
FROM dbo.[Order Details]
GO
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
SELECT *
FROM dbo.od
WHERE UnitPrice >100 AND Quantity<5
GO
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 1 ms.
CREATE INDEX i_od
ON dbo.od (UnitPrice, Quantity)
GO
SELECT *
FROM dbo.od
WHERE UnitPrice >100 AND Quantity<5
GO
Table 'od'. Scan count 1, logical reads 5, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
```

Pomimo tego, że nie utworzyliśmy indeksu zawierającego zapytanie, liczba odczytanych stron danych zmniejszyła się o połowę, chociaż tabela zawierająca najwięcej rekordów ze wszystkich tabel bazy Northwind, liczy jedynie nieco ponad 2000 rekordów.

Zapytania z operatorem OR

Zapytania wykorzystujące alternatywy kilku warunków logicznych przetwarzane są następująco:

1. W pierwszej kolejności Query Optimizer zwraca wszystkie wiersze spełniające poszczególne kryteria wymienione w klauzuli WHERE.
2. Do otrzymanego zbioru kolejno dodawane są wiersze niespełniające wcześniejszych warunków.

Podczas przetwarzania zapytań tego typu Query Optimizer:

1. Wykorzysta wszelkie dostępne indeksy zawierające wszystkie lub część danych wymienionych w klauzuli WHERE. W przypadku braku niegrupującego indeksu powiązanego z choćby jednym warunkiem logicznym przeszukana zostanie cała tabela zawierająca odpowiednie dane;
2. Może wykorzystać różne indeksy, o ile każdy z nich zawiera fragment danych wymienionych w klauzuli WHERE.

> Operator IN na etapie standaryzacji jest przekształcany na odpowiadające mu wyrażenie z operatorami OR — zapytania z tym operatorem są przetwarzane w ten sam sposób co zapytania z operatorem OR.

W wypadku zapytań z operatorem OR istnienie indeksów zawierających wszystkie dane wymienione w klauzuli WHERE zapobiegnie (z wyjątkiem sytuacji, w której tabela przechowująca dane jest na tyle mała, że taniej jest odczytać wszystkie strony danych przechowujące dane z tabeli niż wybierać na podstawie odczytanych indeksów) przeszukaniu całej tabeli oraz zmniejszy liczbę operacji arytmetycznych związanych z sortowaniem i porównywaniem danych.

Przykład:

```
USE Northwind
DROP TABLE dbo.od --jeżeli przykładowa tabeli istnieje, usuwamy ją
SELECT *
INTO dbo.od
FROM dbo.[Order Details]
CREATE INDEX i_odup
ON dbo.od (UnitPrice)
GO
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
SELECT *
FROM dbo.od
WHERE UnitPrice >100 OR Quantity<5
```

```
GO
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 2 ms, elapsed time = 2 ms.
```

Wyświetlmy jeszcze plan wykonania instrukcji:

```
StmtText
------------------------------------------------------------------------------
|--Table Scan(OBJECT:([Northwind].[dbo].[od]), WHERE:([od].[UnitPrice]>100.00 OR
[od].[Quantity]<5))
```

Jak widać, optymalizator przeszukał całą tabelę pomimo tego, że istniał indeks powiązany z danymi wykorzystywanymi w jednym z warunków logicznych. Dodajmy teraz indeks powiązany z drugą tabelą przechowującą dane. na podstawie których wybierany jest wynik zapytania i ponownie wyświetlmy statystyki i plan wykonania instrukcji:

```
CREATE INDEX i_odq
ON dbo.od (Quantity)
GO
SELECT *
FROM dbo.od
WHERE UnitPrice >100 OR Quantity<5
GO
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 3 ms.
```

Liczba odczytanych stron danych się nie zmieniła, co sugeruje, że i tym razem przeszukana została cała tabela. Czas wykonania instrukcji wyraźnie się skrócił, w przeciwieństwie do czasu jej przetworzenia, co sugeruje, że optymalizator tym razem był w stanie opracować bardziej wydajny, choć nieco bardziej skomplikowany, plan jej wykonania:

```
StmtText
------------------------------------------------------------------------------
|--Table Scan(OBJECT:([Northwind].[dbo].[od]), WHERE:([od].[UnitPrice]>100.00 OR
[od].[Quantity]<5))
```

Ponieważ tabela od jest niewielka (mieści się na 10 stronach pamięci) i tym razem optymalizator zdecydował się na przeszukanie wszystkich jej wierszy. Mimo to czas wykonania zapytania uległ skróceniu.

Zapytania pobierające dane z kilku źródeł danych

Jeżeli zapytanie odwołuje się do kilku tabel lub widoków, pierwszą wartością, którą oszacuje optymalizator jest liczba wierszy zwróconych w wyniku złączenia obiektów źródłowych. Liczba ta zależy odwrotnie proporcjonalnie od unikalności wartości danych wykorzystanych do złączenia obiektów (łącząc tabele za pomocą kolumn przechowujących dane np. o województwie otrzymamy (statystycznie) większy zbiór wynikowy niż łącząc tabele za pomocą kolumn przechowujących dane o adresie konkretnego dostawcy).

Ponadto na podstawie szacunkowej ziarnistości danych optymalizator ocenia, ile danych zostanie zduplikowanych w wyniku złączenia i na tej podstawie wybiera typ indeksu wykorzystanego do złączenia:

- W przypadku danych o małej ziarnistości wykorzystany będzie w pierwszej kolejności indeks niegrupujący, w drugiej — grupujący.
- W przypadku danych o dużej ziarnistości wykorzystany będzie wyłącznie indeks grupujący.

Złączenia pośrednie

Wynik złączenia dowolnej liczby obiektów obliczany jest jako suma złączeń par obiektów. Każde złączenie pośrednie może zostać wykonane za pomocą operacji innego typu, wybranego przez optymalizator dla dwóch łączonych obiektów. Dla każdego złączenia pośredniego niezależnie wyznaczana jest tabela zewnętrzna i wewnętrzna złączenia.

Kolejność wykonywania złączeń pośrednich wyznaczana jest przez Query Optimizera i nie musi odpowiadać kolejności wymienionej w instrukcji SELECT.

Klauzula WHERE

Jeżeli zapytanie zawiera klauzulę WHERE, optymalizator może zadecydować o wybraniu wierszy spełniających podane kryteria, zanim wykona operację złączenia. W ten sposób wielokrotnie zmniejsza się liczba wierszy, które będą łączone.

Przykład:

Pierwsze zapytanie zwraca wszystkie rekordy łączonych tabel, drugie — jedynie wybrane rekordy z obu tabel. Pokazano statystyki związane z wykonaniem zapytań i plan ich wykonania:

```
USE Northwind
SELECT LastName, OrderID
FROM dbo.Orders o JOIN dbo.Employees e
ON e.EmployeeID = o.EmployeeID
GO
Table 'Orders'. Scan count 9, logical reads 19, physical reads 0, read-ahead reads 0.
Table 'Employees'. Scan count 1, logical reads 1, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 34 ms.
SQL Server parse and compile time:
```

```
CPU time = 0 ms, elapsed time = 0 ms.
StmtText
------------------------------------------------------------------------------
|--Nested Loops(Inner Join, OUTER REFERENCES:([e].[EmployeeID]))
|--Index Scan(OBJECT:([Northwind].[dbo].[Employees].[LastName] AS [e]))
|--Index Seek(OBJECT:([Northwind].[dbo].[Orders].[EmployeeID] AS [o]),
SEEK:([o].[EmployeeID]=[e].[EmployeeID]) ORDERED FORWARD)
(3 row(s) affected)
USE Northwind
SELECT LastName, OrderID
FROM dbo.Orders o JOIN dbo.Employees e
ON e.EmployeeID = o.EmployeeID
WHERE e.FirstName LIKE 'A%' AND Freight<10
GO
Table 'Employees'. Scan count 176, logical reads 352, physical reads 0, read-ahead reads 0.
Table 'Orders'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 2 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
StmtText
------------------------------------------------------------------------------
|--Nested Loops(Inner Join, OUTER REFERENCES:([o].[EmployeeID]))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Orders].[PK_Orders] AS [o]),
WHERE:([o].[Freight]<10.00))
|--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Employees].[PK_Employees] AS
[e]), SEEK:([e].[EmployeeID]=[o].[EmployeeID]), WHERE:(like([e].[FirstName],
A%')) ORDERED FORWARD)
(3 row(s) affected)
```

W drugim przypadku obie tabele najpierw zostały przeszukane pod kątem zgodności z kryteriami podanymi w klauzuli WHERE (stąd duża liczba przeszukań tabel), ale w rezultacie uzyskano wielokrotnie mniejszy zbiór wierszy, które zostały złączone.

Pętla wyszukiwania

Złączenie przeprowadzone za pomocą operatora pętli wyszukiwania (ang. Nested Loop) jest tym sposobem łączenia tabel, o którym myślimy w pierwszej kolejności, szczególnie jeżeli piszemy programy w językach takich jak C czy Basic. Złączenie tego typu polega na porównaniu przez SQL Server każdego wiersza w wewnętrznej tabeli złączenia (decyzja o tym, która tabela zostanie uznana za wewnętrzną jest podejmowana przez usługę Query Optimizer na podstawie liczby wierszy, ich unikalności i ziarnistości) z kolejnymi wierszami zewnętrznej tabeli złączenia. Najgorszym z możliwych scenariuszów jest ten, w którym nie istnieją ani indeksy powiązane z kolumnami, według których następuje złączenie, ani indeksy związane z kolumnami wymienionymi w klauzuli WHERE — w takim przypadku zbiór danych

źródłowych jest równy całym tabelom. Nie wnikając w opisaną w dalszej części książki strukturę fizycznych plików bazy danych, spróbujemy obliczyć liczbę stron danych, które muszą zostać odczytane w przypadku, gdy tabela t1, składająca się z w1 wierszy zapisanych na p1 stron, zostanie złączona z tabelą t2 zawierającą w2 wierszy zapisanych na p2 stronach — SQL Server musi odczytać p1 + w1 ∗ s2 stron danych. Oznacza to, że łącząc niewielkie tabele zawierające odpowiednio: tabela t2 (wewnętrzna) — 100 stron, tabela t1 (zewnętrzna) — 5 000 wierszy zapisanych na 200 stronach, SQL Server musi odczytać ponad 500 000 stron, czyli ponad 4 MB danych.

Rozwiązaniem jest utworzenie indeksu grupującego na kolumnie wykorzystanej do łączenia tabel. Ponieważ dane w tabeli są uporządkowane według wartości indeksu grupującego, nie ma potrzeby przeszukiwania całej tabeli. W przypadku wewnętrznej tabeli złączenia SQL Server może odczytać jedynie stronę zawierającą dokładnie ten wiersz, który odpowiada kolejnemu wierszowi tabeli zewnętrznej. W ten sposób zastępujemy w naszym wzorze wartość s2 stałą wyliczoną na podstawie ilości poziomów indeksu (będzie to liczba z zakresu od 1 do 3). Oznacza to, że złączenie naszych przykładowych tabel wymaga już jedynie (w najgorszym razie) odczytania 15 200 stron, czyli około 120 KB danych — ponad 30-krotnie mniej.

Dodatkowo, jeżeli na zewnętrzną tabelę złączenia nałożono warunek WHERE, możemy ograniczyć liczbę wierszy odczytanych z tej tabeli, tworząc niegrupujący indeks dla kolumny, według której wybierane są dane.

Query Optimizer tworzy co najmniej cztery plany wykonania złączenia uwzględniające różną rolę tabel w złączeniu oraz kolejność wybierania danych i realizuje plan najtańszy.

Złączenie za pomocą operatora pętli wyszukiwania wybierane jest, jeżeli zewnętrzna tabela złączenia zawiera dużą ilość wierszy, a wewnętrzna jest mała lub zawiera użyteczne indeksy.

Łączenie

Ograniczeniem złączenia przez pętle wyszukiwania jest niewykorzystywanie ewentualnie istniejących indeksów dla kolumny łączącej tabeli zewnętrznej. Niestety podczas projektowania i strojenia bazy danych nie mamy pewności, która z tabel zostanie wybrana przez optymalizator jako tabela zewnętrzna złączenia.

Jeżeli utworzymy indeks grupujący dla kolumny łączącej w obu złączonych tabelach (czyli obie kolumny będą posortowane według tej samej kolumny) SQL Server będzie mógł złączyć tabele poprzez łączenie (ang. Merge). Złączenie tego typu polega na odczytaniu i porównaniu kolejnych wierszy obu tabel.

Złączenie poprzez łączenie wybierane jest wtedy, gdy obie tabele są posortowane według kolumny złączenia.

Łączenie tabel powiązanych związkiem typu jeden do wielu i jeden do jednego przebiega dokładnie według opisanego schematu. Natomiast łączenie tabel powiązanych związkiem typu wiele do wielu wymaga utworzenia tymczasowej tabeli do przechowywania kolejnych wierszy, zanim zostaną sprawdzone pozostałe wiersze drugiej tabeli i zapadnie decyzja o włączeniu lub odrzuceniu danego wiersza z wyniku złączenia. Ponieważ wielokrotnie zwiększa to liczbę operacji wejścia-wyjścia, optymalizator z reguły nie decyduje się na złączenie poprzez łączenie, jeżeli przynajmniej jedna tabela wykorzystywana do złączenia nie zawiera wartości niepowtarzalnych.

Przykład:

Tworzymy kopię tabel Orders i Orders Detail, następnie indeks grupujący dla kolumn OrderID i wykonujemy proste zapytanie łączące obie utworzone tabele:

```
USE Northwind
SELECT *
INTO dbo.o
FROM dbo.Orders
SELECT *
INTO dbo.od
FROM dbo.[Order Details]
CREATE CLUSTERED INDEX i_o
ON o(OrderID)
CREATE CLUSTERED INDEX i_od
ON od(OrderID)
GO
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
SELECT *
FROM o JOIN od
ON o.OrderID = od.OrderID
GO
Table 'od'. Scan count 830, logical reads 1672, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 22, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 10 ms, elapsed time = 98 ms.
```

Liczba odczytanych stron pozwala przypuszczać, że tabele zostały złączone za pomocą pętli wyszukiwania. Wyświetlając plan wykonania przekonamy się, że nasze obawy są słuszne — ponieważ żaden z indeksów grupujących nie został zdefiniowany jako unikalny, optymalizator, który nie analizuje danych zapisanych w obu tabelach pod kątem ich niepowtarzalności, zdecydował się na takie złączenie tabel.

```
StmtText
------------------------------------------------------------------------------
|--Nested Loops(Inner Join, OUTER REFERENCES:([o].[OrderID]))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[o].[i_o]))
|--Clustered Index Seek(OBJECT:([Northwind].[dbo].[od].[i_od]),
SEEK:([od].[OrderID]=[o].[OrderID]) ORDERED FORWARD)
(3 row(s) affected)
```

Zmieńmy definicję jednego indeksu i ponownie wykonajmy zapytanie, porównując koszty jego wykonania:

```
CREATE UNIQUE CLUSTERED INDEX i_o
ON o(OrderID)
WITH DROP_EXISTING
GO
SELECT *
FROM o JOIN od
ON o.OrderID = od.OrderID
GO
Table 'od'. Scan count 1, logical reads 11, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 10 ms, elapsed time = 236 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
StmtText
------------------------------------------------------------------------------
|--Merge Join(Inner Join, MERGE:([o].[OrderID])=([od].[OrderID]),
RESIDUAL:([od].[OrderID]=[o].[OrderID]))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[o].[i_o]), ORDERED FORWARD)
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[od].[i_od]), ORDERED FORWARD)
(3 row(s) affected)
```

Liczba odczytanych stron danych zmniejszyła się z 1694 do 32 (prawie 53 razy), a liczba wykonanych operacji przeszukiwania indeksu z 830 do 1! Pomimo tak spektakularnego zmniejszenia liczby operacji wejścia-wyjścia czas wykonania instrukcji uległ wydłużeniu, co związane jest z tym, że wszystkie potrzebne dane były zapisane w pamięci podręcznej, a same tabele były raczej niewielkie.

Złączenie przez łączenie jest na tyle wydajnym sposobem złączania tabel, że optymalizator może zdecydować się na to rozwiązanie, nawet jeżeli jedna z łączonych tabel nie jest posortowana według odpowiedniej kolumny. W takim przypadku najpierw następuje posortowanie tabeli, a następnie jej złączenie.

Zgodność funkcji skrótu

Złączenie na podstawie zgodności funkcji skrótu (ang. Hash) przeprowadzane jest, jeżeli optymalizator nie może znaleźć użytecznych dla złączenia indeksów. Brak indeksów oznacza między innymi, że dane zawarte w tabelach źródłowych nie są posortowane. O ile w przypadku niewielkich danych ich posortowanie i złączenie przez łączenie może okazać się najtańszym rozwiązaniem, o tyle dla tabel zawierających miliony wierszy próba ich posortowania byłaby operacją wyjątkowo kosztowną. Również sekwencyjne porównywanie wszystkich wierszy jednej tabeli z kolejnymi wierszami drugiej tabeli okazuje się zbyt kosztownym rozwiązaniem. Pozostaje podzielenie danych na grupy, co jest operacją szybszą niż pełne ich posortowanie, a jednocześnie pozwalającą ograniczyć liczbę operacji wejścia-wyjścia.

Dane zawarte w tabelach źródłowych dzielone są na grupy według wartości obliczonych dla kolejnych wierszy funkcji skrótu. Każda grupa zawiera dane, dla których wyliczona

wartość była taka sama, więc jeżeli nastąpi porównanie na podstawie wartości funkcji skrótu, wystarczy, że SQL Server sprawdzi zgodność z wierszami z wybranej grupy.

Na przykład, jeżeli funkcją skrótu byłaby funkcja modulo 13, dane liczbowe zostałyby podzielone na 13 grup (możliwe wartości funkcji modulo 13 należą do zbioru <0, 12>). W rzeczywistości funkcje skrótu stosowane przez SQL Server są bardziej skomplikowane, a liczba grup może sięgać kilku tysięcy. Dzieląc dane źródłowe pomiędzy grupy i porównując, wiersz po wierszu, dane z jednego zbioru wejściowego z odpowiadającymi im danymi ze zbioru danych drugiej tabeli, SQL Server może złączyć obie tabele, wykonując tylko jedną operację przeszukania tabeli (jak w przypadku złączenia przez łączenie).

Przykład:

Wykonamy kopie tabel Orders i Orders Detail i połączymy nowo utworzone, pozbawione jakichkolwiek indeksów tabele:

```
USE Northwind
DROP TABLE o --jeżeli tabele istniały
DROP TABLE od --należy je usunąć
SELECT *
INTO dbo.o
FROM dbo.Orders
SELECT *
INTO dbo.od
FROM dbo.[Order Details]
GO
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
SELECT *
FROM o JOIN od
ON o.OrderID = od.OrderID
GO
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 20 ms, elapsed time = 244 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
StmtText
------------------------------------------------------------------
|--Hash Match(Inner Join, HASH:([o].[OrderID])=([od].[OrderID]))
|--Table Scan(OBJECT:([Northwind].[dbo].[o]))
|--Table Scan(OBJECT:([Northwind].[dbo].[od]))
(3 row(s) affected)
```

Liczba odczytanych stron jest równie mała jak dla złączenia przez łączenie, ale czas wykonania operacji uległ wydłużeniu.

W dokumentacji technicznej SQL Servera możemy przeczytać, że złączenie na podstawie zgodności funkcji skrótu powinno być stosowane jedynie dla zapytań ad hoc i świadczy o konieczności optymalizacji (strojenia) bazy danych.

Wybór najlepszego indeksu

Optymalizacja zapytań związana jest bezpośrednio z optymalizacją bazy danych i nie ma ogólnego schematu pozwalającego na osiągnięcie maksymalnej wydajności konkretnego zapytania. Jedynym rozwiązaniem jest testowanie różnych rozwiązań i porównywanie wyników. Właściwie wyglądający proces testowania i optymalizacji prowadzący do wyboru najlepszego indeksu dla zapytania wybierającego dane z dwóch tabel opisano poniżej.

W poniższych przykładach dodatkowo została włączona opcja wyświetlania graficznego planu wykonania poszczególnych zapytań, co wpłynęło na wydłużony czas ich wykonania i zwiększenie liczby operacji wejścia-wyjścia.

Należy utworzyć linię bazową zawierającą dane, z którymi będziemy porównywali wydajność zapytania korzystającego z różnych indeksów. Kolejno przedstawiono instrukcje (częściowo wypełnione losowymi danymi) tworzące kopię dwóch tabel bazy Northwind oraz zapytanie bazowe.

Tworzymy kopię tabel pozbawionych jakichkolwiek indeksów:

```
USE Northwind
DROP TABLE o -- jeżeli tabele istniały
DROP TABLE od -- należy je usunąć
SELECT *
INTO dbo.o
FROM dbo.Orders
SELECT *
INTO dbo.od
FROM dbo.[Order Details]
GO
```

Tworzymy zapytanie bazowe pobierające dane z tabel o i od:

```
SELECT o.OrderID, CustomerID, OrderDate, ProductID, UnitPrice, Discount
FROM o JOIN od
ON o.OrderID = od.OrderID
WHERE CustomerID IN ('SUPRD', 'HANAR', 'RICSU')
AND UnitPrice >30
GO
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
```

```
CPU time = 10 ms, elapsed time = 22 ms.
```

Statystyki wykonania zapytania bazowego, przede wszystkim liczba milisekund, którą SQL Server potrzebował na jego wykonanie (10 ms), są wartościami bazowymi, które należy poprawić. Niewielka liczba odczytanych stron danych wynika z tego, że SQL Server połączył obie tabele na podstawie zgodności funkcji skrótu.

W opisywanym przykładzie wszystkie argumenty wymienione w klauzuli WHERE to argumenty SARG. Są minimalnym zbiorem argumentów niezbędnym do przeprowadzenia zaplanowanej selekcji, więc pozostaje nam utworzyć brakujące indeksy.

W pierwszej kolejności dodamy indeksy do kolumn łączących tabelę. W przypadku tabeli o będzie to indeks unikalny, ponieważ tworzymy go dla kolumny będącej kluczem głównym tabeli:

```
CREATE UNIQUE INDEX o_zl ON o(OrderID)
CREATE INDEX od_zl ON od(OrderID)
GO
The command(s) completed successfully.
```

Ponowne wykonanie zapytania bazowego zwróci następujące informacje:

```
Table 'od'. Scan count 50, logical reads 235, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 21 ms.
```

Czas wykonania zapytania uległ wyraźnemu skróceniu, ale liczba odczytanych stron danych jest wielokrotnie większa. Sprawdźmy, jaki wpływ na wydajność zapytania będzie miało zastąpienie utworzonych indeksów indeksami utworzonymi dla kolumn wymienionych w klauzuli WHERE:

```
DROP INDEX o.o_zl
DROP INDEX od.od_zl
CREATE INDEX o_sz ON o(CustomerID)
CREATE INDEX od_sz ON od(UnitPrice)
GO
```

Po wykonaniu zapytania okazuje się, że poindeksowanie kolumn, według których wyszukiwane są dane, zamiast kolumn, według których łączone są tabele, nie przyniosło żadnego wzrostu wydajności:

```
Table 'od'. Scan count 1, logical reads 10, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 10 ms, elapsed time = 20 ms.
```

Kolejna próba będzie polegała na utworzeniu indeksów wspólnych dla obu kolumn (tej wykorzystywanej do złączenia i tej wykorzystywanej do wyszukiwania danych) i na ponownym wykonaniu zapytania bazowego:

```
DROP INDEX o.o_sz
DROP INDEX od.od_sz
CREATE UNIQUE INDEX o_ws ON o(OrderID,CustomerID)
CREATE INDEX od_ws ON od(OrderID,UnitPrice)
GO
Table 'od'. Scan count 50, logical reads 209, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 10 ms, elapsed time = 17 ms.
```

Jak widać, utworzenie wspólnego indeksu dla obu kolumn nie wpłynęło w znaczący sposób na wydajność zapytania w porównaniu z indeksem utworzonym dla kolumny, według której łączone są tabele. Czytelnicy mogą samodzielnie przekonać się, że zmiana kolejności kolumn w definicji indeksu wspólnego również nie przynosi żadnych korzyści. Ponieważ to zapytanie wybiera wszystkie dane z łączonych tabel, nie możemy utworzyć indeksu zawierającego zapytanie, który z pewnością gwarantowałby jego najszybsze wykonanie. Zamiast tego spróbujemy utworzyć indeksy grupujące dla kolumn złączenia i niegrupujące dla kolumn przechowujących dane, według których wyszukiwane są wiersze:

```
DROP INDEX o.o_ws
DROP INDEX od.od_ws
CREATE UNIQUE CLUSTERED INDEX o_zl ON o(OrderID)
CREATE CLUSTERED INDEX od_zl ON od(OrderID)
CREATE INDEX o_sz ON o(CustomerID)
CREATE INDEX od_sz ON od(UnitPrice)
GO
Table 'od'. Scan count 50, logical reads 101, physical reads 0, read-ahead reads 0.
Table 'o'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 16 ms.
```

Ostatnie rozwiązanie okazuje się najlepsze. Chociaż liczba odczytanych stron świadczy wyraźnie o tym, że złączenie zostało dokonane poprzez pętle wyszukiwania, to początkowy wybór łączonych danych i porównywanie posortowanych danych zaowocowało krótkim czasem wykonania zapytania przy stosunkowo niewielkiej liczbie odczytanych stron danych.

Zapytania grupujące dane

Zapytania zawierające klauzulę GROUP BY lub DISTINCT wykonywane są według tego samego planu — w obu przypadkach SQL Server zwraca jeden rekord reprezentujący wiersze przechowujące te same wartości. Różnica pomiędzy klauzulami sprowadza się do tego, że w przypadku klauzuli GROUP BY możliwe jest użycie funkcji grupującej, zwracającej jedną wartość na podstawie dowolnej liczby przekazanych argumentów. Wspólne cechy obu klauzul obrazują poniższe zapytania, które nie tylko zwracają te same dane, ale także są wykonane według tego samego planu:

```
USE Northwind
SELECT CustomerID
FROM dbo.Orders
GROUP BY CustomerID
GO
StmtText
------------------------------------------------------------------------------
|--Stream Aggregate(GROUP BY:([Orders].[CustomerID]))
|--Index Scan(OBJECT:([Northwind].[dbo].[Orders].[CustomerID]), ORDERED FORWARD)
(2 row(s) affected)
SELECT DISTINCT CustomerID
FROM dbo.Orders
GO
StmtText
------------------------------------------------------------------------------
|--Stream Aggregate(GROUP BY:([Orders].[CustomerID]))
|--Index Scan(OBJECT:([Northwind].[dbo].[Orders].[CustomerID]), ORDERED FORWARD)
(2 row(s) affected)
```

Aby uwidocznić różnice pomiędzy klauzulami, wykonajmy zapytanie, które oprócz identyfikatorów klientów zwraca datę ostatniego zamówienia:

```
USE Northwind
SELECT CustomerID, MAX (OrderDate)
FROM dbo.Orders
GROUP BY CustomerID
GO
StmtText
------------------------------------------------------------------------------
|--Hash Match(Aggregate, HASH:([Orders].[CustomerID]),
RESIDUAL:([Orders].[CustomerID]=[Orders].[CustomerID])
DEFINE:([Expr1002]=MAX([Orders].[OrderDate])))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Orders].[PK_Orders]))
(2 row(s) affected)
```

Plan wykonania tego zapytania różni się od poprzednich — w tym wypadku SQL Server przeprowadzi grupowanie dla funkcji skrótu (ang. Hash Aggregation), które (w przeciwieństwie do odczytania indeksu) nie spowoduje uporządkowania zwracanych danych.

Inną techniką wykorzystywana do wykonania zapytania zawierającego funkcję grupującą jest grupowanie dla danych (ang. Stream Aggregation) — w tym przypadku najpierw nastąpi posortowanie danych, następnie usunięcie duplikatów i obliczenie wartości funkcji grupującej.

Do wersji 7.0 SQL Server grupował dane i obliczał wartości funkcji grupujących wyłącznie poprzez grupowanie dla danych. W rezultacie wynik zawsze zawierał dane uporządkowane. W wersji 7.0 i następnych dla osiągnięcia takiego samego rezultatu konieczne jest dodanie klauzuli ORDER BY.

Analizując plan wykonania dwóch pierwszych zapytań zauważymy, że zostały one wykonane poprzez odczytanie indeksu zawierającego żądane dane. Dodatkowo wpis ORDERED FORWARD świadczy o tym, że zostały sekwencyjnie odczytane uporządkowane liście indeksu. W rezultacie SQL Server nie potrzebował ponownie sortować danych.

Wykonajmy zapytanie zwracające liczbę pracowników związanych z realizowaniem zamówień poszczególnych klientów. W tym przypadku w bazie Northwind nie ma indeksu zawierającego wynik zapytania:

```
USE Northwind
SELECT CustomerID, count (EmployeeID)
FROM dbo.Orders
GROUP BY CustomerID
GO
StmtText
------------------------------------------------------------------------------------
|--Compute Scalar(DEFINE:([Expr1002]=Convert([Expr1006])))
|--Hash Match(Aggregate, HASH:([Orders].[CustomerID]),
RESIDUAL:([Orders].[CustomerID]=[Orders].[CustomerID])
DEFINE:([Expr1006]=COUNT_BIG([Orders].[EmployeeID])))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Orders].[PK_Orders]))
(3 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 21, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 7 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Porównajmy koszt i plan wykonania tego zapytania po dodaniu brakującego indeksu:

```
CREATE INDEX orders_c_e
ON Orders (CustomerID, EmployeeID)
GO
```

```
SELECT CustomerID, count (EmployeeID)
FROM dbo.Orders
GROUP BY CustomerID
GO
StmtText
------------------------------------------------------------------------------
|--Compute Scalar(DEFINE:([Expr1002]=Convert([Expr1006])))
|--Stream Aggregate(GROUP BY:([Orders].[CustomerID])
DEFINE:([Expr1006]=COUNT_BIG([Orders].[EmployeeID])))
|--Index Scan(OBJECT:([Northwind].[dbo].[Orders].[orders_c_e]), ORDERED
FORWARD)
(3 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Po dodaniu niegrupującego indeksu zawierającego zapytanie koszt jego wykonania zmniejszył się wielokrotnie. Ponieważ zapytania zawierające funkcje grupujące rzadko pobierają dane z wielu kolumn, najlepszym rozwiązaniem jest utworzenie niegrupującego indeksu zawierającego takie zapytanie albo indeksu grupującego, który sortuje dane według wartości kolumny wymienionej w klauzuli GROUP BY lub po słowie kluczowym DISTINCT.

Kurs Transact-SQL, część 5 (Wyszukiwanie pełnotekstowe)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Usługa Microsoft Search pozwala na wyszukiwanie danych tekstowych na podstawie ich zgodności z pojedynczymi słowami, frazami czy różnymi formami danego słowa. Dzięki temu możemy tworzyć bardziej skomplikowane warunki wyszukiwania niż za pomocą operatora LIKE. Drugą zaletą wyszukiwania pełnotekstowego jest łatwość pobierania fragmentów danych zapisanych w pojedynczych kolumnach.

Na przykład policzenie ile razy w kolumnie ContactTitle występuje słowo Manager wymaga napisania niezbyt przejrzystej instrukcji wywołującej dwie systemowe funkcje tekstowe i jedną grupującą:

```
USE Northwind
SELECT SUM((LEN(ContactTitle)-LEN(REPLACE(ContactTitle,'Manager','')))
/LEN('Manager'))
FROM dbo.Suppliers
GO
-----------
13
(1 row(s) affected)
```

Usługa Microsoft Search

SQL Server wykorzystuje do wyszukiwania pełnotekstowego obiekty dwóch typów:

1. Indeks wyszukiwania pełnotekstowego (ang. full-text index) przechowujący poszczególne słowa występujące w tabeli, dla której został stworzony. Indeks wyszukiwania pełnotekstowego, tak jak indeksy innych typów, wykorzystywany jest do wyszukiwania danych spełniających określone kryteria. Indeks wyszukiwania pełnotekstowego może zostać stworzony jedynie dla tabeli zawierającej kolumnę przechowująca unikalne i określone wartości, np. tabeli, dla której został zdefiniowany prosty klucz główny.
2. Katalog wyszukiwania pełnotekstowego (ang. full-text catalog). Indeksy wyszukiwania pełnotekstowego zapisane są w katalogu wyszukiwania pełnotekstowego — folderze utworzonym na dysku NTFS, do którego uprawnienia posiada wyłącznie administrator komputera i usługa Microsoft Search. Z reguły jeden katalog wyszukiwania pełnotekstowego tworzony jest dla jednej bazy danych.

Zadaniem usługi Microsoft Search jest tworzenie indeksów wyszukiwania pełnotekstowego i zarządzanie nimi oraz wykorzystywanie tych indeksów w zapytaniach.

Indeksy wyszukiwania pełnotekstowego

Informacje o indeksach wyszukiwania pełnotekstowego można uzyskać za pomocą:

- procedur systemowych,
- funkcji języka Transact-SQL,
- konsoli Enterprise Menager.

Procedury systemowe związane z usługą Microsoft Serach

Wszystkie przykłady z tego rozdziału dotyczą tabeli Suppliers, dla której utworzono indeks wyszukiwania pełnotekstowego powiązany z kolumnami: CompanyName, ContactName i ContactTitle. Indeksy tego typu można tworzyć za pomocą konsoli Enterprise Manager.

sp_help_fulltext_catalogs

Zwraca informacje o wybranym katalogu wyszukiwania pełnotekstowego.
Składnia:

```
sp_help_fulltext_catalogs [@fulltext_catalog_name =] 'nazwa'
```

Przykład:

```
sp_help_fulltext_catalogs Northwind
GO
ftcatid NAME PATH STATUS
NUMBER_FULLTEXT_TABLES
-------- --------- ------------------------------- ---------- ----------- ------------
----
5 Northwind D:\Program Files\Microsoft SQL Server\MSSQL$MSSQL\FTData 0 1
(1 row(s) affected)
```

gdzie:
STATUS oznacza jeden z poniższych stanów katalogu:
0 — bezczynny,
1 — trwa pełne wypełnianie,
2 — zatrzymany z powodu niewystarczających zasobów systemowych,
3 — wstrzymany,
4 — odtwarzany,
5 — wyłączony,
6 — trwa wypełnianie przyrostowe,
7 — trwa aktualizacja indeksów.

sp_help_fulltext_tables

Zwraca listę tabel, dla których utworzono indeks wyszukiwania pełnotekstowego.
Składnia:

```
sp_help_fulltext_tables [[@fulltext_catalog_name =] 'nazwa'] [, [@table_name =] 'tabela']
```

Przykład:

```
sp_help_fulltext_tables
GO
TABLE_OWNER TABLE_NAME FULLTEXT_KEY_INDEX_NAME FULLTEXT_KEY_COLID
FULLTEXT_INDEX_ACTIVE FULLTEXT_CATALOG_NAME
--------- ----------- --------------------- ------------------ ---------------------- -
------------------
dbo Suppliers PK_Suppliers 1 1
Northwind
(1 row(s) affected)
```

Funkcje języka Transact-SQL związane z usługą Microsoft Search

> Składnia poszczególnych funkcji została przedstawiona w rozdziale 2.

COLUMNPROPERTY

Argument IsFullTextIndexed przyjmuje wartość 1 dla kolumny, dla której istnieje indeks wyszukiwania pełnotekstowego i 0 dla pozostałych kolumn.
Przykład:

```
USE Northwind
SELECT COLUMNPROPERTY (OBJECT_ID ('dbo.Suppliers'),'ContactTitle','IsFullTextIndexed')
GO
-----------
1
(1 row(s) affected)
```

DATABASEPROPERTY

Argument IsFullTextEnabled przyjmuje wartość 1, jeżeli istnieje indeks wyszukiwania pełnotekstowego skojarzony z dowolną tabelą określonej bazy danych, a w przeciwnym wypadku przyjmuje wartość 0.
Przykład:

```
SELECT DATABASEPROPERTY ('Northwind','IsFullTextEnabled')
GO
-----------
1
(1 row(s) affected)
```

INDEXPROPERTY

Argument IsFulltextKey przyjmuje wartość 1, jeżeli istnieje indeks wyszukiwania pełnotekstowego skojarzony z dowolną tabelą określonej bazy danych, a w przeciwnym wypadku przyjmuje wartość 0.

```
USE Northwind
SELECT INDEXPROPERTY (OBJECT_ID('dbo.Suppliers'),'PK_Suppliers','IsFulltextKey')
GO
-----------
1
(1 row(s) affected)
```

OBJECTPROPERTY

Argument TableHasActiveFulltextIndex przyjmuje wartość 1, jeżeli istnieje indeks wyszukiwania pełnotekstowego skojarzony z daną tabelą, a w przeciwnym wypadku przyjmuje wartość 0.
Przykład:

```
USE Northwind
SELECT OBJECTPROPERTY (OBJECT_ID('dbo.Suppliers'),'TableHasActiveFulltextIndex ')
GO
-----------
1
(1 row(s) affected)
```

FULLTEXTCATALOGPROPERTY

Zwraca informacje o katalogu wyszukiwania pełnotekstowego.
Przykład:

```
SELECT FULLTEXTCATALOGPROPERTY ('Northwind','UniqueKeyCount')
GO
-----------
32
```

```
(1 row(s) affected)
```

FULLTEXTSERVICEPROPERTY

Zwraca informacje o usłudze wyszukiwania pełnotekstowego.
Przykład:

```
SELECT FULLTEXTSERVICEPROPERTY ('ResourceUsage')
GO
-----------
3
(1 row(s) affected)
```

Zapytania pełnotekstowe

Zapytania wykorzystujące indeksy wyszukiwania pełnotekstowego są rozszerzeniem języka Transact-SQL względem standardu SQL-2 i umożliwiają wybieranie danych na podstawie słów lub fraz.

Predykat CONTAINS

Predykat umożliwia wybranie wierszy zgodnych z podanym wzorcem na podstawie wyszukiwania pełnotekstowego. Wyszukiwane mogą być: wyrazy lub frazy, prefiksy oraz słowa zbliżone do wzorca na podstawie ich wymowy lub budowy.
Składnia:

```
CONTAINS
({kolumna | *}, '<warunek_wyszukiwania_typu_contains>'
)
<warunek_wyszukiwania_typu_contains> ::=
{<słowo>
| <prefiks>
| <generator>
| <wzorzec>
| <wartość>
}
| {(<warunek_wyszukiwania_typu_contains>)
{AND | AND NOT | OR} <warunek_wyszukiwania_typu_contains> […n]
}
<słowo> ::=
słowo | "fraza"
< prefiks > ::=
{"słowo*" | "fraza*"}
```

```
<generator> ::=
FORMSOF (INFLECTIONAL, <słowo> [,…n])
<wzorzec> ::=
{<słowo> | <prefiks>}
{{NEAR | ~} {<słowo> | <prefiks>}} [,…n]
<wartość> ::=
ISABOUT
({{
<słowo>
| <prefiks>
| <generator>
| <wzorzec>
}
[WEIGHT (wartość)]
} [,…n]
)
```

gdzie:
słowo jest określonym zbiorem słów lub fraz, na podstawie których zostaną wyszukane rekordy,
prefiks jest fragmentem słowa lub frazy, na podstawie których zostaną wyszukane rekordy,
generator jest zbiorem słów utworzonych na podstawie podanego słowa poprzez jego odmianę, na podstawie których zostaną wyszukane rekordy, wzorzec jest zbiorem słów o zbliżonym znaczeniu do wzorca, na podstawie których zostaną wyszukane rekordy,
wartość określa liczbową reprezentację słowa (ang. weighted value), na podstawie której zostaną wyszukane rekordy.
Przykłady:

```
USE Northwind
SELECT ContactName, CompanyName
FROM dbo.Suppliers
WHERE CONTAINS(ContactName, 'Marie')
GO
ContactName CompanyName
------------------------------ ----------------------------------------
Marie Delamare Escargots Nouveaux
(1 row(s) affected)
USE Northwind
GO
SELECT CompanyName, ContactName, ContactTitle
FROM dbo.Suppliers
WHERE CONTAINS(ContactTitle, '"Adm*"') AND CONTAINS(ContactName, '"v*"')
GO
```

```
CompanyName ContactName ContactTitle
---------------------------------------- ------------------------------ ----------------
Cooperativa de Quesos 'Las Cabras' Antonio del Valle Saavedra Export Administrator
(1 row(s) affected)
```

Predykat CONTAINS wyszukuje teksty zgodne z podanym wzorem bez uwzględniania wielkości liter.

```
USE Northwind
GO
SELECT ContactName, ContactTitle
FROM dbo.Suppliers
WHERE CONTAINS(*, 'FORMSOF (INFLECTIONAL, market)')
GO
ContactName ContactTitle
------------------------------ ------------------------------
Beate Vileid Marketing Manager
Ian Devling Marketing Manager
Mayumi Ohno Marketing Representative
Sven Petersen Coordinator Foreign Markets
Martin Bein International Marketing Mgr.
Yoshi Nagase Marketing Manager
Carlos Diaz Marketing Manager
Jean-Guy Lauzon Marketing Manager
(8 row(s) affected)
```

Znak * oznacza, że wzorzec (w tym wypadku odmiany słowa market) będzie wyszukany we wszystkich kolumnach indeksu pełnotekstowego skojarzonego z tabelą.

Predykat FREETEXT

Predykat zwraca wiersze zawierające słowa lub frazy o znaczeniu zbliżonym do znaczenia wzorca. SQL Server dzieli ciąg znaków wzorca na pojedyncze słowa, wyszukuje słowa znaczące i porównuje każde z nich z danymi źródłowymi.

Składnia:

```
FREETEXT ({kolumna | *}, 'wzorzec')
```

Przykład:

```
USE Northwind
SELECT CompanyName, ContactTitle
```

```
FROM dbo.Suppliers
WHERE FREETEXT (*, 'Exotic or Nord Coordinating, Owner')
GO
CompanyName ContactTitle
---------------------------------------- ------------------------------
Nord-Ost-Fisch Handelsgesellschaft mbH Coordinator Foreign Markets
Leka Trading Owner
Exotic Liquids Purchasing Manager
(3 row(s) affected)
```

Funkcja CONTAINSTABLE

Zwraca tabelę składającą się z dowolnej liczby wierszy spełniających określone kryteria wyszukiwania pełnotekstowego.
Składnia:

```
CONTAINSTABLE (tabela, {kolumna | *}, ' <warunek_wyszukiwania> '
[, pierwsze_n_wierszy])
<warunek_wyszukiwania> ::=
{<słowo>
| <prefiks>
| <generator>
| <wzorzec>
| <wartość>
}
| {(<warunek_wyszukiwania>)
{AND | AND NOT | OR} <warunek_wyszukiwania> [,…n]
}
<słowo> ::=
słowo | "fraza"
<prefiks> ::=
{"słowo*" | "fraza*"}
<generator> ::=
FORMSOF (INFLECTIONAL, <słowo> [,…n])
<wzorzec> ::=
{<słowo> | <prefiks>}
{{NEAR | ~} {<słowo> | <prefiks>}} [,…n]
<wartość> ::=
ISABOUT
({{
<słowo>
| <prefiks>
```

```
| <generator>
| <wzorzec>
}
[WEIGHT (wartość)]
} [,…n]
)
```

Funkcja CONTAINSTABLE i predykat CONTAINS wykorzystują takie same warunki wyszukiwania.

Przykład:

```
USE Northwind
SELECT t1.CompanyName, t1.ContactTitle,t2.RANK
FROM dbo.Suppliers AS t1 INNER JOIN
CONTAINSTABLE (dbo.Suppliers, *,
'(Order NEAR Adm*) OR
("Ma*")'
, 10
) AS t2
ON t1.SupplierID = t2.[KEY]
WHERE t2.RANK>16
GO
CompanyName ContactTitle RANK
---------------------------------------- ------------------------------ -----------
Escargots Nouveaux Sales Manager 80
G'day, Mate Sales Representative 80
Mayumi's Marketing Representative 80
Plutzer Lebensmittelgroßmärkte AG International Marketing Mgr. 80
Ma Maison Marketing Manager 64
Nord-Ost-Fisch Handelsgesellschaft mbH Coordinator Foreign Markets 64
(6 row(s) affected)
```

Funkcja FREETEXTTABLE

Zwraca tabelę składającą się z dowolnej liczby wierszy zawierających ciąg znaków przekazany jako kryterium wyszukiwania.

Funkcja FREETEXTTABLE i predykat FREETEXT wykorzystują takie same warunki wyszukiwania.

Przykład:

```
USE Northwind
SELECT t1.CompanyName, t1.ContactTitle,t2.RANK
```

```
FROM dbo.Suppliers AS t1 INNER JOIN
FREETEXTTABLE(dbo.Suppliers, *,
'Exotic or Nord Coordinating, Owner') AS t2
ON t1.SupplierID = t2.[KEY]
GO
CompanyName ContactTitle RANK
---------------------------------------- ------------------------------ -----------
Leka Trading Owner 90
Nord-Ost-Fisch Handelsgesellschaft mbH Coordinator Foreign Markets 63
Exotic Liquids Purchasing Manager 63
(3 row(s) affected)
```

1. Ponieważ funkcje CONTAINSTABLE i FREETEXTTABLE zwracają tabelę liczącą 0 lub więcej wierszy, muszą zostać wymienione w klauzuli FROM.
2. Zwracana tabela zawiera kolumnę KEY zawierającą identyfikatory indeksu wyszukiwania pełnotekstowego.
3. Zwracana tabela zawiera kolumnę RANK zawierającą liczby z przedziału od 0 do 1000 określające stopień zgodności wiersza z warunkiem wyszukiwania.

Kurs Transact-SQL, część 6 (Modyfikowanie danych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Informacje przechowywane w bazie danych mogą być aktualizowane, dodawane lub usuwane. Modyfikowanie danych może być przeprowadzone za pomocą instrukcji języka Transact-SQL UPDATE, INSERT; usuwanie poprzez wydanie instrukcji DELETE lub TRUNCATE. Ponieważ informacje w bazach danych mogą być równocześnie modyfikowane przez wielu użytkowników, w celu zapewnienia ich spójności operacje te przeprowadzane są w ramach transakcji.

Przetwarzanie transakcyjne

Transakcja jest sekwencją operacji traktowanych jak pojedyncza, niepodzielna jednostka, która musi zostać w całości przeprowadzona lub w całości wycofana.

SQL Server pozwala na rozpoczęcie transakcji w jednym z trzech trybów:

1. Jawne rozpoczęcie transakcji (ang. Explicit) — poprzez wydanie polecenia BEGIN TRANSACTION.
2. Automatyczne rozpoczęcie i zatwierdzenie transakcji (ang. Autocommit) — poprzez wykonanie dowolnej instrukcji języka Transact-SQL. Ten tryb jest domyślnym trybem SQL Servera.
3. Niejawne rozpoczęcie transakcji (ang. Implicit) — poprzez włączenie opcji IMPLICIT_TRANSACTIONS. W tym trybie wykonanie kolejnej instrukcji spowoduje automatyczne rozpoczęcie kolejnej transakcji.

Transakcja może zostać zatwierdzona poleceniem COMMIT lub wycofana poceniem ROLLBACK. W pierwszym wypadku wyniki wszystkich operacji składających się na transakcję zostaną na stałe zapisane w bazie, w drugim wszystkie zmiany danych zostaną wycofane, a baza wróci do stanu z momentu rozpoczęcia transakcji. W obu przypadkach ewentualne blokady zostają zdjęte, a zasoby systemowe zwolnione.

Wystąpienie niektórych błędów krytycznych SQL Servera, na przykład zakleszczenia operacji spowoduje automatyczne wycofanie transakcji. Jednak w większości przypadków programista musi samodzielnie napisać program wycofujący błędną transakcję.

Zakleszczenie występuje, jeżeli jedna transakcja T1 po zablokowaniu rekordu R1 chce zmodyfikować rekord R2, który został zablokowany przez inną transakcję T2, która z kolei próbuje uzyskać wyłączny dostęp do rekordu R1. W takiej sytuacji obie transakcje „czekają” nawzajem na zwolnienie blokady, które jednak nigdy nie nastąpi. SQL Server wykrywa zakleszczenia i wycofuje jedną z zakleszczonych transakcji. Wycofana zostaje ta transakcja, której ponowne wykonanie wiąże się z mniejszą liczbą operacji.

Groźniejsze od automatycznie niwelowanych zakleszczeń jest wystąpienie łańcucha blokad utworzonego przez transakcje, które czekając na zwolnienie zasobów, utrzymują blokady na innych zasobach (do tych zasobów odwołują się kolejne transakcje, blokujące inne zasoby itd.) W takiej sytuacji cała baza danych może zostać zablokowana przez jedną rozpoczętą i niezakończoną transakcję (np. transakcję modyfikującą rekordy w kluczowej tabeli rozpoczętą przez użytkownika, który wyszedł z biura i zapominał nacisnąć klawisz OK). Taka sytuacja musi zostać wykryta i rozwiązana przez administratora bazy danych.

Poniższy, klasyczny przykład pokazuje, w jaki sposób dzięki przetwarzaniu transakcyjnemu unikniemy sytuacji, w której z powodu błędu lub awarii, kwota odjęta z konta A nie zostałaby dodana do konta B:

```
USE tempdb
CREATE TABLE konto_A (
klient_id int NOT NULL,
stan int)
CREATE TABLE konto_B (
klient_id int NOT NULL,
stan int)
GO
INSERT INTO konto_A
VALUES (1,2000)
INSERT INTO konto_B
VALUES (1,0)
GO
BEGIN TRANSACTION
UPDATE konto_A
SET stan = stan - 500
WHERE klient_id = 1
IF @@ERROR <> 0
BEGIN
RAISERROR ('Wystąpił błąd w trakcie przesyłania pieniędzy. Operacja zostaje
wycofana',15,1)
ROLLBACK TRANSACTION
END
UPDATE konto_B
SET stan = stan + 500
WHERE klient_id = 1
IF @@ERROR <> 0
BEGIN
RAISERROR ('Wystąpił błąd w trakcie przesyłania pieniędzy. Operacja zostaje
wycofana', 15,1)
ROLLBACK TRANSACTION
END
COMMIT TRANSACTION
```

```
SELECT *
FROM konto_a JOIN konto_b
ON konto_A.klient_id = konto_B.klient_id
GO
klient_id stan klient_id stan
----------- ----------- ----------- -----------
1 1500 1 500
(1 row(s) affected)
```

Wstawianie danych

Język Transact-SQL pozwala na wstawianie danych do pojedynczego wiersza wybranej tabeli, wstawianie dowolnej liczby wierszy wybranych w zapytaniu oraz jednoczesne z wstawianiem danych tworzenie tabeli docelowej.

Instrukcja INSERT

W przeciwieństwie do rozbudowanej instrukcji SELECT, składnia pozostałych instrukcji języka Transact-SQL będzie przedstawiona w jednym punkcie, bez dodatkowego rozbicia na poszczególne klauzule.

Wstawianie pojedynczego wiersza

Dodać pojedynczy wiersz do tabeli możemy za pomocą instrukcji INSERT zawierającej klauzule VALUES.
Składnia:

```
INSERT [INTO]
{tabela WITH (<optymalizacja> [,…n])
| widok
| funkcja_typu_RowSet
}
{ [(lista_kolumn)]
{VALUES
({DEFAULT | NULL | wyrażenie} [,…n])
| instrukcja_SELECT
| dyrektywa_EXECUTE
}
}
| DEFAULT VALUES
< optymalizacja > ::=
{FASTFIRSTROW
| HOLDLOCK
| PAGLOCK
```

```
| READCOMMITTED
| REPEATABLEREAD
| ROWLOCK
| SERIALIZABLE
| TABLOCK
| TABLOCKX
| UPDLOCK
}
```

gdzie:
lista_kolumn jest opcjonalnym określeniem nazw kolumn, do których dopisywane będą dane. Niepodanie listy kolumn spowoduje dodawanie wartości do kolumn według porządku ich występowania w definicji tabeli docelowej.

1. Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę. Na przykład próba dodania rekordu bez podania wartości dla kolumny z nałożonym zawężeniem NOT NULL zakończy się błędem.
2. Dane poszczególnych typów muszą być wyszczególnione w klauzuli VALUES zgodnie z konwencją nazewniczą stosowaną przez SQL Server, na przykład daty muszą zostać zapisane w apostrofach.

Przykład:

```
USE pubs
INSERT INTO dbo.authors (au_id, au_fname, au_lname, contract)
VALUES ('999-99-9999', 'Szczepan', 'Szczęśliwy', 1)
GO
SELECT *
FROM dbo.authors
WHERE au_id >= '999-99-9998'
GO
-------------------------------------------------------------------------------
LOLEK\Administrator
(1 row(s) affected)
au_id au_lname au_fname phone address city state zip contract
----------- ---------- -------- -------- --------- ------- ------ ------- --------
999-99-9999 Szczęśliwy Szczepan UNKNOWN NULL NULL NULL NULL 1
(1 row(s) affected)
```

Informacja o tym, który użytkownik dodał rekord do tabeli, została wyświetlona przez wyzwalacz dbo.t_d (jego działanie polega na wywołaniu funkcji bezpieczeństwa SUSER_SNAME).

Wstawianie danych wybranych w zapytaniu

Instrukcja INSERT z klauzulą SELECT pozwala na dodanie do tabeli dowolnej liczby wierszy wybranych w zapytaniu.

1. Wynik zapytania w całości zostanie dodany do tabeli docelowej.
2. Tabela docelowa musi zostać wcześniej stworzona.
3. Próba dodania danych innych typów niż typy kolumn tabeli docelowej zakończy się błędem.
4. Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę.

Przykład:

```
USE Northwind
INSERT dbo.Customers (CustomerID, ContactName, CompanyName)
SELECT EmployeeID, FirstName + ' ' + LastName, Title
FROM dbo.Employees
WHERE EmployeeID < 3
GO
(2 row(s) affected)
SELECT CustomerID, ContactName, CompanyName
FROM dbo.Customers
WHERE CustomerID < '3'
GO
CustomerID ContactName CompanyName
---------- ------------------------------ ----------------------------------------
1 Nancy Davolio Sales Representative
2 Andrew Fuller Vice President, Sales
(2 row(s) affected)
```

Niewymienienie kolumn tabeli docelowej spowoduje dodanie danych do wszystkich jej kolumn w kolejności określonej podczas tworzenia tabeli.

Tworzenie tabeli za pomocą instrukcji SELECT INTO

Instrukcja SELECT INTO umożliwia utworzenie tabeli, do której zostaną wstawione dane wybrane w zapytaniu.

Przechowywanie wyników zapytania w tymczasowej tabeli jest często spotykanym sposobem na podzielenie skomplikowanego zapytania pomiędzy kilka prostszych i szybciej wykonywanych zapytań.

1. Instrukcja SELECT INTO tworzy tabelę docelową o podanej nazwie, więc próba jej wykonania dla istniejącej tabeli skończy się błędem.
2. Konieczne jest podanie nazwy kolumny lub stworzenie aliasu dla wszystkich wyrażeń wymienionych w poleceniu SELECT.

3. Język Transact-SQL pozwala na tworzenie lokalnych lub globalnych tabel tymczasowych:

- ♣ Nazwa lokalnej tabeli tymczasowej, dostępnej w ramach sesji użytkownika, poprzedzona jest pojedynczym znakiem #.
- ♣ Nazwa globalnej tabeli tymczasowej, dostępnej dla wszystkich użytkowników, poprzedzona jest dwoma znakami ##.

Przykład:

```
USE Northwind
SELECT ProductID, ProductName, UnitPrice, UnitPrice*0.22 AS Vat
INTO ##Podatki
FROM dbo.Products
WHERE CategoryID = 5
GO
SELECT *
FROM ##Podatki
ORDER BY Vat
GO
(7 row(s) affected)
ProductID ProductName UnitPrice Vat
---------- ---------------------------------------- --------------------- -----------
52 Filo Mix 7.0000 1.540000
23 Tunnbröd 9.0000 1.980000
42 Singaporean Hokkien Fried Mee 14.0000 3.080000
57 Ravioli Angelo 19.5000 4.290000
22 Gustaf's Knäckebröd 21.0000 4.620000
64 Wimmers gute Semmelknödel 33.2500 7.315000
56 Gnocchi di nonna Alice 38.0000 8.360000
(7 row(s) affected)
```

Wstawianie wartości domyślnych

Jeżeli dla danej kolumny tabeli docelowej utworzono zawężenie DEFAULT, wpisując słowo kluczowe DEFAULT w instrukcji INSERT, możemy wstawić wartość określoną jako wartość domyślna dla kolumny.

Niewymienienie słowa kluczowego DEFAULT spowoduje wstawienie wartości NULL dla kolumn, dla których nie określono wartości domyślnej lub wstawienie wartości domyślnej dla kolumn, dla których została ona określona.

1. Próba wstawienia wartości domyślnej dla kolumny, dla której nie została ona określona, spowoduje wstawienie wartości nieokreślonej.

2. Próba wstawienia wartości domyślnej dla kolumn typu IDENTITY skończy się błędem.

Przykład (dla kolumny phone zdefiniowana została wartość domyślna (UNKNOWN), dla kolumny adress — nie):

```
USE pubs
INSERT INTO dbo.authors (au_id,au_lname, au_fname, phone, address, contract)
VALUES ('172-32-7776', 'Gibson', 'Fiodor', DEFAULT, DEFAULT,0)
GO
SELECT au_id,au_lname, au_fname, phone, address
FROM dbo.authors
WHERE au_id = '172-32-7776'
GO
-------------------------------------------------------------------------------
LOLEK\Administrator
(1 row(s) affected)
au_id au_lname au_fname phone address
----------- ------------ -------------------- ------------ --------------
172-32-7776 Gibson Fiodor UNKNOWN NULL
(1 row(s) affected)
```

Język Transact-SQL umożliwia nam również, za pomocą słów kluczowych DEFAULT VALUES, wstawienie wiersza zawierającego wyłącznie wartości domyślne, bez konieczności wielokrotnego wymieniania słowa kluczowego DEFAULT.

1. Do wszystkich kolumn, dla których nie zdefiniowano zawężenia DEFAULT, zostaną wstawione wartości nieokreślone.
2. Próba wykonania instrukcji dla tabeli, która zawiera kolumny niezezwalające na wstawienie wartości NULL i nieposiadające zawężenia DEFAULT skończy się błędem.

Przykład:

```
USE pubs
CREATE TABLE #test
(id int IDENTITY(1,1),
id_procesu smallint DEFAULT @@SPID,
data datetime DEFAULT getdate(),
kolumna char(3)
)
GO
INSERT INTO #test
DEFAULT VALUES
```

```
GO
SELECT *
FROM #test
GO
(1 row(s) affected)
id id_procesu data kolumna
----------- ---------- -------------------------- -------
1 51 2002-10-16 17:13:08.850 NULL
(1 row(s) affected)
```

Usuwanie danych

Usuwać informacje przechowywane w bazie danych możemy za pomocą instrukcji DELETE lub TRUNCATE. W pierwszym wypadku istnieje możliwość określenia, które dane zostaną usunięte z tabeli, w drugim — wszystkie dane z tabeli zostają nieodwracalne skasowanie.

Instrukcja DELETE

Instrukcja DELETE usuwa wybrane wiersze tabeli.
Składnia:

```
DELETE
[FROM]
{tabela WITH (<optymalizacja> [,…n])
| widok
| funkcja_typu_RowSet
}
[FROM {<tabela_źródłowa>} [,...n]]
[WHERE
{<warunek_wyszukiwania>
| {[CURRENT OF
{{[GLOBAL] kursor}
| @zmienna_kursora
}
]}
}
]
[OPTION (<optymalizacja> [,…n])]
<tabela_źródłowa> ::=
tabela [[AS] alias] [WITH (<optymalizacja> [,…n])]
| widok[[AS] alias]
| funkcja_typu_rowset [[AS] alias]
```

```
| fragment_tabeli [AS] alias [(alias_kolumny [,…n])]
| <złaczona_tabela>
<złaczona_tabela> ::=
<tabela_źródłowa> <typ_złączenia> <tabela_źródłowa> ON <warunek_wyszukiwania>
| <tabela_źródłowa> CROSS JOIN <tabela_źródłowa>
| <złaczona_tabela>
<typ_złączenia> ::=
[INNER | {{LEFT | RIGHT | FULL} [OUTER]}]
[<optymalizacja_złączenia>]
JOIN
<optymalizacja> ::=
{FASTFIRSTROW
| HOLDLOCK
| PAGLOCK
| READCOMMITTED
| REPEATABLEREAD
| ROWLOCK
| SERIALIZABLE
| TABLOCK
| TABLOCKX
| UPDLOCK
}
<optymalizacja_złączenia> ::=
{{HASH | ORDER} GROUP
| {CONCAT | HASH | MERGE} UNION
| FAST liczba_wierszy
| FORCE ORDER
| MAXDOP
| ROBUST PLAN
| KEEP PLAN
}
```

Język Transact-SQL, w przeciwieństwie do standardu SQL-2, pozwala na usuwanie wierszy wybranych w wyniku złączenia tabel oraz zwróconych przez podzapytanie powiązane. Usunięcie każdego wiersza z tabeli za pomocą polecenia DELETE zostaje zapisane w dzienniku transakcyjnym bazy danych.

Przykłady:

```
USE Northwind
SELECT *
INTO #temp
FROM dbo.Products
GO
```

```
SELECT COUNT(*)
FROM #temp
GO
DELETE #temp
WHERE CategoryID <6
GO
SELECT COUNT(*)
FROM #temp
GO
77 row(s) affected)
-----------
77
(1 row(s) affected)
(54 row(s) affected)
-----------
23
(1 row(s) affected)
```

(usuwanie informacji o 15 najdroższych ze znajdujących się w tabeli #temp produktach):

```
USE Northwind
DELETE #temp
FROM (SELECT TOP 15 *
FROM #temp
ORDER BY UnitPrice DESC) AS t1
WHERE #temp.ProductID = t1.ProductID
GO
(15 row(s) affected)
```

Usuwanie danych wybranych na podstawie danych zapisanych w innej tabeli

Warunek określony w klauzuli WHERE określa, które wiersze zostaną usunięte z tabeli. Język Transact-SQL umożliwia, poprzez wymienienie dodatkowej klauzuli FROM, wybór usuwanych wierszy na podstawie danych zapisanych w innej tabeli.
Przykład:

```
USE Northwind
DELETE FROM dbo.[order details]
FROM dbo.Products AS p
INNER JOIN [order details] AS od
ON p.ProductID = od.ProductID
```

```
WHERE CategoryID = 7
GO
(136 row(s) affected)
```

Instrukcja TRUNCATE

Wszystkie dane możemy usunąć z wybranej tabeli, wykonując instrukcję TRUNCATE TABLE. Chociaż funkcjonalnym odpowiednikiem wykonania instrukcji TRUNCATE jest wykonanie instrukcji DELETE bez klauzuli WHERE, to pierwsza operacja zostanie wykonana szybciej, a informacje o poszczególnych usuwanych wierszach nie zostaną zapisane w dzienniku transakcyjnym.

Składnia:

```
TRUNCATE TABLE tabela
```

1. Próba obcięcia tabeli powiązanej z innymi tabelami poprzez klucze obce skończy się błędem (wykonanie instrukcji DELETE bez klauzuli WHERE spowodowałoby usunięcie danych).
2. Obcięcie tabeli jest najszybszym sposobem usunięcia danych.
3. Obcięcie danych jest operacją niezapisywaną w dzienniku transakcyjnym, co oznacza, że po jej przeprowadzeniu należy wykonać kopię zapasową pliku bazy danych.
4. Wartości danych przechowywanych w kolumnie typu IDENTITY zostaną zresetowane.

Przykład:

```
USE Northwind
SELECT *
INTO dbo.kopia
FROM dbo.Orders
TRUNCATE TABLE dbo.kopia
GO
(830 row(s) affected)
```

Aktualizowane danych

SQL Server możliwa jednoczesne aktualizowanie dowolnej liczby rekordów, zarówno bezpośrednio, jak i poprzez widoki jednoznacznie prezentujące dane zapisane w tabeli.

Za pomocą instrukcji UPDATE możemy aktualizować istniejące dane.

Składnia:

```
UPDATE
{
tabela WITH (<optymalizacja> [,…n])
```

```
| widok
| funkcja_typu_RowSet
}
SET
{kolumna = {wyrażenie | DEFAULT | NULL}
| @zmienna = wyrażenie
| @zmienna = kolumna = wyrażenie} [,…n]
{{[FROM {<tabela_źródłowa>} [,…n]]
[WHERE
<warunek_wyszukiwania>]}
|
[WHERE CURRENT OF
{{[GLOBAL] kursor} | @zmienna_kursora}
]}
[OPTION (<optymalizacja> [,…n])]
<tabela_źródłowa> ::=
tabela [[AS] alias] [WITH (<optymalizacja> [,…n])]
| widok [[AS] alias]
| funkcja_typu_RowSet [[AS] alias]
| fragment_tabeli [AS] alias [(alias_kolumny [,…n])]
| <złączona_tabela>
<złączona_tabela> ::=
<tabela_źródłowa> <typ_złączenia> <tabela_źródłowa> ON <warunek_wyszukiwania>
| <tabela_źródłowa> CROSS JOIN <tabela_źródłowa>
| <złączona_tabela>
<typ_złączenia> ::=
[INNER | {{LEFT | RIGHT | FULL} [OUTER]}]
[<optymalizacja_złączenia>]
JOIN
<optymalizacja> ::=
{ INDEX (index_val [,…n])
| FASTFIRSTROW
| HOLDLOCK
| NOLOCK
| PAGLOCK
| READCOMMITTED
| READPAST
| READUNCOMMITTED
| REPEATABLEREAD
| ROWLOCK
| SERIALIZABLE
| TABLOCK
| TABLOCKX
```

```
| UPDLOCK
}
<optymalizacja_złaczenia> ::=
{ {HASH | ORDER} GROUP
| {CONCAT | HASH | MERGE} UNION
| {LOOP | MERGE | HASH} JOIN
| FAST number_rows
| FORCE ORDER
| MAXDOP
| ROBUST PLAN
| KEEP PLAN
}
```

1. Zaktualizowane zostaną wszystkie dane spełniające warunek podany w klauzuli WHERE. Brak tej klauzuli spowoduje zmianę wartości wszystkich danych zapisanych w tabeli.
2. Próba zmiany typu danych zakończy się błędem, chyba że SQL Server będzie potrafił dokonać niejawnej konwersji nowego typu do typu zdefiniowanego dla danej kolumny.
3. Aktualizowane dane muszą uwzględniać zawężenia nałożone na tabelę.
4. Niemożliwe jest jednoczesne aktualizowanie danych zapisanych w kilku tabelach.

Przykłady:

```
USE pubs
BEGIN TRANSACTION
UPDATE dbo.authors
SET au_lname = UPPER (au_lname)
SELECT au_lname, au_fname
FROM dbo.authors
WHERE state='UT'
ROLLBACK TRAN
SELECT au_lname, au_fname
FROM dbo.authors
WHERE state='UT'
GO
au_lname au_fname
---------------------------------------- --------------------
RINGER Anne
RINGER Albert
(2 row(s) affected)
au_lname au_fname
---------------------------------------- --------------------
```

```
Ringer Anne
Ringer Albert
(2 row(s) affected)
USE Northwind
UPDATE dbo.Products
SET UnitsOnOrder = 10
WHERE ProductID = 1
GO
(1 row(s) affected)
```

Aktualizowanie danych na podstawie danych zapisanych w innej tabeli

Język Transact-SQL pozwala na aktualizowanie danych do wartości wybranych lub obliczonych na podstawie danych zapisanych w innej tabeli. Poniższa instrukcja ustala wartość atrybutu ytd_sales dla wszystkich rekordów zapisanych w tabeli titles jako równych sumie ostatnich wartości sprzedaży zapisanych w tabeli sales:

```
USE pubs
UPDATE titles
SET ytd_sales =
(SELECT SUM(qty)
FROM sales
WHERE sales.title_id = titles.title_id
AND sales.ord_date IN (SELECT MAX(ord_date) FROM sales))
FROM titles, sales
GO
(18 row(s) affected)
```

Aktualizując dane na podstawie wyników podzapytania powiązanego, pamiętaj, że SQL Server w ramach jednej instrukcji UPDATE tylko raz zmieni wartość danych kolumny docelowej.

Aktualizowanie danych wybranych na podstawie danych zapisanych w innej tabeli

Tak jak możemy ustawić nową wartość aktualizowanych danych na podstawie danych zapisanych w innej tabeli, dane z innej tabeli mogą posłużyć jako kryterium wyboru aktualizowanych wierszy. Poniższa instrukcja zwiększa o 10% wartość produktów należących do kategorii, której nazwa rozpoczyna się od litery C:

```
USE Northwind
UPDATE dbo.products
SET unitprice = unitprice * 1.1
```

```
FROM dbo.products p
INNER JOIN dbo.Categories c
ON p.CategoryID = c.CategoryID
WHERE c.CategoryName LIKE 'C%'
GO
(25 row(s) affected)
```

Kurs Transact-SQL, część 7 (Projektowanie relacyjnych baz danych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Celem procesu projektowania bazy danych jest stworzenie poprawnego i spełniającegon wymagania użytkowników schematu bazy danych. Ponieważ cały proces jest dosyć skomplikowany, w przypadku rozbudowanych baz danych dzieli się go na kilka etapów:

1. Przygotowanie diagramu związków E/R.
2. Normalizacja projektu.
3. Implementacja zasad wymuszających integralność danych.

Skoro baza danych to zbiór uporządkowanych danych, to praca projektanta sprowadza się do stworzenia struktury, w której przechowywane będą dane oraz wyboru informacji, które powinny znaleźć się w bazie danych.

W pierwszej kolejności należy określić schemat (strukturę) bazy danych i dopiero wtedy, dysponując gotowym schematem, wybrać te informacje, które będą przechowywane w bazie danych. Tworząc schemat, należy kierować się ogólną regułą, na podstawie której:

- W otaczającym nas świecie można wyróżnić mniej lub bardziej trwałe, ale będące logicznymi całościami obiekty różnych typów.
- Obiekty poszczególnych typów mogą zostać określone za pomocą właściwych im cech (atrybutów, metod i zdarzeń).

Wybór typów obiektów oraz określenie, które informacje powinny być przechowywane w bazie, jest podstawą diagramu związku E/R (Encja/Relacja). Zanim zostanie opisane tworzenie takiego modelu, zapoznamy się z podstawami teorii relacyjnych baz danych.

Model relacyjnych baz danych

Termin relacyjna baza danych oznacza bazę zbudowaną z relacji. Podstawowy obiekt takiej bazy danych, tabela, jest konkretną reprezentacją relacji — technicznego pojęcia matematyki. Wynika z tego, że oba terminy nie są jednoznaczne, a jedna relacja może być odwzorowana za pomocą wielu różnych tabel.

Relacyjny model baz danych stworzony został przez E.F. Codda w 1970 roku i przedstawiony w pracy *Relacyjny model danych dla dużych banków danych*. Nie używa się tam pojęć tabela, kolumna i wiersz, lecz relacja, atrybut i krotka.

Każda tabela składa się z pewnej liczby wierszy i kolumn. Na przecięciu wiersza z kolumną znajduje się pole. W modelu relacyjnym przyjmuje się, że:

- kolejność wierszy i kolumn w tabelach jest nieistotna,
- wiersze zawierające takie same dane są identyczne.

Natomiast w tabeli przedstawiającej konkretny przypadek relacji identyczne dane (wartości pól) będą przechowywane w różnych wierszach. Pole zawiera najmniejszą niepodzielną wartość, czyli taką część informacji, która nie może być dalej dzielona ze względu na spójność logiczną.

Zagadnieniu przechowywania w jednym polu wielu informacji poświęcony jest punkt *Pierwsza postać anormalna*.

Przed podaniem definicji podstawowego terminu opisywanej teorii, relacji, zasadne wydaje się podanie definicji schematu relacji i dziedziny relacji.

Definicja 1. Schemat relacji

Schematem relacji nazywamy zbiór $R = \{A_1, A_2, \ldots, A_n\}$,
gdzie:
$A_1, A_2, \ldots, A_n$ są atrybutami, reprezentowanymi w tabeli poprzez nazwy kolumn.
Przykładowym schematem relacji Northwind..Categories będzie zbiór:

Categories {CategoryID, CategoryName, Description, Picture}

Każdemu atrybutowi ($A_1, A_2, \ldots, A_n$) przyporządkowana jest dziedzina (zakres dopuszczalnych wartości atrybutu) reprezentowana przez typ danych. W tym przypadku dla poszczególnych kolumn zdefiniowano następujące typy danych:

[CategoryID] [int] IDENTITY (1, 1)

[CategoryName] [nvarchar] (15) COLLATE Polish_CI_AS

[Description] [ntext] COLLATE Polish_CI_AS

[Picture] [image]

czyli:

- Wartości atrybutu CategoryID będą liczbami całkowitymi z zakresu od –2 147 483 648 do 2 147 483 647.
- Wartości atrybutu CategoryName będą ciągami 15 znaków, z których każdy zapisany jest w dwóch bajtach.
- Wartości atrybutu Description będą ciągami znaków o długości do 1 073 741 823 znaków, z których każdy zapisany jest w dwóch bajtach.
- Wartości atrybutu Picture będą danymi binarnymi o długości do 1 073 741 823 bajtów.

Definicja 2. Dziedzina relacji

Dziedziną relacji o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy sumę dziedzin wszystkich jej atrybutów $Dom(R)=Dom(A_1) \cup Dom(A_2) \cup \ldots \cup Dom(A_n)$.

Przykładową dziedziną relacji Categories będzie dziedzina

DOM(Categories) = int ∪ nvarchar (15) ∪ ntext ∪ image

Umiejętność oszacowania średniej wielkości pojedynczego wiersza każdej tabeli jest niezbędna do zaplanowania wielkości pliku bazy danych.

W ten sposób możemy już podać definicję podstawowego obiektu relacyjnych baz danych — relacji.

Definicja 3. Relacja

Relacją o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy skończony zbiór $r = \{t_1, t_2, \ldots, t_m\}$ odwzorowań $t_i: R \rightarrow Dom(R)$ takich, że dla każdego j z zakresu $1 <= j <= n$ zachodzi zależność: $t_i(A_j) \in Dom(A_j)$.

Tak zdefiniowane pojedyncze odwzorowanie nosi nazwę krotki i odpowiada mu pojedynczy wiersz tabeli. Wartością krotki jest suma wartości poszczególnych atrybutów. Na przykład wartością pierwszej krotki opisywanej tabeli jest zbiór:

1 ∪ Beverages ∪ Soft drinks, coffees, teas, beers, and ales ∪ 0x151C2F00020000000D000
E0014002100FFFFFFFF4269746D617020496D616765005061696E742E506963747572650001050000020000000700000050427275736800000000000000000A0290000424D98290000000000005600000028000000AC0000007800000001000400000000000000000880B0000880B0000080000

Krotka może zostać ograniczona do wartości wybranych atrybutów.

Lemat 1. Ograniczenie krotki

Ograniczeniem krotki t relacji r o schemacie R do zbioru atrybutów $X \subseteq R$ nazywamy odwzorowanie będące ograniczeniem krotki t do zbioru atrybutów X $t_{IX}: X \rightarrow Dom(X)$.
Przykład: wartością ograniczenia $X = \{CategoryID, CategoryName\}$ przykładowej krotki będzie zbiór wartości.

{1 ∪ Beverages}

Podstawowe zasady implementacji modelu relacyjnych baz danych można podzielić na trzy grupy:

1. Zasady dotyczące struktury danych.
2. Zasady dotyczące przetwarzania danych.
3. Zasady dotyczące integralności danych.

Zasady dotyczące struktury danych

W modelu relacyjnych baz danych informacja o poszczególnych obiektach zapisana jest w tabelach. Jest to model abstrakcyjny, zawsze obsługiwany w ten sam sposób i niezwiązany ze sposobem przechowywania danych. W ten sposób możliwe jest oddzielenie danych od aplikacji klienckiej (interfejsu użytkownika) i platformy sprzętowej. Serwer bazodanowy poprzez system przechowujący dane (ang. Storage engine) zarządza rozmieszczeniem danych w plikach znajdujących się na dysku bądź w pamięci podręcznej oraz metodami dostępu do tych danych.

Cechą charakterystyczną modelu jest wymóg przechowywania w bazie danych tylko konkretnych wartości. W relacyjnych bazach danych nie możemy posługiwać się wskaźnikami do danych.

Twórca relacyjnego modelu baz danych, E.F. Codd, przedstawił zbiór dwunastu postulatów, które powinny zostać uwzględnione przez projektantów systemów zarządzania relacyjnymi bazami danych. Zamieszczone niżej postulaty dotyczą struktury danych, a ich znajomość może okazać się przydatna podczas projektowania baz danych:

- **Postulat informacyjny.** Na poziomie logicznym dane reprezentowane są wyłącznie za pomocą tabel wartości.
- **Postulat dostępu.** Do każdej pojedynczej danej jest dostęp za pomocą nazwy tabel, kolumn i wartości kluczy głównych.
- **Postulat fizycznej niezależności danych.** Zmiany w sposobie przechowywania danych i dostępu do nich nie wpływają na aplikację kliencką.
- **Postulat logicznej niezależność danych.** Zmiany w tabelach, zachowujące informację i dopuszczalne semantycznie, nie mają wpływu na aplikację kliencką.
- **Postulat niezależności dystrybucyjnej.** System i jego język umożliwiają dostęp do danych zapisanych w różnych miejscach, np. na wielu komputerach w sieci.
- **Postulat zabezpieczania przed operacjami na niższym poziomie abstrakcji.** Jeśli system zarządzania bazą danych umożliwia bezpośrednie operacje na niższych poziomach abstrakcji, nie mogą one naruszać reguł relacyjnego modelu baz danych, w szczególności nie mogą pomijać ograniczeń określonych przez więzy spójności.

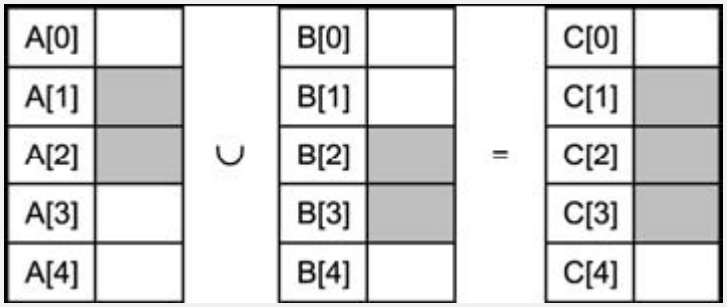
Zasady dotyczące przetwarzania danych

Dane przechowywane w bazie danych nie są niezmienne. Wręcz przeciwnie — aby baza danych była przydatna, przechowywane w niej informacje muszą odpowiadać stanowi faktycznemu, a więc muszą być cały czas aktualizowane. Operacje modyfikowania danych nie mogą jednak naruszać struktury danych. Wynika z tego, że przekształcenie danych dokonywane bądź to w celu ich modyfikacji, bądź pobrania danych, musi przebiegać z zachowaniem wewnętrznej logiki (struktury) powiązań istniejących pomiędzy danymi.

Pierwszy typ przekształceń, które można wykonywać na wartościach atrybutów, wynika z definicji relacji jako zbioru. Operatorom algebry zbiorów, czyli ∪ ∩ —, odpowiadają operatory algebry relacji UNION (suma relacji), INTERSEC (cześć wspólna relacji, przecięcie, przekrój relacji) i EXCEPT (dopełnienie relacji, różnica relacji).

SQL Server pozwala jedynie na łączenia za pomocą operatora UNION wyników dwóch zapytań. Aby obliczyć część wspólną lub dopełnienie wyników zapytań, należy posłużyć się podzapytaniem powiązanym.

Na sumę dwóch relacji r ∪ s składają się wszystkie elementy relacji r i wszystkie elementy relacji s (rysunek 7.1).

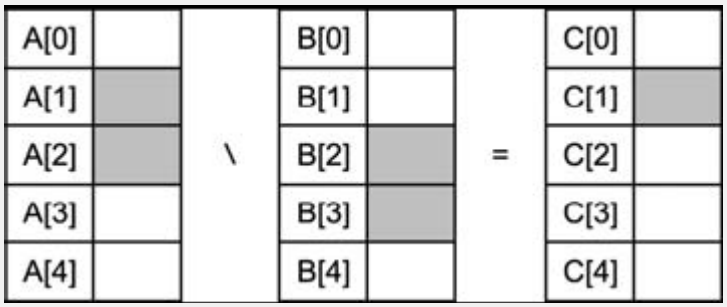


Rysunek 7.1 Suma relacji

Wynikiem przecięcia relacji r ∩ s jest relacja składająca się z elementów wspólnych relacji r i s (rysunek 7.2).

Rysunek 7.2 Przecięcie relacji

Wynikiem odjęcia relacji r – s jest relacja powstała na skutek usunięcia z relacji r wszystkich elementów wchodzących w skład relacji s (rysunek 7.3).



Rysunek 7.3 Dopełnienie relacji

Oprócz operatorów teoriomnogościowych, dla operacji na relacjach zdefiniowane są następujące operatory: selekcji, projekcji, iloczynu kartezjańskiego i złączenia naturalnego. Dwa pierwsze dotyczą wyników pojedynczego zapytania, dwa kolejne — wyników zwracanych przez większa liczbę zapytań.

Definicja 4. Selekcja relacji

Selekcją relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy:

- Zbiór tych krotek relacji *r*, w których wartość atrybutu *A* wynosi *a*, przy założeniu, że $A \in R$ i $a \in Dom(A)$, co zapisujemy $\sigma A=a(r) = \{t \in r \mid t(A) = a\}$.
- Dla dowolnego warunku logicznego *F* — zbiór krotek relacji *r* spełniających warunek *F*, co zapisujemy $\sigma F(r) = \{t \in r \mid t \text{ spełnia warunek } F\}$.

Przykład:

σCategoryID=2(Categories) = {2 ∪ Condiments ∪ Sweet and savory sauces, relishes, spreads,
and seasonings ∪ 0x151C2F00020000000D000E0014002100FFFFFFFF4269746D617020496D6167650
05061696E742E5069637475726500010500000200000007000000504272757368000000000000000000A0
290000424D982900000000000056000000280000000AC000000780000000100040000000000000000000880
B0000880B0000080000}

Dla operatora selekcji prawdziwe są następujące twierdzenia:

Twierdzenie 1. Selekcja sumy

$\sigma F(r \cup s) = \sigma F(r) \cup \sigma F(s)$ — selekcja sumy relacji równa jest sumie selekcji.

Twierdzenie 2. Selekcja przecięcia

$\sigma F(r \cap s) = \sigma F(r) \cap \sigma F(s)$ — selekcja przecięcia relacji równa jest przecięciu selekcji.

Twierdzenie 3. Selekcja różnicy

$\sigma F(r - s) = \sigma F(r) - \sigma F(s)$ — selekcja różnicy relacji równa jest różnicy selekcji.

Na podstawie twierdzeń T1-T3 możemy w dowolny sposób zmieniać kolejność operacji teoriomnogościowych względem operacji selekcji. Zmiana kolejności wykonania operacji nie wpłynie na ich wynik.

Dodatkowo — dzięki temu, że w relacyjnej bazie danych wyniki wszystkich zapytań (wyniki instrukcji SELECT) są zwracane w postaci tabel — możliwe jest zagnieżdżanie zapytań (kierowanie zapytań nie do tabel czy widoków, ale do wyników innych zapytań).

Definicja 5. Projekcja relacji

Projekcją relacji r na zbiór atrybutów $X \subseteq R$ nazywamy zbiór ograniczeń do atrybutów zbioru X wszystkich krotek relacji r, co zapisujemy $\pi X(r) = \{t|X: t \in r\}$.

Projekcja nazywana jest czasami selekcją pionową lub rzutem — tak jak selekcja ogranicza liczbę wierszy wyniku, projekcja ogranicza liczbę kolumn wyniku.
Przykład:

π{CategoryID, CategoryName}(Categories) = {(1,Beverages) ∪ (2,Condiments) ∪ (3,Confections) ∪
(4,Dairy Products) ∪ (5,Grains/Cereals) ∪ (6,Meat/Poultry) ∪ (7,Produce) ∪ (8,Seafood)

Dla operatora projekcji prawdziwe są następujące twierdzenia:

Twierdzenie 4. Projekcja projekcji

$\pi X(\pi Y(r)) = \pi X(r)$ przy założeniu, że $X \subseteq Y$ — projekcja wyniku szerszej projekcji równa jest wynikowi bardziej restrykcyjnej projekcji.

Twierdzenie 5. Projekcja sumy

$\pi X(r \cup s) = \pi X(r) \cup \pi X(s)$ — projekcja sumy relacji równa jest sumie projekcji.

Twierdzenie 6. Selekcja projekcji

$\sigma F(\pi X(r)) = \pi X(\sigma F(r))$ — selekcja projekcji równa jest projekcji selekcji.

Na podstawie twierdzeń T4-T6 możemy w dowolny sposób zmieniać kolejność operacji sumy, selekcji i projekcji. Zmiana kolejności operacji nie wpłynie na ich wynik.

Prostym przykładem łącznego wykorzystania operatorów selekcji i projekcji jest wybranie nazwy i opisu kategorii o identyfikatorze równym 3:

```
USE Northwind
SELECT CategoryName, Description
FROM dbo.Categories
WHERE CategoryID=3
GO
CategoryName Description
---------- -----------------------------------
Confections Desserts, candies, and sweet breads
```

(1 row(s) affected)

Semantyczną definicją powyższego zapytania jest wyrażenie

π(CategoryName, Description}(σCategoryID=3(Categories))

Definicja 6. Iloczyn kartezjański

Iloczynem kartezjańskim dwóch relacji: *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i *s* o schemacie $S = \{B_1, B_2, \ldots, B_m\}$, przy założeniu, że $R \cap S = \emptyset$, nazywamy relację $q = r \times s$ o schemacie $Q = \{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$ składającą się ze wszystkich krotek $t \in q$, dla których istnieją krotki $u \in r$ i $v \in s$ takie, że: $t(A_i) = u(A_i)$ *dla* $1 <= i <= n$ *oraz* $t(B_i) = v(B_i)$ *dla* $1 <= i <= m$.

> Iloczyn kartezjański dwóch relacji jest zbiorem wszystkich możliwych połączeń krotek obu relacji. Jeżeli schematy relacji nie są rozłączne (istnieje niepusty zbiór powstały w wyniku przecięcia obu relacji), najpierw zmienia się nazwy atrybutów jednej z relacji, a następnie stosuje się powyższą definicję.

Dla operatora iloczynu kartezjańskiego prawdziwe są następujące twierdzenia:

Twierdzenie 7. Iloczyn sumy

$(r \cup s) \times q = (r \times q) \cup (s \times q)$ — iloczyn kartezjański sumy równy jest sumie iloczynów kartezjańskich.

Twierdzenie 8. Selekcja iloczynu

$\sigma F \hat{} G(r \times s) = \sigma F(r) \times \sigma G(s)$ przy założeniu że *F(R), G(S)* — selekcja iloczynu kartezjańskiego równa jest iloczynowi kartezjańskiemu selekcji.

Twierdzenie 9. Projekcja iloczynu

$\pi X \cup Y(r \times s) = \pi X(r) \times \pi Y(s)$ przy założeniu, że $X \subseteq R$ *i* $Y \subseteq S$ — projekcja iloczynu kartezjańskiego równa jest iloczynowi kartezjańskiemu projekcji.

Przykładem iloczynu kartezjańskiego dwóch relacji jest poniższe wyrażenie algebry relacji i odpowiadająca mu instrukcja języka Transact-SQL:

π(CategoryName, Description}(Categories)(σSupplierID=9(Categories x Products)

```
USE Northwind
SELECT CategoryName, Description
FROM dbo.Categories, dbo.Products
WHERE SupplierID=9
GO
CategoryName Description
------------- -----------------------------------------------------------
Beverages Soft drinks, coffees, teas, beers, and ales
Beverages Soft drinks, coffees, teas, beers, and ales
Condiments Sweet and savory sauces, relishes, spreads, and seasonings
Condiments Sweet and savory sauces, relishes, spreads, and seasonings
Confections Desserts, candies, and sweet breads
Confections Desserts, candies, and sweet breads
```

```
Dairy Products Cheeses
Dairy Products Cheeses
Grains/Cereals Breads, crackers, pasta, and cereal
Grains/Cereals Breads, crackers, pasta, and cereal
Meat/Poultry Prepared meats
Meat/Poultry Prepared meats
Produce Dried fruit and bean curd
Produce Dried fruit and bean curd
Seafood Seaweed and fish
Seafood Seaweed and fish
(16 row(s) affected)
```

Jak widać, wynikiem iloczynu kartezjańskiego jest zbiór wielu krotek, z których jedynie niektóre zawierają sensowne dane. Dlatego w produkcyjnych bazach danych bardzo rzadko wykorzystuje się tę metodę łączenia relacji.

Definicja 7. Złączenie naturalne

Złączeniem naturalnym relacji: r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i s o schemacie $S = \{B_1, B_2, \ldots, B_m\}$ jest relacja q o schemacie $Q = R \cup S$ taka, że $q = \{t: \exists\, u \in r \; i \; v \in s$ *takie, że* $u|R = t \; i \; v|S = t\}$.

Tak więc złączeniem naturalnym relacji jest zbiór wszystkich możliwych połączeń krotek relacji, przy których ich wspólne atrybuty mają takie same wartości.

> Wynikiem złączenia naturalnego dwóch relacji jest zbiór powiązanych ze sobą krotek obu relacji.

Dla operatora złączenia naturalnego prawdziwe są następujące twierdzenia:

Twierdzenie 10. Złączenie relacji z samą sobą

$q \oplus q = q$ — wynikiem złączenia relacji z samą sobą jest dana relacja.

Twierdzenie 11. Naprzemienność złączenia

$q \oplus r = r \oplus q$ — zmiana kolejności łączonych relacji nie wpływa na wynik złączenia.

Twierdzenie 12. Przechodniość złączenia

$(q \oplus r) \oplus s = q \oplus (r \oplus s)$ — zmiana kolejności operacji złączenia naturalnego kilku relacji nie wpływa na wynik złączenia.

Twierdzenie 13. Projekcja złączenia

$\pi R(r \oplus s) \subseteq r$ oraz $\pi S(r \oplus s) \subseteq s$ — projekcja złączenia naturalnego dwóch relacji zawiera się (a w pewnych okolicznościach jest równa) jednej z tych relacji.

Twierdzenie 14. Rozkład złączenia

$q \subseteq \pi R(q) \oplus \pi S(q)$ — relacja zawiera się (a w pewnych okolicznościach jest równa) złączeniu naturalnemu dwóch projekcji tej relacji

Przykładem złączenia naturalnego dwóch relacji jest poniższe wyrażenie algebry relacji i odpowiadająca mu instrukcja języka Transact-SQL:

```
π(CategoryName, Description}(Categories)(σSupplierID=9(Categories ⊕ Products)
```

```
USE Northwind
SELECT CategoryName, Description
FROM dbo.Categories JOIN dbo.Products
ON dbo.Categories.CategoryID=dbo.Products.CategoryID
WHERE SupplierID=9
GO
CategoryName Description
---------- -----------------------------------
Grains/Cereals Breads, crackers, pasta, and cereal
Grains/Cereals Breads, crackers, pasta, and cereal
(2 row(s) affected)
```

Zasad związanych z przetwarzaniem danych bezpośrednio dotyczą trzy kolejne postulaty Codda:

- **Postulat pełnego języka danych.** W systemie zaimplementowany jest pełny język, który obejmuje:
  - Definiowanie danych.
  - Definiowanie perspektyw (widoków lub kwerend, które nie przechowują danych, ale pobierają i porządkują dane zapisane w tabelach, dzięki czemu pełnią funkcje „okularów", przez które możemy „spoglądać" na dane).
  - Definiowanie więzów spójności (ograniczeń uniemożliwiających wprowadzenie niepoprawnych danych).
  - Modyfikowanie danych.
  - Nadawanie uprawnień użytkownikom.
  - Implementację transakcyjnego modelu przetwarzania danych.
- **Postulat modyfikowania bazy danych przez perspektywy.** System umożliwia modyfikowanie danych poprzez perspektywy, o ile modyfikacja taka jest semantycznie sensowna
- **Postulat modyfikowania danych na wysokim poziomie abstrakcji.** System umożliwia modyfikowanie danych za pomocą operacji, których argumentami są tabele i perspektywy.

Zasady dotyczące integralności danych

Zasady dotyczące integralności danych zapewniają zachowanie logicznej spójności przechowywanych w bazie danych informacji. Podstawową zasadą tego typu jest wymóg jednoznacznego identyfikowania wszystkich atrybutów dowolnego obiektu na podstawie jednego (lub kilku) jego atrybutów.

W teorii relacyjnych baz danych występuje pojęcie zależności funkcyjnej, za pomocą którego opisuje się wspomnianą zasadę. Atrybuty zależą funkcyjnie od pewnej grupy atrybutów, jeżeli każdemu zespołowi wartości z pierwszej grupy przyporządkowany jest jeden i tylko jeden zespół wartości z drugiej grupy. Zależność funkcyjną zapisuje się za pomocą znaku strzałki, która wskazuje, że jeden zespół wartości zależy funkcyjnie od drugiego.

Dla relacji reprezentowanej poprzez tabelę Categories prawdziwe są m.in. następujące zależności funkcyjne:

```
{CategoryID} -> {CategoryName, Description}
```

{CategoryID, CategoryName} -> {Description}

(CategoryName) -> {CategoryID, Description}

{CategoryID, Description} -> {CategoryName}

{CategoryName, Description) -> CategoryID

czyli:

1. Na podstawie identyfikatora możemy jednoznacznie określić krotkę tabeli.
2. Na podstawie identyfikatora i nazwy możemy jednoznacznie określić krotkę tabeli.
3. Na podstawie nazwy możemy jednoznacznie określić krotkę tabeli.
4. Na podstawie identyfikatora i opisu możemy jednoznacznie określić krotkę tabeli.
5. Na podstawie nazwy i opisu możemy jednoznacznie określić krotkę tabeli.

Istotne jest rozróżnienie zależności spełnianych przez istniejące rekordy tabeli od zależności, które będą spełnione nawet po dodaniu nowych wierszy. W tym przypadku w tabeli Categories znajduje się jedynie 8 rekordów i możemy jednoznacznie określić rekord zarówno na podstawie identyfikatora, jak i na podstawie nazwy czy kombinacji nazwy i opisu. Jednak nie można wykluczyć, że do tabeli zostaną dodane nowe kategorie o takiej samej jak jedna z istniejących nazw i w ten sposób trzecia wypisana zależność przestanie być prawdziwa.

Definicja 8. Zależność funkcyjna

Relacja *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ spełnia zależność funkcyjną $X \rightarrow Y$ $(X, Y \subseteq R)$, jeżeli dla każdych dwóch krotek $t, u \in r$ zachodzi warunek: jeśli $t|X = u|X$ *to* $t|Y = u|Y$.

Innymi słowy, atrybuty zależą funkcyjnie od pewnej grupy atrybutów, jeżeli dla każdego zespołu wartości z pierwszej grupy jest przyporządkowany jeden i tylko jeden zespół wartości w drugiej grupie.

Zbiory atrybutów, na podstawie wartości, których możemy jednoznacznie określić obiekt, nazywane są kluczami kandydującymi lub nadkluczami. Prawie dla każdej relacji można znaleźć więcej niż jeden klucz kandydujący (w każdym przypadku kluczem kandydującym dowolnej relacji jest zbiór wszystkich jej atrybutów). Jednak identyfikowanie obiektu za pomocą dużej liczby parametrów świadczy o błędnie zaprojektowanej bazie danych i niekorzystnie wpływa na jej wydajność. Dlatego należy dążyć do odkrycia pojedynczego atrybutu jednoznacznie identyfikującego wszystkie obiekty danego typu. Wielu projektantów baz danych dodaje do każdej relacji atrybut jednoznacznie identyfikujący pozostałe. Wartości tego atrybutu są z reguły automatycznie generowane przez serwer i ukrywane przed użytkownikiem. Taki atrybut pełni funkcję klucza głównego obiektu.

Definicja 9. Nadklucz

Nadkluczem relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy dowolny zbiór atrybutów $X \subseteq R$ taki, że zachodzi zależność funkcyjna $X \rightarrow Y$.

Definicja 10. Klucz

Kluczem relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ nazywamy każdy minimalny nadklucz (nadklucz nie zawierający żadnego innego nadklucza).

> Klucze główne, jako jednoznacznie identyfikujące pozostałe atrybuty są wykorzystywane przy łączeniu relacji ze sobą.

Dla prawie każdej tabeli (wyjątkiem są specjalne tabele reprezentujące związek typu wiele do wielu i niezawierające żadnych dodatkowych kolumn) należy określić klucz główny. W większości wypadków najlepszym rozwiązaniem jest utworzenie prostego klucza (klucza odpowiadającego jednej, wybranej kolumnie. Wyjątkiem od tej reguły są m.in. bazy danych replikowane pomiędzy kilkoma serwerami bazodanowymi działającymi w modelu centralny subskrybent — wielu wydawców.

> Zagadnienia związane z replikacją zostały opisane w rozdziale 21.

> Atrybut lub zbiór atrybutów z innej relacji, która odpowiada kluczowi głównemu powiązanej relacji, nazywany jest kluczem zewnętrznym (ang. Foreign Key).

Dodatkowym mechanizmem wymuszającym integralność danych przechowywanych w bazie są zawężenia (ang. Constraints). Jeżeli do definicji określonej kolumny tabeli dodamy zawężenie, to ilekroć będą modyfikowane lub dodawane nowe wartości tego atrybutu, serwer bazodanowy sprawdzi, czy wprowadzane lub modyfikowane wartości są zgodne z określonym zawężeniem. W przypadku, gdy nowe wartości nie będą zgodne z zadanym zawężeniem, serwer zwróci komunikat o błędzie i zatrzyma wykonywanie takiego polecenia.

Wyróżniamy pięć rodzajów zawężeń:

Zawężenie NOT NULL

Jeżeli dla jakiegoś atrybutu wprowadzimy zawężenia NOT NULL, w odpowiadającej atrybutowi kolumnie nie będzie możliwe przechowywanie pól nieposiadających żadnej wartości (pól niewypełnionych). Domyślnie SQL Server pozwala na niewypełnienie danymi każdej z tworzonych kolumn, ale ponieważ obsługa wartości NULL komplikuje przygotowanie interfejsu bazy danych, a dodatkowo niewypełnienie wartości niektórych atrybutów powoduje wadliwe wyniki przeszukiwania bazy danych, powinniśmy uniemożliwić niewypełnienia niektórych kolumn danymi.

Przykład:

```
USE pubs
INSERT INTO dbo.authors (au_id, au_lname, au_fname, contract)
VALUES ('999-99-9999','Szczęśliwy', 'Szczepan', 1)
GO
-------------------------------
LOLEK\Administrator
(1 row(s) affected)
DECLARE @a int, @b int, @c int
SELECT @a = COUNT (*)
FROM dbo.authors
WHERE city = 'Oakland'
```

```
SELECT @b = COUNT (*)
FROM dbo.authors
WHERE city <> 'Oakland'
SELECT @c = COUNT (*)
FROM dbo.authors
PRINT 'Autorzy z Oakland '
PRINT @a
PRINT 'Autorzy spoza Oakland'
PRINT @b
PRINT 'Wszyscy autorzy'
PRINT @c
PRINT 'Różnica'
PRINT @c-(@a+@b)
GO
-----------------------------------------
Autorzy z Oakland
5
Autorzy spoza Oakland
18
Wszyscy autorzy
25
Różnica
2
```

Zawężenie UNIQUE

Zawężenie wymusza przechowywanie niepowtarzalnych wartości atrybutu. Za jego pomocą definiujemy klucz unikalny tabeli. W kolumnach, na które nałożymy ten warunek, SQL Server będzie przechowywał wyłącznie niepowtarzalne (unikalne) wartości.

W ramach jednej tabeli można zdefiniować dowolną liczbę zawężeń tego samego typu, na przykład zawężeń UNIQUE lub NOT NULL. Wyjątkiem od tej zasady jest zawężenie PRIMARY KEY — tabela może posiadać tylko jeden klucz główny.

Zawężenie PRIMARY KEY (klucz główny)

To zawężenie jest równoważne zawężeniom NOT NULL i UNIQUE. Dla każdej tabeli możemy utworzyć tylko jeden klucz główny, za pomocą którego SQL Server będzie mógł jednoznacznie identyfikować wszystkie pozostałe atrybuty danego obiektu.

Domyślnie SQL Server tworzy indeks grupujący dla kolumny, której zostało nadane zawężenie PRIMARY KEY.

Zawężenie REFERENCES (klucz zewnętrzny)

Poprzez zawężenie REFERENCES możemy zdefiniować klucz zewnętrzny, czyli powiązanie relacji z inną relacją. W wyniku nadania tego zawężenia na dany atrybut ograniczymy listę dopuszczalnych dla niego wartości do wartości przechowywanych w odpowiadającej jej kolumnie w powiązanej relacji. Atrybuty z nadanym zawężeniem REFERENCES mogą posiadać wartości NULL (niewypełnione pola), o ile nie zabraniają tego inne nadane zawężenia. Aby SQL Server mógł utworzyć klucz zewnętrzny, musi wcześniej zostać utworzona tabela, do której klucz będzie się odwoływał. Dodatkowo w tabeli tej musi być utworzona odpowiednia kolumna (lub kolumny), dla której zdefiniowano taki sam typ danych jak dla kolumny z nałożonym warunkiem REFERENCES. Na kolumnę taką musi być także nałożone jedno z dwóch zawężeń: PRIMARY KEY lub UNIQUE.

Jeżeli pewnej kolumnie nałożymy zawężenie REFERENCES, powinniśmy pamiętać, że:

- W tabeli z kluczem zewnętrznym nie można wstawić wiersza o wartościach klucza zewnętrznego, które nie mają odpowiedników w powiązanej tabeli.
- W tabeli z kluczem zewnętrznym nie można bezpośrednio zmodyfikować wiersza klucza zewnętrznego, nadając mu wartości, które nie mają odpowiedników w powiązanej tabeli, o ile podczas tworzenia zawężenia nie została podana dyrektywa ON UPDATE {CASCADE | SET NULL}.
- Z powiązanej tabeli nie można bezpośrednio usunąć wiersza, do którego odwołują się wartości klucza zewnętrznego innej tabeli. Można zażądać usuwania wraz z wierszem wszystkich wierszy w tabeli z kluczem zewnętrznym, do których ten wiersz się odwołuje. W tym celu przy klauzuli definiującej klucz zewnętrzny należy umieścić dyrektywę ON DELETE CASCADE. Aby podczas usuwania wiersza w powiązanych tabelach zamienić wartość klucza zewnętrznego na wartość nieokreśloną, należy posłużyć się dyrektywą ON DELETE SET NULL.

Zawężenie CHECK

Zawężenie CHECK ma najbardziej ogólne zastosowanie ze wszystkich omówionych zawężeń. Jest ono wyrażeniem, które zwraca wartość TRUE lub FALSE (prawda lub fałsz). Jeżeli zawężenie CHECK zwróci wartość FALSE, instrukcja języka Transact-SQL, w ramach której próbowano zmodyfikować dane, zostanie wycofana, a SQL Server zwróci komunikat błędu. Kryteria zawężenia określane są poprzez podanie warunku logicznego. Lista dopuszczalnych operatorów została zamieszczona w tabeli 7.1.

Tabela 7.1. Lista operatorów porównania języka SQL

Operator	Prawda, gdy
=	Równy
<>, !=	Nierówny
>	Większy niż
<	Mniejszy niż
>=	Nie mniejszy
<=	Nie większy
IN	Należy do
BETWEEN … AND	Zawiera się pomiędzy
LIKE	Jest zgodny ze wzorcem
IS	Istnieje
NOT <operator algebry relacji>	Warunek nie jest spełniony

Dodatkowo możliwe jest definiowanie złożonych warunków — warunków składających się z kilku warunków cząstkowych połączonych jednym z operatorów logicznych. Wynik

takiego warunku obliczany jest na podstawie wyników warunków cząstkowych i tabel prawdziwości operatora łączącego te warunki.

Tabela 7.2. Tabele prawdziwości operatorów logicznych

a	NOT a
1	0
0	1

a	b	a OR b
1	1	1
1	0	1
0	1	1
0	0	0

a	b	a AND b
1	1	1
1	0	0
0	1	0
0	0	0

Implementacja warunku logicznego użytego jako parametr zawężenia CHECK musi spełniać kilka ograniczeń:

- Nie może odwoływać się do wartości kolumn z innych tabel.
- Nie może używać funkcji niedeterministycznych (funkcji, które zwracają różne wartości w zależności od okoliczności ich wywołania).
- Musi być prosty — niedozwolone jest używanie podzapytań (zapytań zagnieżdżonych).

Zawężenie DEFAULT

Język Transact-SQL umożliwia automatyczne wprowadzenie określonej wartości do kolumny, jeżeli użytkownik pozostawi ją nieuzupełnioną. Na implementację zawężenia DEFAULT nałożone są ograniczenia bardzo podobne do tych, jakie nałożone są na zawężenie CHECK:

- Wyrażenie nie może odwoływać się do kolumn z innych tabel.
- Wyrażenie musi być proste — niedozwolone jest używanie podzapytań (zapytań zagnieżdżonych).

Natomiast dozwolone jest użycie funkcji, które zwracają różne wartości, w zależności od okoliczności ich wywołania, np. dozwolone jest użycie funkcji GETDATE() zwracającej aktualną datę i czas zegara systemowego.

Ostatnie trzy z dwunastu postulatów Codda związane są z zasadami dotyczącymi integralności danych. Są to:

- **Postulat wartości NULL.** W systemie jest dostępna specjalna wartość reprezentująca wartość nieokreśloną, brakującą lub nieznaną. Jest to wartość różna od wszelkich konkretnych wartości, w szczególności od ciągu pustego ("") i zera (0).
- **Postulat słownika danych.** Informacje o obiektach bazy danych tworzących jej schemat (metainformacje) są zgrupowane w tabelach i dostępne w taki sam sposób jak każde inne dane.
- **Niezależność więzów spójności.** Więzy spójności są definiowane w języku bazy danych i nie muszą być wyrażane w aplikacji.

Diagram związków E/R

Przygotowanie diagramu związków E/R (encja/relacja) jest pierwszym zadaniem projektanta bazy danych. Na jego podstawie określimy:

1. Typy obiektów, o których informacje będziemy przechowywać w bazie danych.

2. Liczbę i typ atrybutów opisujących obiekty każdego typu.
3. Wzajemne powiązania pomiędzy danymi dotyczącymi obiektów różnego typu.

Określanie typów obiektów

Podstawą diagramu E/R są encje. Encja reprezentuje określony typ obiektów. Załóżmy, że mamy zaprojektować bazę danych na zlecenie niewielkiego wydawnictwa sprzedającego swoje książki za pośrednictwem Internetu.

Podstawowym typem obiektów (encją) naszej bazy danych będzie książka.

Z każdą książką związany jest jej autor. Ponieważ trudno uznać dane o nazwisku czy imieniu autora za cechy napisanej przez niego książki, kolejną encją bazy będzie autor.

Na tej samej zasadzie jako odrębny typ obiektu należy uznać dane wydawnictwa które wydało książkę: w przypadku oryginału będzie to wydawnictwo zagraniczne, w przypadku książki wydanej po polsku — firma OCO lub wydawnictwo konkurencyjne. Trzecią encją bazy danych będzie wydawnictwo.

Firma OCO, prowadząc sprzedaż za pośrednictwem Internetu, ma możliwość zbierania podstawowych danych marketingowych o swoich klientach. Dane te, jako powiązane z książkami, również powinny znaleźć się w naszej bazie — czwartą encją będzie klient.

Podczas implementacji modelu okaże się, że encje odpowiadają tabelom tworzonej bazy danych. Okaże się również, że kompletna lista tabel bazy będzie dłuższa niż lista typów obiektów.

Określanie atrybutów obiektów poszczególnych typów

Atrybut jest właściwością, cechą obiektu, która jest przechowywana w bazie danych jako wartość pewnego typu, np. tytuł książki przechowywany jest jako wartość tekstowa, a informacja o liczbie stron jako liczba całkowita.

Po wyliczeniu typów obiektów należy wypisać listę atrybutów obiektów każdego typu. Na tym etapie projektowania należy kierować się zasadą:

Atrybut powinien opisywać obiekt, przy którym się go umieszcza.

Określmy listę atrybutów obiektów każdego typu:

książka — tytuł, numer ISBN, liczba stron, rok wydania, seria wydawnicza, dołączony nośnik, numer ISBN oryginału, notka redakcyjna,
autor — imię, nazwisko, adres, numer telefonu, e-mail, numer konta bankowego,
wydawnictwo — nazwa, adres, osoba kontaktowa, prezes, numer telefonu, e-mail, numer konta bankowego,
klient — imię, nazwisko, e-mail, data urodzenia, wykształcenie, stanowisko, miasto.

Wyodrębnianie danych elementarnych

Charakterystyczny dla relacyjnych baz danych jest wymóg, aby każdy z atrybutów przechowywany był w bazie danych jako niepodzielna, atomowa wartość. Atomową wartością danego atrybutu jest taka jego część, która nie jest dalej dzielona w celu uzyskania bardziej szczegółowych informacji. Na przykład, jeżeli planujemy znaleźć dane o imieniu autora pewnej książki, powinniśmy dane o imieniu traktować jako pojedynczy atrybut (a nie w połączeniu z informacjami o nazwisku). Innym przykładem jest adres wydawnictwa — o ile chcemy wykorzystywać jakikolwiek fragment informacji

składającej się na pełny adres firmy, dane takie powinny zostać zapisane jako odrębny atrybut. Na tym etapie projektowania powinniśmy kierować się zasadą:

Każdy atrybut opisujący dowolny obiekt powinien przyjmować atomową, niepodzielną wartość.

Wypiszmy listę danych elementarnych opisujący obiekty każdego typu:

książka {tytuł, numer ISBN, liczba stron, rok wydania, seria wydawnicza, dołączony nośnik, numer ISBN oryginału, notka redakcyjna, uwagi}

autor {imię, nazwisko, miasto, kod, ulica i nr domu, numer telefonu, e-mail, bank, numer konta bankowego, uwagi}

wydawnictwo {nazwa, kraj, miasto, kod, ulica i nr domu, osoba kontaktowa, prezes, numer telefonu, e-mail, bank, numer konta bankowego, uwagi}

klient {imię, nazwisko, e-mail, data urodzenia, wykształcenie, stanowisko, miasto, uwagi}

Określanie zależności funkcyjnych zachodzących pomiędzy atrybutami

Kolejnym etapem projektowania relacyjnej bazy danych jest określenie zależności funkcyjnych zachodzących pomiędzy atrybutami poszczególnych encji. Przypomnijmy — atrybuty zależą funkcyjnie od pewnej grupy atrybutów, jeżeli dla każdego zbioru wartości z pierwszej grupy jest przyporządkowany jeden i tylko jeden zespół wartości z grupy drugiej.

Na tym etapie należy kierować się zasadą:

Dla każdego typu obiektów (w każdej tabeli) należy zdefiniować jeden klucz główny jednoznacznie wskazujący na pozostałe atrybuty wszystkich obiektów danego typu.

W przypadku projektu bazy wydawnictwo dla każdej encji utworzymy dodatkowy atrybut o nazwie id_nazwa obiektu. Wartościami tego atrybutu będą niepowtarzalne, automatycznie generowane przez SQL Server liczby całkowite.

Przekonaliśmy się, że z każdym schematem relacji związany jest pewien zbiór zależności funkcyjnych określanych dużą literą F.

Dla zbioru atrybutów relacji r przyjmuje się założenie, na podstawie którego dla dowolnego podzbioru zbioru wszystkich atrybutów relacji r można określić zależności funkcyjne, niezależnie od tego, w jakich relacjach atrybuty te występują:

Lemat 2. Założenie o relacji uniwersalnej

Dla zadanego zbioru atrybutów R istnieje jedna relacja r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$, a wszystkie relacje s o schemacie $S = \{B_1, B_2, \ldots, B_m\}$ $(S \subseteq R)$ uzyskiwane są w wyniku projekcji relacji r. Zależność funkcyjna $X \rightarrow Y$ wynika logicznie ze zbioru zależności funkcyjnych F, co zapisujemy $F \models X \rightarrow Y$, jeżeli każda relacja r spełniająca wszystkie zależności ze zbioru F spełnia również zależność $X \rightarrow Y$.

Na podstawie lematu 2. możemy mówić o istnieniu uniwersalnego zbioru zależności funkcyjnych dla relacji r. Lemat 2., choć ma podstawowe znaczenie dla teorii relacyjnych baz danych, nie jest spełniony automatycznie i w praktyce nie zawsze jest przyjmowany, ponieważ wymagałoby to dodawania do schematu relacji dodatkowych atrybutów zbędnych z punktu widzenia użytkownika bazy danych.

Na przykładzie relacji reprezentowanej przez tabelę Categories przekonaliśmy się, że dla każdej relacji można wskazać niepusty zbiór zależności funkcyjnych spełnianych przez tą relację. Dodatkowo z przyjętego na podstawie natury odzwierciedlanej przez relację rzeczywistości zbioru zależności funkcyjnych można wyprowadzać nowe zależności.

Zagadnienie generowania pochodnych zależności funkcyjnych zostało opracowane przez Armstronga, który udowodnił, że możliwe jest wyprowadzanie jednych zależności funkcyjnych z drugich na podstawie analizy czysto syntaktycznej, bez konieczności odwoływania się do samej relacji.

Zagadnienie rozkładalności schematów relacji i związanie z nim aksjomaty Armstronga zostały opisane w dalszej części rozdziału.

Lemat 2. jest podstawą wykorzystywanej w dalszej części rozdziału definicji domknięcia zbioru zależności funkcyjnych.

Definicja 11. Domknięcie zależności funkcyjnych

Domknięciem zbioru zależności funkcyjnych *F* nazywamy zbiór wszystkich zależności logicznych wynikających logicznie z zależności funkcyjnych *F*. Domknięcie zależności funkcyjnych oznacza się symbolem *F+*.

Analizując model bazy danych, należy brać pod uwagę zbiór wszystkich zależności funkcyjnych występujących pomiędzy atrybutami, a więc zbiór F+, nie F.

Posługując się definicją 11., możemy rozszerzyć podane wcześniej definicje nadklucza i klucza relacji:

Definicja 9a. Nadklucz

Nadkluczem relacji *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* nazywamy dowolny zbiór atrybutów $X \subseteq R$ taki, że $X \rightarrow R \in F+$.

Definicja 10a. Klucz

Kluczem relacji *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* nazywamy każdy minimalny nadklucz, tj. dowolny zbiór atrybutów $X \subseteq R$ taki, że $X \rightarrow R \in F+$ i dla żadnego $Y \subseteq X$, $Y \neq X$ nie zachodzi $Y \rightarrow R \in F+$.

Grupowanie danych w tabelach

Tworząc projekt przykładowej bazy danych, założyliśmy, że dane powinny być przechowywane w kilku powiązanych ze sobą tabelach. Innymi słowy, uznaliśmy, że relacyjny model bazy danych jest najlepszym wyborem w naszej sytuacji. I chociaż prawie na pewno mieliśmy rację, należy wiedzieć, że oprócz relacyjnych baz danych szeroko wykorzystywane są jednorodne i obiektowe bazy danych. Przykładem jednorodnej bazy danych (ang. Flat File) jest arkusz z danymi programu Microsoft Excel czy pojedyncza tabela utworzona w Wordzie.

Zalety grupowania danych w odrębnych tabelach najłatwiej opisać, porównując relacyjny i jednorodny model baz danych.

Podstawowym powodem grupowania danych o obiektach różnych typów w odrębnych tabelach jest nadmiarowość danych przechowywanych w bazach jednorodnych. O ile

bardzo proste (opisujące niewielką liczbę atrybutów obiektów tego samego typu) bazy danych można tworzyć, implementując model jednorodny, to każda próba zapisania w takiej bazie większej liczby danych o obiektach różnego typu (np. książkach i autorach) skończy się wielokrotnym przechowywaniem tych samych danych w bazie. Innymi słowy, te same wartości zapisane będą nie raz, ale wiele razy, co nie tylko niepotrzebnie zwiększa rozmiar bazy danych, ale niekorzystnie wpływa na wydajność i utrudnia zarządzanie bazą. Na przykład, gdybyśmy przechowywali dane autorach razem z danymi o książkach, informacje o imieniu, nazwisku, adresie itd. każdego autora musiałyby zostać zapisane tyle razy, ile książek napisał dany autor.

Konsekwencjami nadmiarowości danych są:

- Anomalie występujące podczas modyfikacji danych. W modelu jednorodnym stosunkowo trudno jest zachować spójność i adekwatność danych. Gdyby w zmodyfikowanej (uzupełnionej o informacje o autorach) tabeli książka należało uaktualnić nasze uwagi o autorze, to informację tę musielibyśmy modyfikować tyle razy, ile książek napisał dany autor. W nieco bardziej skomplikowanych przypadkach mogłoby się okazać, że jedna drobna zmiana pociąga za sobą modyfikacje kilkudziesięciu czy kilkuset rekordów. A implementacja mechanizmu niedopuszczającego do sytuacji, w której dane o autorze zmienimy 66 razy, a w 13 komórkach pozostaną niezmienione, może okazać się dość skomplikowana.
- Anomalie występujące podczas dodawania danych. Kolejnym ograniczeniem modelu jednorodnego jest konieczność wprowadzania nowych danych o jednym obiekcie do wielu rekordów. W omawianym przypadku, gdybyśmy w końcu poznali adres e-mail jednego z autorów, SQL Server musiałby wpisać tę informację w liczbę pól równą ilości książek tego autora.
- Anomalie występujące podczas usuwana danych to kolejny powód, dla którego powinno się grupować dane opisujące obiekty różnego rodzaju w osobnych tabelach. W omawianym przykładzie nie byłoby możliwości wprowadzenia do bazy danych o autorze, którego książek wydawnictwo nie ma jeszcze w swojej ofercie. I odwrotnie — gdybyśmy usunęli dane o wszystkich książkach danego autora, automatycznie utracilibyśmy również dane o jego nazwisku, imieniu itd.

Kwestia grupowania danych w odrębnych tabelach została dokładnie opisana w dalszej części rozdziału, w tym miejscu podane zostały jedynie intuicyjne wskazówki, według których przygotujemy model Encja/Relacja przykładowej bazy danych.

Zagadnieniu grupowania danych w tabelach poświęcony jest podrozdział „Normalizacja”. Celem normalizacji jest wyeliminowanie wszystkich zależności funkcyjnych nie od klucza.

Określanie związków (relacji) zachodzących między encjami

Dotychczas przygotowaliśmy cztery zbiory atrybutów opisujących poszczególne typy obiektów. Jednak w bazie danych, oprócz atrybutów opisujących obiekty (atrybutów jednoargumentowych), powinny znaleźć się informacje o związkach zachodzących pomiędzy obiektami różnego typu (atrybuty wieloargumentowe).

W przykładowej bazie danych, tak jak w zdecydowanej większości relacyjnych baz danych, obiekty poszczególnych typów powiązane są ze sobą za pomocą związków dwuargumentowych, np.: książka została napisana przez autora, wydawnictwo wydało książkę, książka została kupiona przez klienta.

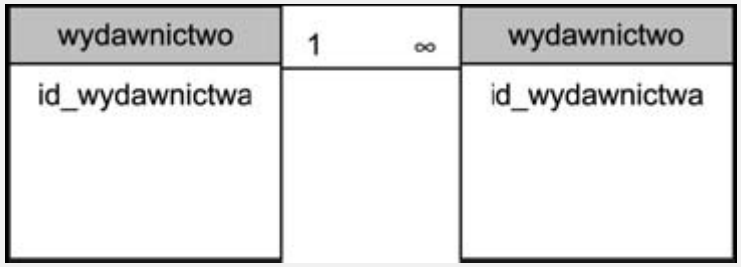
Wyróżnia się trzy typy związków dwuargumentowych:

1. Związek wieloznaczny typu wiele do wielu. Powiązania tego typu podczas implementacji projektu muszą zostać zamienione na równoważne związki typu jeden do wielu. W przykładowej bazie danych w ten sposób powiązane są encje książka i autor (jeden autor może napisać wiele książek i jedna książka może mieć wielu autorów).
2. Związek jednoznaczny typu jeden do wielu. Związki jednoznaczne, w przeciwieństwie do związku wieloznacznego, są bezpośrednio reprezentowane w bazie danych poprzez powiązanie odpowiednich atrybutów kluczowych obiektów.
3. Związek jednoznaczny typu jeden do jednego. Ten typ powiązania odpowiada funkcji jedno-jednoznacznej. Powiązania tego typu z reguły są wynikiem optymalizacji bazy danych, ponieważ na etapie projektowania najczęściej tak powiązane atrybuty opisują obiekty tego samego typu.

Sklasyfikujmy typ związków zachodzących pomiędzy przykładowymi tabelami:

- ♣ Każda książka może zostać napisana przez wielu autorów i jednocześnie każdy autor może napisać dowolną liczbę książek — tabele autor i książka łączy związek typu wiele do wielu.
- ♣ Każda książka może zostać wydana przez dokładnie jedno wydawnictwo, ale każde wydawnictwo może wydać dowolną liczbę książek — tabele wydawnictwo i książka łączy związek typu jeden do wielu.
- ♣ Każdy klient może kupić dowolną liczbę książek i ta sama książka1 może zostać kupiona przez wielu klientów — tabele klient i książka łączy związek typu wiele do wielu.

Jako pierwszy zaimplementujemy związek zachodzący pomiędzy tabelami wydawnictwo i książka — do tabeli książka dodamy kolumnę przechowującą wartości kluczy głównych tabeli wydawnictwo (rysunek 7.4).



Rysunek 7.4. Połączenie tabel wydawnictwo i książka

Ponieważ związki wielo-wieloznaczne nie mogą zostać bezpośrednio odwzorowane w relacyjnych bazach danych, do połączenia pozostałych tabel musimy utworzyć dodatkową tabelę łączącą odpowiednie tabele. Za pomocą tabeli przechowującej jedynie informacje o powiązanych ze sobą wartościach kluczy głównych obu tabel połączymy związkami typu jeden do wielu z oboma tabelami nadrzędnymi (rysunek 7.5).

Rysunek 7.5. Połączenie tabel autor i książka

Łącząc tabele klient i książka możemy dodatkowo wykorzystać fakt, że i tak utworzymy tabelę łącznikową. W takiej tabeli, oprócz kluczy głównych obu tabel, możemy

przechowywać dodatkowe informacje opisujące bezpośrednio samą operację kupna książki, takie jak data transakcji czy liczba zakupionych egzemplarzy (rysunek 7.6).

Rysunek 7.6. Połączenie tabel klient i książka poprzez tabelę zamówienie

W uproszczonym modelu pojedyncze zamówienie może być złożone przez dokładnie jednego klienta, a w ramach takiego zamówienia możliwe jest zamówienie tylko jednej pozycji (tytułu). W tym przypadku operacja zamówienia przez tego samego klienta w tym samym czasie różnych książek musi być zapisana jako zbiór pojedynczych zamówień. Rozwiązaniem tego problemu byłoby np. stworzenie powiązanych związkiem typu jeden do wielu tabel zamówienie i szczegóły zamówienia, tak jak ma to miejsce w bazie Northwind.

Końcowa lista encji bazy danych wydawnictwo wygląda następująco:

książka {id_książki, id_wydawnictwa, tytuł, numer ISBN, liczba stron, rok wydania,
seria wydawnicza, dołączony nośnik, numer ISBN oryginału, notka redakcyjna, uwagi}
autor {id_autora, imię, nazwisko, miasto, kod, ulica i nr domu, numer telefonu,
e-mail, bank, numer konta bankowego, uwagi}
wydawnictwo {id_wydawnictwa, nazwa, kraj, miasto, kod, ulica i nr domu, osoba
kontaktowa, prezes, numer telefonu, e-mail, bank, numer konta bankowego, uwagi}
klient {id_klienta, imię, nazwisko, e-mail, data urodzenia, wykształcenie, stanowisko,
miasto, uwagi}
autor_książka {id_autora, id_książki)
zamówienie (id_zamowienia, id_klienta, id_książki, data, egzemplarzy}

Normalizacja

Normalizacja modelu bazy danych polega na przekształcaniu danych do kolejnych, coraz wyższych postaci normalnych (schematów relacji spełniających coraz bardziej restrykcyjne kryteria logiczne). Każda wyższa postać normalna musi spełniać kryteria wszystkich postaci niższych. Tak więc baza danych spełniająca trzecią postać normalną z definicji spełnia również normy pierwszej i drugiej postaci normalnej. Dla większości baz danych, aby uniknąć problemów związanych ze wstawianiem, modyfikowaniem i usuwaniem danych, wystarcza znormalizowanie projektu do trzeciej postaci normalnej (3PN).

Pierwsza postać normalna 1PN

Mówimy, że encja znajduje się w pierwszej postaci normalnej, jeśli wszystkie jej atrybuty mają pojedynczą, niepodzielną wartość. Jeśli jakikolwiek atrybut składa się z kilku wartości, encja nie znajduje się w pierwszej postaci normalnej. Innymi słowy, baza danych spełnia założenia pierwszej postaci normalnej, jeżeli posiada tylko pola zawierające wartości atomowe (niepodzielne). Gdyby wszystkie tabele bazy danych spełniały pierwszą postać normalną, to związki typu jeden do wielu byłyby określone zawsze pomiędzy kilkoma tabelami (nigdy w obrębie jednej tabeli), a związek typu jeden do jednego mógłby zawierać się w jednej tabeli.

Pierwsza postać anormalna

Encje opisywane przez atrybuty posiadające złożone wartości (encję niespełniającą wymogów 1PN) nazywane są encjami znajdującymi się w pierwszej postaci anormalnej. Chociaż możliwe jest świadome zaplanowanie i utworzenie reprezentującej taką encję tabeli, to w większości przypadków jest to wynikiem błędu projektanta. W kolejnych punktach zostały opisane problemy związane z odczytywaniem i modyfikowaniem danych zapisanych w takiej tabeli oraz technika świadomego tworzenia tabel w pierwszej postaci anormalnej, dla odróżnienia nazywanych tabelami pomocniczymi (ang. Auxiliary Tables).

Anomalie wynikające z przechowywania danych w tabelach w pierwszej postaci anormalnej

Problemy wynikające z przechowywania informacji w zdenormalizowanej bazie danych przedstawimy na najprostszym z możliwych przykładzie: utworzymy bazę, w której zostaną zapisane informacje o przedmiotach charakteryzowanych za pomocą pewnej liczby atrybutów.
Właściwym rozwiązaniem byłoby utworzenie dwóch tabel (przedmiot i cecha) połączonych związkiem typu jeden do wielu. Definicja tych tabel znajduje się poniżej:

```
USE tempdb
CREATE TABLE [przedmiot] (
[przedmiot_ID] [int] NOT NULL,
[nazwa] [varchar] (50) COLLATE Polish_CI_AS NOT NULL,
CONSTRAINT [PK_przedmiot] PRIMARY KEY CLUSTERED
(
[przedmiot_ID]
) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [cecha] (
[przedmiot_ID] [int] NOT NULL,
[opis] [varchar] (50) COLLATE Polish_CI_AS NOT NULL,
CONSTRAINT [FK_cecha_przedmiot] FOREIGN KEY
([przedmiot_ID]
) REFERENCES [przedmiot] (
[przedmiot_ID])
) ON [PRIMARY]
GO
```

```
The command(s) completed successfully.
```

W ten sposób utworzyliśmy powiązane związkiem typu jeden do wielu dwie tabele, co pozwala nam przypisywać dowolną liczbę cech do każdego przedmiotu. Zarówno odczyt, jak i modyfikacja danych jest sprawą łatwą i szybką, np. aby dodać nową cechę przedmiotu o identyfikatorze równym 1, napiszemy:

```
INSERT INTO przedmiot
VALUES (1,'Cudowne cygaro dziadka')
GO
INSERT INTO cecha
VALUES (1,'bezcenny')
GO
(1 row(s) affected)
(1 row(s) affected)
```

Natomiast aby wyświetlić nazwę przedmiotu posiadającym cechę bezcenny, napiszemy:

```
SELECT nazwa
FROM przedmiot JOIN cecha
ON przedmiot.przedmiot_ID = cecha.przedmiot_ID
AND opis = 'bezcenny'
GO
nazwa
--------------------------------------------------
Cudowne cygaro dziadka
(1 row(s) affected)
```

Jak natomiast wyglądałyby te operacje w przypadku utworzenia tabeli w pierwszej postaci anormalnej? Rozpatrzymy kolejno dwa przypadki: pierwszy, w którym utworzymy tabelę przedmiot i dodamy do niej kilka kolumn nazwanych odpowiednio cecha1, cecha2, …, cechan, oraz drugi, w którym poszczególne cechy przedmiotu będziemy zapisywać w tej samej kolumnie o nazwie cechy. Poniżej znajdują się definicje obu tabel:

```
CREATE TABLE [przedmiot1] (
[przedmiot_ID] [int] NOT NULL,
[nazwa] [varchar] (50) COLLATE Polish_CI_AS NOT NULL,
[cecha1] [varchar] (50) COLLATE Polish_CI_AS,
[cecha2] [varchar] (50) COLLATE Polish_CI_AS,
[cecha3] [varchar] (50) COLLATE Polish_CI_AS,
[cecha4] [varchar] (50) COLLATE Polish_CI_AS,
CONSTRAINT [PK_przedmiot1] PRIMARY KEY CLUSTERED
(
```

```
[przedmiot_ID]
) ON [PRIMARY])
CREATE TABLE [przedmiot2] (
[przedmiot_ID] [int] NOT NULL,
[nazwa] [varchar] (50) COLLATE Polish_CI_AS NOT NULL,
[cechy] [varchar] (250) COLLATE Polish_CI_AS,
CONSTRAINT [PK_przedmiot2] PRIMARY KEY CLUSTERED
(
[przedmiot_ID]
) ON [PRIMARY])
GO
The command(s) completed successfully.
```

Oczywistym ograniczeniem tabeli przedmiot1 jest możliwość przechowywania informacji o maksymalnie czterech cechach każdego przedmiotu. Możemy co prawda dodać nowe kolumny, ale to z kolei spowoduje, że liczba wartości NULL przechowywanych w tabeli będzie duża, a w związku z tym wykorzystanie zasobów komputera (przestrzeni dysku, pamięci operacyjnej i procesora) będzie nieefektywne. Dodajmy do tabeli przedmiot1 dane o bezcennym dla nas cygarze:

```
INSERT INTO przedmiot1 (przedmiot_ID, nazwa, cecha1)
VALUES (1,'Cudowne cygaro dziadka','bezcenny')
GO
(1 row(s) affected)
```

Odczytanie danych o nazwach produktów, które uznaliśmy za bezcenne w przypadku tej tabeli będzie wymagać napisania poniższej instrukcji:

```
SELECT nazwa
FROM przedmiot1
WHERE cecha1 = 'bezcenny'
OR cecha2 = 'bezcenny'
OR cecha3 = 'bezcenny'
OR cecha4 = 'bezcenny'
GO
nazwa
--------------------------------------------------
Cudowne cygaro dziadka
(1 row(s) affected)
```

Ponadto modyfikacja danych zapisanych w tej tabeli nie jest już ani łatwa, ani szybka — np., aby zmienić cechę bezcenny towaru o numerze 1 na grzechu warte, napiszemy:

```
UPDATE przedmiot1
```

```
SET cecha1 = 'grzechu warte'
WHERE przedmiot_ID=1 AND cecha1='bezcenny'
UPDATE przedmiot1
SET cecha2 = 'grzechu warte'
WHERE przedmiot_ID=1 AND cecha2='bezcenny'
UPDATE przedmiot1
SET cecha3 = 'grzechu warte'
WHERE przedmiot_ID=1 AND cecha3='bezcenny'
UPDATE przedmiot1
SET cecha4 = 'grzechu warte'
WHERE przedmiot_ID=1 AND cecha4='bezcenny'
GO
(1 row(s) affected)
(0 row(s) affected)
(0 row(s) affected)
(0 row(s) affected)
```

Tylko w ten sposób, sprawdzając kolejno wartości poszczególnych kolumn, możemy mieć pewność, że opis cechy danego przedmiotu zostanie zmodyfikowany. Kolejne kłopoty czekają nas, jeżeli na przykład będziemy chcieli dodać cechę jakiemuś istniejącemu w tabeli przedmiotowi — w tym wypadku czeka nas sprawdzanie, która z kolumn cecha1 … cecha4 zawierała wartość NULL i wpisanie nowej wartości tylko do jednej (pierwszej) znalezionej w ten sposób kolumny. Odpowiednikiem pojedynczej instrukcji INSERT będzie w tym wypadku poniższa instrukcja:

```
UPDATE przedmiot1
SET cecha1 = 'ostatnie takie cygaro'
WHERE przedmiot_ID=1 AND cecha1 IS NULL
IF @@ROWCOUNT = 0
BEGIN
UPDATE przedmiot1
SET cecha2 = 'ostatnie takie cygaro'
WHERE przedmiot_ID=1 AND cecha2 IS NULL
IF @@ROWCOUNT = 0
BEGIN
UPDATE przedmiot1
SET cecha3 = 'ostatnie takie cygaro'
WHERE przedmiot_ID=1 AND cecha3 IS NULL
IF @@ROWCOUNT = 0
BEGIN
UPDATE przedmiot1
SET cecha4 = 'ostatnie takie cygaro'
WHERE przedmiot_ID=1 AND cecha4 IS NULL
END
```

```
END
END
GO
(0 row(s) affected)
(1 row(s) affected)
```

Aby dodać cechę obiektu, musimy sprawdzić, korzystając z funkcji systemowej @@ROWCOUNT i instrukcji warunkowej IF, czy wykonanie przedniej instrukcji UPDATE nie spowodowało zmodyfikowania wiersza,.

Tabela przedmiot2 na pierwszy rzut oka wydaje się pozbawiona niektórych wad tabeli przedmiot1 — np. możemy zapisać informację o dowolnej liczbie cech przedmiotu bez konieczności modyfikowania struktury tabeli. Problemy związane z odczytywaniem lub modyfikacją danych w tej tabeli też wydają się możliwe do rozwiązania, zwłaszcza że SQL Server umożliwia nam pełnotekstowe wyszukiwanie danych oraz oferuje bogaty zestaw systemowych funkcji tekstowych.

Dodajmy do tabeli przedmiot2 informacje o cudownym cygarze dziadka:

```
INSERT INTO przedmiot2
VALUES (1,'Cudowne cygaro dziadka','bezcenny')
GO
(1 row(s) affected)
```

Zobaczmy, jak wyglądałoby w tym przypadku wyświetlenie nazw towarów uznanych za bezcenne:

```
SELECT nazwa
FROM przedmiot2
WHERE cechy LIKE '%bezcenny%'
GO
nazwa
--------------------------------------------------
Cudowne cygaro dziadka
(1 row(s) affected)
```

To rozwiązanie wydaje się przynajmniej proste, skoro już nie jest wydajne (użycie operatora LIKE z dwoma symbolami wieloznacznymi nigdy nie jest dobrym rozwiązaniem). Niestety, zapytanie może nam zwrócić więcej rekordów niż się spodziewaliśmy, np. jeżeli jedną z cech jest duży, a inną bardzo duży, do wyniku wyszukiwania dużych przedmiotów dołączone zostałyby przedmioty bardzo duże. Pokazane poniżej rozwiązanie tego problemu nie ma już nic wspólnego z programowaniem strukturalnym — konieczne okazuje się napisanie instrukcji, która znak po znaku sprawdzi wszystkie cechy wybranych obiektów i na podstawie określonego znaku separatora (w przykładzie takim znakiem jest przecinek) zapisze poszczególne cechy w odrębnych wierszach tabeli tymczasowej:

```
INSERT INTO przedmiot2 --dodajemy do tabeli przedmiot o atrybucie 'bardzo duży'
```

```
VALUES (2,'Szkielet dinozaura','bardzo duży')
GO
(1 row(s) affected)
CREATE TABLE #Przed (
cechy varchar (250))
DECLARE @dl int, @poz int, @znaki varchar (250)
DECLARE @znak char (1), @cechy varchar (250)
SET @poz=1
SET @znaki = ''
SELECT @cechy = cechy FROM przedmiot2
WHERE przedmiot_ID=2 --pobieramy wyłącznie cechy wybranego obiektu
SET @dl = LEN(@cechy)+1
WHILE @poz<=@dl
BEGIN
IF substring (@cechy,@poz,1) = ','
OR @poz=(LEN(@cechy)+1)
BEGIN
INSERT #Przed
VALUES (@znaki)
SET @znaki = ''
SET @poz = @poz +1
END
ELSE
BEGIN
SET @znak = SUBSTRING (@cechy,@poz,1)
SET @znaki = @znaki + @znak
SET @poz = @poz +1
END
END
SELECT nazwa
FROM Przedmiot2
WHERE EXISTS (SELECT *
FROM #Przed
WHERE cechy = 'duży')
AND przedmiot_ID=2
GO
(1 row(s) affected)
nazwa
----------------------------------------------------
(0 row(s) affected)
SELECT nazwa
FROM Przedmiot2
WHERE EXISTS (SELECT *
```

```
FROM #Przed
WHERE cechy = 'bardzo duży')
AND przedmiot_ID=2
GO
nazwa
--------------------------------------------------
Szkielet dinozaura
(1 row(s) affected)
```

Chociaż dodanie nowej cechy okazuje się stosunkowo proste, to już usunięcie jednej z cech obiektu wymaga napisania odrębnej funkcji lub procedury. Poniższa instrukcja dopisze cechę ostatnie takie cygaro obiektowi o identyfikatorze równym 1:

```
UPDATE przedmiot2
SET cechy = cechy + ', ostatnie takie cygaro'
WHERE przedmiot_ID=1
GO
(1 row(s) affected)
```

Próbując usunąć jedną z cech przedmiotu, napotkamy tę samą trudność co przy odczytywaniu zawartości tabeli — nie wystarczy użyć funkcji REPLACE, ponieważ nazwy niektórych cech mogą zawierać takie same słowa. Konieczne okazuje się użycie funkcji lub procedury, która sprawdzi, czy wartość danej cechy zapisana jest pomiędzy znakami separatora (i wtedy zostanie zastąpiona pustym ciągiem znaków), czy też znajduje się w sąsiedztwie innego znaku.

Tabele pomocnicze

Zgodnie z wymogami 1PN, aby w tabeli zapisać informacje o 8 atrybutach, niezbędne jest utworzenie 8 kolumn. Niekiedy jednak wartości poszczególnych atrybutów przyjmują jedną z kilku określonych wartości, a w skrajnych sytuacjach przyjmują tylko jedną z dwóch wartości: 1 (prawda) lub 0 (fałsz). W takim przypadku, zamiast tworzyć 15 kolumn typu bit, możemy zapisać tę samą informację w poszczególnych bitach jednej kolumny typu integer lub binary.

Przykładem tabeli pomocniczej jest występująca w każdej bazie danych tabela systemowa sysprotects przechowująca w kolumnie action informację o nadanych uprawnieniach do wykonywania poszczególnych operacji, takich jak wykonywanie instrukcji SELECT, INSERT czy UPDATE.

Decydując się na takie rozwiązanie, musimy dysponować narzędziem umożliwiającym odczytanie wartości poszczególnych bitów zapisanych w kolumnie. Rozwiązaniem są dwa operatory: OR (|) oraz AND (&). Oba są dwuargumentowymi operatorami analizującymi poszczególne bity przekazanych danych typu integer lub binary.

Załóżmy, że w tabeli pomocniczej będziemy przechowywali dane o uprawnieniach poszczególnych użytkowników. Nadane uprawnienie do wykonania instrukcji SELECT będzie symbolizowane za pomocą pierwszego bitu, instrukcji INSERT — za pomocą drugiego, UPDATE — trzeciego, a DELETE — czwartego. Aby nadać użytkownikowi uprawnienia do odczytu, modyfikowania i usuwania wierszy, należy ustawić wartości 1,

3 i 4 bitu na 1. Binarną reprezentację liczby z zakresu od 0 do 255 obliczamy na podstawie wagi poszczególnych bitów:

$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^7 + 2^1 + 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$

W naszym przypadku należy ustawić wartość 1, 3 i 4 bitu. Najprostszym rozwiązaniem jest wykorzystanie operatora OR do nałożenia maski 1 | 4 | 8 na wstawianą do kolumny wartość liczbową. Otrzymana liczba — 13 — reprezentuje nadanie trzech wspomnianych uprawnień.

Operator OR może zostać również wykorzystany do łączenia uprawnień nadanych użytkownikowi poprzez członkostwo w różnych rolach — jeżeli użytkownik należy do roli r1, której nadano uprawnienia do odczytu i usuwania danych oraz do roli r2, której nadano uprawnienia do odczytu i modyfikowania danych, wynikowe uprawnienia użytkownika obliczymy następująco:

Uprawnienia roli r1 (SELECT i DELETE) powodują ustawienie 1 i 4 bitu, uprawnienia roli r2 (SELECT i UPDATE) — bitu 1 i 3. W pierwszym wypadku liczbową reprezentacją uprawnień będzie liczba 9 (8+1), w drugim — 5 (4+1). Aby odczytać wynikowe uprawnienia, wystarczy wykonać operację 9 | 5, w wyniku której otrzymamy liczbę 13.

Wykorzystując operator & możemy sprawdzić, czy użytkownik ma nadane konkretne uprawnienie (mnożąc logicznie wartość zapisaną w kolumnie przez maskę określającą interesujący nas bit). Na przykład, aby sprawdzić, czy użytkownik, którego uprawnienia reprezentowane są przez liczbę 13, ma możliwość dodawania nowych wierszy (uprawnienie INSERT odzwierciedla drugi bit), wykonamy operację 13 & 2, otrzymując w wyniku 0 — użytkownik nie ma nadanych uprawnień do wykonywania instrukcji INSERT.

Druga postać normalna 2PN

Encja odpowiada wymogom drugiej postaci normalnej, jeżeli spełnia dwa założenia:

- Tabela jest w 1PN.
- Kolumny niewchodzące w skład klucza głównego są zależne od całego klucza (a nie od dowolnej jego części).

Tak więc encja jest w drugiej postaci normalnej, jeśli znajduje się w pierwszej postaci normalnej i wszystkie zależne funkcyjnie atrybuty są zależne od całego niepowtarzalnego identyfikatora (klucza głównego). Jeśli jakikolwiek atrybut nie jest całkowicie zależny od klucza głównego, należy zmodyfikować projekt bazy danych. Atrybuty takie normalizuje się albo poprzez odnalezienie encji, którą bezpośrednio opisują te atrybuty, albo przez utworzenie dodatkowej encji, do której atrybut powinien zostać dodany.

Trzecia postać normalna 3PN

Encja jest w 3PN, jeżeli wszystkie atrybuty spełniają założenia dwóch poprzednich postaci, a ponadto każdy z atrybutów nienależących do klucza głównego nie zależy funkcyjnie od jakiegokolwiek nadklucza. Innymi słowy, encja znajduje się w trzeciej postaci normalnej, kiedy jest już w drugiej postaci normalnej i żadne nieidentyfikujące atrybuty nie są zależne od innych nieidentyfikujących atrybutów (nie występują zależności przechodnie). Atrybuty, które są zależne od innych nieidentyfikujących atrybutów są normalizowane przez przeniesienie do nowej encji zarówno zależnego atrybutu, jak i atrybutu, od którego jest on zależny.

Przekształcenie modelu do 3PN często sprowadza się do wyodrębnienia encji do tej pory traktowanych jako zbiory atrybutów opisujących encje nadrzędne i utworzenie dla nich odrębnych tabel słownikowych.

Tworzenie tabel słownikowych

Tabele słownikowe to tabele pomocnicze przechowujące wartości poszczególnych atrybutów opisujących jeden z typów obiektów. Dobrymi kandydatami na tabele słownikowe są te atrybuty, których zakres dopuszczalnych wartości jest ściśle określony. Na przykład atrybut seria wydawnicza może przyjąć jedną z następujących wartości: od A do Z, tylko dla orłów, nie panikuj!, pisma i traktaty. Tworząc odrębną tabelę przechowującą dopuszczalne wartości tego atrybutu:

- Ułatwiamy użytkownikowi korzystanie z bazy danych. Zamiast wielokrotnie wpisywać zwrot nie panikuj!, użytkownik będzie mógł wybrać go z listy wszystkich serii wydawniczych.
- Zmniejszamy ryzyko utraty spójności danych. Wprowadzając nowe informacje, użytkownicy często popełniają błędy i np. zamiast nie panikuj! mogliby wpisać: nie panikuj, nie panikój!, itp. Gdybyśmy chcieli na podstawie takich danych przygotowywać statystyki, otrzymane wyniki byłyby nieprawdziwe.
- Poprawiamy wydajność bazy danych. Przenosząc wartości tekstowe pozą główną tabelę i wiążąc ją za pomocą kluczy zdefiniowanych jako liczby całkowite, poprawiamy wydajność operacji odczytu oraz modyfikacji danych w tabeli. Dodatkowo niektóre raporty będą mogły być obliczane na podstawie zawartości tabeli słownikowej i nie będą wymagały odczytu danych przechowywanych w tabeli głównej.

W przykładowej bazie danych można wyodrębnić tabele słownikowe dla wszystkich czterech początkowo wyróżnionych encji:

Dla tabeli książka należy zdefiniować dwie tabele słownikowe: tabelę seria wydawnicza i nośnik. Projektując tabele słownikowe, będziemy kierować się uwagami zawartymi wcześniej w bieżącym rozdziale:

seria wydawnicza: {id_serii, seria, uwagi}

nośnik: {id_nośnika, nośnik, uwagi}

Tabele słownikowe powiązane są z tabelą nadrzędną związkiem typu jeden do wielu (książka może być wydana tylko w jednej serii, ale w ramach każdej z serii można wydać dowolną liczbę książek). Związki tego typu implementuje się, dodając do listy kolumn tabeli nadrzędnej tabelę zawierającą klucz główny tabeli słownikowej, czyli tworząc klucz zewnętrzny dla tej tabeli. Końcowy projekt tabeli książka będzie wyglądał następująco:

książka: {id_książki, tytuł, isbn, liczba_stron, rok_wydania, isbn_oryginalu,

notka_redakcyjna, uwagi, id_serii, id_nośnika}

Dla tabeli autor można znaleźć tylko jeden atrybut nadający się do przeniesienia do tabeli słownikowej — miasto. Dodatkową korzyścią z utworzenia tej tabeli słownikowej będzie to, że będzie ona użyteczna podczas projektowania tabel wydawnictwo i klient:

miasto: {id_miasta, miasto, mieszkancow, uwagi}

autor: {id_autora, imię, nazwisko, kod, ulica_nr_domu, telefon, e-mail, bank, konto,

uwagi, id_miasta}

Projektując tabelę wydawnictwo, na liście atrybutów opisujących wydawnictwo umieściliśmy m.in. miasto, kraj, osobę kontaktową i prezesa. Każdy z tych atrybutów powinien zostać przeniesiony poza tabelę nadrzędną. Chwilę uwagi powinniśmy poświęcić dwóm ostatnim z wymienionych atrybutów — prawdopodobnie lista danych o osobie kontaktowej (takich jak imię czy nazwisko) będzie identyczna z listą danych o prezesie danego wydawnictwa. Skoro tak, to zamiast tworzyć dwie tabele, możemy utworzyć jedną, a do listy kolumn dodamy wtedy kolumnę określającą, czy dana osoba jest osobą kontaktową, czy prezesem wydawnictwa:

osoba: {id_osoby, imię, nazwisko, telefon, e-mail, prezes, uwagi}

kraj: {id_kraju, kraj, uwagi}

wydawnictwo: {id_wydawnictwa, nazwa, kod, ulica_nr_domu, telefon, e-mail, bank, konto,

uwagi, id_osoby, id_prezesa, id_kraju, id_miasta}

Ostatnią tabelą nadrzędną jest tabela klient. Na liście atrybutów, które powinny zostać przeniesione do tabel słownikowych, znajdują się wykształcenie, stanowisko oraz miasto. Pierwsze dwie tabele będą typowymi tabelami słownikowymi, przechowującymi wyłącznie listę pojedynczych wartości, trzecia tabela została utworzona już wcześniej:

wyksztalcenie: {id_wyksztalcenia, wykształcenie}

stanowisko: {id_stanowiska, stanowisko}

klient: {id_klienta, imie, nazwisko, e-mail, data_urodzenia, uwagi, id_wyksztalcenia,

id_stanowiska, id_miasta}

Algorytmy sprowadzania relacji do wyższej postaci normalnej

Do tej pory, częściowo dzięki intuicji, utworzyliśmy poprawny model bazy danych i sprowadziliśmy go do 3PN. Bieżący punkt zawiera bardziej formalne podejście do zagadnienia normalizacji modeli baz danych.

Pierwszą czynnością, którą należy wykonać, przekształcając relację do poprawnych postaci, jest sprawdzenie, czy dana zależność funkcyjna wynika logicznie ze zbioru zależności funkcyjnych F.

Definicja 12. Domknięcie tranzytywne

Domknięciem tranzytywnym zbioru $X \subseteq R$ względem zbioru zależności funkcyjnych F jest zbiór atrybutów $X+ = \{A \in R: X \rightarrow A \in F+\}$. Oznacza to, że domknięcie tranzytywne jest zbiorem wszystkich tych atrybutów, których wartości są zdeterminowane przez wartości atrybutów należących do zbioru X.

Twierdzenie 15. Wynikanie logiczne zależności funkcyjnych

Zależność funkcyjna *X -> Y* wynika ze zbioru zależności funkcyjnych *F* wtedy i tylko wtedy, gdy zbiór atrybutów *Y* należy do domknięcia tranzytywnego zbioru *X*, co zapisujemy *X -> Y ∈ F+* wtedy i tylko wtedy, gdy *Y ⊆ X+*.

Na mocy twierdzenia 15., sprawdzenie, czy dana zależność funkcyjna wynika logicznie ze zbioru zależności F, sprowadza się do wyznaczenia domknięcia tranzytywnego zbioru X.

Do wyznaczenia domknięcia tranzytywnego zbioru X posłużymy się poniższym algorytmem:

Konstruujemy następujący ciąg zbiorów:

1. X0 = X,
2. Xi+1 = Xi plus zbiór atrybutów A takich, że istnieje zależność funkcyjna Y -> Z ∈ F, gdzie A ⊆ Z oraz Y ⊆ Xi,
3. Krok 2 powtarzamy, dopóki nie zajdzie równość Xi = Xi+1. Wtedy szukany zbiór X+ = Xi.

Przykład:

Dla relacji *r* o schemacie *R = {A, B, C, D, E}* i zbiorze zależności funkcyjnych *F = {B, E -> C, C -> A, A, B, C -> D, B, C -> D, B, D -> E}* i dla zbioru *X = {B, D}* wyznaczamy:

X0 = {B, D},
X1 = {B, D, E} ponieważ istnieje zależność B, D -> E spełniona przez wygenerowane już atrybuty zbiory Y, mamy prawo wygenerować wszystkie atrybuty zbioru Z, w tym wypadku E,
X2 = {B, C, D, E} ponieważ istnieje zależność B, E -> C,
X3 = {A, B, C, D, E} ponieważ istnieje zależność C -> A.

Okazuje się, że zbiór *X+ = R*, co oznacza, że zbiór *X = {B, D}* jest nadkluczem relacji. Dodatkowo, ponieważ nie istnieje żaden nadklucz będący podzbiorem zbiory X, zbiór ten jest kluczem relacji.

Tworząc model E/R przykładowej bazy danych dzieliliśmy atrybuty pomiędzy odrębne tabele, kierując się intuicyjną wskazówką, według której atrybut powinien opisywać encję, przy której się go umieszcza. Istnieją określone warunki, przy których taki podział (rozkład) relacji jest logicznie poprawny.

Zależności funkcyjne z istniejącego zbioru zależności dla danej relacji możemy wyprowadzać, posługując się aksjomatami Armstronga:

Aksjomat 1. Zwrotność zależności funkcyjnych

Dla relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* będącym podzbiorem wszystkich zależności funkcyjnych o postaci *X -> Y (X, Y ⊆ R)* jeżeli *Y ⊆ X* to *X -> Y ⊆ F+*.

Aksjomat 2. Poszerzalność zależności funkcyjnych

Dla relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* będącym podzbiorem wszystkich zależności funkcyjnych o postaci *X -> Y (X, Y ⊆ R)* jeżeli *X -> Y ∈ F+* to *XZ -> YZ ∈ F+*.

Aksjomat 3. Przechodniość zależności funkcyjnych

Dla relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych F będącym podzbiorem wszystkich zależności funkcyjnych o postaci $X \rightarrow Y$ $(X, Y \subseteq R)$ jeżeli $X \rightarrow Y \in F+$ *i* $y \rightarrow Z \in F+$ *to* $X \rightarrow Z \in F+$.

Z aksjomatów Armstronga można wyprowadzić kolejne trzy reguły przekształcania zależności funkcyjnych:

Aksjomat 4. Pseudoprzechodniość zależności funkcyjnych

Dla relacji *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych F będącym podzbiorem wszystkich zależności funkcyjnych o postaci $X \rightarrow Y$ $(X, Y \subseteq R)$ jeżeli $X \rightarrow Y \in F+$ *i* $YW \rightarrow Z \in F+$ *to* $XW \rightarrow Z \in F+$.

Aksjomat 5. Addytywność zależności funkcyjnych

Dla relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych F będącym podzbiorem wszystkich zależności funkcyjnych o postaci $X \rightarrow Y$ $(X, Y \subseteq R)$ jeżeli $X \rightarrow Y \in F+$ *i* $X \rightarrow Z \in F+$ *to* $X \rightarrow YZ \in F+$.

Aksjomat 6. Dekompozycyjność zależności funkcyjnych

Dla relacji r o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych F będącym podzbiorem wszystkich zależności funkcyjnych o postaci $X \rightarrow Y$ $(X, Y \subseteq R)$ jeżeli $X \rightarrow YZ \in F+$ *to* $X \rightarrow Y \in F+$ *i* $X \rightarrow Z \in F+$.

Ponieważ z powyższych aksjomatów wynika, że zbiór F+ może zostać utworzony (wygenerowany) na podstawie różnych zbiorów zależności funkcyjnych, wprowadzono pojęcie minimalnego generatora zbioru F+.

Definicja 13. Minimalny generator domknięcia tranzytywnego

Minimalnym generatorem zbioru $F+$ nazywamy taki najmniejszy podzbiór F_0 zbioru F $(F_0 \subseteq F)$, dla którego $F_0 + = F+$.

Poprzednio zaznaczyłem, że proces normalizacji polega na wyeliminowaniu ze zbioru F+ wszystkich zależności funkcyjnych nie od klucza. Teraz widzimy, że polega to na wyznaczeniu minimalnego generatora zbioru F+.

Lemat 3. Rozkład relacji

Rozkład relacji jest poprawny logicznie wtedy i tylko wtedy, gdy w jego wyniku zachowane zostały zarówno informacje, jak i zależności funkcyjne istniejące w relacji początkowej.

Definicja 14 Poprawny rozkład relacji

Niech $\pi Z(F) = \{X \rightarrow Y \in F+: X \cup Y \subseteq Z\}$. Rozkład Q relacji R i S, co zapisujemy $Q = R \cup S$, zachowuje:

- ♣ informacje, jeżeli dla każdej relacji q spełniającej zależności funkcyjne F zachodzi równość $q = \pi R(q) \oplus \pi S(q)$,

- ♣ zależności funkcyjne, jeżeli zachodzi równość $F+ = (\pi R(F) \cup \pi S(F))+$.

Twierdzenie 16. Rozkład bez straty danych

Jeżeli relacja *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* rozkłada się bez straty danych na dwa schematy $R = X \cup Y$ *i* $R = X \cup Z$, gdzie $X \cup Y \cup Z = \{A_1, A_2, \ldots, A_n\}$ *i* $Y \cap Z = x$ *to* $X \to Y \in F+$ *lub* $X \to \in F+$.

Ponieważ nie każdy rozkład bez straty zależności jest rozkładem bez straty danych w procesie normalizacji dąży się do rozłożenia schematu relacji na składowe niezależne:

Definicja 15. Rozkład na składowe niezależne

Mówimy, że relacja *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ i zbiorze zależności funkcyjnych *F* rozkłada się na relacje niezależne s o schemacie $S = \{B_1, B_2, \ldots, B_m\}$ i zbiorze zależności funkcyjnych *G* oraz *t* o schemacie $T = \{C_1, C_2, \ldots, C_l\}$ i zbiorze zależności funkcyjnych *H* wtedy i tylko wtedy, gdy:

1. $B \cup C = A$,
2. $F+ = (G \cup H)+$,
3. każda relacja $r \in R = \{A_1, A_2, \ldots, A_n\}$ rozkłada się bez straty danych na dwie relacje $s \in S = \{B_1, B_2, \ldots, B_m\}$ i $t \in T = \{C_1, C_2, \ldots, C_l\}$.

Na podstawie wymienionych praw możliwe jest przekształcenie schematu relacji do dowolnej postaci normalnej. Postacie wyższe od 3PN (postać normalna Boyce'a-Codda oraz 4PN) stosunkowo rzadko spotykane są w produkcyjnych bazach danych.

Metoda dekompozycji Codda-Fagina

Metoda dekompozycji polega na kolejnym wykonaniu następujących operacji:

1. Określenie zbioru atrybutów kluczowych relacji *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$.
2. Wykrycie niepełnych zależności funkcyjnych i rozłożenie relacji *r* o schemacie $R = \{A_1, A_2, \ldots, A_n\}$ na zbiór relacji znajdujących się w 2PN.
3. Określenie zbioru atrybutów kluczowych każdej z otrzymanych relacji.
4. Wykrycie tranzytywnych (przechodnich) zależności funkcyjnych w każdej z otrzymanych relacji i sprowadzenie ich do 3PN.

Metoda ta gwarantuje rozkład z zachowaniem danych, ale w pewnych sytuacjach może prowadzić do utraty zależności funkcyjnych.

Uproszczona metoda Bernsteina

Normalizacja projektu bazy danych według metody Bernsteina polega kolejno na:

1. Wyeliminowaniu zbędnych atrybutów z lewych stron zależności funkcyjnych.
2. Znalezieniu minimalnego generatora otrzymanego w poprzednim punkcie zbioru zależności funkcyjnych (na podstawie wymienionych aksjomatów eliminuje się te zależności, które można wyprowadzić z zależności podstawowych).
3. Takim podziale znalezionego minimalnego generatora na rozłączne zbiory zależności, aby wszystkie otrzymane zależności cząstkowe miały identyczne lewe strony.

4. Pogrupowaniu tych z otrzymanych w poprzednim punkcie zależności, dla których zarówno $X \rightarrow Y \in F+$, jak i $Y \rightarrow X \in F+$.
5. Skonstruowaniu schematów relacji dla wszystkich otrzymanych grup zależności funkcyjnych. W tak otrzymanym schemacie relacji atrybuty znajdujące się po lewej stronie zależności funkcyjnych pełnią funkcję klucza głównego.

Metoda Niekludowej-Calenki

Postępowanie według algorytmu Niekludowej-Calenki gwarantuje, w przeciwieństwie do poprzednio przedstawionych metod, uzyskanie minimalnego rozkładu poprawnego, tj. rozkładu zachowującego zarówno zależności funkcyjne, jak i dane. Kolejno należy przeprowadzić następujące operacje:

1. Utworzyć zbiór wszystkich atrybutów kluczowych.
2. Dokonać projekcji zależności funkcyjnych na otrzymany zbiór atrybutów kluczowych.
3. Usunąć ze zbioru atrybutów kluczowych wszystkie zbędne atrybuty.
4. Usunąć z pozostałego zbioru wszystkie atrybuty zależne od atrybutów kluczowych.
5. Utworzyć zbiór U1, dołączając do niego atrybuty występujące po prawej stronie zależności funkcyjnych usuniętych w punkcie 4. Jeżeli zbiór U1 będzie równy zbiorowi wszystkich atrybutów, to otrzymano schemat będący w 3PN, co kończy działanie algorytmu. W przeciwnym wypadku należy przejść do punktu 6.
6. Sprawdzić, czy zachodzi równość $U1 = \{A\}$ gdzie A jest jedną z zależności funkcyjnych występujących w zbiorze otrzymanym w punkcie 4. Jeżeli zbiór U1 będzie równy zbiorowi wszystkich atrybutów, to należy rozłożyć schemat relacji według metody Bernsteina, w przeciwnym wypadku należy powtórzyć kroki od 1. do 5.

Postać normalna Boyce'a-Codda

Bardziej restrykcyjna niż 3PN jest postać normalna Boyce'a-Codda. Jeżeli schemat relacji spełnia warunki tej postaci, niemożliwe jest określenie wartości jednych danych w oparciu o inne dane, czyli w bazie nie ma informacji nadmiarowych (redundantnych). Ponieważ głównym celem projektanta jest stworzenie wydajnych i poprawnych logicznie modeli bazy danych, w praktyce rzadko doprowadza się normalizację projektu do postaci zgodnej z wymogami postaci normalnej Boyce'a-Codda. Ponadto niektórych relacji o bardzo prostych schematach nie można rozłożyć (przekształcić) bez straty zależności funkcyjnych na relacje znajdujące się w postaci Boyce'a-Codda.

Encja jest w postaci normalnej Boyce'a-Codda, jeżeli dla każdej zależności $X \rightarrow Y \in F+$ gdzie $X \subseteq R$, $A \in R$ albo zachodzi $A \in X$ (zależność trywialna), albo X jest nadkluczem.

Kurs Transact-SQL, część 8 (Bazy danych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

W ramach jednej instancji SQL Servera możliwe jest utworzenie i uruchomienie do 32 767 baz danych użytkownika. Bieżący rozdział opisuje fizyczną organizację plików bazodanowych oraz tworzenie i modyfikowanie baz danych.

Konwencja nazewnicza

Opisana konwencja nazewnicza powinna być stosowana nie tylko w odniesieniu do nazw baz danych, ale również do poszczególnych obiektów tworzonych w ramach baz danych użytkownika.

Nazwy poszczególnych obiektów powinny:

1. Być możliwie krótkie.
2. Oznaczać dany obiekt (użytkownik powinien potrafić na podstawie nazwy określić typ i funkcje obiektu).
3. Być tworzone w ramach jednoznacznej i zrozumiałej dla wszystkich użytkowników konwencji (konwencja powinna zawierać możliwie proste i jednoznaczne reguły, np. nazwy obiektów powinny być zapisywane małymi literami, bez polskich znaków diakrytycznych, znak spacji zostanie zastąpiony podkreśleniem, a same nazwy powinny być rzeczownikami w liczbie pojedynczej).
4. Zawierać przedrostek określający typ obiektu (jest to szczególnie istotne w przypadku widoków).
5. Nie pokrywać się z nazwami kont użytkowników lub ról.

Fizyczna struktura bazy danych

Każda baza danych działająca w środowisku SQL Servera (zarówno systemowa, jak i użytkownika) zapisana jest w przynajmniej dwóch plikach:

- W pliku .mdf, głównym pliku bazy danych, w którym zapisane są informacje o strukturze bazy, metainformacje, takie jak definicje obiektów bazodanowych czy konta użytkowników i ich uprawnienia oraz wszystkie wiersze poszczególnych tabel i indeksy.
- W pliku .ldf, w którym zapisany jest dziennik transakcyjny bazy danych. SQL Server wykorzystuje model zapisywania danych z wyprzedzeniem do dziennika

transakcyjnego i okresowego synchronizowania aktualnych danych zapisanych w dzienniku transakcyjnym z zawartością plików bazodanowych.

Baza danych może (i powinna) zostać zapisana w większej liczbie plików — dane mogą być podzielone pomiędzy dowolną ilość plików .ndf, a dziennik transakcyjny, dla zwiększenia niezawodności systemu, powinien zostać zapisany przynajmniej w dwóch plikach znajdujących się na osobnych dyskach twardych.

Dziennik transakcyjny

Wszystkie transakcje od momentu wydania pierwszej instrukcji wchodzącej w ich skład zapisywane są w dzienniku transakcyjnym. Poszczególnym transakcjom zostaje nadany kolejny numer odzwierciedlający kolejność rozpoczęcia kolejnych transakcji. Instrukcje języka Transact-SQL składające się na transakcje zostają zapisane w znajdującym się na dysku twardym pliku .ldf. Natomiast same strony danych, które zawierają wartości odczytane lub zmodyfikowane przez instrukcje nie są natychmiast zapisywane na dysku. W ten sposób zmniejsza się liczba operacji wejścia-wyjścia, a w przypadku nagłej awarii serwera dane mogą zostać odtworzone na podstawie wpisów z dziennika transakcyjnego (wykonane zostaną wszystkie zatwierdzone instrukcje języka Transact-SQL).

SQL Server okresowo tworzy punkt kontrolny (ang. Checkpoint). Polega to na zapisaniu w plikach bazy danych (plikach .mdf lub .ndf) wszystkich zmian wprowadzonych przez zatwierdzone transakcje i ewentualnym wycofaniu zmian wprowadzonych przez transakcje wycofane. W rezultacie dane zapisane w plikach bazodanowych zostają uaktualnione i zapisane w spójnej postaci. Równocześnie ustalana jest nowa wartość numeru LSN (kolejnego numeru dziennika, ang. Log Sequence Number). Numer ten wskazuje na ostatnią transakcję, która została odzwierciedlona w plikach bazodanowych i oddziela nieaktywną część dziennika transakcyjnego od części aktywnej.

Ponieważ zmiany wprowadzone przez znajdujące się w nieaktywnej części dziennika transakcje zostały trwale zapisane w plikach bazodanowych, ta część dziennika może zostać obcięta. W wersji 7. automatyczne obcinanie nieaktywnej części dziennika transakcyjnego ustawia się, włączając opcję trunc. log on chkpt, w wersji 2000 natomiast — zmieniając model odtwarzania bazy danych na Simple. W ten sposób zapobiegamy zwiększaniu się pliku .ldf, ale uniemożliwiamy odtworzenie bazy danych do określonego momentu w czasie.

> Kwestie związane z wykonywaniem i odtwarzaniem kopii zapasowych baz danych zostały opisane w rozdziałach 18. i 19.

Natomiast aktywna część dziennika jako zawierająca informacje niezapisane w plikach bazodanowych w żadnym wypadku nie może zostać usunięta.

Im częściej tworzony jest punkt kontrolny, tym statystycznie mniejsza liczba transakcji znajduje się w aktywnej części dziennika transakcyjnego. Od tej reguły mogą wystąpić wyjątki — na przykład rozpoczęta i niezakończona (czy to przez zatwierdzenie, czy przez wycofanie) transakcja nie będzie mogła zostać przeniesiona do części nieaktywnej i numer LSN będzie na nią wskazywał, nawet jeżeli setki rozpoczętych po niej transakcji zostało już zakończonych.

Podczas uruchamiania (proces ten może również zostać wywołany programowo) następuje proces odtwarzania bazy danych do spójnej postaci (ang. Recovery). W ramach procesu analizowana jest aktywna część dziennika transakcyjnego (transakcje o numerach większych niż LSN). Wyniki wszystkich zatwierdzonych transakcji zostają zapisane w plikach bazodanowych, a zmiany wprowadzone przez transakcje wycofane

lub przerwane — wyeliminowane. W ten sposób odtworzona jest logiczna spójność danych.

Struktura plików bazodanowych

SQL Server zapisuje dane (wiersze tabeli) w 8 KB ciągłych obszarach przestrzeni nazywanych stronami (ang. Page). Wynika z tego, że 1 MB zawiera 128 stron danych. W celu poprawy wydajności operacji wejścia-wyjścia strony łączone są w większe struktury nazywane zakresami (ang. Extent). Każdy zakres składa się z 8 stron, czyli zawiera 64 KB danych. Tabele i indeksy zapisywane są w zakresach, przy czym małe obiekty mogą dzielić ten sam zakres.

Poszczególne wiersze tabel oraz dane indeksów przechowywane są na stronach. Ponieważ każda strona przechowuje również metainformacje, maksymalna liczba danych zapisanych na pojedynczej stronie wynosi 8060 bajtów.

Tworząc bazę danych, należy w miarę dokładnie określić jej rozmiar początkowy. W ten sposób zapobiegniemy zwiększaniu lub zmniejszaniu rozmiaru plików bazy danych (plików .mdf, .ndf i .ldf). Chociaż rozmiar plików może być zmieniany dynamicznie, bez konieczności zatrzymywania bazy danych, to jest to operacja silnie obciążająca serwer i powinna być przeprowadzana stosunkowo rzadko.

Szacując wielkość plików bazodanowych, należy uwzględnić fakt, że pojedynczy wiersz nie może zostać zapisany na kilku stronach. Wynika z tego, że wielkość pojedynczego wiersza nie może przekroczyć 8060 bajtów.

Szacowanie wielkości plików bazodanowych:

1. Nowa baza danych użytkownika tworzona jest na podstawie systemowej bazy model i jest jej wierną kopią. Należy zapamiętać wielkość bazy model i dodać ją do końcowego wyniku obliczonego w punkcie 6.
2. Na podstawie projektu tabeli należy określić wielkość pojedynczego wiersza tabeli. W przypadku kolumn o stałej „szerokości” (np. kolumn przechowujących dane typu char) wystarczy dodać liczbę bajtów poszczególnych kolumn, w przypadku kolumn o zmiennej „szerokości”, należy przyjąć wartość średnią.
3. Dzielimy 8060 przez otrzymany w poprzednim punkcie wynik (liczbę bajtów jednego wiersza tabeli) i zaokrąglamy wynik w dół. W ten sposób poznamy liczbę wierszy zapisanych na pojedynczej stronie. Na przykład, jeżeli w punkcie 2. otrzymaliśmy wynik 3200 bajtów, to dzielimy 8060 przez 3200, co po zaokrągleniu daje 2 — na jednej stronie będą zapisane średnio 2 wiersze tabeli.
4. Szacunkową liczbę wierszy tabeli dzielimy przez otrzymany w punkcie 2. wynik. W naszym przypadku przewidujemy, że tabela będzie składać się z 5000 wierszy, dzielimy więc 5000 przez 2, otrzymując liczbę 2500 stron, na których będą zapisane dane z tabeli.
5. Obliczoną liczbę stron mnożymy przez 8 196 (8 KB) i w wyniku otrzymujemy szacunkową liczbę bajtów, na których będą zapisane dane przechowywane w tabeli. W naszym przykładzie będzie to 19,5 MB.
6. W analogiczny sposób obliczamy liczbę stron, na których będą zapisane wszystkie indeksy niegrupujące powiązane z tabelą. Przy obliczaniu tej wartości należy pamiętać o uwzględnieniu określonego dla indeksów współczynnika wypełnienia, np. jeżeli z obliczeń wynika, że pojedynczy wiersz indeksu będzie zapisany na 250 bajtach, to po podzieleniu 8060 przez 250 otrzymany wynik (32) mnożymy przez wartość współczynnika wypełnienia (np. 70%) i dopiero otrzymany w ten sposób wynik (22) jest liczbą wierszy indeksu zapisanych na pojedynczej stronie.

7. Powtarzamy całą operację dla kolejnych tabel bazy danych. Po dodaniu wyników dla poszczególnych tabel uzyskamy szacunkową wielkość pliku bazy danych.
8. Niestety, nie można w równie łatwy sposób oszacować wielkości pliku dziennika transakcyjnego. Jego wielkość zależy w dużym stopniu od:
    - Wybranej opcji odtwarzania bazy danych.
    - Liczby przeprowadzanych transakcji.
    - Rodzaju przeprowadzanych transakcji (baza wykorzystywana głównie do odczytu będzie wymagała mniejszego dziennika transakcyjnego od bazy, w której dane są często aktualizowane).

Początkową wielkość dziennika transakcyjnego dla baz OLTP (ang. Online Transaction Processing) należy określić jako 25 – 35 % wielkości plików bazodanowych, a następnie na podstawie obserwacji dostosować do konkretnej bazy danych.

Monitowanie parametrów i wydajności baz danych zostało dokładniej opisane w rozdziale 20.

Struktura wiersza danych

Pojedynczy wiersz tabeli składa się z danych oraz nagłówka:

1. 4-bajtowy nagłówek wiersza zawiera informacje o kolumnach wchodzących w skład wiersza, m.in. lokalizację końca poszczególnych kolumn.
2. Blok danych składa się z maksymalnie czterech segmentów:
    - Blok danych o określonej długości (ang. Fixed-length data). Dane przechowywane w kolumnach o stałej szerokości (np. dane typu char (10)) zapisywane są jako pierwsze za pomocą określonej liczbie bajtów. Wynika z tego, że nawet jeżeli w danym wierszu zapisany jest ciąg znaków „ala”, to i tak do jego zapisania wykorzystano 10 bajtów.
    - Blok wartości NULL (ang. Null block). Poszczególne bity danych bloku informują o tym, czy dana kolumna zawiera wartości NULL. Wynika z tego, że długość bloku (w bajtach) zależy od liczby kolumn tabeli i może być obliczona według wzoru: liczba kolumn tabeli podzielona przez osiem, a wynik zaokrąglony w górę.
    - Blok wartości o zmiennej długości (ang. Variable block). Dwa pierwsze bajty danych bloku przechowują informacje o liczbie kolumn tabeli o zmiennej długości. Kolejne dwa bajty są wskaźnikiem lokalizacji końca poszczególnych kolumn.
    - Blok danych o zmiennej długości (ang. Variable-length data). Dane przechowywane w kolumnach o zmiennej szerokości (np. dane typu varchar (20)) zapisywane są w końcowej części wiersza za pomocą minimalnej liczby bajtów. Wynika z tego, że do zapisania ciągu znaków „ala” zostaną wykorzystane 3 bajty.

W odrębny sposób przechowywane są dane typów text, ntext oraz image. Wszystkie te typy danych pozwalają na przechowywanie danych o długości przekraczającej wielkość strony danych i z tego powodu muszą zostać zapisane odrębnie od pozostałych danych wiersza. W wierszu zapisany zostaje jedynie 16-bajtowy wskaźnik do danych tych typów.

Możliwe jest jednak wymuszenie zapisywania danych tych typów razem z danymi zapisanymi w pozostałych kolumnach, o ile liczba bajtów niezbędnych do zapisania wartości tych typów nie przekracza 7000. Za pomocą procedury systemowej

sp_tableoption możemy ustawić limit wielkości, poniżej której dane typów text, ntext oraz image zostaną zapisane razem z pozostałymi danymi:

```
USE Northwind
EXEC sp_tableoption N'Employees', 'text in row', 2000
GO
The command(s) completed successfully.
```

Tworzenie baz danych

Aby utworzyć własną bazę danych, użytkownik musi należeć do roli serwera sysadmin lub dbcreator albo posiadać uprawnienie do wykonywania instrukcji CREATE DATABSE. Ponieważ metainformacje o wszystkich bazach danych (katalog schematu, ang. Schema Catalog) znajduje się w bazie master, aby utworzyć nową bazę danych, należy połączyć się z bazą master.

Dane na temat wszystkich baz danych znajdują się w tabeli systemowej sysdatabases. Po utworzeniu, zmodyfikowaniu lub usunięciu bazy danych należy wykonać kopię zapasową bazy master.

Przechowywane w katalogu schematu SQL Servera metainformacje możemy poznać, wykonując poniższe zapytanie:

```
USE master
SELECT name, dbid, crdate, filename
FROM dbo.sysdatabases
GO
name dbid crdate filename
------- ------ ------------------------- ----------------------------------------
master 1 2000-04-18 01:51:58.363 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\data\master.mdf
model 3 2000-04-18 02:03:11.240 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\data\model.mdf
msdb 4 2000-04-18 02:03:15.613 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\data\msdbdata.mdf
Northwind 6 2000-04-18 02:03:19.113 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\data\northwnd.mdf
pubs 5 2000-04-18 02:03:17.363 D:\Program Files\Microsoft SQLServer\MSSQL$MSSQL\data\pubs.mdf
tempdb 2 2002-11-01 15:17:18.613 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\data\tempdb.mdf
(6 row(s) affected)
```

Instrukcja CREATE DATABASE

Tworzenie nowej bazy danych polega na podaniu jej nazwy i określeniu lokalizacji plików bazodanowych. Opcjonalnie podczas wykonywania instrukcji CREATE DATABASE możemy ustawić wartość opcji bazodanowych związanych z jej fizyczną strukturą, np. maksymalną wielkość plików bazy danych czy liczbę bajtów, o jaką pliki będą automatycznie zwiększane.

Składnia:

```
CREATE DATABASE nazwa
[ON [PRIMARY]
[<plik> [,…n]]
[, <grupa_plików> [,…n]]
]
[LOG ON {<plik> [,…n]}]
[COLLATE porządek]
[FOR LOAD | FOR ATTACH]
<plik> ::=
([NAME = nazwa_logiczna,]
FILENAME = 'nazwa_fizyczna'
[, SIZE = wielkość]
[, MAXSIZE = {maksymalna_wielkość | UNLIMITED}]
[, FILEGROWTH = przyrost]) [,…n]
*lt;grupa_plików> ::=
FILEGROUP nazwa <plik> [,…n]
```

gdzie:

PRIMARY — określa domyślną grupę plików bazy danych. W grupie PRIMARY znajdują się m.in. wszystkie obiekty systemowe oraz pliki nieprzypisane do innych grup plików. Baza danych może posiadać tylko jedną domyślną grupę plików, w której znajduje się główny plik bazy danych. Domyślne rozszerzenie głównego pliku (.mdf) nie powinno być zmieniane, porządek — określa porządek sortowania i porównywania wartości,
FOR LOAD — opcja zachowana w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera. Jej włączenie powoduje utworzenie bazy danych w trybie ograniczonego dostępu dbo use only,
FOR ATTACH — nie tworzy, a dołącza do instancji istniejącą bazę danych. W tym przypadku konieczne jest określenie lokalizacji plików istniejącej bazy danych. Alternatywnym sposobem przyłączenia bazy danych jest wykonanie procedury systemowej sp_attach_db,
nazwa_logiczna — określa nazwę pliku bazy danych, którą będzie się posługiwać SQL Server,
nazwa_fizyczna — określa lokalizację i nazwę plików bazodanowych. Pliki muszą znajdować się na lokalnym dysku twardym komputera, na którym zainstalowano SQL Server i nie mogą znajdować się w folderze skompresowanym. Zapisanie ich w folderze zaszyfrowanym nie jest zalecane,
wielkość — początkowa wielkość pliku bazy danych (domyślnie 1 MB),
maksymalna_wielkość — określa maksymalną wielkość pliku bazy danych. Jeżeli parametr nie został podany, plik będzie przyrastał aż do wypełnienia dysku logicznego,
przyrost — wyrażona w jednostkach bezwzględnych (KB, MB, GB, TB) lub procentowo wielkość, o jaką będzie zwiększany rozmiar pliku bazodanowego.

Przykład:

```
USE master
GO
CREATE DATABASE Test
ON
PRIMARY (NAME = Test1,
FILENAME = 'd:\data\test1.mdf',
SIZE = 50MB,
MAXSIZE = 200,
FILEGROWTH = 20%),
(NAME = Test2,
FILENAME = 'd:\data\test2.ndf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20MB)
LOG ON
(NAME = Testlog1,
FILENAME = 'd:\data\test1.ldf',
SIZE = 20MB,
MAXSIZE = 100,
FILEGROWTH = 5MB),
(NAME = Testlog2,
FILENAME = 'e:\data\test2.ldf',
SIZE = 20MB,
MAXSIZE = 100,
FILEGROWTH = 20%)
GO
The CREATE DATABASE process is allocating 50.00 MB on disk 'Test1'.
The CREATE DATABASE process is allocating 100.00 MB on disk 'Test2'.
The CREATE DATABASE process is allocating 20.00 MB on disk 'Testlog1'.
The CREATE DATABASE process is allocating 20.00 MB on disk 'Testlog2'.
```

Przykładowa baza danych Wydawnictwo

W tym rozdziale rozpoczniemy tworzenie bazy danych Wydawnictwo, której projekt został opisany w rozdziale 7. Podczas wykonywania ćwiczeń z następnych rozdziałów części II książki będziemy uzupełniać tę bazę i przykładowe bazy Northwind i pubs o kolejne typy obiektów bazodanowych.

Aby utworzyć fizyczną strukturę bazy danych i zarejestrować ją w SQL Serverze:

1. Utwórz folder, w którym będą zapisane pliki bazy danych (np. d:\wydawnictwo).
2. Za pomocą programu Query Analyzer połącz się z bazą master jako użytkownik należący do roli sysadmin.

3. Wykonaj instrukcję języka Transact-SQL:

```
CREATE DATABASE wydawnictwo
ON PRIMARY
(NAME = wydawnictwo_dane,
FILENAME = 'd:\wydawnictwo\wydawnictwo_dane.mdf')
LOG ON
(NAME = wydawnictwo_dziennik,
FILENAME = 'd:\wydawnictwo\wydawnictwo_dziennik.ldf')
GO
The CREATE DATABASE process is allocating 0.75 MB on disk 'wydawnictwo_dane'.
The CREATE DATABASE process is allocating 1.00 MB on disk
'wydawnictwo_dziennik'.
```

Opcje baz danych

Po utworzeniu możemy zmienić lub odczytać ustawienia opcji bazy danych.

Skoro nowa baza danych jest kopią bazy model, aby ustawić opcje obowiązujące we wszystkich nowotworzonych bazach, należy właściwie skonfigurować bazę model.

Procedura sp_dboption

Wywołując procedurę systemową sp_dboption, możemy wyświetlić bieżące ustawienia lub ustawić opcje wybranej bazy danych.
Składnia:

```
sp_dboption [[@dbname =] 'baza_danych']
[, [@optname =] 'opcja']
[, [@optvalue =] 'wartość']
```

gdzie:
opcja — jest jedną z dopuszczalnych nazw opcji. Listę wszystkich opcji zawiera tabela 8.1.
wartość — określa, czy dana opcja ma zostać ustawiona (wartość true lub on), czy wyłączona (wartość false lub off).

Wywołanie procedury bez parametrów spowoduje wyświetlenie listy wszystkich opcji.
Wywołanie procedury z nazwą bazy danych jako jedynym parametrem spowoduje wyświetlenie listy ustawionych opcji.
Wywołanie procedury z nazwą bazy danych i nazwą opcji zwraca bieżące ustawienia danej opcji.

Tabela 8.1. Lista opcji procedury sp_dboption

Opcja	Opis
auto create statistics	Brakujące statystyki będą automatycznie tworzone przez proces optymalizatora wykonania instrukcji Transact-SQL

auto update statistics	Nieaktualne statystyki będą automatycznie aktualizowane
Autoclose	Po rozłączeniu się ostatniego użytkownika baza danych zostanie automatycznie zamknięta
autoshrink	Pliki bazy danych zawierające ponad 25% ciągłej, pustej przestrzeni będą automatycznie zmniejszone
ANSI null default	Nowo utworzone tabele będą zawierały kolumny dopuszczające wartość nieokreśloną w myśl standardu SQL-2
ANSI nulls	Jakiekolwiek porównania z wartością NULL zwracają NULL. Po wyłączeniu porównanie dwóch wartości NULL zwraca wartość TRUE
ANSI warnings	Wykonanie poprawnych, ale niezgodnych ze standardem SQL-2 instrukcji języka Transact-SQL, spowoduje wyświetlenie ostrzeżenia
concat null yields null	Połączenie (konkatenacja) dowolnego ciągu znaków z wartością NULL zwraca wartość NULL
cursor close on commit	Otwarte kursory zostaną automatycznie zamknięte podczas zatwierdzania lub wycofywania transakcji
dbo use only	Tylko członkowie roli dbo mają dostęp do bazy
default to local cursor	Domyślną lokalizacją tworzonych kursorów jest LOCAL
merge publish	Baza danych może brać udział w replikacji scalanej
Offline	Baza danych została wyłączona
Published	Baza danych może zostać opublikowana
quoted identifier	Do identyfikowania obiektów można używać cudzysłowu
read only	Baza danych umożliwia jedynie odczyt danych
recursive triggers	Wyzwalacz może wywoływać inny wyzwalacz
select into/bulkcopy	Dopuszczalne jest wykonywanie operacji masowego ładowania danych
single user	Baza danych pozwala na pojedyncze połączenie klienckie. Uwaga — w przypadku bazy msdb pierwsze połączenie zostanie nawiązane przez rejestrujący się w systemie proces SQL Server Agent
subscribed	Baza danych może pobierać dane w procesie replikacji
torn page detection	Niekompletne (np. uszkodzone podczas zapisywania) strony zostaną oznaczone jako błędne
trunc. Log on chkpt	Podczas punktu kontrolnego nieaktywna część dziennika transakcyjnego jest nieodwracalnie usuwana

Przykłady:

```
sp_dboption Northwind
GO
The following options are set:
----------------------------------
select into/bulkcopy
trunc. log on chkpt.
torn page detection
auto create statistics
```

```
auto update statistics
sp_dboption Northwind, 'concat null yields null'
GO
OptionName CurrentSetting
---------------------------------- --------------
concat null yields null off
sp_dboption Northwind, 'ANSI nulls', 'on'
GO
sp_dboption command succeeded.
```

Procedura sp_helpdb

Zwraca informacje o wybranej lub o wszystkich bazach danych.
Składnia:

```
sp_helpdb [[@dbname =] 'nazwa']
```

Lista zwracanych wartości zależy od tego, czy została podana nazwa bazy danych.
Przykład:

```
sp_helpdb
GO
Name db_size owner dbid created status compatibility_level
-------- ------------ ----- ------ ------------- ---------- --------------------
master 11.44 MB sa 1 Apr 18 2000 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=SIMPLE, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAutoCreateStatistics, IsAutoUpdateStatistics 80
model 1.25 MB sa 3 Apr 18 2000 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=FULL, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAutoCreateStatistics, IsAutoUpdateStatistics 80
msdb 14.00 MB sa 4 Apr 18 2000 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=SIMPLE, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAutoCreateStatistics, IsAutoUpdateStatistics 80
Northwind 6.38 MB sa 6 Apr 18 2000 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=SIMPLE, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAnsiNullsEnabled, IsAutoCreateStatistics, IsAutoUpdateStatistics, IsFullTextEnabled 80
pubs 3.38 MB sa 5 Apr 18 2000 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=FULL, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAutoCreateStatistics, IsAutoUpdateStatistics 80
```

```
tempdb 8.50 MB sa 2 Nov 2 2002 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=SIMPLE, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsAutoCreateStatistics,
IsAutoUpdateStatistics 80
Test 190.00 MB LOLEK\Administrator 7 Nov 2 2002 Status=ONLINE,
Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=FULL, Version=538,
Collation=Polish_CI_AS, SQLSortOrder=0, IsTornPageDetectionEnabled,
IsAutoCreateStatistics, IsAutoUpdateStatistics 80
```

Procedura sp_spaceused

Zwraca informacje o liczbie wierszy, wielkości plików bazy danych lub obiektów bazodanowych i liczbie pustych stron w poszczególnych plikach.
Składnia:

```
sp_spaceused [[@objname =] 'nazwa']
[,[@updateusage =] 'updateusage']
```

gdzie:
updateusage — wywołuje instrukcję DBCC UPDATEUSAGE aktualizującą zawartość tabeli systemowej sysindexes, która przechowuje informacje o wielkości poszczególnych obiektów
Przykład:

```
sp_spaceused @objname='dbo.Categories', @updateusage = 'TRUE'
GO
name rows reserved data index_size unused
---------- -------- --------- -------- ------------ --------
Categories 8 176 KB 112 KB 32 KB 32 KB
```

Obliczanie przybliżonej liczby wierszy tabel

Jedynym stuprocentowo dokładnym sposobem na poznanie liczby wierszy tabeli jest wykonanie instrukcji.

```
SELECT COUNT (*)
FROM nazwa_tabeli
```

Jednak jej wykonanie wiąże się z przeszukaniem całej tabeli, co w przypadku dużych obiektów jest operacją bardzo kosztowną. A skoro dane o wielkości obiektów zapisane są w tabeli systemowej sysindexes, możemy odczytać przybliżoną liczbę wierszy tabeli na podstawie jej wielkości:

```
USE Northwind
SELECT LEFT(object_name(id),30) as nazwa, rowcnt as liczba_wierszy
FROM sysindexes
WHERE indid IN (0,1)
AND OBJECTPROPERTY (id,'IsUserTable') = 1
GO
nazwa liczba_wierszy
------------------------------ --------------------
Orders 830
Products 77
Order Details 2019
CustomerCustomerDemo 0
CustomerDemographics 0
Region 4
Territories 53
EmployeeTerritories 49
dtproperties 7
Employees 9
Categories 8
Customers 91
Shippers 3
Suppliers 29
(14 row(s) affected)
```

We wcześniejszych wersjach SQL Servera informacja o liczbie wierszy tabeli przechowywana była w kolumnie rows tabeli sysindexes. W wersji 2000 kolumna ta została zachowana wyłącznie w celu zapewnienia zgodności z wcześniejszymi wersjami.

Narzędzie DBCC

Integralną częścią SQL Servera jest narzędzie DBCC (ang. DataBase Cosistency Checker), za pomocą którego można zarówno odczytywać jak i konfigurować fizyczną strukturę bazy danych. Poniżej zostaną opisane najczęściej wykorzystywane opcje programu DBCC.

Instrukcje DBCC SHRINKFILE oraz DBCC SHRINKDATABASE zostały opisane w dalszej części rozdziału.

Informacje o wielkości i stopniu wykorzystania dzienników transakcyjnych wszystkich baz danych poznamy wykonując instrukcję DBCC SQLPERF (LOGSPACE).

Przykład:

```
DBCC SQLPERF (LOGSPACE)
GO
Database Name Log Size (MB) Log Space Used (%) Status
```

```
-------- ---------------------- ---------------------- --------------------
master 0.7421875 35.723682 0
tempdb 0.4921875 81.050423 0
model 0.4921875 81.944443 0
msdb 2.2421875 31.903311 0
pubs 0.7421875 50.657894 0
Northwind 0.9921875 43.257874 0
wydawnictwo 0.9921875 45.275589 0
(7 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Sprawdzić spójność fizycznych plików wybranej bazy danych możemy za pomocą instrukcji DBCC CHECKALLOC.
Składnia:

```
DBCC CHECKALLOC
('baza_danych'
[, NOINDEX
|
{REPAIR_ALLOW_DATA_LOSS
| REPAIR_FAST
| REPAIR_REBUILD
}]
) [WITH {[ALL_ERRORMSGS | NO_INFOMSGS]
[, [TABLOCK]]
[, [ESTIMATEONLY]]
}
]
```

gdzie:
NOINDEX spowoduje niesprawdzanie spójności indeksów niegrupujących,
REPAIR_ALLOW_DATA_LOSS spowoduje automatyczne naprawienie wszystkich znalezionych błędów, nawet jeżeli może to doprowadzić do utraty częściowo uszkodzonych danych,
REPAIR_FAST spowoduje wykonanie tylko podstawowych czynności usuwających proste błędy,
REPAIR_REBUILD spowoduje wykonanie również kosztownych operacji, jak np. przebudowy indeksów.

Przed wykonaniem jakiejkolwiek naprawy baza musi zostać przełączona na tryb pojedynczego użytkownika.

Przykład:

```
EXEC sp_dboption 'Northwind', 'single user', 'on'
GO
DBCC CHECKALLOC (Northwind, REPAIR_REBUILD)
```

```
GO
sp_dboption command succeeded.
DBCC results for 'Northwind'.
*****************************************************************
Table sysobjects Object ID 1.
Index ID 1 FirstIAM (1:10) Root (1:11) Dpages 3.
Index ID 1. 5 pages used in 0 dedicated extents.
Index ID 2 FirstIAM (1:25) Root (1:15) Dpages 1.
Index ID 2. 2 pages used in 0 dedicated extents.
Index ID 3 FirstIAM (1:33) Root (1:31) Dpages 1.
Index ID 3. 2 pages used in 0 dedicated extents.
.
.
.
Total number of extents = 48, used pages = 341, reserved pages = 379 in this database.
(number of mixed extents = 32, mixed pages = 251) in this database.
CHECKALLOC found 0 allocation errors and 0 consistency errors in database 'Northwind'.
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Wykonanie instrukcji DBCC CHECKDB spowoduje sprawdzenie spójności struktury wszystkich obiektów wybranej bazy danych.
Składnia:

```
DBCC CHECKDB
('baza_danych'
[, NOINDEX
| {REPAIR_ALLOW_DATA_LOSS
| REPAIR_FAST
| REPAIR_REBUILD
}]
) [WITH {[ALL_ERRORMSGS]
[, [NO_INFOMSGS]]
[, [TABLOCK]]
[, [ESTIMATEONLY]]
[, [PHYSICAL_ONLY]]
}
]
```

Przykład:

```
DBCC CHECKDB (Northwind) WITH PHYSICAL_ONLY
GO
DBCC results for 'Northwind'.
```

```
CHECKDB found 0 allocation errors and 0 consistency errors in database 'Northwind'.
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Modyfikacja baz danych

Fizyczna struktura bazy danych, tak jak wartości związanych z nią opcji, może zostać w dowolnym momencie zmodyfikowana. Domyślnie jedynie użytkownicy należący do ról serwera sysadmin lub dbcreator oraz właściciel konkretnej bazy danych mogą zmodyfikować jej fizyczną strukturę.

Instrukcja ALTER DATABASE

Fizyczną strukturę bazy danych możemy zmienić (np. poprzez dodanie nowych plików lub zmianę rozmiaru plików istniejących), wykonując instrukcję ALTER DATABASE.
Składnia:

```
ALTER DATABASE nazwa
{ADD FILE <plik> [,…n] [TO FILEGROUP grupa_plików]
| ADD LOG FILE <plik> [,…n]
| REMOVE FILE nazwa_pliku [WITH DELETE]
| ADD FILEGROUP nazwa_grupy_plików
| REMOVE FILEGROUP nazwa_grupy_plików
| MODIFY FILE <plik>
| MODIFY NAME = nowa_nazwa
| MODIFY FILEGROUP nazwa_grupy_plików {właściwość | NAME = nowa_nazwa_grupy_plików}
| SET <opcja> [,…n] [WITH <przerwanie>]
| COLLATE <porządek>
}
<plik> ::=
(NAME = nazwa_pliku
[, NEWNAME = nowa_nazwa_pliku]
[, FILENAME = 'nazwa_fizyczna']
[, SIZE = wielkość]
[, MAXSIZE = {maksymalna_wielkość | UNLIMITED}]
[, FILEGROWTH = przyrost])
<opcja> ::=
<opcje_ograniczenia_dostępu>
| <opcje_kursora>
| <opcje_automatyzacji>
| <opcje_standardu_sql>
| <opcje_odtwarzania>
<opcje_ograniczenia_dostępu> ::=
{SINGLE_USER | RESTRICTED_USER | MULTI_USER}
| {OFFLINE | ONLINE}
```

| {READ_ONLY | READ_WRITE}

<przerwanie> ::=

ROLLBACK AFTER liczba [SECONDS]

| ROLLBACK IMMEDIATE

| NO WAIT

<opcje_kursora> ::=

CURSOR_CLOSE_ON_COMMIT {ON | OFF}

| {CURSOR_DEFAULT {LOCAL | GLOBAL}

<opcje_automatyzacji> ::=

{AUTO_CLOSE ON | OFF}

| {AUTO_CREATE_STATISTICS ON | OFF}

| {AUTO_SHRINK ON | OFF}

| {AUTO_UPDATE_STATISTICS ON | OFF}

<opcje_standardu_sql> ::=

ANSI_NULL_DEFAULT {ON | OFF}

| ANSI_NULLS {ON | OFF}

| ANSI_PADDING {ON | OFF}

| ANSI_WARNINGS {ON | OFF}

| ARITHABORT {ON | OFF}

| CONCAT_NULL_YIELDS_NULL {ON | OFF}

| NUMERIC_ROUNDABORT {ON | OFF}

| QUOTED_IDENTIFIER {ON | OFF}

| RECURSIVE_TRIGGERS {ON | OFF}

<opcje_odtwarzania> ::=

RECOVERY {FULL | BULK_LOGGED | SIMPLE}

| TORN_PAGE_DETECTION {ON | OFF}

Zwiększanie rozmiaru plików bazy danych

Wraz z dodawaniem nowych danych lub z modyfikacją danych istniejących niezbędne może okazać się zwiększenie rozmiaru plików danych oraz dziennika transakcyjnego. Ponieważ brak miejsca spowodowałby zatrzymanie bazy danych, należy monitorować procent wykorzystania plików bazodanowych.

Zwiększyć rozmiar plików bazy danych możemy poprzez:

1. Skonfigurowanie automatycznego zwiększania rozmiaru plików. Zaletą tego rozwiązania jest automatyzacja zadań administracyjnych, wadą fakt, że rozmiar plików bazy danych będzie zwiększany w miarę potrzeb i operacja ta może wystąpić w godzinach największego obciążenia serwera. Aby umożliwić ograniczony jedynie rozmiarem dysku twardego przyrost pliku Testlog2, napiszemy:

2.
3. USE master
4. ALTER DATABASE Test

```
MODIFY FILE
(NAME = Testlog2,
MAXSIZE = UNLIMITED,
FILEGROWTH = 5MB)
GO
The command(s) completed successfully.
```

Programowe zwiększenie wielkości pliku bazy danych. Zaletą tego rozwiązania jest zachowanie kontroli nad czasem wykonania operacji, wadą — konieczność nieustannego monitorowania stopnia wykorzystania plików bazodanowych. Aby jednorazowo zwiększyć rozmiar pliku Test1, napiszmy:

```
USE master
ALTER DATABASE Test
MODIFY FILE
(NAME = Test1,
SIZE = 60 MB,
FILEGROWTH = 0)
GO
The command(s) completed successfully.
```

Dodanie kolejnego pliku .ndf. Zaletą tego rozwiązania jest fizyczne podzielenie danych pomiędzy kilka, z reguły znajdujących się na różnych dyskach twardych, plików. Ponieważ SQL Server każdą operację odczytu lub zapisu danych z odrębnego pliku wykonuje w ramach osobnego procesu, prawidłowe podzielenie bazy pomiędzy pliki i grupy plików zwiększa wydajność operacji wejścia-wyjścia.

Tworzenie i zarządzanie grupami plików zostało opisane w znajdującym się w dalszej części rozdziału punkcie „Grupy plików”.

Wadą tego rozwiązania są: skomplikowanie fizycznej struktury bazy danych i związanych z nią zadań administracyjnych oraz konieczność nieustannego monitorowania stopnia wykorzystania plików bazodanowych. Aby dodać nowy plik do bazy Test, napiszemy:

```
USE master
ALTER DATABASE Test
ADD FILE
(NAME = Test3,
FILENAME = 'f:\data\test3.ndf',
SIZE = 25MB,
MAXSIZE = 100MB,
FILEGROWTH = 5MB)
GO
Extending database by 25.00 MB on disk 'Test3'.
```

Wypełnienie pliku dziennika transakcyjnego spowoduje niemożność jakiejkolwiek modyfikacji zapisanych w bazie danych. Techniki monitorowania parametrów pracy bazy danych zostały opisane w rozdziale 20.

Zmniejszanie rozmiaru plików bazy danych

Jeżeli bazie danych przydzielono zbyt dużo przestrzeni na dysku, możemy zmniejszyć rozmiar plików bazy danych i zwrócić cześć zasobów systemowi operacyjnemu. Niemożliwe jest zmniejszenie bazy danych poniżej jej rozmiaru początkowego i poniżej rozmiaru zapisanych w niej w określonej chwili danych.

Instrukcja DBCC SHRINKFILE

Wykonanie instrukcji spowoduje obcięcie pliku bazodanowego. O ile rozmiar plików danych zmniejszany jest natychmiast, o tyle rozmiar plików dziennika transakcyjnego zostanie zmniejszony podczas usuwania nieaktywnej części dziennika transakcyjnego.
Składnia:

```
DBCC SHRINKFILE
({plik | id_pliku}
{[, wielkość]
| [, {EMPTYFILE | NOTRUNCATE | TRUNCATEONLY}]
}
)
```

gdzie:
EMPTYFILE — powoduje przeniesienie wszystkich danych do innych plików danych i umożliwia usunięcie pliku za pomocą instrukcji ALTER DATABASE,
NOTRUNCATE — powoduje wyłącznie uporządkowanie stron danych zapisanych w pliku, bez zmniejszenia jego rozmiaru,
TRUNCATEONLY — powoduje obcięcie wolnej przestrzeni znajdującej się na końcu pliku, bez wcześniejszego uporządkowania stron. W rezultacie, jeżeli w pliku znajdowało się nawet ponad 25% wolnej przestrzeni, ale puste strony nie były zapisane na końcu pliku, zmniejszenie rozmiaru pliku nie uda się. Obcięcie rozmiaru bez uporządkowania stron jest operacją o wiele mniej obciążającą serwer od uporządkowania i obcięcia pliku.

Instrukcja DBCC SHRINKDATABASE

Wykonanie instrukcji spowoduje zmniejszenie wielkości bazy danych.
Składnia:

```
DBCC SHRINKDATABASE
(nazwa [, wielkość_procentowo]
[, {NOTRUNCATE | TRUNCATEONLY}]
)
```

Parametrem instrukcji DBCC SHRINKDATABASE jest procent wolnego miejsca, jaki ma zostać w plikach bazy danych po ich obcięciu.

Zmniejszyć rozmiar plików bazy danych również możemy na trzy sposoby:

1. Poprzez ustawienie opcji automatycznego obcinania plików bazy danych. Zaletą tego rozwiązania jest automatyzacja zadań administracyjnych, wadą fakt, że rozmiar plików bazy danych będzie zmniejszany w miarę potrzeb i operacja ta może wystąpić w godzinach największego obciążenia serwera. Aby włączyć automatyczne zmniejszanie rozmiaru bazy danych Northwind, napiszemy:

```
2.
3. USE master
4. ALTER DATABASE Northwind
5. SET AUTO_SHRINK ON
6. GO
7. The command(s) completed successfully.
```

8. Poprzez programowe zmniejszenie wielkości bazy danych lub konkretnego pliku bazy danych. Zaletą tego rozwiązania jest zachowanie kontroli nad czasem wykonania operacji, wadą — konieczność monitorowania stopnia wykorzystania plików bazodanowych. Aby jednorazowo zmniejszyć rozmiar pliku Test1 (w przypadku dużych baz danych, aby rozłożyć silnie obciążającą serwer operację obcinania rozmiaru plików w czasie, możemy kolejno zmniejszać rozmiary poszczególnych plików), napiszemy:

```
9.
10. USE Test
11. DBCC SHRINKFILE (Test3, 10)
12. GO
13. DbId FileId CurrentSize MinimumSize UsedPages EstimatedPages
14. ------ ------ ----------- ----------- ----------- --------------
15. 7 5 1280 1280 0 0
16. (1 row(s) affected)
17. DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

gdzie:

Test3 jest nazwą obcinanego pliku bazy danych Test,

10 jest wyrażona w MB wielkością docelowa obcinanego pliku. Opuszczenie tego parametru spowoduje obcięcie pliku do najmniejszej dopuszczalnej wielkości.

Aby jednorazowo zmniejszyć rozmiar bazy Test tak, aby zostało jedynie 10% wolnego miejsca, napiszmy:

```
DBCC SHRINKDATABASE (Test, 10)
GO
DbId FileId CurrentSize MinimumSize UsedPages EstimatedPages
------ ------ ----------- ----------- ----------- --------------
7 1 6400 6400 96 96
```

```
(1 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

18. Poprzez usunięcie jednego z plików .ndf. Zaletą tego rozwiązania jest uproszczenie zadań administracyjnych, wadą — zmiana fizycznej struktury produkcyjnej bazy danych. Ponieważ niemożliwe jest usunięcie pliku zawierającego dane, przed wykonaniem instrukcji ALTER DATABASE należy usunąć wszystkie dane z pliku:

```
19.
20. USE Test
21. DBCC SHRINKFILE (Test3, EMPTYFILE)
22. GO
23. USE master
24. ALTER DATABASE Test
25. REMOVE FILE Test3
26. GO
27. DbId FileId CurrentSize MinimumSize UsedPages EstimatedPages
28. ------ ------ ----------- ----------- ----------- --------------
29. 7 5 1280 1280 0 0
30. (1 row(s) affected)
31. DBCC execution completed. If DBCC printed error messages, contact your system administrator.
32. The file 'Test3' has been removed.
33.
```

Usuwanie baz danych

Nieużywana i niepotrzebna baz danych powinna zostać usunięta. W ten sposób zwrócone zostaną do systemu wszystkie zarezerwowane na jej potrzeby zasoby.
Baza danych nie może zostać usunięta, jeżeli:

- Jest bazą systemową. Chociaż możliwe jest usunięcie bazy msdb, to jej usunięcie spowoduje automatyczne zablokowanie:
  - usługi SQL Server Agent (procesu zarządzającego m.in. alarmami, zaplanowanymi zadaniami administracyjnymi i powiadamianiem operatorów),
  - replikacji,
  - usługi DTS (ang. Data Transformation Service),
  - kreatora publikacji w sieci Web (ang. SQL Server Web Wizard).
- Istnieje otwarte połączenie jakiegokolwiek użytkownika z bazą.
- Dane z bazy są podstawą dowolnej publikacji (dane podlegają replikacji).

Instrukcja DROP DATABASE

Wykonanie instrukcji spowoduje usunięcie wybranych baz danych.
Składnia:

```
DROP DATABASE nazwa [,…n]
```

Po usunięciu bazy danych użytkownicy, dla których była to baza domyślna nie będą mieli określonej domyślnej bazy danych.

Przykład:

```
USE master
DROP DATABASE Test
GO
Deleting database file 'd:\data\test2.ndf'.
Deleting database file 'd:\data\test1.ldf'.
Deleting database file 'd:\data\test1.mdf'.
Deleting database file 'e:\data\test2.ldf'.
```

Grupy plików

Grupa plików jest logiczną strukturą, na którą składa się pewna liczba plików danych. Pliki dziennika transakcyjnego nie mogą należeć do jakiejkolwiek grupy plików. Łącząc pliki w grupy, umożliwiamy:

1. Wykonanie kopii zapasowej określonej grupy plików.
2. Fizyczne zapisanie indeksu w innym pliku niż plik przechowujący powiązaną z indeksem tabelę.
3. Równoległe przeprowadzanie operacji odczytu i zapisu danych przechowywanych w tabelach należących do różnych grup plików.
4. Przełączenie całej grupy plików w tryb „tylko do odczytu”.

Grupy plików nie powinny być traktowane jak ekwiwalent macierzy dyskowych i nie powinny być wykorzystywane wyłącznie do poprawy wydajności bazy danych.

Przykłady:

Aby dodać nową grupę plików do bazy pubs i umieścić w niej plik .ndf, należy wykonać poniższy skrypt:

```
USE master
ALTER DATABASE pubs
ADD FILEGROUP pubs_G1
GO
ALTER DATABASE pubs
ADD FILE
(NAME = pubs2,
FILENAME = 'D:\Program Files\Microsoft SQL Server\MSSQL$MSSQL\Data\pubs2.ndf',
SIZE = 5MB,
MAXSIZE = 100MB,
FILEGROWTH = 1MB)
TO FILEGROUP pubs_G1
```

```
GO
Extending database by 5.00 MB on disk 'pubs2'.
```

Aby przełączyć grupę pubs_G1 w tryb „tylko do odczytu”:

```
USE master
ALTER DATABASE pubs
MODIFY FILEGROUP pubs_G1 READONLY
GO
The filegroup property 'READONLY' has been set.
```

Procedura sp_helpfile

Procedura zwraca informacje o nazwie i atrybutach pliku bazy danych.
Składnia:

```
sp_helpfile [[@filename =] 'nazwa']
```

Przykład:

```
USE Northwind
EXEC sp_helpfile @filename = 'Northwind_log'
GO
name filename
filegroup size maxsize growth usage
---------- --------------------------------------------------------------------
------------------ -------------------- ------------------ -----------------
-------------
Northwind_log D:\Program Files\Microsoft SQL Server\MSSQL$MSSQL\data\northwnd.ldf
NULL 1024 KB Unlimited 10% log only
```

Procedura sp_helpfilegroup

Procedura zwraca informacje o nazwie i atrybutach grupy pliku bazy danych.
Składnia:

```
sp_helpfilegroup [[@filegroupname =] 'nazwa']
```

Przykład:

```
USE Pubs
EXEC sp_helpfilegroup 'pubs_G1'
GO
groupname groupid filecount
------------------------ ------- -----------
pubs_G1 2 1
```

```
file_in_group fileid filename
size maxsize growth
--------------- ------- --------------------------------------------------------------
------------------- ----------------- ----------------------
pubs2 3 D:\Program Files\Microsoft SQL
Server\MSSQL$MSSQL\Data\pubs2.ndf
5120 KB 102400 KB 1024 KB
```

Kurs Transact-SQL, część 9 (Tabele)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Podstawowymi obiektami każdej bazy danych są tabele. SQL Server 2000 pozwala na utworzenie ok. 2 bilionów tabel w pojedynczej bazie danych, przy czym każda tabela może składać się z nie więcej niż 1024 kolumn, a wielkość pojedynczego wiersza nie może przekroczyć 8092 bajtów (to ograniczenie nie dotyczy danych typu text, ntext lub image, dla których w tabeli zapisuje się jedynie wskaźniki do danych).

Każda z kolumn przechowuje dane określonego typu. SQL Server, oprócz udostępnienia kilkunastu predefiniowanych typów danych, pozwala na tworzenie typów danych przez użytkowników.

Typy danych użytkownika

Podstawą (szablonem) wszystkich typów danych użytkownika jest predefiniowany typ danych. Zakresem stosowania typu danych użytkownika jest określona baza danych.

Procedura systemowa sp_addtype

Możemy utworzyć własny typ danych, wywołując procedurę systemową sp_addtype.

Składnia:

```
sp_addtype [@typename =] typ,
[@phystype =] typ_predefiniowany
[, [@nulltype =] 'NULL']
[, [@owner =] 'właściciel']
```

Przykład:

```
USE wydawnictwo
EXEC sp_addtype nazwisko, 'nvarchar(25)'
EXEC sp_addtype kod, 'nchar(6)', NULL
GO
(1 row(s) affected)
Type added.
```

```
(1 row(s) affected)
Type added.
```

Procedura systemowa sp_droptype

Wywołanie procedury spowoduje usunięcie wybranych typów danych użytkownika.
Składnia:

```
sp_droptype [@typename =] 'typ'
```

Przykład:

```
USE wydawnictwo
EXEC sp_droptype nazwisko
EXEC sp_droptype kod
GO
(1 row(s) affected)
(0 row(s) affected)
Type has been dropped.
(1 row(s) affected)
(0 row(s) affected)
Type has been dropped.
```

Instrukcja CREATE TABLE

Wykonanie instrukcji spowoduje dodanie nowej tabeli do bazy danych. Instrukcja może zostać wykonana przez członków roli db_owner oraz przez użytkowników, którym nadano uprawnienie do jej wykonywania.

> Jeżeli względy innego typu (np. polityka firmy) nie stoją na przeszkodzie, należy ograniczyć prawo do tworzenia nowych obiektów, w tym tabel, do konta właściciela bazy danych. Kwestia sprawdzania przez SQL Server uprawnień do powiązanych ze sobą obiektów (np. tabel i widoków) została omówiona w rozdziale 16.

Składnia:

```
CREATE TABLE
[nazwa_bazy_danych. | właściciel.] nazwa_tabeli
({<definicja_kolumny>
| nazwa_kolumny AS wynik_wyrażenia
| <zawężenie_tabeli>} [,…n]
)
[ON {grupa_plików | DEFAULT}]
[TEXTIMAGE_ON {grupa_plików | DEFAULT}]
<definicja_kolumny> ::= {nazwa typ}
[[DEFAULT wyrażenie]
```

```
| [IDENTITY [(wartość_początkowa, przyrost) [NOT FOR REPLICATION]]]
]
[ROWGUIDCOL]
[COLLATE <porządek>]
[<zawężenie_kolumny>] [,…n]
<zawężenie_kolumny> ::= [CONSTRAINT nazwa]
{[NULL | NOT NULL]
| [{PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED]
[WITH FILLFACTOR = współczynnik_wypełnienia]
[ON {grupa_plików | DEFAULT}]]
]
| [[FOREIGN KEY]
REFERENCES powiązana_tabela [(powiązana_kolumna)]
[ON DELETE {CASCADE | NO ACTION}]
[ON UPDATE {CASCADE | NO ACTION}]
[NOT FOR REPLICATION]
]
| CHECK [NOT FOR REPLICATION]
(warunek_logiczny)
}
<zawężenie_tabeli> ::= [CONSTRAINT nazwa]
{[{PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED]
{(kolumna [ASC | DESC] [,…n])}
[WITH FILLFACTOR = współczynnik_wypełnienia]
[ON {grupa_plików | DEFAULT}]
]
| FOREIGN KEY
[(kolumna [,…n])]
REFERENCES ref_table [(ref_column [,…n])]
[ON DELETE {CASCADE | NO ACTION}]
[ON UPDATE {CASCADE | NO ACTION}]
[NOT FOR REPLICATION]
| CHECK [NOT FOR REPLICATION]
(warunek_logiczny)
}
```

gdzie:
IDENTITY oznacza, że wartości poszczególnych pól kolumny będą niepowtarzalnymi, generowanymi przez SQL Server identyfikatorami, najczęściej wykorzystywanymi jako klucz główny tabeli. Dla jednej tabeli można zdefiniować tylko jedną kolumnę typu IDENTITY. Wartości tak zdefiniowanej kolumny nie mogą być aktualizowane, nie mogą być nieokreślone, a sama kolumna musi zostać zdefiniowana jako przechowująca

wartości jednego z poniższych typów: integer, numeric lub decimal, przy czym skala dwóch ostatnich typów musi być równa 0.

Podczas tworzenia tabel i indeksów możemy określić, w której grupie plików obiekty zostaną zapisane, ale nie możemy określić pliku przechowującego dane. Jedynym rozwiązaniem prowadzącym do określenia pliku docelowego jest utworzenie grup plików zawierających pojedyncze pliki.

Przykłady:

Tworzymy tabelę seria_wydawnicza przykładowej bazy danych wydawnictwo:

```
USE wydawnictwo
CREATE TABLE seria_wydawnicza
(id_serii int IDENTITY (100,1)
CONSTRAINT pk_sw PRIMARY KEY,
seria varchar (50) NOT NULL,
uwagi varchar (250))
GO
The command(s) completed successfully.
```

Dodajemy drugą tabelę również będącą słownikiem tabeli ksiazka — tabelę nosnik:

```
CREATE TABLE wydawnictwo..nosnik
(id_nosnika int IDENTITY
CONSTRAINT pk_n PRIMARY KEY,
nosnik varchar (50) NOT NULL DEFAULT 'książka w miękkiej oprawie',
uwagi varchar (250))
GO
The command(s) completed successfully.
```

Tworzymy tabelę nadrzędną ksiazka:

```
CREATE TABLE wydawnictwo.dbo.ksiazka
(id_ksiazki int IDENTITY (3200,1)
CONSTRAINT pk_k PRIMARY KEY NONCLUSTERED,
tytul varchar (50) NOT NULL,
isbn char (12) NOT NULL UNIQUE,
liczba_stron int NOT NULL
CONSTRAINT ch_k_ls CHECK (liczba_stron BETWEEN 50 AND 3000),
rok_wydania smalldatetime NOT NULL DEFAULT LEFT(GETDATE(),7),
isbn_oryginalu char(12) UNIQUE,
id_serii int REFERENCES seria_wydawnicza(id_serii)
ON UPDATE CASCADE ON DELETE NO ACTION,
id_nosnika int REFERENCES nosnik(id_nosnika)
ON UPDATE CASCADE ON DELETE NO ACTION,
```

```
uwagi varchar (250),
notka_redakcyjna text)
GO
The command(s) completed successfully.
```

SQL Server automatycznie tworzy indeks grupujący dla kolumny określonej jako klucz główny tabeli. Ponieważ można utworzyć najwyżej jeden indeks grupujący dla tabeli, a w przypadku tabeli ksiazka dane powinny zostać uporządkowane według serii lub tytułów, a nie wewnętrznych identyfikatorów, jawnie podano typ indeksu zawężenia PRIMARY KEY).

Przechowywanie wartości obliczonych na podstawie innych wartości

Raporty i zestawienia, oprócz danych odczytanych z kolumn tabeli z reguły zawierają wartości obliczone na ich podstawie, na przykład obliczoną wartość podatku czy średnią wartość sprzedaży danego towaru. Wartości te mogą być obliczane podczas wykonywania instrukcji SELECT lub pobierane z dodatkowych kolumn tabeli. Na przykład możemy dodać do definicji tabeli miasto kolumnę zawierającą informację o przyporządkowaniu na podstawie liczby mieszkańców miasta do jednej z pięciu kategorii:

```
USE wydawnictwo
CREATE TABLE miasto
(id_miasta int IDENTITY
CONSTRAINT [pk_m] PRIMARY KEY NONCLUSTERED,
miasto varchar (30),
mieszkancow int,
uwagi varchar (250),
kategoria AS CASE
WHEN mieszkancow < 1000 THEN 'A'
WHEN mieszkancow between 1000 and 5000 THEN 'B'
WHEN mieszkancow between 5000 and 20000 THEN 'C'
WHEN mieszkancow between 20000 and 50000 THEN 'D'
WHEN mieszkancow > 50000 THEN 'E'
END
)
GO
The command(s) completed successfully.
```

SQL Server nie przechowuje informacji o przypisanej każdemu miastu kategorii, ale wyznacza ją dynamicznie. Wersja 2000 pozwala jednak na utworzenie indeksu powiązanego z kolumną kategoria tabeli miasto i zapisanie go na dysku. Dzięki temu operacje wybierania i sortowania danych z tej kolumny będą mniej kosztowne.

Dla kolejnej tabeli przykładowej bazy danych, tabeli autor, również zdefiniowano kolumnę zawierającą wartości obliczone na podstawie innych wartości:

```
USE wydawnictwo
CREATE TABLE autor
(id_autora int IDENTITY
CONSTRAINT pk_a PRIMARY KEY NONCLUSTERED,
imie varchar (20) NOT NULL,
nazwisko varchar (25) NOT NULL,
kod char (6),
ulica_nr_domu varchar (50),
telefon varchar (15),
e_mail varchar (40)
CONSTRAINT ch_a_m CHECK (LEN(e_mail)-LEN(REPLACE(e_mail,'@',''))>0),
bank varchar (40),
konto varchar (30),
id_miasta int REFERENCES miasto(id_miasta)
ON UPDATE CASCADE ON DELETE NO ACTION,
uwagi varchar(250),
skrot AS UPPER(LEFT(imie,2)) + UPPER(LEFT(nazwisko,2)))
GO
The command(s) completed successfully.
```

Możliwe jest również przechowywanie innego wyrażenia dla poszczególnych wierszy tabeli, np. dla wierszy o identyfikatorach mniejszych od 100 wyznaczana będzie wartość średnia określonego parametru, a dla pozostałych — suma kilku wartości. Wymaga to jedynie wstawienia do kolumny przechowującej definicje wyrażenia różnych wartości dla różnych wierszy (np. za pomocą instrukcji UPDATE z klauzulą WHERE).

Czasami (np. jeżeli planujemy replikację scalaną) niezbędne jest jednoznaczne identyfikowanie wiersza tabeli nie tylko w skali tej tabeli, ale w skali całego systemu bazodanowego. W takim przypadku do definicji kolumny należy dodać kolumnę przechowującą dane typu uniqueidentifier, a następnie dla każdego wiersza wygenerować wartość identyfikatora. Przykładem takiego rozwiązania jest tabela wydawnictwo (na początku utworzymy niezbędne tabele słownikowe):

```
USE wydawnictwo
CREATE TABLE kraj
(id_kraju int IDENTITY
CONSTRAINT pk_kr PRIMARY KEY,
kraj varchar (20) NOT NULL UNIQUE,
uwagi varchar(250),
)
CREATE TABLE osoba
```

```
(id_osoby int IDENTITY
CONSTRAINT pk_o PRIMARY KEY,
imie varchar (20) NOT NULL,
nazwisko varchar (25) NOT NULL,
telefon varchar (15),
e_mail varchar (40)
CONSTRAINT ch_o_m CHECK (LEN(e_mail)-LEN(REPLACE(e_mail,'@',''))>0),
prezes bit DEFAULT 0,
uwagi varchar(250),
)
GO
CREATE TABLE wydawnictwo
(id_wydawnictwa uniqueidentifier DEFAULT NEWID()
CONSTRAINT pk_w PRIMARY KEY NONCLUSTERED,
nazwa varchar (40) NOT NULL,
kod char (6),
ulica_nr_domu varchar (50),
telefon varchar (15),
e_mail varchar (40)
CONSTRAINT ch_w_m CHECK (LEN(e_mail)-LEN(REPLACE(e_mail,'@',''))>0),
bank varchar (40),
konto varchar (30),
id_miasta int REFERENCES miasto(id_miasta)
ON UPDATE CASCADE ON DELETE NO ACTION,
id_kraju int REFERENCES kraj(id_kraju)
ON UPDATE CASCADE ON DELETE NO ACTION,
id_osoby int REFERENCES osoba(id_osoby)
ON UPDATE NO ACTION ON DELETE NO ACTION,
id_prezesa int REFERENCES osoba(id_osoby)
ON UPDATE NO ACTION ON DELETE NO ACTION,
uwagi varchar(250),
)
GO
The command(s) completed successfully.
```

Funkcja NEWID zwraca przypadkową wartość 36-bajtowego globalnego identyfikatora obiektu.

Opcje tabel

Dokładne informacje o poszczególnych tabelach poznamy, wywołując procedurę systemową sp_help z parametrem będącym nazwą tabeli:

```
USE wydawnictwo
EXEC sp_help kraj
GO
Name Owner Type Created_datetime
----------- ---------- -------------- -------------------------
kraj dbo user table 2002-11-10 15:25:41.880
Column_name Type Computed Length Prec Scale Nullable TrimTrailingBlanks
ªFixedLenNullInSource Collation
----------- ------ -------- ------ ---- ----- -------- -------------------
------------------- -----------
id_kraju int no 4 10 0 no (n/a)
ª(n/a) NULL
kraj varchar no 20 no no no Polish_CI_AS
uwagi varchar no 250 yes no no Polish_CI_AS
Identity Seed Increment Not For Replication
-------------- ------------ ---------------- -----------------------
id_kraju 1 1 0
RowGuidCol
---------------------------
No rowguidcol column defined.
Data_located_on_filegroup
---------------------------
PRIMARY
index_name index_description index_keys
------------------- ------------------------------------------------------------ --------
pk_kr clustered, unique, primary key located on PRIMARY id_kraju
UQ__kraj__6EF57B66 nonclustered, unique, unique key located on PRIMARY kraj
constraint_type constraint_name delete_action update_action status_enabled
ªstatus_for_replication constraint_keys
----------------------- ------------------ -------------- -------------- ---------------
ª--------------------- -----------------
PRIMARY KEY (clustered) pk_kr (n/a) (n/a) (n/a)
ª(n/a) id_kraju
UNIQUE (non-clustered) UQ__kraj__6EF57B66 (n/a) (n/a) (n/a)
ª(n/a) kraj
Table is referenced by foreign key
----------------------------------
wydawnictwo.dbo.wydawnictwo: FK__wydawnict__id_kr__7F2BE32F
No views with schemabinding reference this table.
```

Możemy również odczytać wyłącznie ustawienia i wartości identyfikatorów typu identity:

```
USE wydawnictwo
SELECT LEFT(TABLE_NAME,30) AS nazwa, IDENT_SEED(TABLE_NAME) AS wartość_początkowa,
IDENT_INCR (TABLE_NAME) AS przyrost, IDENT_CURRENT (TABLE_NAME) AS wartość_bieżąca
FROM INFORMATION_SCHEMA.TABLES
WHERE IDENT_SEED(TABLE_NAME) IS NOT NULL
GO
nazwa wartość_początkowa przyrost wartość_bieżąca
--------------- ---------------------- -------------- ----------------
autor 1 1 2
dtproperties 1 1 1
kraj 1 1 1
ksiazka 3200 1 3200
miasto 1 1 2
nosnik 1 1 1
osoba 1 1 1
seria_wydawnicza 100 1 100
(8 row(s) affected)
```

Możemy sprawdzić spójność danych przechowywanych w określonej tabeli, wykonując instrukcję DBCC CHECKTABLE.

Składnia:

```
DBCC CHECKTABLE
('tabela'
[, NOINDEX
| id_indeksu
| {REPAIR_ALLOW_DATA_LOSS
| REPAIR_FAST
| REPAIR_REBUILD}
]
) [WITH {[ALL_ERRORMSGS | NO_INFOMSGS]
[, [TABLOCK]]
[, [ESTIMATEONLY]]
[, [PHYSICAL_ONLY]]
}
]
```

Przykład:

```
USE Northwind
DBCC CHECKTABLE ('Orders')
GO
DBCC results for 'Orders'.
```

```
There are 830 rows in 28 pages for object 'Orders'.
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
```

Opcje zawężeń

Spójność i poprawność przechowywanych danych wymuszają zawężenia. Informacje o zawężeniach zdefiniowanych dla tabeli poznamy, wywołując procedurę składowaną sp_helpconstraint:

```
USE wydawnictwo
EXEC sp_helpconstraint autor
GO
Object Name
-------------
autor
constraint_type constraint_name delete_action update_action
[a]status_enabled status_for_replication constraint_keys
---------------------- -------------------------------- ------------- -------------
[a]-------------- -------------------- ----------------------------------------------
[a]---CHECK on column e_mail ch_a_m (n/a) (n/a)
[a]Enabled Is_For_Replication (len([e_mail]) - len(replace([e_mail],'@',''')) > 0)
FOREIGN KEY FK__autor__id_miasta__2C3393D0 No Action Cascade
[a]Enabled Is_For_Replication id_miasta
REFERENCES wydawnictwo.dbo.miasto (id_miasta)
PRIMARY KEY pk_a (n/a) (n/a)
[a](n/a) (n/a) id_autora
Table is referenced by foreign key
--------------------------------------
wydawnictwo.dbo.autor_ksiazka: ref_a
```

Instrukcja ALTER TABLE

Za pomocą instrukcji ALTER TABLE możemy dodawać i usuwać kolumny z definicji tabeli oraz włączać i wyłączać sprawdzanie zawężeń i uruchamianie wyzwalaczy.
Składnia:

```
ALTER TABLE tabela
{[ALTER COLUMN kolumna
{typ [(precyzja [, skala])]
[COLLATE <porządek>]
[NULL | NOT NULL]
```

```
| {ADD | DROP} ROWGUIDCOL}
]
| ADD
{[<definicja_kolumny>]
| kolumna AS wynik_wyrażenia
} [,…n]
| [WITH CHECK | WITH NOCHECK] ADD
{<zawężenie_tabeli>} [,…n]
| DROP
{[CONSTRAINT] zawężenie_kolumny
| COLUMN kolumna} [,…n]
| {CHECK | NOCHECK} CONSTRAINT
{ALL | zawężenie [,…n]}
| {ENABLE | DISABLE} TRIGGER
{ALL | wyzwalacz [,…n]}
}
```

gdzie:
składnia klauzul *definicja_kolumny, zawężenie_tabeli i zawężenie_kolumny* jest taka sama jak dla instrukcji CREATE TABLE.
Przykłady:
Po utworzeniu tabeli łączącej tabele autor i ksiazka wyłączamy sprawdzanie zawężeń FOREGIN KEY, dzięki czemu będziemy mogli powiązać informacje o nieistniejących książkach i autorach:

```
USE wydawnictwo
CREATE TABLE autor_ksiazka
(id_autora int CONSTRAINT ref_a REFERENCES autor(id_autora)
ON UPDATE CASCADE ON DELETE CASCADE,
id_ksiazki int CONSTRAINT ref_k REFERENCES ksiazka(id_ksiazki)
ON UPDATE CASCADE ON DELETE CASCADE,)
GO
ALTER TABLE autor_ksiazka
NOCHECK CONSTRAINT ref_a, ref_k
GO
The command(s) completed successfully.
```

Tworzymy tabele słownikowe tabeli klient, po czym do pierwszej dodajemy, a z drugiej usuwamy jedną kolumnę:

```
USE wydawnictwo
CREATE TABLE wyksztalcenie
(id_wyksztalcenia int IDENTITY
CONSTRAINT pk_wy PRIMARY KEY
```

```
)
CREATE TABLE stanowisko
(id_stanowiska int IDENTITY
CONSTRAINT pk_st PRIMARY KEY,
stanowisko varchar (50) NOT NULL,
uwagi varchar (250)
)
GO
ALTER TABLE wyksztalcenie
ADD wykształcenie varchar (50) NOT NULL DEFAULT 'wyższe'
ALTER TABLE stanowisko
DROP COLUMN uwagi
GO
The command(s) completed successfully.
```

Zmiana nazwy tabeli

Zmienić nazwę obiektu bazy danych (w tym tabeli) możemy przez wywołanie procedury składowanej sp_rename. Poniższy program tworzy tabelę czytelnik, a następnie zmienia jej nazwę na klient. Ponieważ w ten sposób niemożliwa jest zmiana obiektu powiązanego referencyjnymi więzami spójności, dwie kolumny przechowujące klucze obce tabel słownikowych zostały dodane do tabeli po zmianie jej nazwy.

Za pomocą procedury sp_rename możemy zmienić nazwę zarówno bazy danych, jak i pojedynczej kolumny wybranej tabeli.

```
USE wydawnictwo
CREATE TABLE czytelnik
(id_klienta int IDENTITY
CONSTRAINT pk_kl PRIMARY KEY,
imie varchar (20) NOT NULL,
nazwisko varchar (25) NOT NULL,
e_mail varchar (40),
data_urodzenia smalldatetime,
uwagi varchar(250),
)
GO
EXEC sp_rename 'czytelnik', 'klient'
GO
ALTER TABLE klient
ADD id_wyksztalcenia int REFERENCES wyksztalcenie(id_wyksztalcenia)
ON UPDATE CASCADE ON DELETE NO ACTION,
```

```
id_miasta int REFERENCES miasto(id_miasta)
ON UPDATE CASCADE ON DELETE NO ACTION
GO
Caution: Changing any part of an object name could break scripts and stored procedures.
The object was renamed to 'klient'.
```

Instrukcja DROP TABLE

Nieużywana i niepotrzebna tabela może zostać usunięta. Usunięcie tabeli spowoduje utratę wszystkich zapisanych w niej danych.

Składnia:

```
DROP TABLE nazwa_tabeli
```

Przykład:

```
USE wydawnictwo
CREATE TABLE zamowienie
(id_zamowienia uniqueidentifier DEFAULT NEWID()
CONSTRAINT pk_z PRIMARY KEY,
id_klienta int NOT NULL REFERENCES klient(id_klienta)
ON UPDATE CASCADE,
id_ksiazki int NOT NULL REFERENCES ksiazka(id_ksiazki)
ON UPDATE CASCADE,
data smalldatetime DEFAULT GETDATE(),
egzemplarzy smallint,
)
GO
DROP TABLE zamowienie
GO
The command(s) completed successfully.
```

Kurs Transact-SQL, część 10 (Indeksy)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Jedynym powodem tworzenia indeksów jest poprawa wydajności bazy danych. Indeksy, tak jak statystyki, nie wpływają na wynik instrukcji języka Transact-SQL, a jedynie na plan i koszt ich wykonania.

Po co tworzyć indeksy?

W rozdziale 8. została opisana fizyczna struktura bazy danych. To właśnie sposób przechowywania danych na dysku twardym komputera ma kluczowe znaczenie dla wydajności całej bazy i jest powodem, dla którego tworzy się takie obiekty jak indeksy i statystyki.

Przypomnijmy sobie fizyczną strukturę baz danych SQL Servera:

1. Dana zapisane są w 8-kilobajtowych ciągłych obszarach nazywanych stronami. Pojedynczy wiersz tabeli nie może być zapisany na kilku stronach, ale małe wiersze mogą dzielić stronę.
2. Strony są łączone w 64-kilobajtowe zakresy. Małe tabele mogą dzielić ten sam zakres.
3. Dane zapisane w stronach nie są w żaden sposób uporządkowane, chyba że został utworzony indeks grupujący.
4. Próba dodania danych do pełnej strony powoduje podzielenie już zapisanych w niej danych pomiędzy dwie strony.

Podstawą planowania i tworzenia indeksów jest zrozumienie sposobu, w jaki SQL Server pobiera i zapisuje żądane przez klienta dane. Jeżeli nie istnieją indeksy, wyszukanie pojedynczego wiersza tabeli wiąże się z koniecznością odczytania wszystkich stron, na których tabela została zapisana. Pełne przeszukanie tabeli rozpoczyna się od odczytania pierwszej strony tabeli, po czym następuje odczytanie wszystkich pozostałych stron i wybór wierszy spełniających podane kryteria. Wynika z tego, że brak indeksu może oznaczać odczytanie megabajtów danych tylko po to, aby np. znaleźć numer czyjegoś telefonu.

Jeżeli natomiast istnieją powiązane z tabelą indeksy (różnica pomiędzy indeksami grupującymi a niegrupującymi została opisana w dalszej części rozdziału), znalezienie żądanych danych sprowadza się do znalezienia w indeksie (który z reguły jest obiektem wielokrotnie mniejszym niż tabela) wierszy tabeli spełniających podane kryteria i odczytania wyłącznie tych stron, na których zostały zapisane znalezione wiersze.

Optymalizator na podstawie statystyk szacuje liczbę wierszy spełniających podane kryteria i tworzy różne plany wykonania instrukcji z wykorzystaniem różnych, dostępnych indeksów.

Typy indeksów

SQL Server pozwala na tworzenie indeksów grupujących (ang. Clustered) i niegrupujących. Indeksy pierwszego typu nie są odrębnym, zapisanym na dysku obiektem, ale narzucają określony porządek wierszom tabeli zapisanym na kolejnych stronach. Indeksy niegrupujące są odrębnymi, zapisanymi na osobnych stronach, obiektami.

Indeksy grupujące

Odpowiednikiem indeksu grupującego jest dowolna encyklopedia czy książka telefoniczna — dane w nich zapisane są w pewnym ściśle określonym porządku (alfabetycznym w przypadku encyklopedii, a tematycznym w przypadku większości dostępnych dzisiaj książek telefonicznych). Dzięki temu, szukając określonego hasła w encyklopedii, nie musimy jej całej wertować (odpowiednik pełnego przeszukania tabeli), wystarczy znaleźć stronę zawierającą hasła rozpoczynające się taką samą sekwencją znaków jak szukane hasło. Po znalezieniu takiej strony lub kilku stron wystarczy odczytać ich zawartość.

SQL Server po utworzeniu indeksu grupującego reorganizuje wszystkie strony tabeli, zapisując wiersze w kolejności określonej przez indeks grupujący. Na przykład, aby uporządkować dane o książkach według ich tytułów, należy utworzyć indeks grupujący dla kolumny tytul tabeli wydawnictwo..ksiazka. Ponieważ niemożliwe jest zapisanie tych samych danych uporządkowanych na kilka różnych sposobów dla tabeli, można utworzyć tylko jeden indeks grupujący.

Utworzenie indeksu grupującego wymaga zmiany fizycznej struktury tabeli i wymaga 120% przestrzeni zajmowanej przez indeksowaną tabelę dostępnego miejsca na dysku. Przestrzeń ta zostanie automatycznie zwolniona po utworzeniu indeksu.

Indeks grupujący powinien być utworzony dla kolumny, według której użytkownicy często sortują dane odczytane z tabeli (skoro dane będą zapisane w tym porządku, niepotrzebne będzie ich sortowanie) oraz dla kolumn przechowujących wartości, na podstawie których zwracane są zbiory danych (np. książki wydane w latach 2000 – 2002). W tym przypadku, ponieważ dane o książkach wydanych w tym samym roku zapisane są na kolejnych stronach, wybranie żądanych danych będzie wymagać odczytania jedynie niektórych, kolejnych stron tabeli.

Szczególnym przypadkiem drugiej z opisywanych sytuacji jest wybieranie danych na podstawie atrybutu, który przyjmuje jedynie określoną, małą w porównaniu do liczby wierszy, liczbę wartości. Na przykład utworzenie indeksu grupującego dla kolumny id_serii pozwoli na szybkie wyszukanie książek należących do określonej serii wydawniczej.

Niestety, prawie na pewno każda tabela produkcyjnej bazy danych zawiera kilka kolumn spełniających wymienione kryteria, a administrator musi wybrać tę jedną, dla której utworzy indeks grupujący. Podejmując decyzję, można kierować się takimi czynnikami jak częstotliwość wykonywania zapytań pobierających dane z poszczególnych tabel czy pozycja wykonującej je osoby (na przykład: Czy indeks utworzyć dla kolumny przechowującej dane, które wchodzą w skład uruchomianego przez kierownika raportu, czy też dla kolumny wykorzystywanej przez codziennie generowane zestawienia i dzięki temu kończyć pracę 15 minut wcześniej?)

Istotną wskazówką jest to, że indeksy niegrupujące są prawie tak samo wydajne jak grupujące w przypadku zwracania pojedynczych wierszy przez zapytanie. Na przykład znalezienie książki o określonym numerze ISBN (kolumnie, na którą nałożono zawężenie typu UNIQUE) będzie równie szybkie za pomocą indeksu niegrupującego jak za pomocą indeksu grupującego. Dopiero przy zwracaniu zbioru wierszy indeksy grupujące okazują się wydajniejsze niż niegrupujące.

Indeksy niegrupujące

Indeks niegrupujący jest zapisany na odrębnych stronach i fizycznie jest całkowicie oddzielony od tabeli, dla której został utworzony. Odpowiednikiem indeksu tego typu jest indeks znajdujący się na końcu niektórych książek — ponieważ poszczególne hasła indeksu zapisane są w określonym (alfabetycznym) porządku, znalezienie żądanych danych jest szybkie i sprowadza się do przeszukania indeksu, znalezienia hasła i umieszczonych przy nim odnośników, wreszcie do znalezienia na wskazanej stronie szukanych informacji.

W przeciwieństwie do „książkowego” indeksu, pojedyncza pozycja niegrupującego indeksu SQL Servera zawsze wskazuje na jeden wiersz tabeli. Ponieważ indeks niegrupujący nie wpływa w żaden sposób na uporządkowanie stron tabeli (uporządkowane są jedynie pozycje samego indeksu), dla każdej tabeli można ich utworzyć maksymalnie 249.

Przed utworzeniem indeksów niegrupujących należy utworzyć indeks grupujący dla danej tabeli. Indeksy niegrupujące używają indeksu grupującego do pobierania żądanych danych z tabeli.

Indeksy kompozytowe

Indeksy obu typów mogą być tworzone dla jednej lub kilku kolumn. Indeksy kompozytowe mogą być tworzone dla maksymalnie 16 kolumn.

Kolumny przechowujące wartości, do których odwołujemy się najczęściej jednocześnie (np. imię i nazwisko) powinny zostać powiązane z jednym indeksem kompozytowym. W ten sposób utworzymy indeks zawierający zapytanie, co jest optymalnym sposobem poprawy wydajności bazy danych.

Wszystkie kolumny wchodzące w skład indeksu kompozytowego muszą należeć do tej samej tabeli.

Kolejność, w jakiej wymienione są poszczególne kolumny indeksu kompozytowego ma kluczowe znaczenie dla wykorzystania indeksu przez SQL Server (z reguły jako pierwszą należy określić kolumnę przechowującą bardziej różnorodne wartości) — tak jak książka telefoniczna jest dla nas bezużyteczna, jeżeli znamy jedynie imię, a nie nazwisko osoby, tak optymalizator nie jest w stanie skorzystać z indeksu kompozytowego uporządkowanego według wartości nie pierwszej kolumny indeksu. Dlatego poniższe zapytanie zawsze wykonywane jest według tego samego planu, pomimo że utworzyliśmy indeks zawierający to zapytanie:

```
USE Northwind
SELECT ContactTitle
FROM dbo.Customers
```

```
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Customers].[PK_Customers]))
(1 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0.
CREATE
INDEX Suppliers_cn_ct ON dbo.Suppliers (ContactName, ContactTitle)
GO
SELECT ContactTitle
FROM dbo.Customers
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Customers].[PK_Customers]))
(1 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0.
```

Natomiast w przypadku konkatenacji argumentów SARG kolejność kolumn indeksu zawierającego zapytanie jest nieistotna — tak czy inaczej dane są uporządkowane według jednego z kryteriów i to odpowiadający mu warunek logiczny będzie sprawdzony jako pierwszy. Tak ograniczony zbiór danych będzie następnie sprawdzony pod kątem zgodności z kolejnymi kryteriami.

Dla których kolumn tworzyć indeksy?

Indeksy skracają czas potrzebny na wyszukanie i odczytanie żądanych danych. Jednak, z wyjątkiem indeksów grupujących, mogą wydłużyć czas operacji wstawiania i modyfikowania danych. Zmiana lub dodanie danych do poindeksowanych kolumn powoduje automatyczną aktualizację indeksu, z drugiej strony jednak duża cześć operacji aktualizacji danych wymaga również ich odczytania (np. instrukcja UPDATE z klauzulą WHERE), a koszt znalezienia poindeksowanych danych i aktualizacji zarówno danych, jak i indeksu może okazać się niższy niż koszta wyszukania danych na stronach tabeli.

> Dla baz OLTP nie należy tworzyć więcej niż 8 – 10 indeksów związanych z pojedynczą tabelą.

Po wybraniu kolumny (lub kolumn), dla której zostanie utworzony indeks grupujący, należy określić kolumny powiązane z indeksami niegrupującymi. Odpowiednimi kandydatami są kolumny, które:

1. Przechowują wartości częściej odczytywane niż modyfikowane.
2. Wykorzystywane są do łączenia lub wyszukiwania danych.
3. Przechowują różnorodne wartości, dzięki czemu ta sama wartość indeksu nie wskazuje na dużą liczbę wierszy tabeli.

W praktyce indeksy tworzy się dla:

1. Kolumn, na które nałożono ograniczenie typu PRIMARY KEY.

2. Kolumn, na które nałożono ograniczenie typu FOREIGN KEY oraz kolumn wykorzystywanych przy łączeniu tabel.
3. Kolumn przechowujących dane wykorzystywane jako argumenty wyszukiwania.
4. Kolumn przechowujących często sortowane dane.

Niemożliwe jest utworzenie indeksu dla kolumn przechowujących dane typu bit, text, ntext lub image.

Wpływ indeksów grupujących na sortowanie danych

Poszczególne wiersze tabeli zapisane są na przypadkowych stronach, chyba że dla tabeli został utworzony indeks grupujący. Natomiast dane indeksu zapisane są zawsze w określonym porządku. Właśnie z tego powodu, a także dlatego, że wielkość indeksu z reguły nie przekracza 10% wielkości wiersza, odwoływanie się do indeksów ma tak duży wpływ na czas wykonania zapytania.

Porównajmy plan i czas wykonania zapytania zwracającego wszystkie dane z tabeli Order Details uporządkowane według kolumny, dla której utworzono indeks z planem wykonania zapytania zwracającego te same, ale inaczej uporządkowane dane:

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY OrderID
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
[a]Details].[PK_Order_Details]), ORDERED FORWARD)
(1 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 1, read-ahead
[a]reads 9.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 148 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Ponieważ dane są już uporządkowane, SQL Server jedynie odczytał kolejne strony danych. Ten sam plan zostanie wykorzystany, jeżeli dane będą uporządkowane malejąco według wartości kolumny OrderID:

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY OrderID DESC
```

```
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
[a]Details].[PK_Order_Details]), ORDERED BACKWARD)
(1 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 0, read-ahead
[a]reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 137 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Jedyną różnicą (krótszy czas wykonania zapytania spowodowany jest odczytaniem wszystkich stron z bufora) jest, sygnalizowana zapisem ORDERED BACKWARD, zmiana kolejności odczytywanych stron. Natomiast uporządkowanie danych według wartości przechowywanych w niepowiązanej z żadnym indeksem kolumnie wymaga wykonania innego planu:

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY UnitPrice
GO
StmtText
------------------------------------------------------------------------------
|--Sort(ORDER BY:([Order Details].[UnitPrice] ASC))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
[a]Details].[PK_Order_Details]))
(2 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 0, read-ahead
[a]reads 0.
SQL Server Execution Times:
CPU time = 20 ms, elapsed time = 149 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Wystąpiły dwie różnice w wykonanym planie: po pierwsze, oprócz odczytania indeksu grupującego, niezbędne jest przeprowadzenie dodatkowej operacji sortowania danych (stąd wyższy koszt użycia procesora dla zapytania), po drugie — indeks grupujący odczytywany jest w dowolnej kolejności (brak wpisu ORDERED BACKWARD lub ORDERED FORWARD).

Wróćmy do kwestii sortowania danych w porządku malejącym — SQL Server umożliwia tworzenie indeksów przechowujących dane w porządku rosnącym (domyślnie) lub malejącym. Wybór uporządkowania danych nie jest bez znaczenia — jeżeli indeks jest

indeksem kompozytowym, to tylko zapytania sortujące dane uporządkowane tak, jak zostało to określone podczas tworzenia indeksu, będą mogły z niego skorzystać (poniższy przykład odwołuje się do powiązanego z kolumnami OrderID i ProductID indeksu grupującego):

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY OrderID, ProductID
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
ªDetails].[PK_Order_Details]), ORDERED FORWARD)
(1 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 0, read-ahead
ªreads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 136 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Oczywiście zapytanie sortujące dane według wartości kolumn ProductID, a następnie według wartości OrderID zostanie wykonane według innego planu — tego, który był wykorzystany podczas wykonywania zapytania sortującego dane według wartości kolumny UnitPrice.

Jeżeli wykonamy zapytanie różniące się od poprzedniego jedynie porządkiem sortowania, przekonamy się, że również w tym wypadku niezbędne okazało się sortowanie odczytanych danych:

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY OrderID, ProductID DESC
GO
StmtText
------------------------------------------------------------------------------
|--Sort(ORDER BY:([Order Details].[OrderID] ASC, [Order Details].[ProductID] DESC))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
ªDetails].[PK_Order_Details]))
(2 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 0, read-ahead
ªreads 0.
SQL Server Execution Times:
```

```
CPU time = 20 ms, elapsed time = 139 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Ponieważ dane fizycznie zapisane są w innej kolejności, ich uporządkowanie wymaga posortowania.

> Jeżeli większość zapytań sortuje dane w porządku malejącym, podczas tworzenia indeksu grupującego należy określić ten właśnie porządek przechowywania danych.

Wpływ indeksów niegrupujących na sortowanie danych

W praktyce indeksy niegrupujące nie są wykorzystywane przez SQL Server do sortowania danych. Wyjątkiem od tej reguły są indeksy zawierające zapytania, ponieważ taką sytuację należy rozpatrywać jako przykład pobierania danych z projekcji tabeli, dla której utworzono indeks grupujący. Ponieważ algorytm sortowania SQL Servera jest optymalizowany pod kątem sortowania kompletnych zbiorów danych, a nie następujących po sobie grup wskazanych przez wskaźniki indeksu, odczytanie wszystkich wierszy, a następnie ich posortowanie pozwala na osiągniecie maksymalnej wydajności.

Dlatego odczytanie danych z tabeli Order Details uporządkowanych według wartości kolumny ProductID (z którą powiązane są indeksy niegrupujące) będzie wykonane następująco:

```
USE Northwind
SELECT *
FROM dbo.[Order Details]
ORDER BY ProductID
GO
StmtText
------------------------------------------------------------------------------
|--Sort(ORDER BY:([Order Details].[ProductID] ASC))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Order
ªDetails].[PK_Order_Details]))
(2 row(s) affected)
SQL Server Execution Times:
CPU time = 10 ms, elapsed time = 140 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Możliwe jest wymuszenie użycia przez SQL Server indeksu niegrupującego do sortowania danych. W ten sposób skróci się czas zwrócenia pierwszego wiersza wyniku, ale czas wykonania całej operacji ulegnie wydłużeniu:

```
USE Northwind
```

```
SELECT *
FROM dbo.[Order Details](FASTFIRSTROW)
ORDER BY ProductID
GO
StmtText
------------------------------------------------------------------------------
|--Bookmark Lookup(BOOKMARK:([Bmk1000]), OBJECT:([Northwind].[dbo].[Order Details]))
|--Index Scan(OBJECT:([Northwind].[dbo].[Order Details].[ProductID]), ORDERED
ªFORWARD)
(2 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 4315, physical reads 0, read-ahead
ªreads 0.
SQL Server Execution Times:
CPU time = 11 ms, elapsed time = 144 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Wpływ indeksów na modyfikowanie danych

To, że indeksy przyspieszają pobieranie danych jest oczywiste (różne aspekty wpływu indeksów na wynik zapytania opisane są w bieżącym rozdziale oraz w rozdziale 4.). Przeanalizujmy jednak nieco dokładniej wpływ indeksów na modyfikowanie danych.

Utworzymy kopię liczącej najwięcej wierszy tabeli bazy Northwind — Order Details i porównamy koszt zmodyfikowania usunięcia danych z tabeli nieposiadającej indeksów oraz posiadającej indeksy.

```
USE Northwind
SELECT *
INTO od
FROM dbo.[Order Details]
GO
(2155 row(s) affected)
```

Kluczem głównym tabeli Order Details jest klucz kompozytowy złożony z kolumn OrderID i ProductID. Klucz główny (co jest typowe dla SQL Servera) powiązany jest z indeksem grupującym. Tak więc wiersz tabeli może zostać jednoznacznie zidentyfikowany jedynie poprzez podanie obu wartości — identyfikatora zamówienia i identyfikatora produktu.

Zmodyfikujmy wybrane wiersze tabeli od:

```
USE Northwind
UPDATE dbo.od
SET UnitPrice = UnitPrice*1.10
WHERE OrderID IN (10803,10360) AND ProductID>25
```

```
GO
(6 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Table Update(OBJECT:([Northwind].[dbo].[od]),
ªSET:([od].[UnitPrice]=RaiseIfNull([Expr1004])))
|--Compute Scalar(DEFINE:([Expr1004]=Convert(Convert([od].[UnitPrice])*1.10)))
|--Top(ROWCOUNT est 0)
|--Table Scan(OBJECT:([Northwind].[dbo].[od]),
ªWHERE:([od].[ProductID]>25 AND ([od].[OrderID]=10360 OR
ª[od].[OrderID]=10803)) ORDERED)
(4 row(s) affected)
Table 'od'. Scan count 1, logical reads 16, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 2 ms, elapsed time = 2 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Wykonanie tej samej instrukcji dla tabeli źródłowej przebiegać będzie następująco:

```
USE Northwind
UPDATE dbo.[Order Details]
SET UnitPrice = UnitPrice*1.10
WHERE OrderID IN (10803,10360) AND ProductID>25
GO
(6 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Assert(WHERE:(If ([Expr1004]>=0.00=0) then 0 else NULL))
|--Clustered Index Update(OBJECT:([Northwind].[dbo].[Order Details].
ª[PK_Order_Details]), SET:([Order Details].[UnitPrice]=RaiseIfNull([Expr1004])))
|--Compute Scalar(DEFINE:([Expr1004]=Convert(Convert([Order Details].
ª[UnitPrice])*1.10)))
|--Top(ROWCOUNT est 0)
|--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Order Details].
ª[PK_Order_Details]), SEEK:([Order Details].[OrderID]=10360 AND
ª[Order Details].[ProductID] > 25 OR [Order Details].[OrderID]=
ª10803 AND [Order Details].[ProductID] > 25)
(5 row(s) affected)
Table 'Order Details'. Scan count 2, logical reads 16, physical reads 0,
ªread-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
```

```
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Najbardziej kosztowną operacją w wykonywaniu drugiej instrukcji jest przeszukanie indeksu grupującego i znalezienie wierszy spełniających podane kryteria. Koszt pozostałych, w tym aktualizacji danych, jest pomijany. Istotne jest to, że całkowity czas wykonania operacji dla tabeli powiązanej z indeksem grupującym okazał się niższy.

Koszt operacji podziału stron indeksu nie jest uwzględniony w statystykach wykonania instrukcji.

Porównajmy teraz plan i koszt wykonania operacji usunięcia pojedynczego wiersza tabeli:

```
USE Northwind
DELETE dbo.od
WHERE OrderID =11077 AND ProductID=10
GO
(1 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Table Delete(OBJECT:([Northwind].[dbo].[od]))
|--Top(ROWCOUNT est 0)
|--Table Scan(OBJECT:([Northwind].[dbo].[od]), WHERE:([od].[OrderID]=
[a]Convert([@1]) AND [od].[ProductID]=Convert([@2])) ORDERED)
(3 row(s) affected)
Table 'od'. Scan count 1, logical reads 11, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 1 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Choć tylko jeden wiersz spełniał sformułowane za pomocą argumentów SARG kryteria SQL Server odczytał 11 stron danych.

```
USE Northwind
DELETE dbo.[Order Details]
WHERE OrderID =11077 AND ProductID=10
GO
(1 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Clustered Index Delete(OBJECT:([Northwind].[dbo].[Order Details].
[a][PK_Order_Details]))
```

```
|--Index Seek(OBJECT:([Northwind].[dbo].[Order Details].[ProductsOrder_Details]),
ªSEEK:([Order Details].[ProductID]=Convert([@2]) AND [Order Details].[OrderID]=
ªConvert([@1])) ORDERED FORWARD)
(2 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 2, physical reads 0,
ªread-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

W przypadku uporządkowanej tabeli źródłowej wystarczyło odczytanie 2 kolejnych stron. Tak więc indeksy, szczególnie grupujące lub pojedyncze niegrupujące mogą przyspieszyć również modyfikowanie danych w tabeli.

Wpływ indeksów na dodawanie danych

Dodanie nowego wiersza do powiązanej z dowolnymi indeksami tabeli zawsze jest bardziej kosztowne niż wykonanie tej samej operacji dla tabeli niepowiązanej z jakimkolwiek indeksem. W przypadku indeksu grupującego SQL Server musi odnaleźć właściwą stronę, dodać do niej nowy wiersz i, jeżeli jest to konieczne, podzielić znalezioną stronę.

```
USE Northwind
INSERT INTO dbo.[Order Details] (OrderID, ProductID, UnitPrice, Quantity, Discount)
VALUES (11077, 10, 56.3, 10, 0)
GO
(1 row(s) affected)
StmtText
------------------------------------------------------------------------------------
|--Assert(WHERE:(If (([Expr1002]>=0 AND [Expr1002]<=1)=0) then 0 else If
ª([Expr1001]>0=0) then 1 else If ([Expr1000]>=0.00=0) then 2 else If
ª(NOT([Pass1010]) AND ([Expr1009] IS NULL)) then 3 else If (NOT([Pass1012]) AND
ª([Expr1011] IS NULL)) then 4 else
|--Nested Loops(Left Semi Join, WHERE:([@2] IS NULL), DEFINE:([Expr1011] =
ª[PROBE VALUE]))
|--Nested Loops(Left Semi Join, WHERE:([@1] IS NULL), DEFINE:([Expr1009] =
ª[PROBE VALUE]))
| |--Clustered Index Insert(OBJECT:([Northwind].[dbo].[Order Details].
ª[PK_Order_Details]), SET:([Order Details].[Discount]=RaiseIfNull
ª([Expr1002]), [Order Details].[Quantity]=RaiseIfNull([Expr1001]),
ª[Order Details].[UnitPrice]=RaiseIfNull([
| |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Orders].
ª[PK_Orders]), SEEK:([Orders].[OrderID]=[@1]) ORDERED FORWARD)
```

```
|--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Products].
[PK_Products]), SEEK:([Products].[ProductID]=[@2]) ORDERED FORWARD)
(6 row(s) affected)
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
Table 'Products'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.
Table 'Orders'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.
Table 'Order Details'. Scan count 0, logical reads 10, physical reads 0, read-ahead
reads 0.
```

Odczytanie tabel Products i Orders związane było z nałożeniem na odpowiednie kolumny tabeli Order Details zawężenia FOREIGN KEY.

W przypadku tabeli niepowiązanej z indeksem grupującym SQL Server, na podstawie statystyk wypełniania poszczególnych stron tabeli, może wybrać stronę, dla której dopisanie wiersza będzie najmniej kosztowne.

```
USE Northwind
INSERT INTO dbo.od (OrderID, ProductID, UnitPrice, Quantity, Discount)
VALUES (11077, 10, 56.3, 10, 0)
GO
(1 row(s) affected)
StmtText
------------------------------------------------------------------------------
|--Table Insert(OBJECT:([Northwind].[dbo].[od]), SET:([od].[Discount]=RaiseIfNull
([Expr1002]), [od].[Quantity]=RaiseIfNull([Expr1001]), [od].[UnitPrice]=
RaiseIfNull([Expr1000]), [od].[ProductID]=RaiseIfNull([@2]), [od].[OrderID]=
RaiseIfNull([@1])), DEFIN
(1 row(s) affected)
Table 'od'. Scan count 0, logical reads 1, physical reads 0, read-ahead reads 0.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Liczba odczytanych stron zmniejszyła się dziesięciokrotnie.

Indeksy zawierające zapytanie

Indeks zawiera zapytanie, jeżeli do jego wykonania wystarczy odczytanie zawartości indeksu, bez konieczności odwoływania się do tabeli źródłowej. Utworzenie indeksu zawierającego zapytanie jest najlepszym sposobem na skrócenie czasu jego wykonania.

Na przykład, aby wyświetlić listę miast klientów z Niemiec, optymalizator przeszuka indeks grupujący:

```
USE Northwind
SELECT DISTINCT City
FROM dbo.Customers
WHERE Country = 'Germany'
GO
StmtText
------------------------------------------------------------------------------
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].[Customers].[PK_Customers]),
ªWHERE:([Customers].[Country]=Convert([@1])))
(1 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0.
```

Wykonanie tego samego zapytania po dodaniu indeksu utworzonego dla kolumn Country i City będzie polegać na odczytaniu zawartości nowo utworzonego indeksu:

```
CREATE INDEX Customers_co_ci
ON Customers(Country, City)
GO
SELECT City
FROM dbo.Customers
WHERE Country = 'Germany'
GO
StmtText
------------------------------------------------------------------------------
|--Index Seek(OBJECT:([Northwind].[dbo].[Customers].[Customers_co_ci]),
ªSEEK:([Customers].[Country]=Convert([@1])) ORDERED FORWARD)
(1 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 1, physical reads 0, read-ahead reads 0.
```

Po znalezieniu (przeszukiwanie pionowe poprzez poziomy indeksu (ang. Index seek)) pierwszej wartości spełniającej podane kryteria SQL Server odczytywał kolejne wiersze indeksu (przeszukiwanie poziome poprzez liście indeksu (ang. Index scan)), aż do natrafienia na wartość niespełniającą podanego kryterium. W rezultacie liczba odczytanych stron zmniejszyła się czterokrotnie.

Jeżeli czas wykonania zapytania jest tak krótki, że Query Analyzer podaje czas równy 0, nie będzie umieszczany w książce.

Porównajmy powyższy plan wykonania zapytania z zapytaniem, które nie zawiera się w nowym indeksie:

```
SELECT City, ContactTitle
FROM dbo.Customers
WHERE Country = 'Germany'
```

```
GO
StmtText
------------------------------------------------------------------------------
|--Bookmark Lookup(BOOKMARK:([Bmk1000]), OBJECT:([Northwind].[dbo].[Customers]))
|--Index Seek(OBJECT:([Northwind].[dbo].[Customers].[Customers_co_ci]),
ªSEEK:([Customers].[Country]=Convert([@1])) ORDERED FORWARD)
(2 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 23, physical reads 0, read-ahead reads 0.
```

Ponieważ tym razem część danych musiała zostać odczytana z tabeli, SQL Server użył nowego indeksu do odczytania wskaźników do tabeli źródłowej. W konsekwencji liczba odczytanych stron zwiększyła się wielokrotnie.

Dzięki temu, że indeksy niegrupujące odwołują się do danych tabeli poprzez indeks grupujący, wykonanie poniższego zapytania nie będzie wymagało odwoływania się poprzez wskaźniki do tabeli — w rzeczywistości plan wykonania tego zapytania jest taki sam jak dla zapytania zawierającego się w indeksie Customers_co_ci, pomimo tego, że indeks ten nie jest powiązany z kolumną CustomerID:

```
SELECT CustomerID, City
FROM dbo.Customers
WHERE Country = 'Germany'
GO
StmtText
------------------------------------------------------------------------------
|--Index Seek(OBJECT:([Northwind].[dbo].[Customers].[Customers_co_ci]),
ªSEEK:([Customers].[Country]=Convert([@1])) ORDERED FORWARD)
(1 row(s) affected)
Table 'Customers'. Scan count 1, logical reads 1, physical reads 0, read-ahead reads 0.
```

Przeszukanie poziome indeksu zawierającego zapytanie

Oprócz przedstawionego wyżej sposobu użycia indeksu zawierającego zapytanie (polegającego na znalezieniu pierwszego liścia zawierającego żądane dane i odczytaniu zawartości kolejnych liści indeksu), SQL Server korzysta z indeksów tego typu podczas odczytywania żądanych wartości (następuje wyłącznie przeszukiwanie poziome).

Wykonajmy poniższe zapytanie i porównajmy plan jego wykonania z planem wykonania tego zapytania po utworzeniu zawierającego je indeksu:

```
SELECT AVG (Quantity)
FROM dbo.[Order Details]
GO
StmtText
------------------------------------------------------------------------------
|--Compute Scalar(DEFINE:([Expr1002]=If ([Expr1003]=0) then NULL else
```

```
a([Expr1004]/Convert([Expr1003])))
|--Stream Aggregate(DEFINE:([Expr1003]=Count(*),
a[Expr1004]=SUM([Order Details].[Quantity])))
|--Clustered Index Scan(OBJECT:([Northwind].[dbo].
a[Order Details].[PK_Order_Details]))
(3 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 10, physical reads 0,
aread-ahead reads 0.
```

Po przeszukaniu indeksu grupującego SQL Server zliczył wszystkie wiersze tabeli i dodał wartości zapisane w kolumnie Quantity. Następnie wyliczył i zwrócił wartość średnią.

Po utworzeniu indeksu zawierającego zapytanie do jego wykonania wystarczy odczytanie wybranych liści indeksu i wykonanie tych samych obliczeń co poprzednio:

```
CREATE INDEX od_q
ON [Order Details](Quantity)
GO
SELECT AVG (Quantity)
FROM dbo.[Order Details]
GO
StmtText
------------------------------------------------------------------------------
|--Compute Scalar(DEFINE:([Expr1002]=If ([Expr1003]=0) then NULL else
a([Expr1004]/Convert([Expr1003])))
|--Stream Aggregate(DEFINE:([Expr1003]=Count(*),
a[Expr1004]=SUM([Order Details].[Quantity])))
|--Index Scan(OBJECT:([Northwind].[dbo].[Order Details].[od_q]))
(3 row(s) affected)
Table 'Order Details'. Scan count 1, logical reads 5, physical reads 0,
aread-ahead reads 0.
```

W rezultacie liczba odczytanych stron zmniejszyła się o połowę.

Instrukcja CREATE INDEX

Wykonanie instrukcji spowoduje utworzenie indeksu powiązanego z wybraną tabelą. Tylko właściciel obiektu może tworzyć powiązane z nim indeksy.

Możliwe (choć w praktyce rzadko wykorzystywane) jest zapisanie indeksu w innej bazie danych niż baza zawierająca tabelę źródłową. W tym celu należy podać pełną nazwę tworzonego indeksu, wraz z nazwą docelowej bazy danych.

Składnia:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX indeks
```

```
ON {tabela (kolumna [ASC | DESC] [,…n])
[WITH
[PAD_INDEX]
[[,] FILLFACTOR = współczynnik_wypełnienia]
[[,] IGNORE_DUP_KEY]
[[,] DROP_EXISTING]
[[,] STATISTICS_NORECOMPUTE]
[[,] SORT_IN_TEMPDB]
]
[ON grupa_plików]
```

gdzie:

PAD_INDEX oznacza utworzenie indeksu, którego środkowe poziomy (poziomy pomiędzy korzeniem a liściem) zostaną zapisane na stronach niewypełnionych w 100%, dzięki czemu dodanie lub modyfikacja danych nie będą oznaczały konieczności przebudowy tego indeksu. Parametry wykorzystywane są w połączeniu z parametrem FILLFACTOR. Domyślnie SQL Server na każdej stronie indeksu zostawi przynajmniej tyle wolnej przestrzeni, ile wystarcza, aby możliwe było zapisanie dwóch dodatkowych wierszy indeksu,
FILLFACTOR określa procent wolnej przestrzeni na stronach przechowujących wartości liści indeksu,
IGNORE_DUP_KEY (opcja dostępna tylko dla indeksów unikatowych) sprawi, że próba wstawienia wartości już istniejącej w jednej z kolumn indeksu unikatowego spowoduje wyświetlenie ostrzeżenia, a naruszająca warunki integralności instrukcja zostanie zignorowana, ale transakcja będzie kontynuowana. Brak tego parametru powoduje przy próbie wstawienia istniejących wartości wyświetlenie komunikatu błędu i wycofanie całej transakcji,
DROP_EXISTING zastępuje istniejący indeks nowym o tej samej nazwie,
STATISTICS_NORECOMPUTE wyłącza automatyczne aktualizowanie statystyk dla indeksu,
SORT_IN_TEMPDB powoduje, że wewnętrzne operacje sortowania danych indeksu będą przeprowadzane w bazie tempdb. Jeżeli baza tempdb znajduje się na osobnym, szybkim dysku, może to spowodować skrócenie czasu sortowania danych indeksu.

SQL Server nie umożliwia tworzenia indeksów o wielkości jednego wiersza przekraczającej 900 bajtów.

Przykłady:

Dodajemy indeks grupujący dane zapisane w tabeli autor według wartości kolumn nazwisko i imie. Kolejność, w której podano argumenty, ma kluczowe znaczenie dla sposobu uporządkowania danych na dysku:

```
USE wydawnictwo
CREATE CLUSTERED
INDEX autor_n_i ON dbo.autor (nazwisko, imie)
GO
The command(s) completed successfully.
```

Utworzenie indeksu grupującego z określonym współczynnikiem wypełnienia wymusi zapisywanie wierszy tabeli na stronach wypełnionych w 80%:

```
USE wydawnictwo
CREATE CLUSTERED
INDEX autor_ksiazka_a ON dbo.autor_ksiazka (id_autora)
WITH
PAD_INDEX
,FILLFACTOR = 80
GO
The command(s) completed successfully.
```

Tworzymy zbiór indeksów przyspieszających łączenie tabeli klient z tabelami słownikowymi oraz pobieranie wybranych danych o klientach:

```
USE wydawnictwo
CREATE
INDEX klient_w ON dbo.klient (id_wyksztalcenia)
CREATE
INDEX klient_m ON dbo.klient (id_miasta)
CREATE
INDEX klient_n_i_m ON dbo.klient (nazwisko, imie, e_mail)
GO
The command(s) completed successfully.
```

Uporządkowanie wierszy tabeli ksiazka, a następnie dodanie indeksu przyspieszającego wyszukiwanie tytułów według numerów ISBN nie spowoduje natychmiastowego aktualizowania statystyk indeksu niegrupującego:

```
CREATE CLUSTERED
INDEX ksiazka_s ON dbo.ksiazka (id_serii)
GO
CREATE UNIQUE
INDEX ksiazka_i_t ON dbo.ksiazka (isbn, tytul)
WITH
IGNORE_DUP_KEY
GO
The command(s) completed successfully.
```

Po utworzeniu indeksu należy wykonać kopię zapasową plików bazy danych, nawet jeżeli przyjęto strategię wykonywania kopii dziennika transakcyjnego. Odtworzenie indeksu na podstawie wpisu do dziennika polega na jego ponownym utworzeniu, co może być operacją bardzo kosztowną. Tematowi tworzenia kopii zapasowych poświęcony jest rozdział 18.

Współczynnik wypełnienia

SQL Server tworzy indeksy niegrupujące według zmodyfikowanego algorytmu drzewa binarnego (ang. B-tree). Jedna strona każdego indeksu jest korzeniem (ang. Root), stroną, która zawiera wskaźniki do dwóch lub więcej stron indeksu drugiego poziomu. Każda ze stron drugiego poziomu zawiera wskaźniki do pewnej liczby stron poziomu trzeciego itd., aż do osiągnięcia ostatniego poziomu indeksu, poziomu liścia (ang. Leaf). Strony-liście przechowują uporządkowane (SQL Server 2000 pozwala na zapisanie ich w porządku rosnącym lub malejącym), kompletne dane indeksu.

Odpowiednio dobrany współczynnik wypełniania może znacząco przyspieszyć operacje wstawiania oraz modyfikowania danych w poindeksowanych kolumnach. Jeżeli wystąpi żądanie dopisania danych do pełnej strony, SQL Server musi podzielić dane pomiędzy strony, a następnie zapisać informacje (ponieważ liście indeksu zapisane są w określonym porządku, np. zmiana nazwiska spowoduje dopisanie do właściwej strony nowego wskaźnika i usunięcie go ze strony oryginalnej). Z drugiej strony duży procent niewykorzystanej przestrzeni zwiększa liczbę operacji wejścia-wyjścia i pogarsza wydajność bazy danych (indeksy powinny być jak „najwęższe” — poszczególne liście powinny być zapisane na jak najmniejszej liczbie bajtów).

Zabronienie wypełniania danymi całych stron (domyślne ustawienie SQL Servera) umożliwi modyfikowanie danych bez automatycznej konieczności podziału strony. Domyślna wartość współczynnika (0) powoduje całkowite wypełnienie danymi stron-liści i pozostawienie wolnego miejsca na co najmniej dwa nowe wiersze na stronach wyższego poziomu.

Możliwe jest również ustawienie nowej domyślnej wartości współczynnika wypełnienia liści indeksu na poziomie serwera. W tym celu należy wywołać procedurę składowaną sp_configure, a następnie zrekonfigurować SQL Server:

```
sp_configure 'fill factor', '70'
GO
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'fill factor (%)' changed from 0 to 70. Run the RECONFIGURE
[a]statement to install.
```

Istotne jest to, że wartość współczynnika FILL FACTOR określa początkowy współczynnik wypełniania dla indeksu, zapisując tę wartość w kolumnie OrigFillFactor tabeli systemowej sysindexes (sposób wyświetlenia tych informacji pokazany został poniżej).

```
USE wydawnictwo
SELECT Name, OrigFillFactor
FROM dbo.sysindexes
GO
```

Zmienić współczynnik wypełnienia możemy jedynie poprzez rekonstrukcję indeksu. Opcjonalnym parametrem polecenia DBCC DBREINDEX jest wartość współczynnika

wypełniania. Poniższa instrukcja rekonstruuje wszystkie indeksy powiązane z tabelą Orders na każdej ze stron liści indeksu, zostawiając 30% wolnej przestrzeni:

```
USE Northwind
DBCC DBREINDEX ('dbo.Orders', '',70)
GO
DBCC execution completed. If DBCC printed error messages, contact your system
ªadministrator.
```

Pełna składnia polecenia DBCC DBREINDEX:

```
DBCC DBREINDEX
( ['baza_danych.właściciel.tabela'
[, indeks
[, współczynnik_wypełnienia]
]
]
) [WITH NO_INFOMSGS]
```

Klauzula PAD_INDEX wymieniona podczas tworzenia indeksu spowoduje, że również strony, na których zapisane są wyższe poziomy indeksu, zostaną wypełnione danymi w stopniu określonym przez współczynnik FILL FACTOR.

Opcje indeksów

Odczytać aktualne informacje o powiązanych z tabelą indeksach możemy poprzez wywołanie procedury składowanej sp_helpindex. Poniższa instrukcja zwraca informacje o indeksach powiązanych z tabelą wydawnictwo..autor:

```
USE wydawnictwo
EXEC sp_helpindex autor
GO
index_name index_description index_keys
------------ ---------------------------------------------------- -------------------
autor_n_i clustered located on PRIMARY nazwisko, imie
pk_ nonclustered, unique, primary key located on PRIMARY id_autora
```

Zdefragmentować jeden lub wszystkie indeksy związane z określoną tabelą możemy poprzez wykonanie instrukcji DBCC INDEXDEFRAG. Na przykład, aby uporządkować strony indeksu CustomerID, napiszemy:

```
DBCC INDEXDEFRAG (Northwind, Orders, CustomerID)
GO
Pages Scanned Pages Moved Pages Removed
-------------- ------------ --------------
```

```
0 0 0
(1 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system
ªadministrator.
```

Należy pamiętać o tym, że nie istnieje żaden mechanizm, który zapobiegałby przypadkowemu tworzeniu indeksów niegrupujących powiązanych z tymi samymi kolumnami, ale noszących inne nazwy. Możemy się o tym przekonać, wywołując procedurę sp_helpindex dla tabeli Orders bazy danych Northwind — indeksy CustomerID, CustomersOrders powiązane są z kolumną CustomerID, a indeksy EmployeeID i EmployeesOrders z kolumną EmployeeID. Ponieważ zdublowanie indeksów w żaden sposób nie wpływa na przyspieszenie wykonania zapytania, natomiast wydłuża czas potrzebny na modyfikowanie poindeksowanych wartości, należy unikać takich sytuacji.

Możliwe jest również wymuszanie użycia indeksu, nawet jeżeli optymalny plan opracowany przez optymalizator go nie uwzględnił. Z możliwość tej należy jednak korzystać ostrożnie, zawsze sprawdzając koszt zrealizowania różnych planów wykonania instrukcji. Poniższy przykład pokazuje, jak wymusić użycie indeksu powiązanego z kolumną CustomerID tabeli Orders:

```
USE Northwind
SELECT MAX (OrderDate)
FROM dbo.Orders (INDEX = CustomerID)
WHERE CustomerID = 'FRANK'
GO
StmtText
------------------------------------------------------------------------------
|--Stream Aggregate(DEFINE:([Expr1002]=MAX([Orders].[OrderDate])))
|--Bookmark Lookup(BOOKMARK:([Bmk1000]), OBJECT:([Northwind].[dbo].[Orders]))
|--Index Seek(OBJECT:([Northwind].[dbo].[Orders].[CustomerID]),
ªSEEK:([Orders].[CustomerID]='FRANK') ORDERED FORWARD)
(3 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 48, physical reads 1, read-ahead reads 15.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 21 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Natomiast opracowany przez optymalizator plan wykonania instrukcji wyglądał następująco:

```
USE Northwind
SELECT MAX (OrderDate)
FROM dbo.Orders
WHERE CustomerID = 'FRANK'
```

```
GO
StmtText
------------------------------------------------------------------------------
|--Stream Aggregate(DEFINE:([Expr1002]=MAX([Orders].[OrderDate])))
|--Top(1)
|--Filter(WHERE:(([Orders].[OrderDate] IS NOT NULL) AND
[Orders].[CustomerID]='FRANK'))
|--Bookmark Lookup(BOOKMARK:([Bmk1000]),
OBJECT:([Northwind].[dbo].[Orders]))
|--Index Scan(OBJECT:([Northwind].[dbo].[Orders].[OrderDate]),
ORDERED BACKWARD)
Table 'Orders'. Scan count 1, logical reads 213, physical reads 1, read-ahead reads 5.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 19 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Instrukcja DROP INDEX

Wykonanie instrukcji spowoduje usunięcie określonego indeksu z bazy danych. Przestrzeń zajmowana przez indeks zostaje zwrócona systemowi. Indeksy utworzone automatycznie dla kolumn podczas nadawania im zawężeń PRIMARY KEY oraz UNIQUE nie mogą zostać w ten sposób usunięte — należy je usunąć, wykonując odpowiednią instrukcję ALTER TABLE.

Usunięcie tabeli spowoduje automatyczne usunięcie powiązanych z nią indeksów. Niemożliwe jest usunięcie indeksów powiązanych z tabelami systemowymi.

Składnia:

```
DROP INDEX 'tabela.indeks' [,…n]
```

Przykład:

```
USE Northwind
DROP INDEX dbo.Orders.EmployeesOrders
GO
The command(s) completed successfully.
```

Statystyki

W kilku miejscach książki (m.in. w punkcie „SARG” rozdziału 4. i w bieżącym rozdziale) został opisany pewien szczególny aspekt pracy optymalizatora — wybór optymalnego planu wykonania instrukcji. Niektóre z opisywanych przykładów pokazywały, że w zależności od liczby i typu zapisanych w tabelach danych, optymalizator może zdecydować się np. na użycie indeksu niegrupującego albo na odczytanie całej tabeli.

Skąd optymalizator pobiera informacje o danych bez konieczności wykonania zapytania? Właśnie ze statystyk indeksu.

Statystyki przechowują w kolumnie statblob typu image tabeli sysindexes określoną próbkę danych zapisanych w poindeksowanych kolumnach.

SQL Server umożliwia również utworzenie statystyk dla kolumn niepowiązanych z żadnym indeksem. W rzeczywistości, jeżeli tylko włączona jest (co jest ustawieniem domyślnym) opcja auto create statistics, SQL automatycznie utworzy statystyki dla kolumn, do których nastąpiło odwołanie w klauzuli WHERE. Poniższy przykład pokazuje, jak sprawdzić bieżące ustawienie opcji dla wybranej bazy danych:

```
sp_dboption 'Northwind', 'auto create statistics'
GO
OptionName CurrentSetting
---------------------------------- --------------
auto create statistics ON
```

Możemy również poznać dokładniejsze informacje o statystykach utworzonych dla wybranej tabeli:

```
USE Northwind
EXEC sp_autostats [Order Details]
GO
Global statistics settings for [Northwind]:
Automatic update statistics: ON
Automatic create statistics: ON
Settings for table [Order Details]
Index Name AUTOSTATS Last Updated
-------------------------------- ----------- ------------------------------
[PK_Order_Details] ON 2002-11-11 17:02:24.347
[OrderID] ON 2002-11-11 17:02:24.347
[OrdersOrder_Details] ON 2002-11-11 17:02:24.357
[ProductID] ON 2002-11-11 17:02:24.357
[ProductsOrder_Details] ON 2002-11-11 17:02:24.357
(5 row(s) affected)
```

Natomiast, aby wyświetlić informacje o danych przechowywanych w poszczególnych statystykach, należy wykonać instrukcję DBCC SHOW_STATISTICS.

```
USE Northwind
DBCC SHOW_STATISTICS (Orders, ShippedDate)
GO
Statistics for INDEX 'ShippedDate'.
Updated Rows Rows Sampled Steps Density Average key length
------------------- ---------- ------------- ------ ------------------ --------------
```

```
Nov 13 2002 9:00AM 830 830 199 1.5124327E-3 11.79759
(1 row(s) affected)
All density Average Length Columns
---------------------- ------------------------ -----------------
2.5773195E-3 7.7975903 ShippedDate
1.2048193E-3 11.79759 ShippedDate, OrderID
(2 row(s) affected)
RANGE_HI_KEY RANGE_ROWS EQ_ROWS DISTINCT_RANGE_ROWS AVG_RANGE_ROWS
--------------------------- ------------ -------- -------------------- ----------------
1996-07-10 00:00:00.000 0.0 1.0 0 0.0
1996-07-15 00:00:00.000 2.0 2.0 2 1.0
.
.
.
(199 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
```

Instrukcja CREATE STATISTICS

Możliwe jest samodzielne utworzenie statystyki dla wybranych kolumn. Chociaż wydawałoby się, że tworzenie statystyki dla niepowiązanej z żadnym indeksem kolumny jest bezcelowe („Przecież nawet kiedy optymalizator prawidłowo oceni, że podane kryteria spełnia tylko kilka wierszy, to i tak nie ma indeksu, który mógłby wykorzystać do ich wybrania.”), to informacje odczytane z takich statystyk mogą być wykorzystywane podczas łączenia tabel.

```
CREATE STATISTICS statystyka
ON {tabela (kolumna [,…n])
[WITH
[[FULLSCAN
| SAMPLE liczba {PERCENT | ROWS}] [,]]
[NORECOMPUTE]
]
```

gdzie:
FULLSCAN oznacza utworzenie statystyki na podstawie wszystkich danych przechowywanych w wybranych kolumnach,
SAMPLE pozwala określić procentową lub bezwzględną liczbę danych, na podstawie których zostanie utworzona statystyka,
NORECOMPUTE powoduje nieaktualizowanie danych statystyki i jest zalecane wyłącznie w przypadku kolumn zawierających wyłącznie odczytywane dane.
Przykład:

```
USE wydawnictwo
CREATE STATISTICS s_k_e
ON dbo.klient (e_mail)
WITH SAMPLE 75 PERCENT
GO
The command(s) completed successfully.
```

Instrukcja UPDATE STATISTICS

Jeżeli ustawiona jest opcja bazy danych auto update statistics, SQL Server automatycznie, w zależności od liczby zmian danych, aktualizuje powiązane z tabelą statystyki. Możliwa jest jednak ręczna aktualizacja wybranych statystyk.
Składnia:

```
UPDATE STATISTICS tabela
[
indeks
| (statystyka [,…n])
]
[ WITH
[
[FULLSCAN]
| SAMPLE liczba {PERCENT | ROWS}]
| RESAMPLE
]
[[,] [ALL | COLUMNS | INDEX]
[[,] NORECOMPUTE]
]
```

gdzie:
RESAMPLE wymusza pełne odczytanie wszystkich powiązanych ze statystyką danych, w tym danych źródłowych pozostałych statystyk powiązanych z tabelą.
Przykład:

```
USE wydawnictwo
UPDATE STATISTICS klient
WITH FULLSCAN
GO
The command(s) completed successfully.
```

Instrukcja DROP STATISTICS

W szczególnych okolicznościach należy, wykonując instrukcję DROP STATISTICS, usunąć nieużywane statystyki.

Fakt, że SQL Server utworzył statystykę świadczy o tym, że była ona niezbędna do oszacowania liczby lub rodzaju danych.

Składnia:

```
DROP STATISTICS tabela.statystyka [,…n]
```

Przykład:

```
USE wydawnictwo
DROP STATISTICS dbo.klient.s_k_e
GO
The command(s) completed successfully.
```

Kurs Transact-SQL, część 11 (Widoki)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Widok jest predefiniowanym, zapisanym po stronie serwera zapytaniem, którego wynik może być wielokrotnie odczytywany.

Widoki wykorzystywane są, aby:

- ♣ Ułatwić odczytywanie danych użytkownikom. Na przykład, jeżeli widok pobiera dane z siedmiu tabel bazowych, odczytanie danych poprzez widok będzie wymagało wykonania prostej instrukcji SELECT.
- ♣ Ułatwić odczytywanie użytkownikom danych obliczonych na podstawie zapisanych w tabelach informacji. Na przykład odczytanie widoku obliczającego średnią liczbę sprzedanych towarów i grupującego wyniki według nazw kategorii produktów będzie wymagało wykonania prostej instrukcji SELECT.
- ♣ Ograniczyć, ze względów bezpieczeństwa, dostęp użytkowników do poufnych danych. Na przykład, jeżeli widok pobiera dane jedynie z wybranych kolumn tabeli, przez ten widok możliwe będzie odczytanie jedynie wybranych danych (np. imienia i nazwiska, ale nie pensji pracowników).
- ♣ Ukryć przed użytkownikami strukturę tabel bazy danych. Informacja ta nie jest niezbędna użytkownikom do pracy z bazą, a przez niektóre osoby może zostać wykorzystana w niewłaściwy sposób.
- ♣ Ułatwić zarządzanie uprawnieniami użytkowników.

Zagadnieniu bezpieczeństwa baz danych poświęcony jest rozdział 16.

Instrukcja CREATE VIEW

Wykonanie instrukcji spowoduje utworzenie wirtualnej tabeli lub widoku pobierających dane z określonych tabel lub widoków. Tabele bazowe muszą zostać utworzone przed utworzeniem widoku.

Domyślnie widoki mogą tworzyć członkowie ról sysadmin, db_ddladmin oraz właściciele bazy danych (członkowie roli db_owner). Aby utworzyć widok, trzeba mieć dodatkowo nadane uprawnienia do odczytywania tabel bazowych.

Instrukcja CREATE VIEW nie może być wykonana w ramach jednego polecenia wsadowego z innymi instrukcjami języka Transact-SQL.

Składnia:

```
CREATE [<właściciel>] VIEW widok [(kolumna [,…n])]
[WITH <atrybut> [,…n]]
AS
instrukcja_SELECT
[WITH CHECK OPTION]
<atrybut> ::=
{ENCRYPTION | SCHEMABINDING | VIEW_METADATA}
```

gdzie:

instrukcja_SELECT jest poprawną instrukcją języka Transact-SQL SELECT, z kilkoma dodatkowymi ograniczeniami:

- ♣ Instrukcja nie może zawierać klauzuli COMPUTE lub COMPUTE BY.
- ♣ Instrukcja nie może zawierać klauzuli ORDER BY, chyba że w klauzuli SELECT zostało wymienione słowo kluczowe TOP.
- ♣ Instrukcja nie może zawierać klauzuli INTO.
- ♣ Instrukcja nie może pobierać danych z tabel tymczasowych.
- ♣ Instrukcja może pobierać dane z maksymalnie 1024 kolumn.

WITH CHECK OPTION uniemożliwi takie zmodyfikowanie danych, które spowodowałby usunięcie wiersza z widoku. Na przykład, jeżeli widok zawiera informacje o towarach, których cena nie przekracza 500 zł, zmiana ceny towaru przez widok na 505 zł będzie niemożliwa,
ENCRYPTION powoduje zapisanie definicji widoku w postaci zaszyfrowanej. Zaszyfrowane widoki nie mogą być replikowane,
SCHEMABINDING powoduje powiązanie definicji widoku ze schematem bazy danych, w rezultacie czego niemożliwe będzie usunięcie lub takie zmodyfikowanie obiektów bazowych (tabel lub widoków), które wpłynęłyby na utworzony widok,
VIEW_METADATA powoduje, że klientom łączącym się poprzez interfejs OLE DB zwracane będą metainformacje o widoku, zamiast metainformacji o obiektach bazowych.

Tworzymy widok zawierający ogólnodostępne dane o wybranych autorach:

```
USE pubs
GO
CREATE VIEW dbo.w_autor (nazwisko,imie)
WITH ENCRYPTION
AS
SELECT au_lname, au_fname
FROM dbo.authors
WHERE contract = 1
GO
The command(s) completed successfully.
SELECT *
FROM w_autor
WHERE nazwisko LIKE 'Ring%'
GO
nazwisko imie
---------------------------------------- --------------------
Ringer Albert
Ringer Anne
(2 row(s) affected)
```

Jeżeli w definicji widoku użyto symbolu wieloznacznego (*) w klauzuli SELECT, a następnie dodano kolumny do definicji tabeli bazowej, nowe kolumny nie zostaną powiązane z wcześniej utworzonym widokiem.

Tworzymy widok zawierający wszystkie dane o autorach mieszkających w wybranych stanach. Użycie klauzuli WITH CHECK OPTION uniemożliwi wykonanie pierwszej instrukcji UPDATE:

```
USE pubs
GO
CREATE VIEW dbo.w_sasiedzi
AS
SELECT *
FROM dbo.authors
```

```
WHERE state IN ('MI', 'MD')
WITH CHECK OPTION
GO
The command(s) completed successfully.
UPDATE w_sasiedzi
SET state = 'CA'
GO
Server: Msg 550, Level 16, State 1, Line 1
The attempted insert or update failed because the target view either specifies WITH
aCHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows
aresulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.
UPDATE w_sasiedzi
SET City = 'SOUTH PARK'
GO
(2 row(s) affected)
```

Tworzymy widok zawierający obliczoną wartość:

```
USE Northwind
GO
CREATE VIEW dbo.w_produkt (identyfikator, nazwa, cena, vat)
WITH SCHEMABINDING
AS SELECT ProductID, ProductName, UnitPrice, UnitPrice*0.22
FROM dbo.Products
GO
The command(s) completed successfully.
USE Northwind
GO
SELECT *
FROM w_produkt
WHERE vat > 50
GO
identyfikator nazwa cena vat
------------ ---------------------------------------- --------------------- ---------
38 Côte de Blaye 263.5000 57.970000
(1 row(s) affected)
```

Należy dbać o to, aby wszystkie obiekty bazy danych, w szczególności obiekty bazowe i pobierające z nich dane widoki, były własnością tego samego użytkownika (z reguły jest to użytkownik dbo). Uprości to i przyspieszy sprawdzanie uprawnień poszczególnych użytkowników do danych prezentowanych przez widok.

Złączenie zewnętrzne w definicji widoków

Tworząc widoki pobierające dane z kilku obiektów bazowych, należy unikać łączenia tych obiektów poprzez jakiekolwiek złączenie zewnętrzne. Chociaż poniższa instrukcja jest poprawną instrukcją jeżyka Transact-SQL, to odczytanie danych poprzez widok może mieć nieoczekiwane rezultaty:

```
USE pubs
GO
CREATE VIEW dbo.w_autor_k
AS SELECT au_lname, COUNT(title_id) AS [liczba tytułów]
FROM dbo.authors AS a LEFT JOIN dbo.titleauthor ta ON a.au_id = ta.au_id
GROUP BY a.au_lname
GO
```

Gdyby w tabelach bazowych znajdowały się rekordy zawierające wartości nieokreślone w kolumnach, z których widok pobiera dane, wyświetlenie poprzez widok wszystkich danych, w których wartością kolumny pobieranej z wewnętrznej tabeli złączenia jest NULL, spowodowałby przypisanie wartości nieokreślonej do wszystkich wierszy wewnętrznej tabeli.

Uporządkowywanie danych poprzez widoki

Widoki są często najprostszym sposobem na „poprawę" bazy danych, której projekt nie spełnia wymogów 1PN. Z reguły administrator wykorzystuje wcześniej i przez kogoś innego utworzoną bazę danych, a polityka firmy nie pozwala na gruntowną przebudowę struktury produkcyjnej bazy danych.

Ponieważ instrukcja SELECT umożliwia, za pomocą operatora teoriomnogościowego UNION, łączyć wyniki wielu zapytań, możemy, bez konieczności tworzenia niewydajnych kursorów po stronie serwera, „uporządkować" tabelę znajdującą się w pierwszej postaci anormalnej. Ponadto, za pomocą prostej „sztuczki" polegającej na dodaniu klauzuli TOP 100 PERCENT, możemy posortować dane widoku.

Załóżmy, że zastaliśmy poniższą tabelę w bazie danych:

```
kursant {imie, kurs_stopnia_1, kurs_stopnia_2, kurs_stopnia_3}
```

Pomijając wszystkie opisane w rozdziale 7. niedogodności związane z tym, że tabela nie spełnia wymogów 1PN, pobieranie i prezentowanie użytkownikom danych z tej tabeli rodzi dodatkowe trudności. Proszę sobie wyobrazić kod programu, który na podstawie nazwy kursu (dowolnego stopnia) wyświetli dodatkowe informacje o kursie, takie jak data rozpoczęcia czy czas trwania kursu. A teraz proszę sobie wyobrazić ten sam kod, przy założeniu, że w tabeli kursant zapisane są informacje o kursach 15 różnych stopni.

Rozwiązaniem części naszych problemów będzie utworzenie poniższego widoku:

```
CREATE VIEW w_kursant
AS SELECT TOP 100 PERCENT *
FROM (
SELECT kurs_stopnia_1 as nazwa_kursu
FROM kursant
UNION
SELECT kurs_stopnia_2 as nazwa_kursu
FROM kursant
UNION
SELECT kurs_stopnia_3 as nazwa_kursu
FROM kursant
) n
ORDER BY n.nazwa_kursu
GO
```

W ten sposób nazwy wszystkich kursów można odczytać, odwołując się do jednej kolumny.

Opcje widoków

Informacje o widoku, tak jak informacje o obiektach innych typów, przechowywane są w tworzących katalog bazy danych tabelach systemowych.
Nazwy wszystkich widoków istniejących w wybranej bazie danych odczytamy wykonując poniższe zapytanie:

```
USE pubs
SELECT LEFT(name,30)
FROM sysobjects
WHERE xtype = 'V'
GO
------------------------------
titleview
w_autor
w_sasiedzi
w_autor_k
syssegments
sysconstraints
(6 row(s) affected)
```

Dodatkowe informacje o widokach należących do aktualnie zalogowanego użytkownika poznamy, odczytując zawartość widoku systemowego INFORMATION_SCHEMA.VIEW_TABLE_USAGE:

```
USE pubs
SELECT *
FROM INFORMATION_SCHEMA.VIEW_TABLE_USAGE
GO
VIEW_CATALOG VIEW_SCHEMA VIEW_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
------------ ---------- --------------- --------------- ------------------ -------------
pubs dbo sysconstraints pubs dbo sysobjects
pubs dbo titleview pubs dbo authors
pubs dbo titleview pubs dbo titleauthor
pubs dbo titleview pubs dbo titles
pubs dbo w_autor pubs dbo authors
pubs dbo w_autor_k pubs dbo authors
pubs dbo w_autor_k pubs dbo titleauthor
pubs dbo w_sasiedzi pubs dbo authors
(8 row(s) affected)
```

Definicja widoków przechowywana jest w tabeli syscomments, z której pobiera dane m.in. widok systemowy INFORMATION_SCHEMA.VIEWS:

```
USE pubs
SELECT VIEW_DEFINITION
```

```
FROM INFORMATION_SCHEMA.VIEWS
WHERE TABLE_NAME = 'titleview '
GO
VIEW_DEFINITION
--------------------------------------------------------------------------------
CREATE VIEW titleview
AS
select title, au_ord, au_lname, price, ytd_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
AND titles.title_id = titleauthor.title_id
(1 row(s) affected)
```

Innym sposobem odczytania definicji widoku jest wywołanie procedury składowanej sp_helptext:

```
USE pubs
EXEC sp_helptext titleview
GO
Text
----------------------------------------------------------------------------------
CREATE VIEW titleview
AS
select title, au_ord, au_lname, price, ytd_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
AND titles.title_id = titleauthor.title_id
```

Informacje o nazwach poszczególnych kolumn zawierających dane pobierane przez widok przechowywane są w tabeli systemowej syscolumns, z której pobiera dane widok systemowy INFORMATION_SCHEMA.VIEW_COLUMN_USAGE. Poprzez widok możemy odczytać informacje wyłącznie o widokach należących do aktualnie zalogowanego użytkownika:

```
USE pubs
SELECT LEFT(VIEW_NAME,30), LEFT(TABLE_NAME,30), LEFT(COLUMN_NAME,30)
FROM INFORMATION_SCHEMA.VIEW_COLUMN_USAGE
GO
------------------------------ ------------------------------ ------------------------
titleview titleauthor au_id
titleview titleauthor title_id
titleview titleauthor au_ord
titleview authors au_id
titleview authors au_lname
titleview titles title_id
.
```

```
.
.
(31 row(s) affected)
```

SQL Server umożliwia również szybkie sprawdzenie wszystkich obiektów bazowych powiązanych z wybranym widokiem. Wywołanie procedury systemowej sp_depends z nazwą widoku jako parametrem spowoduje zwrócenie listy wszystkich obiektów bazowych wybranego widoku:

```
USE pubs
EXEC sp_depends titleview
GO
In the current database, the specified object references the following:
name type updated selected column
---------- ---------- --------- -------- --------------
dbo.titles user table no no ytd_sales
dbo.titles user table no no title
dbo.titles user table no no pub_id
dbo.titles user table no no price
dbo.authors user table no no au_id
dbo.authors user table no no au_lname
dbo.titles user table no no title_id
dbo.titleauthor user table no no au_id
dbo.titleauthor user table no no title_id
dbo.titleauthor user table no no au_ord
```

Modyfikowanie danych poprzez widoki

Ponieważ widok nie przechowuje kopii danych, ale jest jedynie „pryzmatem", poprzez który odczytujemy dane zapisane w tabelach, modyfikacja danych widoku jest w rzeczywistości modyfikacją danych zapisanych w tabelach.
Modyfikowanie danych jest możliwe, o ile:

- Widok jest powiązany z tabelami bazowymi związkiem jedno-jednoznacznym. Wynika z tego, że niemożliwe jest modyfikowanie danych poprzez widoki, w których wykorzystano jakąkolwiek funkcję grupującą.
- Dane zostaną jednocześnie zmodyfikowane w tylko jednej tabeli bazowej. Nie oznacza to, że widok musi pobierać dane z tylko jednej tabeli bazowej, ale że jednocześnie mogą być zmodyfikowane wartości kolumn jednej tabeli.
- Modyfikowane dane nie są wartościami obliczonymi na podstawie wartości innych kolumn. Generalnie SQL Server nie pozwala na programową zmianę takich danych.
- Nie narusza zawężeń zdefiniowanych dla tabeli bazowej. Na przykład, jeżeli widok nie pobiera danych z tabeli, na którą nałożono zawężenie NOT NULL, wstawianie nowych wierszy poprzez widok będzie niemożliwe.
- Jest zgodne z opcjonalnym, opisanym wcześniej warunkiem WITH CHECK OPTION.

Instrukcja ALTER VIEW

Zmienić definicję widoku możemy poprzez wykonanie instrukcji ALTER VIEW. Bezpośrednie modyfikowanie zawartości kolumny text tabeli syscomments jest niemożliwe.
Składnia:

```
ALTER VIEW widok [(kolumna [,…n])]
```

[WITH <atrybut> [,…n]]

AS

instrukcja_SELECT

[WITH CHECK OPTION]

<atrybut> ::=

{ENCRYPTION | SCHEMABINDING | VIEW_METADATA}

Przykład:

USE pubs GO ALTER VIEW w_sasiedzi AS SELECT * FROM dbo.authors WHERE au_lname LIKE 'G%' WITH CHECK OPTION GO The command(s) completed successfully.

Instrukcja DROP VIEW

Niepotrzebne widoki mogą zostać usunięte z bazy danych. Ponieważ usunięcie tabeli bazowej nie spowoduje automatycznego usunięcia powiązanych z nią widoków, z reguły instrukcje DROP TABLE i DROP VIEW wykonywane są łącznie.

Składnia:

DROP VIEW {widok} [,…n]

Przykład:

USE pubs

GO

DROP VIEW w_sasiedzi, w_autor

GO

The command(s) completed successfully.

Kurs Transact-SQL, część 12 (Procedury składowane)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Procedury składowane są zbiorami instrukcji języka Transact-SQL zapisanymi pod wspólną nazwą i wywoływanymi jak pojedyncza instrukcja. Domyślnie jedynie członkowie ról sysadmin, db_owner oraz db_ ddladmin mogą tworzyć nowe procedury składowane.

SQL Server posiada zbiór predefiniowanych procedur składowanych zapisanych albo w bazie master (nazwy tych procedur rozpoczynają się od prefiksu sp_, a same procedury służą do pobierania i konfigurowania opcji serwera i baz danych), albo jako pliki bibliotek dynamicznych dll (nazwy tych procedur rozpoczynają się od prefiksu xp_).

Ponadto możliwe jest tworzenie własnych procedur składowanych przechowywanych w bazie użytkownika oraz tymczasowych procedur składowanych (nazwy tych procedur rozpoczynają się pojedynczym lub podwójnym znakiem #).

Procedury składowane umożliwiają:

- Przekazywanie parametrów wywołania.
- Wykonywanie prawie wszystkich instrukcji języka Transact-SQL, w tym wywoływania innych procedur składowanych.
- Zwracanie dowolnej liczby wyników do programu, który wywołał procedurę.
- Zwracanie informacji o udanej lub niewykonanej procedurze.

Ze względu na swoją uniwersalność procedury składowane są najczęściej wykorzystywanym przez programistów bazy danych typem obiektów.

Procedury składowane są powszechnie wykorzystywane w celu:

- Implementacji reguł logiki biznesowej.
- Zabezpieczenia obiektów bazy danych przed bezpośrednim dostępem użytkowników.
- Poprawy wydajności często wykonywanych instrukcji.
- Zminimalizowania obciążenia sieci (zamiast wysyłać całe instrukcje języka transact-sql użytkownik wywołuje jedynie procedurę, wysyłając jej nazwę i przekazując parametry jej wywołania).
- Automatycznego wykonania pewnych operacji podczas uruchamiania sql servera.

Przetwarzanie procedur przez SQL Server

Przetwarzanie procedury składowanej przez SQL Server rozpoczyna się w momencie jej utworzenia, poprzez jej optymalizację i kompilację, aż po jej pierwsze wykonanie. Skompilowany plan wykonania procedury składowanej przechowywany jest w buforze procedury (ang. Procedure cache) i jest wykorzystywany do ponownego wykonania tej samej procedury.

Tworzenie

Po sprawdzeniu poprawności syntaktycznej nowo utworzonej procedury jej ciało zostaje zapisane w tabeli systemowej syscomments, a jej nazwa w tabeli sysobjects. Ponieważ na tym etapie nie jest sprawdzana poprawność semantyczna, możliwe jest utworzenie procedur odwołujących się do nieistniejących jeszcze obiektów.

Wykonanie

Pierwsze wywołanie procedury przez użytkownika powoduje przygotowanie planu jej wykonania i skompilowanie wybranego przez optymalizator planu. W ramach tego etapu następuje:

1. Sprawdzenie poprawności semantycznej. Jeżeli procedura odwołuje się do nieistniejących lub niedostępnych obiektów, jej wykonanie zostaje na tym etapie przerwane.
2. Optymalizacja planu wykonania procedury. Na podstawie statystyk optymalizator analizuje liczbę pobieranych danych, na podstawie dostępnych metainformacji — strukturę tabel źródłowych, a na podstawie kodu ciała procedury — metody łączenia i wybierania danych. Wszystkie te dane są wykorzystane do przygotowania planu wykonania procedury.
3. Kompilacja wybranego planu wykonania. Parametrami wpływającymi na wybór planu wykonania są również: wielkość skompilowanego kodu oraz czas potrzebny na kompilację procedury.

Ponieważ raz skompilowany plan wykonania może ulec dezaktualizacji, SQL Server ponownie wykona wszystkie trzy powyższe operacje, jeżeli:

- ♣ Zmieni się struktura obiektów źródłowych.
- ♣ Dostępne będą nowe statystyki powiązane z obiektami źródłowymi.
- ♣ Zostanie dodany indeks powiązany z dowolnym obiektem źródłowym.
- ♣ Zasadniczo zmieni się liczba lub charakter danych przechowywanych w tabelach źródłowych.

Wywołanie

Kolejne wywołanie procedury składowanej dokonane w ramach tego samego środowiska użytkownika sprowadza się do odczytania z bufora skompilowanego planu wykonania procedury i wykonania go. O wywołaniu w ramach tego samego środowiska użytkownika mówimy, jeśli:

1. Nie zmieniły się: nazwa serwera, nazwa bazy danych oraz parametry przekazane podczas nawiązania połączenia klienta z bazą danych (nie tylko parametry obiektu CONNECTION, ale również parametr sesji użytkownika QUOTED_IDENTIFIER oraz niektóre ze zbioru parametrów ANSI_DEFAULTS).
2. Odwołanie się procedury do obiektów bazy danych nie wymaga rozwiązywania nazw. Rozwiązywanie nazw przeprowadzane jest wtedy, gdy obiekt o podanej nazwie jest własnością innego użytkownika, a odwołanie do niego nie odbywa się poprzez podanie zarówno nazwy użytkownika, jak i nazwy obiektu.

Plan wykonania procedury przygotowany przez SQL Server w rzeczywistości składa się z dwóch komponentów: planu wykonania procedury oraz kontekstu jej wykonania. W buforze przechowywane są wszystkie unikalne kombinacje tych dwóch komponentów.

Konwencja nazewnicza procedur składowanych

Ponieważ nazwy wszystkich predefiniowanych procedur składowanych rozpoczynają się prefiksem sp_, wielu administratorów stosuje tę konwencję również przy tworzeniu własnych procedur składowanych. To powszechny błąd — ze względu na wydajność nie należy postępować w ten sposób, chyba że procedury użytkownika zostaną zapisane w bazie master (oczywiście, w przypadku odwoływania się do procedury za pomocą jej pełnej nazwy, opisany poniżej problem nie wystąpi).

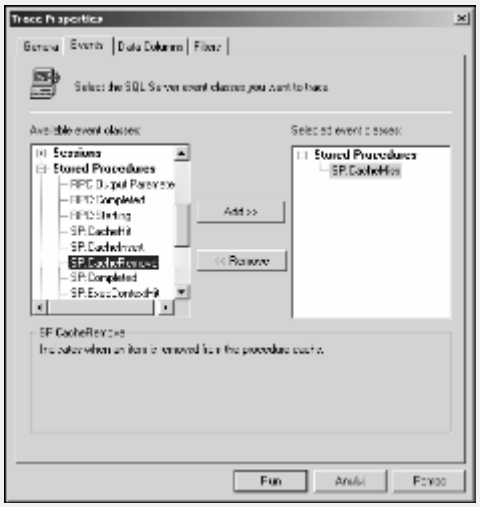
Aby przekonać się, w jaki sposób SQL Server przeprowadza rozwiązywanie nazw, utwórzmy dwie przykładowe procedury, których działanie sprowadza się do podniesienia liczby 2 do kwadratu:

```
USE tempdb
GO
CREATE PROCEDURE dbo.test AS
SELECT 2*2
GO
CREATE PROCEDURE dbo.sp_test AS
SELECT 2*2
GO
```

Wykonajmy każdą z procedur, aby SQL Server miał okazję skompilować plan ich wykonania:

```
USE tempdb
EXEC dbo.test
GO
EXEC dbo.sp_test
GO
-----------
4
(1 row(s) affected)
-----------
4
(1 row(s) affected)
```

Aby poznać różnice pomiędzy wykonaniem obu, różniących się jedynie nazwą, procedur, musimy uruchomić instalowany automatycznie z SQL Serverem program SQL Server Profiler. Następnie wybieramy tworzenie nowego pliku śledzenia (ang. New Trace), łączymy się z serwerem bazodanowym, na zakładce Event usuwamy wszystkie domyślnie dodane klasy zdarzeń i dodajemy zdarzenie klasy Stored Procedur Cache Miss (rysunek 12.1).



Rysunek 12.1. *Program SQL Server Profiler jest narzędziem do śledzenia bieżącej aktywności SQL Servera*

Uruchamiamy śledzenie, przełączamy się do programu Query Analyzer i wywołujemy procedurę dbo.test. Profiler nie zanotował żadnego zdarzenia. Następnie wywołujemy procedurę dbo.sp_test i ponownie sprawdzamy wynik śledzenia. Tym razem wystąpiło zdarzenie SP:CacheMiss. Ponieważ SQL Server domyślnie szuka planu wykonania procedur o nazwie rozpoczynającej się od prefiksu sp_ w buforze bazy master, założył, że nie istnieje plan jej wykonania i skompilował ponownie procedurę sp_test.

Instrukcja CREATE PROCEDURE

Wykonanie instrukcji CREATE PROC [EDURE] spowoduje dodanie nowej procedury składowanej.

```
CREATE PROC [EDURE] procedura [; numer]
[{@parametr typ_danych}
[VARYING] [= wartość_domyślna] [OUTPUT]
] [,...n]
[WITH
{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
```

```
AS instrukcja_SQL [...n]
```

gdzie:

numer jest opcjonalnym numerem określającym różne procedury o tej samej nazwie. Na przykład możliwe jest utworzenie procedur test;1 i test;2. Wszystkie procedury o tej samej nazwie mogą zostać usunięte jedną instrukcją DROP PROCEDURE test, *@parametr* jest nazwą przekazanego (i zadeklarowanego w ciele procedury) parametru. SQL Server pozwala na przekazanie do procedury maksymalnie 1024 parametrów, *typ_danych* jest typem danych parametru, *VARYING* pozwala na zwrócenie wyników w postaci zbioru danych. Parametr jest dostępny, jeśli przekazano do procedury parametr typu CURSOR, *wartość_domyślna* jest domyślną wartością parametru wywołania. Jeżeli została określona, możliwe będzie wywołanie procedury bez przekazania obowiązkowego parametru, *OUTPUT* określa, że parametr zwraca wynik działania procedury, *RECOMPILE* powoduje, że SQL Server nie będzie przechowywał planu wykonania procedury — za każdym razem będzie ona na nowo kompilowana, *ENCRYPTION* powoduje, że ciało procedury zostanie zapisane w postaci zaszyfrowanej, *FOR REPLICATION* określa, że procedura nie będzie mogła być wykonana na serwerach pełniących funkcję subskrybenta.

Tworząc procedury składowane, należy pamiętać, że:

- ♣ Możliwe jest utworzenie procedury składowanej wyłącznie w bieżącej bazie danych (wyjątkiem jest utworzenie procedury w bazie tempdb).
- ♣ Instrukcja CREATE PROCEDURE musi zostać oddzielona od pozostałych instrukcji języka Transact-SQL dyrektywą GO.
- ♣ W ciele procedury niedozwolone jest umieszczanie następujących instrukcji: CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER oraz CREATE VIEW. Wszystkie pozostałe instrukcje języka Transact-SQL są dopuszczalne.
- ♣ Tak jak w przypadku widoków, nie należy dopuszczać do sytuacji, w której właścicielem procedury jest inny użytkownik niż właściciel tabel źródłowych.
- ♣ Wielkość pojedynczej procedury nie może przekroczyć 128 MB.

Przykłady:

Tworzymy prostą procedurę pobierającą dane z tabeli Products:

```
USE Northwind
GO
CREATE PROCEDURE dbo.p_produkty
WITH ENCRYPTION
AS
SELECT Products.ProductName, Products.UnitPrice
FROM Products
WHERE Products.UnitPrice>(SELECT AVG(UnitPrice) From Products)
GO
The command(s) completed successfully.
```

Tworzymy prostą procedurę z dwoma parametrami. Jeżeli nie zostaną one podane, procedura zwróci informacje o wszystkich pracownikach:

```
USE Northwind
GO
CREATE PROCEDURE p_pracownicy
@nazwisko varchar(40) = '%',
@imie varchar(30) = '%'
AS
SELECT LastName, FirstName, Title, BirthDate, HireDate, HomePhone
FROM dbo.Employees
WHERE LastName LIKE @nazwisko
```

```
AND FirstName LIKE @imie
GO
The command(s) completed successfully.
```

Tworzymy procedurę przypisującą obliczoną liczbę produktów z podanej kategorii do zmiennej @@liczba:

```
USE Northwind
GO
CREATE PROCEDURE p_prod_kat @@kategoria varchar(40) = '%', @@liczba int OUTPUT
AS
SELECT @@liczba = COUNT (ProductID)
FROM dbo.Categories JOIN dbo.Products
ON dbo.Categories.CategoryID = dbo.Products.CategoryID
WHERE CategoryName LIKE @@kategoria
GO
The command(s) completed successfully.
```

Automatyczne uruchamianie procedur

Wywołując procedurę systemową sp_procoption, możemy wymusić automatyczne wykonywanie określonych procedur podczas uruchamiania SQL Servera. Jedynie procedury zapisane w bazie master i będące własnością użytkownika dbo mogą być automatycznie uruchamiane.
Składnia:

```
sp_procoption [@ProcName =] 'procedura'
, [@OptionName =] 'opcje'
, [@OptionValue =] 'wartość'
```

gdzie jedyną prawidłowa wartością parametru opcje jest startup.

Zagnieżdżanie procedur

Procedury składowane mogą być zagnieżdżane (jedna procedura może wywoływać inna procedurę). SQL Server pozwala na zagnieżdżanie procedur do 32 poziomu włącznie. Próba wywołania 33 procedury składowanej zakończy się zgłoszeniem błędu i wycofaniem wszystkich instrukcji.

> Procedury wywołane z poziomu innej procedury mogą pobierać i modyfikować dane z obiektów utworzonych przez procedurę wyższego poziomu zagnieżdżenia.

Bieżący poziom zagnieżdżenia zwraca funkcja systemowa @@nestlevel.

Opcje procedur składowanych

Listę wszystkich procedur składowanych powiązanych z wybraną bazą danych możemy odczytać, wywołując procedurę systemową sp_stored_procedures:

```
USE pubs
EXEC sp_stored_procedures
```

```
GO
PROCEDURE_QUALIFIER PROCEDURE_OWNER PROCEDURE_NAME NUM_INPUT_PARAMS
ªNUM_OUTPUT_PARAMS NUM_RESULT_SETS REMARKS PROCEDURE_TYPE
------------------ --------------- --------------------- --------------------------
ª------------------ ---------------- ------------ ----------------
pubs dbo byroyalty;1 -1
ª-1 -1 NULL 2
pubs dbo reptq1;1 -1
ª-1 -1 NULL 2
pubs dbo reptq2;1 -1
ª-1 -1 NULL 2
pubs dbo reptq3;1 -1
ª-1 -1 NULL 2
pubs dbo titles_sum;1 -1
ª-1 -1 NULL 2
(5 row(s) affected)
```

Listę wszystkich obiektów powiązanych z daną procedurą składowaną poznamy, wywołując procedurę systemową sp_depends z parametrem określającym nazwę procedury systemowej:

```
USE Northwind
EXEC sp_depends p_produkty
GO
In the current database, the specified object references the following:
name type updated selected column
-------------- ----------------- ------------------------------------------------------
dbo.Products user table no no ProductName
dbo.Products user table no no UnitPrice
```

Wykonywanie procedur składowanych

Procedura składowana może zostać wykonana niezależnie albo jako klauzula instrukcji INSERT. Wywołanie procedury polega na wykonaniu dyrektywy EXEC [UTE], po której następuje nazwa procedury i ewentualnie lista parametrów.

> Jeżeli wywołanie procedury jest jedyną instrukcją wydzieloną przez dyrektywy GO, można pominąć słowo kluczowe EXEC. W innych wypadkach jego opuszczenie spowoduje wywołanie błędu.

Aby wywołać procedurę p_produkty, napiszemy:

```
EXEC Northwind.dbo.p_produkty
```

Aby natomiast zapisać wynik procedury do tabeli, należy wywołać ją jako klauzulę instrukcji INSERT. Na przykład, aby dodać do tabeli Employees dane zwrócone przez procedurę p_pracownicy wywołaną z parametrami '%', 'M%', napiszemy:

```
USE Northwind
INSERT INTO dbo.Employees (LastName, FirstName, Title, BirthDate, HireDate, HomePhone)
EXEC Northwind.dbo.p_pracownicy '%', 'M%'
GO
```

```
(2 row(s) affected)
```

Możliwe jest również posługiwanie się nazwami parametrów, zamiast ich pozycją. Poniższe wywołanie procedury jest funkcjonalnym odpowiednikiem poprzedniej instrukcji:

```
USE Northwind
INSERT INTO dbo.Employees (LastName, FirstName, Title, BirthDate, HireDate, HomePhone)
EXEC Northwind.dbo.p_pracownicy
@nazwisko = '%',
@imie = 'M%'
GO
```

Wywołanie procedury zwracającej obliczone wartości wymaga wcześniejszego zadeklarowania zmiennych odpowiedniego typu:

```
USE Northwind
DECLARE @@liczba1 int
DECLARE @@liczba2 int
EXEC p_prod_kat 'Grains/Cereals', @@liczba1 OUTPUT
EXEC p_prod_kat 'Seafood', @@liczba2 OUTPUT WITH RECOMPILE
SELECT @@liczba1, @@liczba2, @@liczba1 - @@liczba2
GO
----------- ----------- -----------
7 12 -5
(1 row(s) affected)
```

Klauzula WITH RECOMPILE wymieniona przy wywołaniu procedury wymusza jej ponowne skompilowanie według na nowo utworzonego planu wykonania i stosowana jest, jeżeli wartości parametrów wywołania procedury w dużym stopniu różnią się od typowych wartości.

Instrukcja ALTER PROCEDURE

Zmodyfikować istniejącą procedurę mogą jedynie członkowie ról sysadmin, db_owner oraz db_ ddladmin — niemożliwe jest nadanie uprawnienia do wykonania tej instrukcji innym użytkownikom.
Składnia:

```
ALTER PROC [EDURE] procedura [; numer]
[{@parametr typ_danych}
[VARYING] [= wartość_domyślna] [OUTPUT]
] [,…n]
[WITH
{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
AS instrukcja_SQL […n]
```

Przykład:

```
USE Northwind
GO
ALTER PROCEDURE dbo.p_produkty
WITH ENCRYPTION
AS
SELECT ProductName, UnitPrice
FROM Products
WHERE Products.UnitPrice>(SELECT AVG(UnitPrice) From Products)
AND Products.UnitPrice<(SELECT MAX(UnitPrice) From Products)
GO
The command(s) completed successfully.
```

Instrukcja DROP PROCEDURE

Usunąć nieużywane procedury składowane możemy poprzez instrukcję DROP PROCEDURE.
Składnia:

```
DROP PROCEDURE {procedura} [,…n]
```

Kurs Transact-SQL, część 13 (Wyzwalacze)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Wyzwalacze są specjalnym typem procedur składowanych powiązanych z wybranymi tabelami i wywoływanych wykonaniem instrukcji języka Transact-SQL INSERT, UPDATE albo DELETE.

> Niemożliwe jest bezpośrednie wywołanie wyzwalacza za pomocą dyrektywy EXECUTE.

Instrukcje wykonane w ramach ciała wyzwalacza traktowane są jako fragment transakcji jawnie lub niejawnie rozpoczętej przez użytkownika, który odwołał się do danych przechowywanych w powiązanej z wyzwalaczem tabeli. Wynika z tego, że wyzwalacz może zatwierdzić (wykonując instrukcję COMMIT TRANSACTION) lub wycofać (instrukcją ROLLBACK TRANSACTION) zmiany wprowadzone przez użytkownika.

Podstawowym zastosowaniem wyzwalaczy jest wymuszenie integralności danych, zwłaszcza ich zgodności z regułami logiki biznesowej. Wyzwalacze umożliwiają m.in.:

1. Kaskadowe aktualizowanie danych w powiązanych tabelach.
2. Sprawdzanie poprawności danych na podstawie wartości przechowywanych w dowolnych tabelach (w przeciwieństwie do zawężenia CHECK, za pomocą którego możemy odwołać się jedynie do bieżącej tabeli).
3. Jednoczesne sprawdzanie danych zmodyfikowanych w dowolnej liczbie wierszy tabeli.
4. Wywoływanie predefiniowanych lub zdefiniowanych przez użytkownika komunikatów błędu.
5. Monitorowanie aktywności użytkowników.
6. Modyfikacje danych w bazach niespełniających wymogów trzeciej postaci normalnej. W bazach tego typu prawdopodobnie przechowywane są informacje nadmiarowe (redundantne) i modyfikacja np. numeru telefonu w jednej tabeli może wiązać się z koniecznością zmiany tego numeru w innych tabelach.

Wyzwalacze a zawężenia

Obiekty obu tych typów w pewnym podstawowym zakresie mogą pełnić tę samą funkcję — wymuszenia integralności przechowywanych danych. Jednak zakres i typ przeprowadzanych przez nie akcji są różne:

1. Wyzwalacze, w przeciwieństwie do zawężeń, wywoływane są w odpowiedzi na akcje użytkownika. Wynika z tego, że dopiero po wykonaniu instrukcji wyzwalany jest wyzwalacz, natomiast warunki zawężenia sprawdzane są przed wykonaniem instrukcji języka Transact-SQL.
2. Konsekwencją poprzedniego punktu jest kolejność, w jakiej wywoływane są wyzwalacze i zawężenia — najpierw sprawdzane są warunki zdefiniowane w zawężeniach, a po ich pomyślnym sprawdzeniu wywoływany jest wyzwalacz.
3. Wyłącznie właściciel tabeli może utworzyć powiązany z nią wyzwalacz. Uprawnienie do tworzenia wyzwalaczy nie może zostać nikomu udzielone czy przekazane.
4. Wyzwalacze nie mogą zostać powiązane z widokami oraz tabelami tymczasowymi.
5. Wyzwalacze mogą przetwarzać jednocześnie wiele wierszy tabeli. Możliwe jest również warunkowe przetwarzanie poszczególnych wierszy.

Natomiast, tak jak możliwe jest zdefiniowanie dla tabeli dowolnej liczby zawężeń (wyjątkiem jest zawężenie PRIMARY KEY), możliwe jest również utworzenie dowolnej liczby powiązanych z nią wyzwalaczy.

> Tym, co odróżnia wyzwalacze od procedur składowanych (oprócz sposobu ich wywoływania) jest fakt, że wyzwalacze z reguły nie zwracają żadnych danych.

Typy wyzwalaczy

SQL Server umożliwia tworzenie wyzwalaczy wywoływanych wykonaniem instrukcji INSERT, UPDATE oraz DELETE. Typ instrukcji wywołującej wyzwalacz jest podstawą wyodrębnienia trzech typów wyzwalaczy.

Wyzwalacze wywoływane wykonaniem instrukcji INSERT

Wywołanie wyzwalacza typu INSERT powoduje wstawienie nowego wiersza w tabeli źródłowej oraz w tabeli INSERTED — wirtualnej tabeli przechowującej dokładną kopię wstawionych wierszy. Rozwiązanie to pozwala na porównanie danych oryginalnych z danymi zmodyfikowanymi przez wyzwalacz.

Wyzwalacze wywoływane wykonaniem instrukcji DELETE

Wywołanie wyzwalacza typu DELETE powoduje zapisanie w wirtualnej tabeli DELETED kopii usuniętych wierszy z tabeli źródłowej. Ponieważ przed zapisaniem wierszy do tabeli DELETED zostały one usunięte z tabeli źródłowej, obie tabele nigdy nie zawierają wspólnych informacji. Tworząc wyzwalacz typu DELETE, pamiętaj, że obcięcie tabeli za pomocą nielogowalnej instrukcji TRUNCATE nie spowoduje jego wywołania.

Wyzwalacze wywoływane wykonaniem instrukcji UPDATE

Wykonanie instrukcji UPDATE można podzielić na dwa etapy: najpierw usuwana jest oryginalna zawartość określonych wierszy tabeli (wykonywana jest instrukcja DELETE), a następnie wstawiana jest ich nowa wartość (wykonywana jest instrukcja INSERT). Dlatego wywołanie wyzwalacza typu UPDATE powoduje przeniesienie do tabeli DELETED usuniętych wierszy z tabeli źródłowej, a następnie zapisanie zmodyfikowanych wartości w tabeli źródłowej i tabeli INSERTED. W rezultacie wyzwalacze tego typu mogą odwoływać się do obu wirtualnych tabel.

Tabele DELETED i INSERTED w celu zapewnienia maksymalnej wydajności bazy danych zawsze przechowywane są w buforze.

Klauzula IF UPDATE

Wywołanie wyzwalaczy typu UPDATE może zostać ograniczone do zmiany wartości przechowywanej w wybranych kolumnach tabeli źródłowej. W rezultacie jedynie zmiany określonych atrybutów obiektu spowodują wywołanie wyzwalacza.

Instrukcja CREATE TRIGGER

Wykonanie instrukcji spowoduje utworzenie nowego wyzwalacza. W ciele wyzwalacza niedopuszczalne jest:

- Tworzenie, modyfikowanie i usuwanie obiektów bazodanowych (używanie instrukcji języka Transact-SQL ALTER DATABASE, CREATE DATABASE, DROP DATABASE).
- Inicjowanie i modyfikowanie urządzeń pamięci masowych (używanie instrukcji DISK INIT, DISK RESIZE).
- Odczytywanie i odtwarzanie plików bazy danych (używanie instrukcji LOAD DATABASE, LOAD LOG, RESTORE DATABASE i RESTORE LOG).
- Rekonfigurowanie serwera bazodanowego (używanie instrukcji RECONFIGURE).
- Zarządzanie uprawnieniami użytkowników (używanie instrukcji GRANT, REVOKE i DENY).

Składnia:

```
CREATE TRIGGER wyzwalacz
ON {tabela}
[WITH ENCRYPTION]
{
{{FOR | AFTER | INSTEAD OF}
```

```
{[DELETE] [,] [INSERT] [,] [UPDATE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS
instrukcja_SQL […n]
}
|
{(FOR | AFTER | INSTEAD OF) {[INSERT] [,] [UPDATE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS
{IF UPDATE (kolumna)
[{AND | OR} UPDATE (kolumna)]
[…n]
| IF (COLUMNS_UPDATED () {operator_binarny} maska_uaktualnienia)
{operator_porównania} maska […n]
}
instrukcja_SQL […n]
}
}
```

gdzie:

AFTER (wartość domyślna) powoduje wywołanie wyzwalacza po pomyślnym zakończeniu wykonywania instrukcji użytkownika,
INSTEAD OF powoduje wykonanie wyzwalacza zamiast oryginalnej instrukcji użytkownika. W rezultacie wszystkie zmiany wprowadzone przez użytkownika zostaną zastąpione zmianami wprowadzonymi przez wyzwalacz,
[DELETE] [INSERT] [UPDATE] określają instrukcję języka Transact-SQL, której wykonanie wywoła wyzwalacz,
WITH APPEND jest opcją używaną wyłącznie w systemach SQL SERVER 6.5 i starszych,
NOT FOR REPLICATION powoduje, że wyzwalacz nie będzie wywoływany przez zmiany danych spowodowane ich replikacją,
IF UPDATE powoduje sprawdzanie wykonania instrukcji UPDATE lub INSERT,
Wykonanie instrukcji DELETE nie spowoduje wykonania żadnej akcji,
IF (COLUMNS_UPDATED()) jest funkcją zwracającą w postaci ciągu bitów informacje o tym, w których kolumnach zmodyfikowano lub dodano dane,
operator_binarny jest operatorem porównania wykorzystywanym do odczytania wyniku funkcji COLUMNS_UPDATED (). Lewy, najmniej znaczący bit wyniku, odpowiada pierwszej kolumnie tabeli,
maska_uaktualnienia określa, za pomocą liczby całkowitej, które kolumny będą sprawdzane przy pomocy funkcji COLUMNS_UPDATED(),
operator_porównania pozwala na sprawdzenie wszystkich (=) lub tylko wybranych (>), określonych poprzez maskę_uaktualnienia, kolumn,
maska określa, za pomocą liczby całkowitej, dla których kolumn modyfikacja lub dodawanie danych wywoła określoną akcję.

Przykłady:

Tworzymy wyzwalacz, który wyśle wiadomość e-mail do użytkownika kontroler1 przy jakiejkolwiek modyfikacji danych w tabeli Orders:

```
USE Northwind
GO
CREATE TRIGGER t_zamowienie
ON dbo.Orders
FOR INSERT, UPDATE, DELETE
AS
```

```
EXEC master..xp_sendmail 'kontroler1',
'Zmieniono zawartość tabeli Orders - proszę uaktualnić zestawienia'
GO
The command(s) completed successfully.
```

Przed wywołaniem procedury xp_sendmail należy skonfigurować usługę SQL Agent Mail. Konfiguracja tej usługi została opisana w rozdziale 17.

Tworzymy wyzwalacz sprawdzający, czy zmodyfikowano zawartość wybranych kolumn:

```
USE Northwind
GO
CREATE TRIGGER t_test ON dbo.[Order Details]
FOR UPDATE AS
IF (COLUMNS_UPDATED()& 14) > 0 --2^2 + 2^3 + 2^4
PRINT 'Zmodyfikowano zawartość kolumn 2,3 lub 4'
GO
UPDATE dbo.[Order Details]
SET ProductID = ProductID,
Quantity=Quantity
GO
Zmodyfikowano zawartość kolumn 2,3 lub 4
(2155 row(s) affected)
```

Wyzwalacze a monitorowanie aktywności użytkowników

Popularnym wykorzystaniem wyzwalaczy jest zapisywanie w bazie informacji o użytkowniku, który zmodyfikował dane w tabeli oraz o czasie przeprowadzenia tej modyfikacji.

W pierwszej kolejności należy zmodyfikować schemat tabeli, dodając do niej dwie kolumny, w których przechowywane będą informacje o użytkowniku i czasie zmiany:

```
USE Northwind
ALTER TABLE dbo.Categories
ADD
uzytkownik varchar(200),
czas datetime
GO
The command(s) completed successfully.
```

Następnie tworzymy wyzwalacz wywoływany wykonaniem instrukcji UPDATE modyfikującej jakiekolwiek dane w tabeli Categories (funkcja @@ROWCOUNT zwraca liczbę zmodyfikowanych wierszy) oraz upewniamy się, że ustawiania bazy Northwind pozwalają na rekursywne wywoływanie wyzwalaczy:

```
CREATE TRIGGER t_cat_audit
```

```
ON dbo.Categories AFTER UPDATE AS
BEGIN
IF (@@ROWCOUNT>0)
UPDATE dbo.Categories
SET uzytkownik = SYSTEM_USER + ' ' + user_name(),
czas = CURRENT_TIMESTAMP
FROM dbo.Categories C JOIN DELETED D
ON C.CategoryID = D.CategoryID
END
GO
sp_dbOption Northwind, 'recursive triggers', 'on'
GO
The command(s) completed successfully.
```

Jeżeli teraz spróbujemy zmodyfikować dane w tabeli Categories, przekonamy się (co można było wywnioskować ze sposobu wywoływania wyzwalacza), że wystąpił błąd — nowoutworzony wyzwalacz wywołany instrukcją UPDATE również zmodyfikował dane, więc nastąpiło jego kolejne wywołanie itd., aż do osiągnięcia maksymalnego poziomu zagnieżdżenia wyzwalaczy:

```
UPDATE dbo.Categories
SET Description = Description
WHERE CategoryID = 8
GO
Server: Msg 217, Level 16, State 1, Procedure t_cat_audit, Line 5
Maximum stored procedure, function or trigger nesting level exceeded (limit 32).
```

Najprostszym rozwiązaniem problemu wydaje się wyłączenie zagnieżdżania wyzwalaczy dla monitorowanej bazy danych — wystarczyłoby wykonanie poniższej instrukcji, aby nasz wyzwalacz zaczął prawidłowo działać:

```
sp_dbOption Northwind, 'recursive triggers','off'
GO
UPDATE dbo.Categories
SET Description = Description
WHERE CategoryID = 8
GO
sp_dboption command succeeded.
(1 row(s) affected)
(1 row(s) affected)
```

Najprostsze rozwiązania nie zawsze jednak bywają najlepsze — w tym wypadku ograniczyliśmy funkcjonalność całej bazy, aby zapewnić prawidłowe funkcjonowanie jednego wyzwalacza. Spróbujmy raczej zmodyfikować sam wyzwalacz, a nie bazę danych.

Skoro nastąpiło automatyczne wywołanie wyzwalacza, to może wystarczy dodać odpowiedni warunek logiczny, który musi zostać spełniony dla jego wywołania, aby rozwiązać problem? Niestety, takie rozwiązanie nie gwarantuje nam, że nie wystąpi rekurencyjne wywołanie wyzwalacza, wprowadza niepotrzebne warunki logiczne, których sprawdzanie jest dość kosztowne i — jako „nieeleganckie” — nie powinno być stosowane. Alternatywą mogłoby być ograniczenie liczby kolumn, w których zmiana danych spowodowałby wywołanie wyzwalacza, do oryginalnych kolumn tabeli. Ale takie

rozwiązanie z kolei nie realizuje założonego celu — monitorowania aktywności użytkowników. Zamiast tego należy zmienić typ wyzwalacza na INSTEAD OF — wyzwalacze tego typu z definicji wywoływane są tylko raz:

```
EXEC sp_dboption Northwind, 'recursive triggers','on'
GO
ALTER TRIGGER t_cat_audit
ON dbo.Categories INSTEAD OF UPDATE AS
BEGIN
IF (@@ROWCOUNT>0)
UPDATE dbo.Categories --kolumna CategoryID nie może być aktualizowana
SET CategoryName = I.CategoryName,
Description = I.Description,
Picture = I.Picture,
uzytkownik = SYSTEM_USER + ' ' + user_name(),
czas = CURRENT_TIMESTAMP
FROM dbo.Categories C JOIN INSERTED I
ON C.CategoryID = I.CategoryID
END
GO
UPDATE dbo.Categories
SET Description = Description
WHERE CategoryID = 2
GO
(1 row(s) affected)
(1 row(s) affected)
```

Opcje wyzwalaczy

W poprzednim przykładzie został zmodyfikowany parametr bazy danych określający, czy możliwe jest rekursywne wywoływanie wyzwalaczy. Opcja ta, jak wiele innych opcji poziomu bazy danych, może zostać ustawiona za pomocą instrukcji ALTER DATABASE, a odczytana lub ustawiona poprzez wywołanie procedury systemowej sp_dboption:

```
USE Northwind
EXEC sp_dboption Northwind, 'recursive triggers', 'on'
GO
sp_dboption command succeeded.
```

Włączyć lub wyłączyć zagnieżdżanie wyzwalaczy możemy poprzez rekonfigurację serwera bazodanowego (rekursywne wywoływanie wyzwalaczy występuje, jeżeli wyzwalacz modyfikuje dane w tej samej tabeli, z którą został powiązany. Zagnieżdżone wywoływanie wyzwalaczy jest terminem szerszym, obejmującym każdy przypadek, w którym wywołanie jednego wyzwalacza spowodowało wywołanie innego wyzwalacza).

```
USE Northwind
EXEC sp_configure 'nested triggers', 0
GO
RECONFIGURE
```

```
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'nested triggers' changed from 1 to 0. Run the RECONFIGURE
[a]statement to install.
```

Wyświetlić listę wszystkich wyzwalaczy powiązanych z daną tabelą możemy poprzez wywołanie procedury systemowej sp_helptrigger:

```
USE Northwind
EXEC sp_helptrigger Categories
GO
trigger_name trigger_owner isupdate isdelete isinsert isafter isinsteadof
------------ -------------- ----------- ----------- ------------ ----------- -----------
t_cat_audit dbo 1 0 0 0 1
(1 row(s) affected)
```

Poszczególne wyzwalacze mogą być włączane i wyłączane za pomocą instrukcji ALTER TABLE. Na przykład, aby czasowo wyłączyć wyzwalacz t_cat_audit, napiszemy:

```
USE Northwind
ALTER TABLE dbo.Categories
DISABLE TRIGGER t_cat_audit
GO
The command(s) completed successfully.
```

Instrukcja ALTER TRIGGER

Wykonanie instrukcji spowoduje zmianę definicji istniejącego wyzwalacza.

```
ALTER TRIGGER wyzwalacz
ON {tabela}
[WITH ENCRYPTION]
{
{{FOR | AFTER | INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS
instrukcja_SQL […n]
}
|
{(FOR | AFTER | INSTEAD OF) {[INSERT] [,] [UPDATE]}
[WITH APPEND]
```

```
[NOT FOR REPLICATION]
AS
{IF UPDATE (kolumna)
[{AND | OR} UPDATE (kolumna)]
[…n]
| IF (COLUMNS_UPDATED () {operator_binarny} maska_uaktualnienia)
{operator_porównania} maska […n]
}
instrukcja_SQL […n]
}
}
```

Przykład:

```
USE Northwind
GO
ALTER TRIGGER t_zamowienie
ON dbo.Orders
FOR DELETE
AS
EXEC master..xp_sendmail 'kontroler1',
'USUNięto wiersz z tabeli Orders '
GO
The command(s) completed successfully.
```

Instrukcja DROP TRIGGER

Niepotrzebne i nieużywane wyzwalacze mogą zostać usunięte z bazy danych. Domyślnie uprawnienie do usuwania wyzwalaczy mają członkowie ról sysadmin oraz db_owner.

Składnia:

```
DROP TRIGGER {wyzwalacz} [,…n]
```

Przykład:

```
USE Northwind
DROP TRIGGER t_zamowienie
GO
The command(s) completed successfully.
```

Kurs Transact-SQL, część 14 (Funkcje użytkownika)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Jedną z nowych możliwości SQL Servera 2000 jest możliwość tworzenia przez użytkowników własnych funkcji. Funkcjom, tak jak procedurom, można przekazać pewną liczbę parametrów, ale funkcja nie tylko wykonuje pewne operacje, ale także zwraca obliczony na podstawie przekazanych parametrów wynik. Zwracane mogą być zarówno dane skalarne, jak i zbiory danych.

Typy funkcji użytkownika

Typ zwracanych danych oraz struktura funkcji są podstawą podziału funkcji użytkownika na trzy typy:

1. Funkcje skalarne — funkcje tego typu zwracają pojedynczą wartość i przypominają opisane w rozdziale drugim funkcje systemowe.
2. Proste funkcje tabelaryczne — funkcje zwracające wynik pojedynczej instrukcji SELECT i przypominające widoki, ale różniące się od nich możliwością określania dodatkowych parametrów wywołania.
3. Złożone funkcje tabelaryczne — funkcje tego typu zwracają wynik dowolnej liczby instrukcji języka Transact-SQL i najbardziej przypominają procedury składowane. Jednak, w przeciwieństwie do procedur składowanych, można się do nich odwoływać w klauzuli FROM instrukcji SELECT.

Instrukcja CREATE FUNCTION

Wykonanie instrukcji spowoduje utworzenie funkcji użytkownika. Domyślnie wyłącznie członkowie ról sysadmin, db_owner oraz db_ddladmin mogą tworzyć własne funkcje. Członkowie dwóch pierwszych ról mogą nadać innym użytkownikom uprawnienie do wykonywania instrukcji CREATE FUNCTION.

Niemożliwe jest tworzenie funkcji użytkownika zawierających w ciele jakąkolwiek niedeterministyczną funkcję systemową, to znaczy funkcję, która zwraca różne wartości, w zależności od okoliczności ich wywołania. Przykładem systemowych funkcji niedeterministycznych są funkcje CURRENT_USER (zwracająca nazwę zalogowanego użytkownika) oraz GETDATE (zwracająca bieżącą datę systemową).

Składnia:

```
--Funkcje skalarne
CREATE FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS skalarny_typ_danych
[WITH <opcje> [,…n]]
[AS]
BEGIN
ciało
RETURN wyrażenie
END
--Proste funkcje tabelaryczne
```

```
CREATE FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS TABLE
[WITH <opcje> [,…n]]
[AS]
RETURN [(] instrukcja_SELECT [)]
--Złożone funkcje tabelaryczne
CREATE FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS @wynik TABLE <definicja_tabeli>
[WITH <opcje> [,…n]]
[AS]
BEGIN
ciało
RETURN
END
<opcje> ::=
{ENCRYPTION | SCHEMABINDING}
<definicja_tabeli> :: =
({definicja_kolumny | definicja_zawężenia} [,…n])
```

Przykłady:
Tworzymy funkcję skalarną zamieniającą wartości NULL na tekst „Brak danych!”:

```
USE pubs
GO
CREATE FUNCTION fn_brakdanych
(@tekst char(12))
RETURNS char(12)
BEGIN
IF @tekst IS NULL
SET @tekst = 'Brak danych!'
RETURN @tekst
END
GO
The command(s) completed successfully.
```

Wywołanie skalarnej funkcji użytkownika sprowadza się do podania jej nazwy i przekazania odpowiednich parametrów:

```
USE pubs
SELECT pub_name, dbo.fn_brakdanych(state) as wynik_funkcji, state
FROM dbo.publishers
WHERE pub_id>9000
```

```
GO
pub_name wynik_funkcji state
---------------------------------------- ---------------------------------- --------
GGG&G Brak danych! NULL
Scootney Books NY NY
Lucerne Publishing Brak danych! NULL
(3 row(s) affected)
```

Tworzymy prostą funkcję tabelaryczną zwracającą informacje o identyfikatorach i nazwach firm klienckich z wybranego miasta. Ponieważ kolumna City tabeli Customers przechowuje dane typu nvarchar (15), parametr funkcji został zadeklarowany jako ciąg tego typu:

```
USE Northwind
GO
CREATE FUNCTION fn_klienci_z_miasta
(@miasto nvarchar(15))
RETURNS TABLE
AS
RETURN (
SELECT CustomerID, CompanyName
FROM dbo.Customers
WHERE City = @miasto)
GO
USE Northwind
SELECT *
FROM fn_klienci_z_miasta (N'México D.F.')
GO
CustomerID CompanyName
---------- ----------------------------------------
ANATR Ana Trujillo Emparedados y helados
ANTON Antonio Moreno Taquería
CENTC Centro comercial Moctezuma
PERIC Pericles Comidas clásicas
TORTU Tortuga Restaurante
(5 row(s) affected)
```

Proste funkcje tabelaryczne charakteryzuje klauzula RETURN zawierająca pojedynczą instrukcję SELECT. Format danych zwracanych przez funkcję określony jest poprzez tę właśnie instrukcję.

Tworzymy złożoną funkcję tabelaryczną zwracającą dane pobierane z dwóch tabel, przy czym liczba zwracanych danych zależy od wartości argumentu dane:

```
USE pubs
GO
CREATE FUNCTION fn_autorzy
```

```
(@dane tinyint)
RETURNS @fn_autorzy TABLE
(id_autora varchar (40) PRIMARY KEY NOT NULL,
autor varchar(60) NOT NULL,
ksiazek smallint)
AS
BEGIN
IF @dane = 1 --minimalna liczba danych
INSERT @fn_autorzy
SELECT a.au_id, au_lname, COUNT (title_id)
FROM dbo.authors a JOIN dbo.titleauthor t
ON a.au_id = t.au_id
GROUP BY a.au_id, au_lname
ELSE IF @dane = 2 --średnia liczba danych
INSERT @fn_autorzy
SELECT a.au_id, au_lname + ' ' + au_fname, COUNT (title_id)
FROM dbo.authors a JOIN dbo.titleauthor t
ON a.au_id = t.au_id
GROUP BY a.au_id, au_lname + ' ' + au_fname
ELSE IF @dane = 3 --maksymalna liczba danych
INSERT @fn_autorzy
SELECT a.au_id, au_lname + ' ' + au_fname + ' ' + address, COUNT (title_id)
FROM dbo.authors a JOIN dbo.titleauthor t
ON a.au_id = t.au_id
GROUP BY a.au_id, au_lname + ' ' + au_fname + ' ' + address
RETURN
END
GO
The command(s) completed successfully.
```

Ponieważ funkcja jest funkcją tabelaryczną, można się do jej wyników odwoływać tak jak do widoku, a ponieważ jest funkcja złożoną — można przekształcać dane tak jak za pomocą funkcji skalarnej lub procedury:

```
USE pubs
SELECT *
FROM dbo.fn_autorzy (3)
WHERE id_autora LIKE '2%'
ORDER BY ksiazek DESC
GO
id_autora autor ksiazek
---------- ---------- -------
213-46-8915 Green Marjorie 309 63rd St. #411 2
267-41-2394 O'Leary Michael 22 Cleveland Av. #14 2
```

```
274-80-9391 Straight Dean 5420 College Av. 1
238-95-7766 Carson Cheryl 589 Darwin Ln. 1
(4 row(s) affected)
SELECT *
FROM dbo.fn_autorzy (2)
WHERE ksiazek > 1
GO
id_autora autor ksiazek
---------- ---------------------------------------------------------- -------
213-46-8915 Green Marjorie 2
267-41-2394 O'Leary Michael 2
486-29-1786 Locksley Charlene 2
724-80-9391 MacFeather Stearns 2
899-46-2035 Ringer Anne 2
998-72-3567 Ringer Albert 2
(6 row(s) affected)
```

Aby użytkownik mógł wywołać funkcję, musi posiadać uprawnienie EXECUTE do wybranej funkcji oraz uprawnienie (co najmniej) do sprawdzania warunków referencyjnej spójności danych (uprawnienie DRI) dla obiektów źródłowych funkcji.

Tworzenie funkcji powiązanych ze schematem bazy danych

Tak jak w przypadku obiektów innych typów utworzenie, za pomocą opcji SCHEMABINDING, funkcji powiązanej ze schematem bazy danych spowoduje, że obiekty źródłowe funkcji nie będą mogły zostać zmodyfikowane lub usunięte.
Aby możliwe było utworzenie funkcji użytkownika powiązanej ze schematem bazy danych:

- ♣ Wszystkie powiązane z funkcją widoki oraz funkcje użytkownika zostały powiązane ze schematem bazy danych.
- ♣ Wszystkie powiązane z funkcją obiekty znajdują się w tej samej bazie danych.
- ♣ W definicji funkcji nie występują odwołania do obiektów typu właściciel.nazwa_obiektu,
- ♣ Osoba tworząca funkcję ma nadane uprawnienie (co najmniej) DRI do wszystkich powiązanych z funkcją obiektów.

Opcje funkcji

Podstawowe informacje o istniejących funkcjach użytkownika uzyskamy, wywołując procedurę systemową sp_help:

```
USE pubs
EXEC sp_help fn_brakdanych
GO
Name Owner Type Created_datetime
------------------- ------------ --------------------- -------------------------
fn_brakdanych dbo scalar function 2002-11-19 16:23:16.340
```

Definicję funkcji (tak jak obiektów innego typu) zwraca procedura systemowa sp_helptext:

```
USE pubs
EXEC sp_helptext fn_brakdanych
GO
Text
--------------------------------------------------------------------------------
CREATE FUNCTION fn_brakdanych
(@tekst nvarchar(50))
RETURNS nvarchar(50)
BEGIN
IF @tekst IS NULL
SET @tekst = 'Brak danych!'
RETURN @tekst
END
```

Instrukcja ALTER FUNCTION

Wykonanie instrukcji pozwala na zmianę definicji funkcji, bez zmiany nadanych jej uprawnień i modyfikowania powiązanych obiektów.
Składnia:

```
--Funkcje skalarne
ALTER FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS skalarny_typ_danych
[WITH <opcje> [,…n]]
[AS]
BEGIN
ciało
RETURN wyrażenie
END
--Proste funkcje tabelaryczne
ALTER FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS TABLE
[WITH <opcje> [,…n]]
[AS]
RETURN [(] instrukcja_SELECT [)]
--Złożone funkcje tabelaryczne
ALTER FUNCTION [właściciel.] funkcja
([{@parametr parametr_skalarny [= wartość_domyślna]} [,…n]])
RETURNS @wynik TABLE <definicja_tabeli>
[WITH <opcje> [,…n]]
[AS]
```

```
BEGIN
ciało
RETURN
END
<opcje> ::=
{ENCRYPTION | SCHEMABINDING}
<definicja_tabeli> :: =
({definicja_kolumny | definicja_zawężenia} [,…n])
```

Przykład:

```
USE pubs
GO
ALTER FUNCTION fn_brakdanych
(@tekst nvarchar(50))
RETURNS nvarchar(50)
BEGIN
IF LEN(@tekst)-LEN(REPLACE(@tekst,'@','')) = 0
SET @tekst = 'Brak poprawnych danych!'
RETURN @tekst
END
GO
SELECT dbo.fn_brakdanych ('szef@firma.com'), dbo.fn_brakdanych ('niby_adres')
GO
---------------------------------------------- ------------------------------------
szef@firma.com Brak poprawnych danych!
(1 row(s) affected)
```

Instrukcja DROP FUNCTION

Wykonanie instrukcji spowoduje usunięcie wybranych funkcji użytkownika.
Składnia:

```
DROP FUNCTION {[właścicel.] funkcja} [,…n]
```

Przykład:

```
USE pubs
GO
DROP FUNCTION fn_brakdanych, dbo.fn_autorzy
GO
The command(s) completed successfully.
```

Kurs Transact-SQL, część 15 (Microsoft SQL Server 2000)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

SQL Server może pełnić funkcję zarówno serwera OLTP (ang. Online Transaction Processing), jak i (po zainstalowaniu znajdujących się na płycie instalacyjnych dodatkowych usług) serwera OLAP (ang. Online Analytical Processing).

W obu przypadkach rola SQL Servera polega przede wszystkim na:

- Zapewnieniu spójności przechowywanych danych.
- Odtwarzaniu spójnego obrazu danych w przypadku wystąpienia awarii dowolnego typu.
- Realizowaniu żądań użytkownika, w szczególności żądań pobrania i zmodyfikowania danych.

Wszystkie zadania realizowane są przez współpracujące ze sobą podstawowe usługi serwera.

Usługi SQL Servera

Podczas domyślnej instalacji SQL Servera instalowane są cztery częściowo niezależne usługi:

1. MSSQLServer — główna usługa serwera bazodanowego odpowiedzialna m.in. za:
   - Przydzielanie i zwalnianie zasobów systemowych na potrzeby realizowania zadań użytkowników.
   - Utrzymanie spójnego obrazu danych, w tym zarządzanie blokadami danych i wycofywanie transakcji.
   - Przechowywanie i modyfikowanie danych.
2. SQLServerAgent — usługa zarządzająca obiektami trzech typów:
   - Operatorami, które mogą być poinformowane o wystąpieniu określonego zdarzenia.
   - Zadaniami, które mogą być automatycznie wykonywane.
   - Alarmami, które są wyzwalane wystąpieniem określonego błędu.

   Definicje obiektów tych typów przechowywane są w bazie msdb i są wspólne dla wszystkich baz danych użytkownika.

3. MS DTC (ang. Distributed Transaction Coordinator) — usługa, której zadaniem jest zatwierdzanie lub wycofywanie transakcji rozproszonych na wszystkich połączonych serwerach bazodanowych.
4. Microsoft Search — w wersji 2000 podczas domyślnej instalacji instalowana jest również usługa, której zadaniem jest obsługa katalogów i indeksów wyszukiwania pełnotekstowego.

Usługi pomocnicze SQL Mail oraz SQLAgentMail zostały przedstawione w rozdziale 17.

Wymiana danych pomiędzy usługami MSSQLServer i SQLServerAgent

Zanim będzie możliwe skorzystanie z usługi SQLServerAgent, konieczne jest jej poprawne skonfigurowanie. Ponieważ książka nie jest poświecona konfiguracji SQL Servera, zostaną jedynie wymienione wymagania, które należy spełnić, aby możliwe było wykonanie ćwiczeń znajdujących się w dalszej części książki:

1. W systemach Microsoft Windows NT/2000/XP usługa SQLServerAgent (tak jak pozostałe usługi SQL Servera) działa jako usługa, a nie jako aplikacja kliencka. Usługa ta musi być cały czas uruchomiona, jej wstrzymanie (ang. Pause) jest równoznaczne z całkowitym zatrzymaniem.
2. W wymienionych systemach każdy program i usługa uruchamiane są w określonym kontekście bezpieczeństwa użytkownika. Aby świadczyć wszystkie usługi, SQLServerAgent musi zostać uruchomiony w

kontekście konta domenowego posiadającego uprawnienia lokalnego administratora systemu, w którym zainstalowany został SQL Server. Rozwiązanie takie umożliwia:

- ♣ Automatyczne wznawianie niespodziewanie zatrzymanych usług SQLServerAgent i MSSQLServer przez usługę SQLServerAgent.
- ♣ Integracje z innymi serwerami firmy Microsoft, w tym z serwerem pocztowym i serwerem WWW (możliwe będzie m.in. wysyłanie wiadomości e-mail do określonych operatorów).
- ♣ Uruchamianie określonych zadań w momencie bezczynności systemu operacyjnego (procent, poniżej którego musi spaść obciążenie procesora i czas, w jakim musi się utrzymywać poniżej tego poziomu, określa podczas konfiguracji usługi SQLServerAgent).

Uruchomienie usługi SQLServerAgent z uprawnieniami administratora lokalnego upraszcza zarówno zarządzanie serwerem, jak i łączenie serwerów i replikację danych. Właściwie jedyną wadą tego rozwiązania jest zwiększenie ryzyka przejęcia kontroli nad całym systemem przez hakera, który może włamać się do serwera bazodanowego.

3. Login usługi SQLServerAgent musi należeć do roli serwera sysadmin, inaczej usługa nie będzie w stanie zarejestrować się w SQL Serverze.

Konta wszystkich lokalnych administratorów (użytkowników należących do wbudowanej grupy lokalnej Administratorzy) podczas instalacji SQL Servera są automatycznie dodawane do roli sysadmin.

Domyślnie usługa SQLServerAgent uwierzytelnia się w SQL Serverze za pomocą mechanizmu uwierzytelniania systemu Windows, ale możliwe jest uwierzytelnienie usługi na podstawie loginu SQL Servera.

Bezpieczeństwo SQL Servera (w tym modele uwierzytelniania i zarządzanie loginami oraz kontami użytkowników) zostało opisane w rozdziale 16.

Usługi MSSQLServer i SQLServerAgent wymieniają między sobą dane poprzez dziennik aplikacji systemu operacyjnego. Wymiana danych przebiega następująco:

1. Jeżeli wystąpił dowolny błąd, usługa MSSQLServer sprawdza, czy w tabeli sysmessages znajduje się wiersz zawierający numer danego błędu. Komunikaty błędów o numerach mniejszych niż 50 000 są błędem systemowym, wyższe numery mogą być nadawane komunikatom błędów zdefiniowanym przez użytkownika. Jeżeli znaleziony komunikat błędu miał określony poziom ważności (ang. Severity) z zakresu od 19 do 25, informacja o wystąpieniu błędu zostanie automatycznie zapisana do dziennika aplikacji systemu operacyjnego. Możliwe jest również wymuszanie zapisania w dzienniku błędów o niższych poziomach ważności. Poniższe zapytanie zwraca przykładowe komunikaty błędów zapisane w tabeli systemowej sysmessages.

```
SELECT *
FROM master.dbo.sysmessages
WHERE severity = 17 AND error > 9000
GO
error severity dlevel description
msglangid
----------- -------- ------ ----------------------------------------------------
a----------------
15319 17 0 Error: DBCC DBREPAIR REMAP failed for database '%s'
a(device '%s').
1033
15498 17 127 Inside txn_1a_, update failed. Will roll back (1a1).
1033
18666 17 128 Could not free up descriptor in rel_desclosed()
asystem function.
1033
(3 row(s) affected)
```

2. Usługa SQLServerAgent podczas startu rejestruje się w dzienniku aplikacji systemu operacyjnego i w momencie zapisania przez usługę MSSQLServer do dziennika informacji o błędzie pobiera te dane i porównuje z zawartością tabeli sysalerts.

Zawartość tabel sysmessages i sysalerts, w celu zapewnienia maksymalnej niezawodności i wydajności, przechowywana jest w buforze. Ponieważ obie usługi serwera wymieniają dane poprzez systemowy dziennik aplikacji, wystąpienie błędu nie odnotowane w tym dzienniku nie zostanie odczytane i przetworzone przez usługę SQLServerAgent.

Jeżeli w tabeli sysalerts znajduje się rekord przechowujący aktywny (ang. Enabled) komunikat zdefiniowany dla błędu o określonym numerze lub dla wszystkich błędów z określonego poziomu ważności, zostaje on odczytany i porównany z zawartością tabel sysnotifications oraz sysoperators. Poniższe zapytanie zwraca przykładowe, predefiniowane definicje alarmów.

```
SELECT LEFT (name,20), message_id, severity, enabled
FROM msdb.dbo.sysalerts
GO
message_id severity enabled
-------------------- ----------- ----------- -------
Demo: Full msdb log 9002 0 1
Demo: Full tempdb 9002 0 1
Demo: Sev. 19 Errors 0 19 1
Demo: Sev. 20 Errors 0 20 1
Demo: Sev. 21 Errors 0 21 1
Demo: Sev. 22 Errors 0 22 1
Demo: Sev. 23 Errors 0 23 1
Demo: Sev. 24 Errors 0 24 1
Demo: Sev. 25 Errors 0 25 1
(9 row(s) affected)
```

3. Tabela sysoperators zawiera definicje operatorów (osób, do których automatycznie mogą zostać wysłane komunikaty):

```
SELECT LEFT (name,5), enabled, email_address, pager_address, netsend_address
FROM msdb.dbo.sysoperators
GO
enabled email_address pager_address netsend_address
----- ------- --------------- ---------------- ------------------------------
a-------
(0 row(s) affected)
```

4. W tabeli sysnotifications znajdują się informacje o tym, którego z operatorów i za pomocą jakiej metody należy automatycznie powiadomić o wystąpieniu danego błędu. W ten sposób administrator może zostać automatycznie powiadomiony o wystąpieniu błędu:

```
SELECT *
FROM msdb.dbo.sysnotifications
```

```
GO
alert_id operator_id notification_method
---------- ---------- ------------------
(0 row(s) affected)
```

5. Dzięki opisanemu wyżej mechanizmowi, wystąpienie dowolnego błędu może spowodować wywołanie określonego alarmu, a to z kolei może wiązać się z powiadomieniem określonego operatora. Usługa SQLServerAgent zarządza obiektami jeszcze jednego typu — zadaniami, dzięki którym możliwe jest osiągnięcie prawie całkowitej automatyzacji pracy SQL Servera.
6. Zadaniem (ang. Job) jest zbiór instrukcji języka Transact-SQL, skryptów powłoki lub plików wykonywalnych, których wzajemne powiązania zostały określone podczas tworzenia zadania. Zadanie może np. wykonać kopię dziennika transakcyjnego, a następnie skopiować plik z kopią na zdalny komputer.

   Pomiędzy wykonaniem zadania, powiadomieniem operatora i wywołaniem alarmu istnieją następujące zależności:
   - ♣ Wywołanie alarmu może spowodować powiadomienie operatora.
   - ♣ Wywołanie alarmu może spowodować uruchomienie zadania.
   - ♣ Zakończone sukcesem wykonanie zadania może spowodować powiadomienie operatora.
   - ♣ Zakończone niepowodzeniem wykonanie zadania może spowodować powiadomienie operatora.
   - ♣ Zakończone sukcesem wykonanie zadania może spowodować wywołanie alarmu.
   - ♣ Zakończone niepowodzeniem wykonanie zadania może spowodować wywołanie alarmu.

Instancje SQL Servera

Jedną z nowości wersji 2000 SQL Servera jest możliwość uruchomienia na jednym komputerze do 16 instancji serwera bazodanowego. Pierwsza instancja jest instancją domyślną i można się do niej odwoływać za pomocą nazwy komputera, np. serwer2k. Pozostałe instancje są instancjami nazwanymi i należy się do nich odwoływać za pomocą nazwy komputera uzupełnionej o nazwę instancji, np. serwer2k\instancja_SQL.

Poszczególne instancje są niezależnymi serwerami bazodanowymi, każda z nich uruchamia własne usługi działające po stronie serwera i dysponuje własnym zestawem baz danych (systemowych i użytkownika). Jednak niektóre komponenty są wspólne dla wszystkich uruchomionych na tym samym komputerze serwerów bazodanowych. Należą do nich:

- ♣ Usługa Microsoft Search.
- ♣ Opcjonalnie zainstalowane usługi OLAP oraz English Query.
- ♣ Grupa narzędzi Microsoft SQL Server dodawana podczas instalacji do menu Start (za pomocą każdego z tych narzędzi można zarządzać dowolną instancją).
- ♣ Opcjonalnie instalowane przez programistów biblioteki i pliki nagłówkowe.

Instancje nazwane są nowością wersji 2000 i w związku z tym nie są w pełni zgodne z wcześniejszymi wersjami SQL Servera. Na przykład w środowisku mieszanym tylko instancja domyślna może brać udział w replikacji, a za pomocą multiprotokołu możliwe jest nawiązanie połączenia wyłącznie z instancją domyślną.

Bazy danych SQL Servera

Wyróżnia się dwa rodzaje baz danych SQL Servera: bazy systemowe i bazy użytkownika. Do tych pierwszych należą:

1. Baza *master* — tworzona podczas instalacji SQL Servera i przechowująca katalog serwera bazodanowego. Jej brak lub uszkodzenie uniemożliwia uruchomienie SQL Servera. W bazie master znajdują się m.in. informacje o parametrach pracy SQL Servera, definicje wszystkich pozostałych baz danych, loginy użytkowników i komunikaty błędów.
2. Baza *msdb* — tworzona podczas instalacji SQL Servera. Przechowuje niezbędne do pracy usługi SQLServerAgent informacje o operatorach, zdefiniowanych zadaniach i alarmach. Jest również wykorzystywana podczas replikacji danych.
3. Baza *model* — tworzona podczas instalacji SQL Servera pełni rolę szablonu (wzorca) wszystkich baz danych użytkownika. Utworzenie nowej bazy danych powoduje skopiowanie do niej zawartości bazy model.
4. Baza *tempdb* — tworzona podczas uruchamiania SQL Servera i usuwana podczas jego zatrzymywania. W bazie tempdb przechowywane są wszystkie obiekty tymczasowe, baza ta może być również wykorzystywana do przechowywania cząstkowych wyników instrukcji języka Transact-SQL.
5. Baza *distribution* — tworzona podczas konfigurowania SQL Servera jako serwera pełniącego funkcję dystrybutora w replikacji danych. Przechowuje replikowane dane i historię replikacji.

Model wydawca — dystrybutor — subskrybent został opisany w rozdziale 21., poświęconym replikacji danych.

Oprócz baz systemowych podczas instalacji SQL Servera tworzone są dwie przykładowe bazy użytkownika: Northwind i pubs.

Każda baza danych, zarówno systemowa, jak i baza użytkownika, zawiera pewną liczbę obiektów systemowych, w tym tabel systemowych. Zbiór tych obiektów nazywany jest katalogiem bazy danych. Nazwy wszystkich tabel systemowych rozpoczynają się prefiksem sys.

Firma Microsoft nie zaleca bezpośredniego odczytywania (a tym bardziej modyfikacji) zawartości tabel systemowych.

Ponieważ struktura tabel systemowych zmienia się z wersji na wersję SQL Servera, zamiast bezpośrednio odczytywać czy modyfikować ich zawartość, należy skorzystać z:

1. Procedur systemowych — opisywane w poprzednich rozdziałach procedury systemowe, np. procedura sp_help, pozwalają na pobieranie metadanych zapisanych w katalogu systemu lub bazy danych.
2. Opisanych w rozdziale 2. funkcji: konfiguracyjnych, metadanych, statystyki systemu lub systemowych. Za ich pomocą (na przykład za pomocą funkcji COLUMNPROPERTY) można pobierać i modyfikować metadane.
3. Predefiniowanych widoków systemowych — w przeciwieństwie do tabel systemowych widoki systemowe w kolejnych wersjach SQL Servera prezentują dane w takiej samej postaci.

Do najczęściej wykorzystywanych widoków systemowych należą:

1. Przechowujący informacje o tabelach bazy danych widok INFORMATION_SCHEMA.TABLES:

```
USE Northwind
SELECT LEFT (TABLE_NAME,30), LEFT (TABLE_TYPE,20)
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME LIKE 'C%'
GO
------------------------------ ----------
Categories BASE TABLE
Category Sales for 1997 VIEW
Current Product List VIEW
Customer and Suppliers by City VIEW
CustomerCustomerDemo BASE TABLE
CustomerDemographics BASE TABLE
Customers BASE TABLE
(7 row(s) affected)
```

2. Przechowujący informacje o poszczególnych kolumnach tabel widok INFORMATION_SCHEMA.COLUMNS:

```
USE Northwind
SELECT LEFT (TABLE_NAME,30), LEFT (COLUMN_NAME, 20), LEFT (COLUMN_DEFAULT, 5),
IS_NULLABLE AS 'Null', LEFT (DATA_TYPE,10) AS Typ, CHARACTER_MAXIMUM_LENGTH
ªAS Długość
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Customers'
GO
Null Typ Długość
------------------------------ -------------------- ----- ---- ---------- -------
```

```
Customers CustomerID NULL No nchar 10
Customers CompanyName NULL No nvarchar 80
Customers ContactName NULL YES nvarchar 60
Customers ContactTitle NULL YES nvarchar 60
Customers Address NULL YES nvarchar 120
Customers City NULL YES nvarchar 30
Customers Region NULL YES nvarchar 30
Customers PostalCode NULL YES nvarchar 20
Customers Country NULL YES nvarchar 30
Customers Phone NULL YES nvarchar 48
Customers Fax NULL YES nvarchar 48
(11 row(s) affected)
```

3. Przechowujący informacje o widokach widok INFORMATION_SCHEMA.VIEW_TABLE_USAGE:

```
USE Northwind
SELECT LEFT (VIEW_NAME, 30),LEFT (TABLE_NAME, 30), LEFT (TABLE_CATALOG,20)
FROM INFORMATION_SCHEMA.VIEW_TABLE_USAGE
WHERE TABLE_NAME = 'Customers'
GO
------------------------------ ------------------------------ --------------------
Customer and Suppliers by City Customers Northwind
Invoices Customers Northwind
Orders Qry Customers Northwind
Quarterly Orders Customers Northwind
Sales Totals by Amount Customers Northwind
(5 row(s) affected)
```

4. Przechowujący informacje o nadanych do poszczególnych tabel uprawnieniach widok INFORMATION_SCHEMA.TABLE_PRIVILEGES:

```
USE Northwind
SELECT LEFT (GRANTOR, 5),LEFT (TABLE_NAME, 30), PRIVILEGE_TYPE, IS_GRANTABLE
FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
WHERE TABLE_NAME = 'Customers'
GO
PRIVILEGE_TYPE IS_GRANTABLE
----- ------------------------------ -------------- ------------
dbo Customers REFERENCES NO
dbo Customers SELECT NO
dbo Customers INSERT NO
```

```
dbo Customers DELETE NO
dbo Customers UPDATE NO
(5 row(s) affected)
```

Kurs Transact-SQL, część 16 (Bezpieczeństwo)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Uzyskanie dostępu do informacji przechowywanych w bazach danych jest jednym z najczęstszych celów hakerów — z reguły właśnie w bazach danych znajduje się większość poufnych lub ważnych dla działania firmy informacji. Ponadto ścisłe zintegrowanie SQL Servera z systemem operacyjnym oraz możliwość wywoływania procedur rozszerzonych, które pozwalają m.in. na wykonanie dowolnego programu lub odczyt czy modyfikację zawartości rejestru systemowego, powodują, że haker, który włamał się do SQL Servera, zdobył tym samym doskonałą pozycję do przejęcia kontroli nad systemem operacyjnym.

Model bezpieczeństwa SQL Servera

Podstawową zasadą, na której opera się model bezpieczeństwa SQL Servera jest umożliwianie jedynie uwierzytelnionym użytkownikom nawiązywania połączenia z serwerem bazodanowym. Uwierzytelnienie (sprawdzenie tożsamości) użytkownika może przebiegać w dwóch trybach: trybie Windows NT/2000 lub w trybie mieszanym.

> Model bezpieczeństwa SQL Servera w wersjach 7 i 2000 nie zawiera trybu, w którym możliwe byłoby uwierzytelnianie wyłącznie użytkowników serwera bazodanowego. Natomiast odblokowując konta gościa, możemy zezwolić na dostęp do serwera anonimowym użytkownikom.

W obu trybach próba połączenia zostanie zaakceptowana, jeżeli w tabeli master.. sysxlogins znajduje się odpowiedni wpis uprawniający użytkownika do jego nawiązania. Poniższe zapytanie zwraca listę loginów i haseł użytkowników serwera:

```
SELECT LEFT(name,25), LEFT(password,20)
FROM master.dbo.sysxlogins
GO
------------------------- --------------------
BUILTIN\Administratorzy NULL
LOLEK\Administrator NULL
LOLEK\klientvpn NULL
sa !1"*&A\X&[H!]$J(
NULL NULL
(5 row(s) affected)
```

Pojedynczy wiersz tabeli przechowuje informacje o jednym loginie (tożsamości użytkownika, na podstawie której możliwe jest połączenie z serwerem bazodanowym). Wyróżniamy trzy typy loginów:

1. Login użytkownika SQL Servera (w powyższym przykładzie loginem tego typu jest login sa). Na podstawie tego loginu użytkownik, którego tożsamość nie została potwierdzona przez system operacyjny może (po podaniu prawidłowego, przechowywanego w postaci zaszyfrowanej hasła) nawiązać połączenie z serwerem bazodanowym.
2. Login użytkownika systemu NT lub 2000 (np. login LOLEK\klientvpn) pozwalający wybranemu użytkownikowi uwierzytelnionemu podczas logowania się do systemu na nawiązanie połączenia z serwerem bazodanowym (ponieważ SQL Server „ufa” systemowi operacyjnemu, w tym wypadku nie jest sprawdzana ponownie tożsamość użytkownika, a więc nie ma potrzeby przechowywania jego hasła w tabeli sysxlogins).

3. Login grupy użytkowników systemu NT lub 2000 (np. login BUILTIN\ Administratorzy), na podstawie którego wszyscy użytkownicy należący do danej grupy mogą nawiązać połączenie z SQL Serverem. W tym wypadku również nie jest wymagane ponowne potwierdzenie tożsamości przez użytkowników należących do danej grupy.

Tryb Windows NT/2000

Użytkownik, którego tożsamość została potwierdzona przez system operacyjny i którego konto znajduje się w tej samej domenie co konto serwera bazodanowego (lub w innej, zaufanej domenie), podczas próby nawiązania połączenia z SQL Serverem może otworzyć połączenie zaufane. Poprzez takie połączenie przesyłane są informacje uwierzytelniające danego użytkownika.

Wynegocjowana przez klienta i SQL Server technika uwierzytelniania zależy od konfiguracji systemu komputerowego. Najbezpieczniejszą metodą jest wzajemne uwierzytelniane za pomocą technologii Kerberos, ale aby było ono możliwe, musi być spełnionych kilka warunków, m.in. oba komputery, kliencki i serwer, muszą działać pod kontrolą systemu Windows 2000 lub nowszego, do nawiązania połączenia musi być użyty protokół TCP/IP, a SQL Server musi mieć prawidłowo skonfigurowaną i zarejestrowaną w Active Directory wartość atrybutu SPN (ang. Service Principal Name).

Ponieważ fakt nawiązania połączenia zaufanego jest dla SQL Servera wystarczającym dowodem na to, że tożsamość użytkownika została potwierdzona przez system operacyjny, hasło użytkownika nie musi być sprawdzone przez serwer bazodanowy, a w konsekwencji nie musi zostać wysłane przez nawiązane połączenie zaufane. Tak więc jedyną operacją, jaka zostanie przeprowadzona w tym wypadku jest sprawdzenie, czy w tabeli sysxlogins znajduje się wpis, na podstawie którego dany użytkownik ma prawo do nawiązania połączenia z serwerem bazodanowym.

Konta użytkowników i grup identyfikowane są na podstawie unikalnego numeru SID, a nie na podstawie ich nazwy, tak więc usunięcie i odtworzenie konta o tej samej nazwie spowoduje, że SQL Server nie pozwoli na nawiązanie połączenia użytkownikom, których loginów nie będzie w stanie zidentyfikować.

Tryb Windows NT/2000 charakteryzuje:

- Możliwość skorzystania z rozbudowanych opcji systemu operacyjnego pozwalających m.in. na wymuszenie:
  - tworzenia skomplikowanych haseł,
  - ustalenia minimalnej długości hasła,
  - automatycznego blokowania konta,
  - automatycznego wygasania ważności konta,
  - zmiany hasła co określony czas,
  - przechowywania historii haseł.
- Uproszczenie zarządzania SQL Serverem dzięki powiązaniu grup użytkowników z pojedynczym loginem.
- Brak konieczności przesyłania poprzez sieć nazwy konta użytkownika i jego hasła.
- Łatwiejszy dostęp dla użytkowników, którzy muszą znać i pamiętać tylko jedno hasło, aby zalogować się do systemu operacyjnego i połączyć z serwem bazodanowym.

Uwierzytelniane za pomocą technologii Kerberos

Jeżeli do uwierzytelniania użytkowników systemu 2000 zostanie wykorzystany protokół Kerberos, możliwe będzie wzajemne uwierzytelnienie się użytkownika i usługi. Proces uwierzytelniania będzie wówczas przebiegał następująco:

1. Po poprawnym zalogowaniu użytkownika do komputera pracującego pod kontrolą systemu Windows 2000 usługa uwierzytelniania Kerberos przydziela użytkownikowi identyfikujący go w usłudze centrum dystrybucji kluczy KDC (ang. Key Distribution Center) bilet TGT (ang. Ticket-Granting Ticket).
2. Podczas próby nawiązania połączenia z SQL Serverem do centrum KDC zostaje wysłany bilet TGT użytkownika.
3. Na podstawie przesłanego biletu TGT centrum KDC wystawia użytkownikowi ważny przez określony okres czasu (z reguły przez osiem godzin) bilet sesji.
4. Bilet sesji (a nie hasło użytkownika) zostaje przesłany do SQL Servera.
5. Jeżeli wymagane jest wzajemne uwierzytelnienie się użytkownika i usługi, SQL Server wysyła użytkownikowi zaszyfrowaną wiadomość, na podstawie której użytkownik może potwierdzić tożsamość serwera bazodanowego.

Jeżeli ten sam użytkownik zgłosi ponowne żądanie dostępu do zasobów SQL Servera przed wygaśnięciem ważności posiadanego biletu sesji, ponowne uwierzytelnienie odbywa się na podstawie posiadanego biletu sesji i nie wymaga połączenia z centrum KDC.

Tryb mieszany

Jeżeli SQL Server działa w trybie mieszanym, użytkownicy będą mogli zostać zweryfikowani albo na podstawie loginów kont lub grup użytkowników systemu NT/2000, albo na podstawie loginów serwera bazodanowego. W drugim przypadku SQL Server sprawdza, czy podana przez użytkownika nazwa występuje w tabeli sysxlogins, a jeżeli tak — czy zostało podane prawidłowe hasło. Po pomyślnej weryfikacji tożsamości użytkownik uzyskuje dostęp do serwera bazodanowego.

Zabezpieczyć się przed podsłuchiwaniem haseł użytkowników (ang. Sniff) możemy poprzez wymuszenie komunikacji z SQL Serverem za pomocą technologii SSL (ang. Secure Socket Layer).

Tryb mieszany charakteryzuje:

- ♣ Możliwość udostępnienia zasobów SQL Servera użytkownikom nieposiadającym kont w systemie NT/2000, w tym klientom internetowym.
- ♣ Możliwość nawiązania bezpiecznego (niewymagającego przesyłania poufnych informacji) połączenia przez użytkowników posiadających konta w niezaufanych domenach lub w domenach spoza obrębu lasu domen, w którym znajduje się konto SQL Servera.

Niemożliwa jest zmiana trybu uwierzytelniania użytkowników przez SQL Server za pomocą instrukcji języka Transact-SQL. Możliwe jest to albo poprzez konsolę Enterprise Manager, programowo, poprzez model ADO lub za pomocą bezpośredniej modyfikacji wartości LoginMode znajdującej się w kluczu HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL\MSSQLServer rejestru systemowego, gdzie MSSQL jest nazwą instancji SQL Servera. Wartość 2 oznacza tryb mieszany, 1 — tryb Windows NT/2000.

Delegacja uprawnień

Delegacja uprawnień umożliwia korzystanie z zasobów wielu serwerów bazodanowych na podstawie pojedynczego uwierzytelnienia się użytkownika w jednym z nich. Na przykład, jeżeli w sieci znajdują się dwa serwery bazodanowe, LOLEK i ZSZServer2K, użytkownik klientvpn po uwierzytelnieniu się w serwerze LOLEK może skorzystać ze wszystkich dostępnych temu użytkownikowi zasobów serwera ZSZServer2K.

Aby możliwa była delegacja uprawnień:

- ♣ Wszystkie serwery bazodanowe muszą pracować w tej samej lub zaufanej domenie Active Directory.
- ♣ Wszystkie serwery bazodanowe muszą zostać uruchomione na komputerach pracujących pod kontrolą systemu Windows 2000, a do uwierzytelniania muszą wykorzystywać protokół Kerberos.
- ♣ Konto użytkownika musi być kontem domenowym i nie może zostać określone jako konto użytkownika, którego uprawnienia nie mogą zostać delegowane.
- ♣ Konta użytkowników, w kontekście których zostały uruchomione serwery bazodanowe, muszą zostać określone jako zaufane w kwestii delegowania uprawnień.
- ♣ Konta komputerów, na których zostały uruchomione serwery bazodanowe, muszą zostać określone jako zaufane w kwestii delegowania uprawnień.
- ♣ Serwery bazodanowe muszą mieć prawidłowo skonfigurowaną i zarejestrowaną w Active Directory wartość atrybutu SPN.

Przedstawienie uprawnień

Mechanizm przedstawiania uprawnień pozwala SQL Serverowi na posłużenie się tożsamością zalogowanego użytkownika w celu dostępu do niektórych zasobów, na przykład folderów czy plików znajdujących się na dyskach NTFS. W ten sposób unika się nieuprawnionego rozszerzenia uprawnień użytkownika, który ma nadane uprawniania do logowania się w serwerze bazodanowym, ale np. ma zakaz odczytu zawartości określonego folderu.

Dostęp do baz danych

Istotną cechą modelu bezpieczeństwa SQL Servera jest wydzielenie dwóch poziomów:

1. Tożsamość użytkownika musi zostać potwierdzona, aby mógł on połączyć się z SQL Serverem, co nie oznacza jeszcze, że dany użytkownik będzie mógł połączyć się z jakąkolwiek bazą danych użytkownika.
2. Połączenie z konkretną bazą danych jest możliwe, o ile login użytkownika został powiązany z kontem utworzonym w obrębie wybranej bazy. Wyjątkiem są bazy, w których znajduje się odblokowane konto gościa (ang. Quest).

Tak więc wyłącznie uwierzytelnieni użytkownicy będą mieli dostęp do wybranych baz danych. Kolejny poziom zabezpieczeń tworzą, opisane w dalszej części rozdziału, uprawnienia nadane poszczególnym użytkownikom.

Zarządzanie dostępem do SQL Servera

Aby użytkownik mógł nawiązać połączenie z SQL Serverem, w tabeli master.. sysxlogins musi istnieć login danego użytkownika. Podczas instalacji serwera bazodanowego automatycznie tworzone są loginy:

1. Dla grupy lokalnych administratorów systemu operacyjnego (BUILTIN\ Administratorzy) — domyślnie wszyscy administratorzy lokalnego systemu
2. Dla konta użytkownika, w kontekście którego została uruchomiona usługa MSSQLServer (LOLEK\Administrator).
3. Jeżeli został wybrany tryb mieszany, dla użytkownika sa (ang. System Administrator).

Wszyscy ci użytkownicy domyślnie mają nieograniczone uprawnienia i mogą przeprowadzić każdą operację we wszystkich bazach danych uruchomionych na serwerze.

Tworzenie loginów

W zależności od tego, czy chcemy utworzyć login użytkownika uwierzytelnionego przez system operacyjny, czy login użytkownika, którego tożsamość będzie potwierdzana przez SQL Server, musimy wykonać jedną z dwóch procedur systemowych: sp_grantlogin lub sp_addlogin.

Domyślnie jedynie członkowie ról sysadmin oraz securityadmin mogą zarządzać dostępem do SQL Servera.

Procedura sp_grantlogin

Wykonanie procedury powoduje dodanie loginu dla użytkownika lub grupy użytkowników systemu operacyjnego.
Składnia:

```
sp_grantlogin [@loginame =] 'login'
```

Przykłady:

```
sp_grantlogin [lolek\klientvpn]
GO
Granted login access to 'lolek\klientvpn'.
sp_grantlogin [lolek\klienci_SQL]
GO
Granted login access to 'lolek\klienci_SQL'.
```

- Konto użytkownika lub grupy użytkowników musi zostać utworzone w systemie przed wykonaniem procedury sp_grantlogin.
- Nazwa konta musi zostać poprzedzona nazwą domeny lub komputera i znakiem ukośnika — SQL Server nie akceptuje nazw zapisanych w formacie UPN (ang. User Principal Names), czyli nazw typu konto@pełna_nazwa_domeny.
- Maksymalna długość loginu (nazwa domeny + nazwa konta użytkownika lub grupy) nie może przekroczyć 128 znaków, a więc SQL Server nadaje dodatkowe ograniczenia na długość nazw kont użytkowników systemu operacyjnego.
- Usunięcie konta systemowego nie powoduje automatycznego usunięcia utworzonego dla niego loginu.
- Dodawanie lub usuwanie kont użytkowników z grupy, dla której utworzono login, nie wpływa na sam login.

Procedura sp_addlogin

Wykonanie procedury spowoduje utworzenie loginu użytkownika SQL Servera.
Składnia:

```
sp_addlogin [@loginame =] 'login'
```

[, [@passwd =] 'hasło']

[, [@defdb =] 'baza_danych']

[, [@deflanguage =] 'język']

[, [@sid =] sid]

[, [@encryptopt =] 'opcje']

gdzie:
baza_danych oznacza domyślną bazę danych użytkownika, a więc bazę, z którą spróbuje się on połączyć po uzyskaniu dostępu do serwera,
język określa domyślny język użytkownika identyfikowanego za pomocą loginu, a więc język, w którym będą wyświetlane danemu użytkownikowi, o ile tylko będą dostępne, komunikaty błędów,
opcje mogą przyjmować jedną z trzech wartości: NULL (wartość domyślna) określa, że hasło zostanie zapisane w tabeli sysxlogins w postaci zaszyfrowanej,
skip_encryption określa, że hasło zostanie zapisane jawnym tekstem,
skip_encryption_old określa, że hasło, które w poprzednich wersjach SQL Servera było zaszyfrowane, zostanie zapisane jawnym tekstem.
Przykłady:

sp_addlogin 'jacek'

GO

New login created.

sp_addlogin 'agatka', 'do_bra+no=cka', 'Northwind'

GO

New login created.

1. Maksymalna długość loginu nie może przekroczyć 128 znaków.
2. Login nie może zawierać znaku ukośnika.
3. Login nie może być pustym lub nieokreślonym ciągiem znaków.
4. Login nie może być zastrzeżonym identyfikatorem, np. ciągiem znaków sa.

Procedura sp_password

Każdy użytkownik SQL Servera może zmienić swoje hasło. Administrator (użytkownik należący do roli sysadmin) może zmienić hasło każdemu użytkownikowi, podając NULL jako wartość poprzedniego hasła.
Składnia:

sp_password [[@old =] 'poprzednie_hasło',]

{[@new =] 'nowe_hasło'}

[, [@loginame =] 'login']

Przykład:

sp_password @old = NULL, @new = 'ko_ka^na', @loginame = 'jacek'

GO

Password changed.

Procedura sp_defaultdb

Domyślnie każdy użytkownik (członek roli public) może sobie przypisać wybraną bazę danych jako bazę domyślną. Tylko członkowie ról sysadmin oraz securityadmin mogą przypisywać wszystkim użytkownikom domyślne bazy danych.
Składnia:

```
sp_defaultdb [@loginame =] 'login',
[@defdb =] 'baza_danych'
```

Przykład:

```
sp_defaultdb 'jacek', 'pubs'
GO
Default database changed.
```

Procedura sp_defaultlanguage

Domyślne uprawnienia do zmiany domyślnego języka są takie same jak do zmiany domyślnej bazy danych — każdy użytkownik może zmienić język dla swojego loginu, członkowie ról sysadmin oraz securityadmin mogą zmienić wszystkim użytkownikom domyślny język.
Składnia:

```
sp_defaultlanguage [@loginame =] 'login'
[, [@language =] 'język']
```

Przykład:

```
sp_defaultlanguage 'lolek\klientvpn', 'English'
GO
lolek\klientvpn's default language is changed to English.
```

Opcje loginów

SQL Server umożliwia pobieranie informacji o loginach za pomocą procedur systemowych i predefiniowanych funkcji.

Procedura xp_logininfo

Zwraca komplet informacji o loginie.
Składnia:

```
xp_logininfo [[@acctname =] 'konto_systemowe'] [,[@option =] 'all' | 'members']
[,[@privelege =] zmienna OUTPUT]
```

gdzie:
zmienna jest nazwą zmiennej, do której zostanie zapisany wynik działania procedury.
Przykład:

```
USE master
GO
xp_logininfo 'LOLEK\Administrator'
GO
(1 row(s) affected)
(2 row(s) affected)
account name type privilege mapped login name permission path
-------------------- ----- --------- -------------------- ----------------------------
LOLEK\Administrator user admin LOLEK\Administrator NULL
(1 row(s) affected)
```

Procedura xp_loginconfig

Procedura zwraca informacje o bieżącej konfiguracji autoryzacji użytkowników przez SQL Server.
Składnia:

```
xp_loginconfig ['opcja']
```

gdzie:
opcja jest nazwą parametru, którego wartość ma zostać wyświetlona i może przyjmować jedną z poniższych wartości:

- ♣ login mode określa tryb uwierzytelniania użytkowników.
- ♣ default login jest parametrem wykorzystywanym w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera i określa login, który będzie użyty w przypadku próby nawiązania anonimowego połączenia.
- ♣ default domain jest parametrem wykorzystywanym w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera i określa nazwę domeny, o jaką będą uzupełniane loginy użytkowników próbujących nawiązać połączenie zaufane.
- ♣ audit level określa liczbę informacji zapisywanych do dziennika błędów SQL Servera i dziennika systemowego. Domyślną wartością jest none, co oznacza brak śledzenia udanych oraz nieudanych prób nawiązania połączenia.
- ♣ set hostname powoduje zastępowanie nazwy loginu nazwą konta systemowego, dla którego został utworzony dany login.
- ♣ map _ jest parametrem wykorzystywanym w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera i określa sposób interpretowania przez SQL Server niezgodnych z jego konwencją nazewniczą symboli systemu operacyjnego oraz dokonuje zamiany tych symboli na znak podkreślenia.
- ♣ map $ jest parametrem wykorzystywanym w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera i określa sposób interpretowania przez SQL Server niezgodnych z jego konwencją nazewniczą symboli systemu operacyjnego oraz dokonuje zamiany tych symboli na znak dolara.
- ♣ map # jest parametrem wykorzystywanym w celu zachowania zgodności z wcześniejszymi wersjami SQL Servera i określa sposób interpretowania przez SQL Server niezgodnych z jego konwencją nazewnicza symboli systemu operacyjnego oraz dokonuje zamiany tych symboli na znak #.

Przykład:

```
USE master
GO
xp_loginconfig
GO
name config_value
-------------------------- --------------------------
login mode Mixed
default login guest
default domain LOLEK
```

```
audit level failure
set hostname false
map _ domain separator
map $ NULL
map # -
(8 row(s) affected)
```

Procedury sp_helpuser oraz sp_helplogins zwracające m.in. informacje o loginach zostały opisane w dalszej części rozdziału. Zwracające informacje o loginach i użytkownikach funkcje bezpieczeństwa zostały opisane w rozdziale 2.

Usuwanie loginów

Loginy użytkowników, którym odebrano uprawnienie do korzystania z zasobów SQL Servera, powinny zostać usunięte.

Procedura sp_revokelogin

Wykonanie procedury usuwa login powiązany z użytkownikiem lub grupą użytkowników systemu operacyjnego. Ponieważ konto użytkownika może jednocześnie należeć do wielu grup, usunięcie loginu nie jest jednoznaczne z odmową dostępu danemu użytkownikowi do SQL Servera. Usunięcie loginu użytkownika należącego do grupy, której niezależnie przyznano dostęp do zasobów SQL Servera, spowoduje, że użytkownik, na mocy członkostwa w grupie, pomyślnie nawiąże połączenie z serwerem bazodanowym.

Składnia:

```
sp_revokelogin [@loginame =] 'login'
```

Przykład:

```
sp_revokelogin [lolek\klientvpn]
GO
Revoked login access from 'lolek\klientvpn'.
```

Procedura sp_denylogin

Wykonanie procedury spowoduje odmówienie użytkownikowi lub grupie użytkowników systemu operacyjnego dostępu do zasobów SQL Servera. Ponieważ uprawnienie deny (odmów) nadpisuje wszystkie nadane uprawnienia, użytkownicy którym odmówiono dostępu, nie nawiążą połączenia z SQL Serverem, nawet jeśli należą do grupy lokalnych administratorów.

Składnia:

```
sp_denylogin [@loginame =] 'login'
```

Przykład:

```
sp_denylogin [lolek\klientvpn]
GO
Denied login access to 'lolek\klientvpn'.
```

Procedura sp_droplogin

Uniemożliwić korzystanie z zasobów serwera bazodanowego użytkownikom uwierzytelnianym przez SQL Server możemy poprzez usunięcie danego loginu.

Składnia:

sp_droplogin [@loginame =] 'login'

Przykład:

sp_droplogin @loginame = 'agatka'

GO

Login dropped.

Zarządzanie uprawnieniami użytkowników

Aby uwierzytelnieni użytkownicy mogli połączyć się z wybraną bazą danych, należy w niej utworzyć powiązane z odpowiednim loginem konto użytkownika. Następnym krokiem jest nadanie danemu kontu wymaganych uprawnień do obiektów określonej bazy danych. W większości wypadków uprawnienia nie są nadawane bezpośrednio kontom użytkowników, ale odpowiednikom grup użytkowników systemu operacyjnego — rolom.

Zarządzanie dostępem do baz danych

Aby uwierzytelniony i reprezentowany poprzez login użytkownik mógł połączyć się z wybraną bazą danych, w tabeli sysusers tej bazy musi znajdować się wpis wiążący login z kontem użytkownika. Poniższe zapytanie zwraca listę kont użytkowników bazy Northwind:

SELECT uid, LEFT (name,30), status, hasdbaccess

FROM Northwind.dbo.sysusers

GO

uid status hasdbaccess

------ ------------------------------ ------ -----------

0 public 0 0

16384 db_owner 0 0

16385 db_accessadmin 0 0

16386 db_securityadmin 0 0

16387 db_ddladmin 0 0

16389 db_backupoperator 0 0

16390 db_datareader 0 0

16391 db_datawriter 0 0

16392 db_denydatareader 0 0

16393 db_denydatawriter 0 0

2 guest 2 1

1 dbo 2 1

(12 row(s) affected)

Zapytanie zwróciło listę ról bazy danych oraz nazwy dwóch kont: quest (gość) i dbo (właściciel bazy danych). W każdej bazie danych znajdują się dwa predefiniowane konta użytkowników:

- *dbo*, które jest domyślnym właścicielem wszystkich obiektów danej bazy danych i z którym powiązane są: rola serwera sysadmin oraz login sa. Konto dbo nie może zostać usunięte.
- *guest*, które jest kontem umożliwiającym dostęp do bazy anonimowym użytkownikom nieposiadającym konta w danej bazie danych. W przykładowych bazach danych Northwind i pubs konto quest, jako należące do roli public, ma pełne uprawnienia do wszystkich obiektów tych baz danych. Konto quest może zostać usunięte ze wszystkich baz, z wyjątkiem baz: master i tempdb.

Procedura sp_grantdbaccess

Wykonanie procedury spowoduje dodanie do bieżącej bazy danych konta użytkownika reprezentowanego przez określony login. Domyślnie jedynie członkowie roli serwera sysadmin oraz ról bazy danych db_accessadmin oraz db_owner mogą tworzyć nowe konta użytkowników bazy danych.
Składnia:

```
sp_grantdbaccess [@loginame =] 'login'
[,[@name_in_db =] 'konto_użytkownika' [OUTPUT]]
```

Przykłady:

```
USE Northwind
EXEC sp_grantdbaccess 'jacek'
GO
Granted database access to 'jacek'.
USE Northwind
EXEC sp_grantdbaccess [LOLEK\klienci_SQL], 'klienci_systemowi'
GO
Granted database access to 'LOLEK\klienci_SQL'.
```

Procedura sp_change_users_login

Zmienić powiązania pomiędzy loginami a istniejącymi kontami użytkowników możemy poprzez wykonanie procedury sp_change_users_login.
Składnia:

```
sp_change_users_login [@Action =] 'operacja'
[, [@UserNamePattern =] 'konto_użytkownika']
[, [@LoginName =] 'login']
```

gdzie:
operacja określa operację wykonaną przez procedurę i może przyjąć jedną z poniższych wartości:

- *Auto_Fix* powodującą automatyczne powiązanie loginów z tabeli master.. sysxlogins z mającymi identyczną nazwę kontami użytkowników zapisanych w bieżącej bazie danych w tabeli sysusers.
- *Report* powodującą wyświetlenie informacji o wszystkich kontach użytkowników niepowiązanych z jakimkolwiek loginem.
- *Update_One* powodującą aktualizację powiązania wybranego loginu z określonym kontem użytkownika. W przeciwieństwie do dwóch poprzednich wartości, przeprowadzanie tej operacji wymaga określenia loginu i konta użytkownika.

Przykład:

```
USE Northwind
EXEC sp_addlogin 'agatka', 'do_bra+no=cka', 'Northwind'
GO
EXEC sp_change_users_login 'Update_One', 'jacek', 'agatka'
GO
```

```
New login created.
The number of orphaned users fixed by updating users was 1.
```

Procedura sp_helplogins

Wykonanie procedury zwraca informacje o wszystkich loginach lub o wybranym loginie oraz o powiązanych z nimi kontach użytkowników.

Składnia:

```
sp_helplogins [[@LoginNamePattern =] 'login']
```

Przykład:

```
USE Northwind
EXEC sp_helplogins 'agatka'
GO
LoginName SID DefDBName DefLangName AUser ARemote
------------ ---------------------------------- ---------- ------------- ----- -------
agatka 0x7D23FF0970D03948B4BA4594453735AC Northwind us_english yes no
(1 row(s) affected)
LoginName DBName UserName UserOrAlias
------------ ------------------ ---------- -----------
agatka Northwind jacek User
(1 row(s) affected)
```

Procedura sp_revokedbaccess

Wykonanie procedury spowoduje usunięcie konta użytkownika z bieżącej bazy danych.

Składnia:

```
sp_revokedbaccess [@name_in_db =] 'konto_użytkownika'
```

Przykład:

```
USE Northwind
EXEC sp_revokedbaccess 'klienci_systemowi'
GO
User has been dropped from current database.
```

Uprawnienia użytkownika bazy danych

Użytkownik może wykonać tylko takie operacje, do których wykonania nadano mu uprawnienia. Lista uprawnień poszczególnych użytkowników baz danych znajduje się w tabeli sysprotects. Poniższe zapytanie zwraca listę nadanych wszystkim użytkownikom (identyfikator 0 oznacza rolę bazy danych public) uprawnień do tabeli Categories. Zapisane w kolumnie action uprawnienia oznaczają odpowiednio: 26 — DRI, 193 — SELECT, 195 — INSERT, 196 — DELETE, 197 — UPDATE.

```
USE Northwind
```

```
SELECT *
FROM dbo.sysprotects
WHERE OBJECT_NAME (id) = 'Categories'
GO
id uid action protecttype columns grantor
----------- ------ ------ ----------- ---------- ------------------------------
917578307 0 26 205 0x01 1
917578307 0 193 205 0x01 1
917578307 0 195 205 NULL 1
917578307 0 196 205 NULL 1
917578307 0 197 205 0x01 1
(5 row(s) affected)
```

Role standardowe

Role SQL Servera pełnią funkcję analogiczną do grup systemu operacyjnego — upraszczają zarządzanie uprawnieniami użytkowników poprzez możliwość pogrupowania loginów i kont użytkowników w logiczne jednostki (role) oraz poprzez nadanie uprawnień rolom, a nie poszczególnym użytkownikom.

Wyróżnia się trzy typy ról standardowych:

1. Predefiniowane role serwera.
2. Predefiniowane role bazy danych.
3. Tworzone przez użytkownika role bazy danych.

Predefiniowane role serwera

Role serwera grupują loginy użytkowników, którzy mają takie same uprawnienia w zakresie całego serwera bazodanowego. Ponieważ role są zbiorem loginów, a członkowie ról serwera mogą wykonywać określone akcje w każdej bazie danych, informacje o rolach przechowywane są w tabeli bazy master — sysxlogins.

> Predefiniowanych ról serwera nie można modyfikować ani usuwać. Niemożliwe jest też tworzenie własnych ról serwera.

Podczas instalacji SQL Servera tworzone są następujące role serwera:

1. *bulkadmin* — członkowie roli mogą przeprowadzać masowe ładowanie danych (wykonać instrukcję BULK INSERT).
2. *dbcreator* — członkowie roli mogą tworzyć i modyfikować bazy danych.
3. *diskadmin* — członkowie roli mogą zarządzać plikami i urządzeniami pamięci masowej.
4. *processadmin* — członkowie roli mogą zarządzać procesami SQL Servera (wykonać instrukcję KILL).
5. *securityadmin* — członkowie roli mogą zarządzać dostępem użytkowników do SQL Servera.
6. *serveradmin* — członkowie roli mogą konfigurować parametry pracy serwera bazodanowego, w tym zatrzymywać serwer.
7. *setupadmin* — członkowie roli mogą zarządzać replikacją danych pomiędzy serwerami.
8. *sysadmin* — członkowie roli są administratorami serwera bazodanowego i mogą wykonywać każdą operację.

> W każdym przypadku użytkownik należący do dowolnej z ról serwera może dodać do tej roli innych użytkowników.

Procedura sp_helpsrvrole

Wykonanie procedury zwraca listę wszystkich predefiniowanych ról serwera wraz z ich opisem.

Składnia:

```
sp_helpsrvrole [[@srvrolename =] 'rola']
```

Przykład:

```
sp_helpsrvrole 'serveradmin'
GO
ServerRole Description
---------------------------------- ----------------------------------
serveradmin Server Administrators
(1 row(s) affected)
```

Procedura sp_srvrolepermission

Listę uprawnień przydzielonych poszczególnym predefiniowanym rolom serwera możemy poznać, wykonując procedurę sp_srvrolepermission.

Składnia:

```
sp_srvrolepermission [[@srvrolename =] 'rola']
```

Przykład:

```
sp_srvrolepermission 'diskadmin'
GO
ServerRole Permission
---------------------------------- ----------------------------------
diskadmin Add member to diskadmin
diskadmin DISK INIT
diskadmin sp_addumpdevice
diskadmin sp_diskdefault
diskadmin sp_dropdevice
(5 row(s) affected)
```

Procedura sp_addsrvrolemember

Wykonanie procedury spowoduje dodanie loginu do określonej roli serwera. Procedura sp_addsrvrolemember, tak jak pozostałe procedury związane z bezpieczeństwem SQL Servera, nie może zostać wywołana w ramach rozpoczętej transakcji.

Składnia:

```
sp_addsrvrolemember [@loginame =] 'login'
, [@rolename =] 'rola'
```

Przykład:

```
sp_addsrvrolemember 'jacek', 'dbcreator'
GO
'jacek' added to role 'dbcreator'.
```

Procedura sp_dropsrvrolemember

Login użytkownika z wybranej roli możemy usunąć, wykonując procedurę sp_dropsrvrolemember.
Składnia:

```
sp_dropsrvrolemember [@loginame =] 'login'
, [@rolename =] 'rola'
```

Przykład:

```
sp_dropsrvrolemember @loginame = 'jacek',
@rolename = 'dbcreator'
GO
'jacek' dropped from role 'dbcreator'.
```

Procedura sp_helpsrvrolemember

Poznać listę loginów należących do wybranej roli serwera możemy za pomocą procedury sp_helpsrvrolemember.
Składnia:

```
sp_helpsrvrolemember [[@srvrolename =] 'rola']
```

Przykład:

```
sp_helpsrvrolemember 'sysadmin'
GO
ServerRole MemberName MemberSID
---------- ---------- ----------------------------------
sysadmin BUILTIN\Administratorzy 0x0102000000000005200000020020000
sysadmin LOLEK\Administrator 0x0105000000000005150000000A1F40462625
??CBC0616C0EA32F4010000
sysadmin sa 0x01
(3 row(s) affected)
```

Predefiniowane role bazy danych

Role bazy danych grupują konta użytkowników danej bazy danych o takich samych uprawnieniach i zapisane są w tabeli sysusers.
Podczas instalacji w każdej bazie danych tworzone są następujące role:

1. *db_accessadmin* — członkowie roli mogą zarządzać dostępem użytkowników do bazy danych.
2. *db_backupoperator* — członkowie roli mogą tworzyć kopie zapasowe bazy danych.
3. *db_datareader* — członkowie roli mogą odczytywać dane zapisane w dowolnej tabeli bazy danych.
4. *db_datawriter* — członkowie roli mogą modyfikować dane zapisane w dowolnej tabeli bazy danych.
5. *db_ddladmin* — członkowie roli mogą tworzyć, modyfikować i usuwać obiekty bazodanowe.
6. *db_denydatareader* — członkowie roli nie mogą odczytywać danych zapisanych w jakiejkolwiek tabeli bazy danych.
7. *db_denydatawriter* — członkowie roli nie mogą modyfikować danych zapisanych w jakiejkolwiek tabeli bazy danych.
8. *db_owner* — członkowie roli są administratorami bazy danych i mogą w niej wykonywać każdą operację.
9. *db_security admin* — członkowie roli mogą nadawać i odbierać uprawnienia użytkownikom.

10. *public* — specjalna rola serwera, której członkami są wszyscy użytkownicy bazy danych. Do roli public należą wszyscy użytkownicy bazy danych, rola ta nie może zostać usunięta, a jej członkowie domyślnie mogą jedynie wykonywać instrukcje, do wykonania których nie potrzebne są żadne uprawnienia (przykładem takiej instrukcji jest instrukcja PRINT).

W każdym przypadku użytkownik należący do dowolnej spośród ról bazy danych może dodać do tej roli konta innych użytkowników.

Procedura sp_helprole

Procedura zwraca informacje o wszystkich lub wybranej roli bazy danych.
Składnia:

```
sp_helprole [[@rolename =] 'rola']
```

Przykład:

```
USE Northwind
EXEC sp_helprole 'public'
GO
RoleName RoleId IsAppRole
------------------------------------------------------------------ ------ -----------
public 0 0
(1 row(s) affected)
```

Procedura sp_addrolemember

Wykonanie procedury spowoduje dodanie określonego konta użytkownika do wybranej roli.
Składnia:

```
sp_addrolemember [@rolename =] 'rola'
, [@membername =] 'konto_użytkownika'
```

Przykład:

```
USE Northwind
EXEC sp_addrolemember 'db_datareader', 'jacek'
GO
'jacek' added to role 'db_datareader'.
```

Procedura sp_droprolemember

Usunąć konta użytkowników z wybranej roli możemy za pomocą procedury sp_droprolemember.
Składnia:

```
sp_droprolemember [@rolename =] 'rola'
, [@membername =] 'konto_użytkownika'
```

Przykład:

```
USE Northwind
EXEC sp_droprolemember 'db_datareader', 'jacek'
GO
'jacek' dropped from role 'db_datareader'.
```

Procedura sp_helprolemember

Procedura zwraca informacje o kontach użytkowników należących do określonej roli bazy danych.
Składnia:

```
sp_helprolemember [[@rolename =] 'rola']
```

Przykład:

```
USE Northwind
EXEC sp_helprolemember 'db_owner'
GO
DbRole MemberName MemberSID
---------- ------------------------ ------------------------------------------
db_owner dbo 0x01
(1 row(s) affected)
```

Procedura sp_helpuser

Procedura zwraca informacje o loginach powiązanych z użytkownikami bieżącej bazy danych oraz o kontach użytkowników i rolach istniejących w bazie.
Składnia:

```
sp_helpuser [[@name_in_db =] 'konto_użytkownika']
```

Przykład:

```
USE Northwind
EXEC sp_helpuser
GO
UserName GroupName LoginName DefDBName UserID SID
---------- ---------------- ------------ ------------------ ------ ----------
dbo db_owner sa master 1 0x01
guest public NULL NULL 2 0x00
jacek public agatka Northwind 5 0x
7D23FF0970D03948B4BA4594453735AC
```

Tworzone przez użytkownika role bazy danych

Tworzenie własnych ról bazy danych pozwala na utworzenie grupy użytkowników, którym należy nadać takie same uprawnienia i w praktyce stosowane jest w dwóch przypadkach:

1. Gdy użytkownicy uwierzytelniani są poprzez SQL Server.
2. Gdy nie ma możliwości utworzenia grupy systemu operacyjnego, do której należeliby określeni użytkownicy.

Do roli utworzonej przez użytkownika można w ten sam sposób dodawać lub usuwać z niej konta użytkowników.

Procedura sp_addrole

Wykonanie procedury spowoduje dodanie nowej roli bazy danych.
Składnia:

```
sp_addrole [@rolename =] 'rola'
[, [@ownername =] 'właściciel']
```

Przykład:

```
USE Northwind
EXEC sp_addrole 'postacie_kreskówek'
GO
New role added.
```

Procedura sp_droprole

Po usunięciu z roli wszystkich kont użytkowników możliwe jest usunięcie samej roli.
Składnia:

```
sp_droprole [@rolename =] 'rola'
```

Przykład:

```
USE Northwind
EXEC sp_droprole 'postacie_kreskówek'
GO
Role dropped.
```

Role aplikacyjne

Role aplikacyjne umożliwiają:

- ograniczenie dostępu do wybranych obiektów,
- ograniczenie możliwości wykonywania określonych operacji poprzez udostępnienie jej tylko tym użytkownikom, którzy łączą się z serwerem bazodanowym za pośrednictwem wybranej aplikacji klienckiej.

Na przykład możliwe jest nadanie prawa do odczytu i modyfikacji zawartości tabeli tajne wyłącznie użytkownikom, którzy połączyli się za pośrednictwem autorskiego programu klienckiego o nazwie herkules.

W przeciwieństwie do ról standardowych, rolom aplikacyjnym nie przypisuje się użytkowników — uprawnienia roli aplikacyjnej nadawane są automatycznie w momencie wykonania, z reguły wywoływanej podczas nawiązywania połączenia, procedury systemowej sp_setapprole. W momencie uaktywnienia roli aplikacyjnej użytkownik traci wszystkie nadane mu uprawnienia, zyskując w zamian uprawnienia nadane roli aplikacyjnej. Ponadto role aplikacyjne:

1. Wymagają podania prawidłowego hasła podczas aktywacji.
2. Nie umożliwiają wykonania instrukcji USE DATABASE (jedynym sposobem na pobranie danych zapisanych w innej bazie danych lub przechowywanych na innym serwerze jest odwołanie się do nich za pomocą pełnej nazwy obiektu, a więc poprzedzenie nazwy obiektu nazwą serwera lub bazy danych).
3. Nie mogą zostać wyłączone (jedynym sposobem na odzyskanie uprawnień nadanych użytkownikowi jest rozłączenie się i ponowne połączenie z serwerem bazodanowym).

Procedura sp_addapprole

Utworzyć rolę aplikacyjną możemy, wykonując procedurę sp_addapprole. Po utworzeniu roli aplikacyjnej nadaje się jej uprawnienia w taki sam sposób (opisany w dalszej części rozdziału) jak roli standardowej lub użytkownikowi.
Składnia:

```
sp_addapprole [@rolename =] 'rola'
, [@password =] 'hasło'
```

gdzie:
hasło nie może być nieokreślone.
Przykład:

```
USE Northwind
EXEC sp_addapprole 'herkules', '5A647A6972612058656E6E610D0A'
GO
New application role added.
```

Procedura sp_setapprole

Aby użytkownik mógł uzyskać uprawnienia nadane roli aplikacyjnej, musi wykonać procedurę sp_setapprole i podać prawidłowe hasło roli (z reguły obie czynności wykonywane są automatycznie przez program kliencki). Ustawiając tożsamość roli aplikacyjnej należy o tym pamiętać, że procedura sp_setapprole musi zostać wywołana bezpośrednio z poziomu języka Transact-SQL, niemożliwe jest jej wywołanie w ramach innej procedury czy w ramach rozpoczętej transakcji użytkownika.
Składnia:

```
sp_setapprole [@rolename =] 'rola',
[@password =] {N'hasło'} | 'hasło'
[,[@encrypt =] 'metoda']
```

gdzie:
metoda określa metodę szyfrowania przesyłanego do SQL Servera hasła i może przyjmować jedną z dwóch wartości:

1. none — wartość domyślna, hasło będzie przesłane jawnym tekstem.
2. odbc — hasło zostanie zaszyfrowane za pomocą standartowej funkcji ODBC Encrypt. Aby hasło mogło zostać zaszyfrowane, musi zostać zapisane w postaci ciągu znaków UNICODE przez aplikację łączącą się z SQL Serverem poprzez ODBC lub OLE DB.

Przykład:

```
USE Northwind
EXEC sp_setapprole 'herkules', '5A647A6972612058656E6E610D0A'
GO
The application role 'herkules' is now active.
```

Procedura sp_approlepassword

Wywołanie procedury spowoduje zmianę hasła dla istniejącej w bazie danych roli aplikacyjnej. Domyślnie jedynie użytkownicy należący do roli serwera sysadmin oraz ról bazy danych db_securityadmin i db_owner mogą zmienić hasło roli aplikacyjnej.

Składnia:

```
sp_approlepassword [@rolename =] 'rola', [@newpwd =] 'hasło'
```

gdzie:

hasło nie może być nieokreślone.

Przykład:

```
USE Northwind
EXEC sp_approlepassword @rolename = 'herkules',
@newpwd = 'to_nie_prawda'
GO
(1 row(s) affected)
The password for application role 'herkules' has been changed.
```

Procedura sp_dropapprole

Usunąć rolę aplikacyjną możemy, wykonując procedurę sp_dropapprole. Jeżeli rola jest właścicielem jakiegokolwiek obiektu, jej usunięcie będzie niemożliwe. W takim wypadku należy wcześniej usunąć obiekt albo zmienić jego właściciela, wykonując procedurę sp_changeobjectowner.

Składnia:

```
sp_dropapprole [@rolename =] 'rola'
```

Przykład:

```
USE Northwind
EXEC sp_dropapprole 'herkules'
GO
Application role dropped.
```

Przypisywanie uprawnień użytkownikom i rolom

Uwierzytelniony, posiadający konto w wybranej bazie danych użytkownik, może jedynie połączyć się z tą bazą i wykonać te operacje, do których wykonania nie są potrzebne żadne uprawnienia oraz te, do których wykonania nadano uprawnienia roli public. Zadaniem administratora bazy danych jest nadanie odpowiednich uprawnień poszczególnym rolom oraz użytkownikom.

Każda baza danych posiada własny, niezależny od uprawnień zdefiniowanych dla pozostałych baz danych i zapisany w tabeli systemowej sysprotects, zbiór uprawnień użytkowników.

Wyróżnia się trzy typy uprawnień:

1. Uprawnienia do wykonania instrukcji języka Transact-SQL — na ich podstawie użytkownicy mogą wykonać jedną lub kilka z poniższych instrukcji:
   - CREATE DATABASE,
   - CREATE TABLE,
   - CREATE VIEW,
   - CREATE PROCEDURE,
   - CREATE RULE,

- ♣ CREATE DEFAULT,
- ♣ CREATE FUNCTION,
- ♣ BACKUP DATABASE,
- ♣ BACKUP LOG.

2. Uprawnienia obiektowe — na ich podstawie użytkownicy mogą odwoływać się i wykonywać określone operacje związane z wybranymi obiektami bazy danych. Możliwe do nadania uprawnienia związane są z typem obiektu:
   - ♣ Do tabel oraz widoków można nadać uprawnienia SELECT, INSERT, UPDATE, DELETE oraz DRI umożliwiające odpowiednio: pobieranie, wstawianie, modyfikowanie, usuwanie danych oraz sprawdzanie warunków referencyjnej spójności (wartości przechowywanych w powiązanych polach innych tabel).
   - ♣ Do kolumn można nadać uprawnienia SELECT oraz UPDATE,
   - ♣ Do procedur składowanych oraz funkcji można nadać uprawnienie EXEC.
3. Uprawnienia predefiniowane — użytkownicy należący do predefiniowanych ról serwera lub bazy danych oraz właściciele obiektów bazodanowych na podstawie predefiniowanych (niejawnych) uprawnień mogą dodatkowo wykonywać określone operacje:
   - ♣ Predefiniowane role serwera umożliwiają wykonanie określonych czynności administracyjnych w dowolnej bazie danych, na przykład członkowie roli sysadmin mogą wykonać każdą operację we wszystkich bazach danych.
   - ♣ Użytkownik należący do roli serwera może wykonać wszystkie operacje, do których przeprowadzenia nadano uprawnienia tej roli.
   - ♣ Właściciel danego obiektu może wykonać każdą operację związaną z tym obiektem.

Instrukcja GRANT

Nadać uprawnienia rolom lub użytkownikom możemy za pomocą instrukcji języka Transact-SQL GRANT.
Składnia:

Uprawnienia do wykonania instrukcji:

```
GRANT {ALL | instrukcja [,…n]}
TO użytkownik [,…n]
```

Uprawnienia obiektowe:

```
GRANT
{ALL [PRIVILEGES] | uprawnienie [,…n]}
{
[(kolumna [,…n])] ON {tabela | widok}
| ON {tabela | widok} [(kolumna [,…n])]
| ON {procedura}
| ON {funkcja}
}
TO użytkownik [,…n]
[WITH GRANT OPTION]
[AS {grupa | rola}]
```

gdzie:
ALL oznacza wszystkie możliwe uprawnienia danego typu.
instrukcja określa instrukcję języka Transact-SQL, do której wykonania nadaje się użytkownikowi uprawnienia. Może być jedną lub kilkoma z wymienionych na poprzedniej stronie instrukcji.
użytkownik określa, komu zostaną nadane uprawnienia i może wskazywać na:

- ♣ konto użytkownika bazy danych,
- ♣ rolę bazy danych,
- ♣ konto użytkownika systemu operacyjnego,
- ♣ grupę użytkowników systemu operacyjnego.

WITH GRANT OPTION może być użyta wyłącznie w przypadku nadawania uprawnień obiektowych i przekazuje użytkownikowi prawo do nadania otrzymanego uprawnienia innym użytkownikom.

AS grupa | *rola* określa nazwę grupy lub roli w bieżącej bazie danych, której członkom nadano uprawnienie do wykonywania instrukcji GRANT. Ponieważ tylko użytkownik może wykonać instrukcję GRANT, jeżeli uprawnienia do jej wykonania nadane są roli, należy podczas jej wykonywania powołać się na uprawnienia nadane określonej roli.
Przykład:

```
USE Northwind
GRANT SELECT, INSERT
ON dbo.Categories
TO jacek
WITH GRANT OPTION
GO
The command(s) completed successfully.
```

Wynikowe uprawnienia użytkownika, tak jak w przypadku uprawnień NTFS, są sumą wszystkich uprawnień nadanych temu użytkownikowi, zarówno bezpośrednio, jak poprzez jego członkostwo w określonych rolach. Wyjątkiem od tej zasady, znowu analogicznie do uprawnień NTFS, jest jawne odebranie danego uprawnienia (wykonanie instrukcji DENY), które zastępuje posiadane uprawnienia. Poniższe instrukcje dodają nową rolę bazy Northwind, do nowoutworzonej roli dodają użytkownika jacek i nadają jej uprawnienie do usuwania danych z tabeli Categories:

```
USE Northwind
EXEC sp_addrole 'jacki_agatki'
EXEC sp_addrolemember 'jacki_agatki', 'jacek'
GRANT DELETE
ON dbo.Categories
TO jacki_agatki
GO
New role added.
'jacek' added to role 'jacki_agatki'.
```

Po wykonaniu skryptu lista uprawnień do tabeli Categories powinna wyglądać następująco (wartość 205 zapisana w kolumnie protecttype oznacza nadane uprawnienie, wartość 204 — nadane uprawnienie z możliwością przekazania tego uprawnienia innym użytkownikom):

```
USE Northwind
SELECT LEFT (USER_NAME (uid),20), action, protecttype
FROM dbo.sysprotects
WHERE OBJECT_NAME (id) = 'Categories'
GO
action protecttype
-------------------- ------ -----------
public 26 205
public 193 205
public 195 205
public 196 205
public 197 205
jacek 193 204
jacek 195 204
```

```
jacki_agatki 196 205
(8 row(s) affected)
```

Alternatywną metodą pobrania listy uprawnień nadanych danemu użytkownikowi lub uprawnień do określonego obiektu jest wywołanie procedury systemowej sp_helprotect.

Procedura sp_helprotect

Procedura zwraca listę uprawnień danego użytkownika lub uprawnień związanych z danym obiektem.
Składnia:

```
sp_helprotect [[@name =] 'obiekt_lub_instrukcja']
[, [@username =] 'użytkownik']
[, [@grantorname =] 'przyznający_uprawnienie']
[, [@permissionarea =] 'typ']
```

gdzie:
obiekt_lub_instrukcja określa albo obiekt bazy danych, albo jedno z uprawnień do wykonywania instrukcji języka Transact-SQL.
Przykłady:

```
USE Northwind
EXEC sp_helprotect 'Categories'
GO
Owner Object Grantee Grantor ProtectType Action Column
------ ------------ ------------------------ ------- ----------- ---------- -----------
dbo Categories jacek dbo Grant_WGO Insert .
dbo Categories jacek dbo Grant_WGO Select (All+New)
dbo Categories jacki_agatki dbo Grant Delete .
dbo Categories public dbo Grant Delete .
dbo Categories public dbo Grant Insert .
dbo Categories public dbo Grant References (All+New)
dbo Categories public dbo Grant Select (All+New)
dbo Categories public dbo Grant Update (All+New)
(8 row(s) affected)
USE Northwind
EXEC sp_helprotect NULL, 'jacek'
GO
Owner Object Grantee Grantor ProtectType Action Column
------ --------------------- ---------- ------- ----------- ------ ------------------
dbo Categories jacek dbo Grant_WGO Insert .
dbo Categories jacek dbo Grant_WGO Select (All+New)
(2 row(s) affected)
```

Ponieważ jacek należy zarówno do roli jacki_agatki, jak i do roli public, posiada on uprawnienia do pobierania, modyfikowania i usuwania danych przechowywanych w tabeli Categories. Wykonując ćwiczenia z dalszej części

rozdziału przekonamy się, jak na wynikowe uprawnienia użytkownika wpłynie usunięcie uprawnień roli public, co jest sytuacją typową dla produkcyjnych baz danych.

Instrukcja DENY

Wykonanie instrukcji spowoduje jawne odebranie określonych uprawnień użytkownikowi. Ponieważ jawne odebranie uprawnień nadpisuje posiadane uprawnienia, odebranie jakichkolwiek uprawnień grupie public spowoduje niemożność wykonania danej operacji przez kogokolwiek, z wyjątkiem właściciela bazy danych.
Składnia:

```
Uprawnienia do wykonania instrukcji:
DENY {ALL | instrukcja [,…n]}
TO użytkownik [,…n]
Uprawnienia obiektowe:
DENY
{ALL [PRIVILEGES] | uprawnienie [,…n]}
{
[(kolumna [,…n])] ON {tabela | widok}
| ON {tabela | widok} [(kolumna [,…n])]
| ON {procedura}
}
TO użytkownik [,…n]
[CASCADE]
```

gdzie:
CASCADE powoduje odebranie uprawnień użytkownikowi i wszystkim użytkownikom, którzy otrzymali od niego dane uprawnienie i jest wymaganą klauzulą podczas odbierania uprawnień nadanych z klauzulą WITH GRANT OPTION.
Przykład:

```
USE Northwind
SELECT *
INTO c
FROM dbo.Categories
DENY ALL
ON c
TO public
GO
(8 row(s) affected)
```

Ponieważ odebraliśmy uprawnienia do pobierania, modyfikowania i usuwania danych z tabeli c wszystkim użytkownikom, próba wykonania poniższego zapytania przez użytkownika jacek zakończy się błędem:

```
USE Northwind
PRINT USER
SELECT *
FROM c
GO
```

```
jacek
Server: Msg 229, Level 14, State 5, Line 1
SELECT permission denied on object 'c', database 'Northwind', owner 'dbo'.
```

Po połączeniu się z bazą jako członek roli sysadmin możemy wyświetlić listę uprawnień nadanych do tabeli c (wartość 206 zapisana w kolumnie protecttype oznacza odebranie danego uprawnienia):

```
SELECT LEFT (USER_NAME (uid),20), action, protecttype
FROM Northwind.dbo.sysprotects
WHERE OBJECT_NAME (id) = 'C'
GO
action protecttype
-------------------- ------ -----------
public 26 206
public 193 206
public 195 206
public 196 206
public 197 206
(5 row(s) affected)
```

Instrukcja REVOKE

Usunąć wcześniej nadane lub odebrane uprawnienia możemy poprzez wykonanie instrukcji REVOKE. Ponieważ brak nadanych uprawnień uniemożliwia wykonanie operacji, użytkownicy, którym odebrano uprawnienia, będą mogli wykonać daną operację tylko w przypadku, gdy należą do roli, której nadano dane uprawnienie.
Składnia:

```
Uprawnienia do wykonania instrukcji:
REVOKE {ALL | instrukcja [,…n]}
FROM użytkownik[,…n]
Uprawnienia obiektowe:
REVOKE [GRANT OPTION FOR]
{ALL [PRIVILEGES] | uprawnienie [,…n]}
{
[(kolumna [,…n])] ON {tabela | widok}
| ON {tabela | widok} [(kolumna [,…n])]
| {procedura}
}
{TO | FROM}
użytkownik [,…n]
[CASCADE]
[AS {grupa | rola}]
```

Przykłady:

```
USE Northwind
REVOKE ALL
ON dbo.Categories
FROM public
GO
The command(s) completed successfully.

USE Northwind
REVOKE SELECT
ON dbo.Categories
FROM jacek
CASCADE
GO
The command(s) completed successfully.
```

W wyniku wykonania instrukcji odebraliśmy wszystkim użytkownikom (roli public) posiadane przez nich uprawnienia do tabeli Categories oraz odebraliśmy użytkownikowi jacek nadane uprawnienie do pobierania danych z tabeli. W rezultacie jedynie jacek ma nadane uprawnienia do dodawania danych do tabeli, a członkowie roli jacki_agatki — do usuwania danych z tabeli:

```
USE Northwind
EXEC sp_helprotect 'Categories'
GO
Owner Object Grantee Grantor ProtectType Action Column
------ -------------------- ------------------------ ------- ----------- ------ ------
dbo Categories jacek dbo Grant_WGO Insert .
dbo Categories jacki_agatki dbo Grant Delete .
(2 row(s) affected)
```

Właściciel obiektu

Każdy obiekt bazodanowy ma swojego właściciela, który ma nieograniczone uprawnienia do obiektu. Z reguły właścicielem obiektu jest użytkownik, który go utworzył, ale możliwe jest, poprzez poprzedzenie nazwy obiektu nazwą właściciela, jawne określenie właściciela nowo utworzonego obiektu. Możliwa jest również zmiana właściciela istniejącego obiektu.

> Jeżeli tylko nie jest to sprzeczne z polityką firmy, właścicielem wszystkich obiektów bazodanowych należy uczynić tego samego użytkownika dbo.

Z możliwości określenia właściciela nowo utworzonego obiektu bazodanowego powinny skorzystać te osoby, które nie należą do roli sysadmin, a mają nadane uprawnienia do wykonywania instrukcji CREATE (obiekty utworzone przez członków tej roli stają się własnością użytkownika dbo). W ten sposób zapobiega się powstawaniu obiektów należących do różnych właścicieli.

Poniższe instrukcje nadają prawo do tworzenia widoków użytkownikom należącym do roli jacki_agatki, a następnie tworzą widok, którego właścicielem zostanie jacek:

```
USE Northwind
GRANT CREATE VIEW
```

```
TO jacki_agatki
GO
CREATE VIEW jacek.w_customer AS
SELECT CompanyName,ContactName
FROM dbo.Customers
WHERE Region IS NULL
GO
The command(s) completed successfully.
```

Procedura sp_changeobjectowner

Wykonanie procedury spowoduje zmianę właściciela wybranego obiektu bazodanowego.
Składnia:

```
sp_changeobjectowner [@objname =] 'obiekt', [@newowner =] 'właściciel'
```

Przykład:

```
USE Northwind
EXEC sp_changeobjectowner 'jacek.w_customer', 'dbo'
GO
Caution: Changing any part of an object name could break scripts and stored procedures.
```

Po zmianie właściciela obiektu konieczna jest zmiana nie tylko wszystkich odwołań do tego obiektu, ale również nadanie odpowiednich uprawnień do obiektu.

Ograniczanie uprawnień za pomocą widoków i procedur składowanych

Chociaż SQL Server umożliwia nadanie uprawnień do odczytywania oraz aktualizacji poszczególnych kolumn tabeli, jeżeli administrator chce ograniczyć użytkownikom dostęp do wybranych kolumn (na przykład umożliwić poznanie imion i nazwisk, ale nie pensji pracowników), z reguły, zamiast nadawać uprawnienia na poziomie poszczególnych kolumn, tworzy widok lub procedurę pobierające wybrane dane z tabeli źródłowej i nadaje odpowiednie uprawnienia jedynie do widoku lub procedury, uniemożliwiając bezpośrednie pobieranie danych z tabeli.

Jeżeli tylko właścicielem obu obiektów — tabeli źródłowej i pobierających dane z tabeli procedury czy widoku — jest ta sama osoba, SQL Server sprawdza wyłącznie uprawnienia nadane do obiektu, do którego odwołał się użytkownik. W rezultacie możliwe jest odebranie uprawnień do tabel źródłowych i umożliwienie pobierania czy modyfikowania danych wyłącznie poprzez widoki czy procedury. Jednak jeżeli właścicielem widoku czy procedury będzie inny użytkownik niż właściciel tabeli źródłowej, próba pobrania danych poprzez widok spowoduje sprawdzenie uprawnień nie tylko do widoku, ale również do wszystkich tabel źródłowych.

Dodatkową korzyścią wynikającą ze stosowania widoków lub procedur jest ukrycie przed użytkownikami często skomplikowanej lub poufnej struktury tabel — użytkownik, zamiast wybierać za pomocą skomplikowanego złączenia dane przechowywane w pięciu tabelach, będzie mógł wykonać prostą instrukcję SELECT pobierającą dane z widoku.

Poniższe instrukcje tworzą widok pobierający dane z dwóch tabel i nadają użytkownikom należącym do roli jacki_agatki uprawnienie do pobierania danych poprzez ten widok. Zarówno widok, jak i tabele źródłowe należą do tego samego użytkownika dbo:

```
USE Northwind
GO
CREATE VIEW dbo.w_Categories AS
SELECT CategoryName, COUNT (ProductID) AS liczba_produktow
```

```
FROM dbo.Categories JOIN dbo.Products
ON dbo.Categories.CategoryID = dbo.Products.CategoryID
GROUP BY CategoryName
GO
GRANT SELECT
ON w_Categories
TO jacki_agatki
GO
The command(s) completed successfully.
```

Następnie łączymy się z SQL Serverem jako użytkownik należący do tej roli i próbujemy kolejno pobrać dane z tabel źródłowych oraz poprzez widok:

```
SELECT CategoryName, COUNT (ProductID) AS liczba_produktow
FROM dbo.Categories JOIN dbo.Products
ON dbo.Categories.CategoryID = dbo.Products.CategoryID
GROUP BY CategoryName
GO
Server: Msg 229, Level 14, State 5, Line 1
SELECT permission denied on object 'Categories', database 'Northwind', owner 'dbo'.
SELECT *
FROM dbo.w_Categories
GO
CategoryName liczba_produktow
--------------- ----------------
Beverages 12
Condiments 12
Confections 13
dairy products 10
Grains/Cereals 7
Meat/Poultry 6
Produce 5
Seafood 12
(8 row(s) affected)
```

Kurs Transact-SQL, część 17 (Automatyzacja typowych zadań administracyjnych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Typowe zadania administratora SQL Servera można podzielić na dwie grupy:

1. Systematycznie wykonywane zadania, takie jak tworzenie kopii zapasowych czy import danych.
2. Zadania wykonywane w reakcji na występujące lub potencjalne problemy w pracy serwera bazodanowego, takie jak usuwanie nieaktywnej części wypełnionego dziennika transakcyjnego bazy danych czy zwolnienie zablokowanych przez użytkownika zasobów.

Usługą, która umożliwia automatyzację zadań obu typów jest opisana w rozdziale 15. usługa SQLServerAgent. Ponieważ spora część automatycznie wykonywanych zadań związana jest z powiadamianiem operatorów za pośrednictwem poczty elektronicznej lub wykonywaniem przesyłanych przez wiadomości e-mail instrukcji języka Transact-SQL, do ich wykonania wymagana jest poprawna konfiguracja usług SQLAgentMail oraz SQL Mail. Dodatkowo publikowanie bądź modyfikowanie danych opublikowanych na stronach WWW wymaga odpowiedniej konfiguracji serwera WWW firmy Microsoft (serwera IIS), a zdalne zarządzanie serwerem — połączenia serwerów bazodanowych.

Integracja SQL Servera z serwerem poczty elektronicznej

SQL Server korzysta z usługi SQLAgentMail (pomocniczej usługi SQLServerAgenta) do wysyłania wiadomości e-mail do określonych operatorów. Wiadomości wysyłane przez usługę z reguły informują o pomyślnym lub niepomyślnym zakończeniu wykonywania zadania lub o wystąpieniu określonego alarmu.

Próba uruchomienia usługi SQLAgentMail podejmowana jest za każdym razem, kiedy uruchomiana jest usługa SQLServerAgent i jej błędna konfiguracja (np. niemożność zlokalizowania serwera pocztowego) jest jedną z częstszych przyczyn problemów z uruchomieniem usługi SQLServerAgent.

> Aby możliwe było wysyłanie wiadomości do operatorów, SQLServerAgent musi działać w kontekście zabezpieczeń użytkownika domeny, który posiada konto na serwerze poczty elektronicznej.

Usługa SQL Mail odpowiada za przetwarzanie i wysyłanie wiadomości e-mail za pomocą predefiniowanego zbioru procedur rozszerzonych. Umożliwia ona m.in. przetwarzanie pojedynczych, przesłanych w postaci wiadomości e-mail instrukcji języka Transact-SQL oraz wysyłanie, na przykład w momencie wywołania wyzwalacza, wiadomości przez SQL Server.

> Aby możliwe było wysyłanie i przetwarzanie wiadomości, usługa SQLServer musi działać w kontekście zabezpieczeń użytkownika domeny, który posiada konto na serwerze poczty elektronicznej.

Konfiguracja usług SQLAgentMail oraz SQL Mail

> SQL Server współpracuje z serwerami poczty elektronicznej zgodnymi ze standardem MAPI-1.

Aby możliwe było korzystanie z obu usług, należy:

1. Na komputerze, na którym działa serwer bazodanowy, zainstalować program klienta poczty elektronicznej.
2. Uruchomić usługi SQL Server oraz SQLServerAgent w kontekście zabezpieczeń użytkownika domenowego posiadającego konto na serwerze pocztowym.
3. Stworzyć profil klienta poczty elektronicznej dla każdego z kont, w których kontekście działają wymienione usługi (jeżeli obie usługi uruchomione są w kontekście tego samego konta, wymagany będzie tylko jeden profil),
4. Za pomocą konsoli Enterprise Manager skonfigurować usługi SQLAgentMail oraz SQL Mail.

Jeżeli na liście dostępnych profili pocztowych obu usług nie znajdują się nowo dodane profile pocztowe, należy zatrzymać i ponownie uruchomić SQL Server.

Procedury rozszerzone usługi SQL Mail

Z usługi SQL Mail możemy skorzystać za pośrednictwem poniższych procedur rozszerzonych (zarządzać usługą SQL Mail domyślnie mogą jedynie administratorzy serwera).

Procedura xp_startmail

Wykonanie procedury uruchamia usługę SQL Mail. Możliwe jest również automatyczne uruchamianie usługi podczas uruchamiania SQL Servera.

Składnia:

```
xp_startmail [[@user =] 'użytkownik'] [,[@password =] 'hasło']
```

gdzie:

użytkownik określa nazwę użytkownika, dla którego stworzono profil klienta poczty elektronicznej.

Przykład:

```
USE master
EXEC xp_startmail
GO
SQL mail session started.
```

Procedura xp_stopmail

Zatrzymać usługę SQL Mail możemy poprzez wykonanie procedury xp_stopmail.

Składnia:

```
xp_stopmail
```

Przykład:

```
USE master
EXEC xp_stopmail
GO
Stopped SQL mail session.
```

Procedura xp_sendmail

Wykonanie procedury spowoduje wysłanie komunikatu lub wyniku zapytania do określonych użytkowników. SQL Server wykona tylko jedną procedurę xp_sendmail w tym samym czasie, tak więc należy unikać wywoływania w ten sposób zapytań, które długotrwale blokują dostęp do danych.

Składnia:

```
xp_sendmail {[@recipients =] 'odbiorca [;…n]'}
[,[@message =] 'komunikat']
[,[@query =] 'zapytanie']
[,[@attachments =] załączniki]
[,[@copy_recipients =] 'do_wiadomości_użytkowników [;…n]'
```

```
[,[@blind_copy_recipients =] 'do_ukrytej_wiadomości_użytkowników [;…n]'
[,[@subject =] 'temat']
[,[@type =] 'typ']
[,[@attach_results =] 'typ_wyniku']
[,[@no_output =] 'potwierdzenie']
[,[@no_header =] 'nagłówek']
[,[@width =] szerokość]
[,[@separator =] 'separator']
[,[@echo_error =] 'komunikaty_błędów']
[,[@set_user =] 'kontekst']
[,[@dbuse =] 'baza_danych']
```

gdzie:
komunikat określa treść wiadomości,
zapytanie jest poprawną instrukcją SELECT której wynik zostanie wysłany do użytkownika,
załączniki określają pliki dołączone do wiadomości,
typ określa typ MAPI wiadomości,
typ_wyniku określa, czy wynik zapytania ma zostać zapisany w wiadomości czy w załączniku i może przyjmować wartości: FALSE (wartość domyślna, wynik zapytania zostanie zwrócony w wiadomości) lub TRUE (wynik zapytania zostanie zapisany w załączniku),
potwierdzenie określa, czy usługa SQL Mail otrzyma wiadomości związane np. z niemożnością dostarczenia wiadomości i może przyjmować wartości: FALSE (wartość domyślna, komunikaty będą dostarczane) lub TRUE (brak potwierdzeń),
nagłówek określa, czy wynik zapytania będzie zawierał nagłówki kolumn (domyślna wartość FALSE), czy nie,
szerokość określa liczbę znaków w wierszu zapytania (wartością domyślną jest 80),
separator określa znak końca kolumny,
komunikaty_błędów określa, czy do wiadomości zostaną dołączone komunikaty ewentualnych błędów wykonania zapytania oraz liczba zwróconych przez nie wierszy (wartość TRUE),
kontekst określa konto użytkownika z uprawnieniami, którego zapytanie zostanie wykonane (wartością domyślną jest quest).
Przykład:

```
EXEC xp_sendmail @recipients = 'jacek',
@query = 'SELECT LastName, FirstName, HomePhone
FROM Northwind.dbo.Employees',
@subject = 'Lista pracowników',
@message = 'W załączniku znajduje się aktualna lista pracowników',
@attach_results = 'TRUE'
GO
Mail sent.
```

Procedura xp_readmail

Wiadomości otrzymane przez usługę SQL Mail możemy odczytać, wywołując procedurę xp_readmail.
Składnia:

```
xp_readmail [[@msg_id =] 'wiadomość'] [, [@type =] 'typ' [OUTPUT]]
[,[@peek =] 'przeczytane']
[,[@suppress_attach =] 'pliki_tymczasowe']
[,[@originator =] 'nadawca' OUTPUT]
```

[,[@subject =] 'tytuł' OUTPUT]

[,[@message =] 'komunikat' OUTPUT]

[,[@recipients =] 'odbiorca [;…n]' OUTPUT]

[,[@cc_list =] 'do_wiadomości_użytkowników [;…n]' OUTPUT]

[,[@bcc_list =] 'do_ukrytej_wiadomości_użytkowników [;…n]' OUTPUT]

[,[@date_received =] 'data' OUTPUT]

[,[@unread =] 'przeczytane' OUTPUT]

[,[@attachments =] 'załączniki [;…n]' OUTPUT])

[,[@skip_bytes =] opuszczone_bajty OUTPUT]

[,[@msg_length =] długość OUTPUT]

[,[@originator_address =] 'adres_nadawcy' OUTPUT]]

gdzie:

przeczytane określa, czy status odczytanej wiadomości ma pozostać niezmieniony (wartość TRUE), czy wiadomość ma zostać oznaczona jako odczytana (domyślna wartość FALSE),
pliki_tymczasowe określa, czy SQL Server będzie mógł utworzyć pliki tymczasowe dla odczytywanych wiadomości z załącznikami (domyślna wartość TRUE nie pozwala na tworzenie plików tymczasowych),
przeczytane określa, czy wiadomość nie została oznaczona jako przeczytana (wartość TRUE),
opuszczone_bajty określają, ile (jeżeli wartość jest różna od domyślnej wartości 0) początkowych bajtów wiadomości zostanie opuszczonych.

Procedura xp_deletemail

Wykonanie procedury spowoduje usunięcie wybranej wiadomości ze skrzynki odbiorczej usługi SQL Mail.
Składnia:

xp_deletemail {'wiadomość'}

Przykład:

USE master

EXEC xp_deletemail 'XA4'

GO

The command(s) completed successfully.

Integracja SQL Servera z serwerem WWW

Aby możliwe było pobieranie i modyfikowanie danych przechowywanych w bazach SQL Servera poprzez witrynę sieci Web opublikowaną przez serwer IIS firmy Microsoft (ang. Internet Information Services), konieczne jest utworzenie odpowiedniego katalogu wirtualnego. Utworzyć i skonfigurować katalog wirtualny można za pomocą instalowanej wraz z SQL Serverem konsoli IIS Virtual Directory Managent for SQL Server.

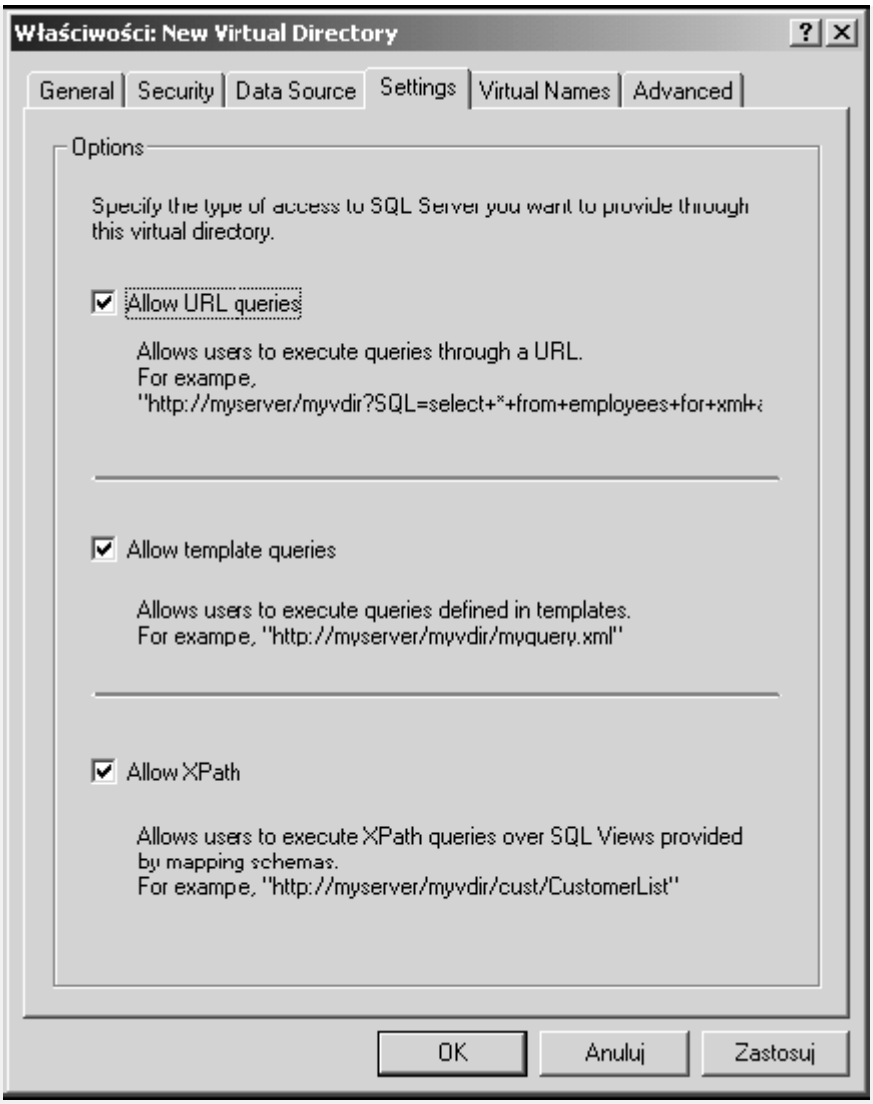
Katalog wirtualny umożliwia nawiązanie połączenia z serwerem bazodanowym poprzez filtr ISAPI sqlisapi.dll.

Konfiguracja katalogu wirtualnego

Po uruchomieniu konsoli należy utworzyć nowy katalog wirtualny.

1. Na zakładce General podać nazwę, za pomocą której klienci internetowi będą mogli połączyć się z bazą danych, a także lokalizację plików dostępnych poprzez stronę WWW.
2. Na zakładce Security określić sposób uwierzytelniania klientów internetowych przez SQL Server.
3. Na zakładce Data Source podać nazwę serwera bazodanowego i nazwę bazy danych przechowującej dane publikowane poprzez witrynę.

4. Na zakładce Settings określić dowolne metody pobierania i modyfikowania danych poprzez witrynę WWW. Możliwe jest:
   - ♣ Zezwolenie na bezpośredni dostęp użytkowników do obiektów bazodanowych poprzez umieszczenie instrukcji języka Transact-SQL w adresie URL.
   - ♣ Wykonywanie instrukcji zapisanych w szablonach umieszczonych w katalogu wirtualnym. Szablonami są poprawne dokumenty XML.
   - ♣ Wykonywanie zapytań typu Xpatch (rysunek 17.1).



Rysunek 17.1. Bezpośredni dostęp użytkowników do bazy danych z powodów bezpieczeństwa powinien zostać zablokowany

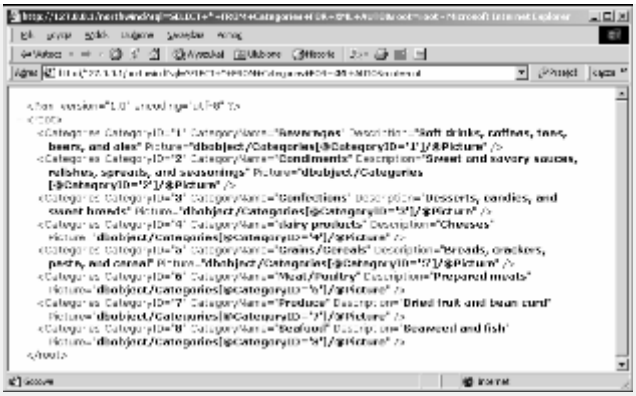
5. Na zakładce Virtual Names należy powiązać wirtualne nazwy folderów z folderami znajdującymi się na dyskach twardych, w których zostaną zapisane np. szablony XML.

Umieszczanie instrukcji języka Transact-SQL bezpośrednio w adresie URL

Najprostszym sposobem na pobieranie danych poprzez serwer WWW jest umieszczanie zapytań w adresie URL. Zapytania tego typu umieszcza się bezpośrednio za nazwą katalogu wirtualnego, wstawiając za nazwą serwera ciąg znaków ?sql= i oddzielając poszczególne słowa instrukcji znakiem plusa. Ponieważ poprawny dokument XML musi posiadać element najwyższego poziomu, zapytanie należy zakończyć, przypisując wartość parametrowi &root. Tak więc, aby wyświetlić wszystkie dane zapisane w tabeli Categories, napiszemy:

http://127.0.0.1/northwind?sql=SELECT+*+FROM+Categories+FOR+XML+AUTO&root=root

gdzie 127.0.0.1 jest adresem serwera WWW, a northwind nazwą katalogu wirtualnego. Wynik zapytania został pokazany na rysunku 17.2.



Rysunek 17.2. Bezpośredni dostęp do bazy danych poprzez stronę WWW

Wykonywanie instrukcji zapisanych w szablonach XML

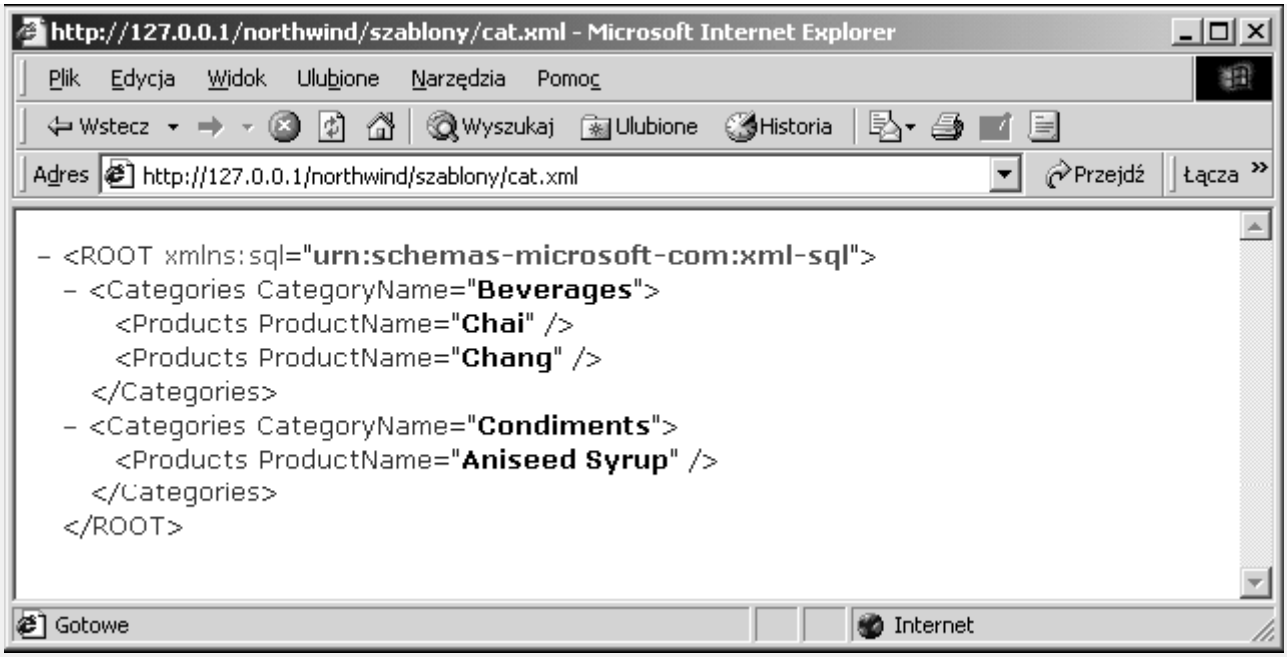
O wiele bezpieczniejsze i prostsze od bezpośredniego wykonywania instrukcji języka Transact-SQL przez klientów internetowych jest zapisanie instrukcji w plikach szablonów i zezwolenie wyłącznie na wykonywanie tych dokumentów. W tym celu należy:

1. Utworzyć poprawny szablon XML zawierający instrukcję języka Transact-SQL:

2.
3. <ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
4. <sql:query>
5. SELECT CategoryName, ProductName
6. FROM Categories JOIN Products
7. ON Categories.CategoryID = Products.CategoryID
8. WHERE SupplierID=1
9. FOR XML AUTO
10. </sql:query>
11. </ROOT>
12.
13. Zapisać utworzony szablon w folderze powiązanym z folderem wirtualnym.
14. Jako adres URL podać lokalizację pliku z szablonem:

    http://127.0.0.1/northwind/szablony/cat.xml

gdzie: szablony są nazwą katalogu wirtualnego, a cat.xml — nazwą pliku szablonu. Wynik zapytania został pokazany na rysunku 17.3.



Rysunek 17.3. Wywoływanie predefiniowanych szablonów XML

Wykonywanie zapytań typu XPATCH

Xpatch jest bazującym na standardzie XML językiem zapytań, który umożliwia przetwarzanie danych poprzez dokumenty XML. Zapytania typu XPATCH odwołują się do obiektów bazodanowych (tabel i widoków), tak jakby to były dokumenty XML, a poszczególne kolumny traktowane są jako atrybuty dokumentu XML.

Zagadnienie integracji SQL Servera z serwerem IIS i wykorzystania dokumentów XML w celu pobierania i modyfikowania danych wykracza poza zakres książki. Tematom tym poświecona jest wydana przez wydawnictwo Helion książka „XML i SQL Server 2000”.

Łączenie serwerów bazodanowych

Połączenie serwerów bazodanowych umożliwia zdalne wykonywanie na nich instrukcji języka Transact-SQL. Połączone serwery wymieniają dane poprzez mechanizm OLE DB.

Tworzenie powiązań między serwerami

Procedura sp_addlinkedserver

Połączyć serwery bazodanowe mogą użytkownicy należący do ról sysadmin lub setupadmin, wykonując procedurę sp_addlinkedserver.
Składnia:

```
sp_addlinkedserver [@server =] 'serwer'
[, [@srvproduct =] 'typ_źródła_danych']
[, [@provider =] 'dostawca']
[, [@datasrc =] 'źródło_danych']
[, [@location =] 'lokalizacja']
[, [@provstr =] 'połaczenie']
[, [@catalog =] 'katalog']
```

gdzie:
typ_źródła_danych określa nazwę OLE DB źródła danych.
połaczenie jednoznacznie określa źródło danych.

Przykład:
Łączymy SQL Server z instancją Kopia uruchomioną na tym samym komputerze:

```
EXEC sp_addlinkedserver @server='LOLEK_Kopia', @srvproduct='',
@provider='SQLOLEDB', @datasrc='LOLEK\kopia'
GO
(1 row(s) affected)
(1 row(s) affected)
```

Procedura sp_addlinkedsrvlogin

Po utworzeniu połączenia należy skonfigurować metodę uwierzytelniania użytkowników poprzez połączone serwery. Loginy użytkowników połączonych serwerów możemy powiązać, wykonując procedurę sp_addlinkedsrvlogin. Tylko użytkownicy ról sysadmin oraz securityadmin mogą wykonać tę procedurę.
Składnia:

```
sp_addlinkedsrvlogin [@rmtsrvname =] 'zdalny_serwer'
[, [@useself =] 'własne']
[, [@locallogin =] 'login']
[, [@rmtuser =] 'zdalny_login']
[, [@rmtpassword =] 'hasło']
```

gdzie:
własne oznacza, że określony login lokalnego użytkownika zostanie wykorzystany do jego uwierzytelnienia na zdalnym serwerze (domyślna wartość TRUE),
login określa login zdalnego użytkownika (domyślna wartość NULL oznacza wszystkie loginy lokalnych użytkowników serwera bazodanowego).
Przykłady:
Umożliwiamy wszystkim użytkownikom lokalnego serwera uwierzytelnienie poprzez serwer LOLEK_Kopia na podstawie ich lokalnych loginów:

```
EXEC sp_addlinkedsrvlogin 'LOLEK_Kopia'
GO
(0 row(s) affected)
(1 row(s) affected)
(0 row(s) affected)
(1 row(s) affected)
```

Umożliwiamy użytkownikowi uwierzytelnianemu lokalnie na podstawie loginu agatka uwierzytelnianie przez zdalny serwer jako użytkownik sa, po podaniu określonego hasła:

```
EXEC sp_addlinkedsrvlogin @rmtsrvname = 'LOLEK_Kopia',
@locallogin = 'agatka',
@rmtuser = 'sa',
@rmtpassword = 'sToi=NA_stacji:)'
GO
(0 row(s) affected)
```

(1 row(s) affected)

(0 row(s) affected)

(1 row(s) affected)

Procedura sp_linkedservers

Procedura zwraca listę połączonych z lokalnym serwerem serwerów bazodanowych.
Składnia:

sp_linkedservers

Przykład:

EXEC sp_linkedservers

GO

SRV_NAME SRV_PROVIDERNAME SRV_PRODUCT SRV_DATASOURCE SRV_PROVIDERSTRING

ªSRV_LOCATION SRV_CAT

------------- ---------------- ------------ -------------- -----------------------------

ª---------------------- ------------

LOLEK\MSSQL SQLOLEDB SQL Server LOLEK\MSSQL NULL NULL

NULL

LOLEK_Kopia SQLOLEDB SQL Server LOLEK\KOPIA NULL NULL

NULL

(2 row(s) affected)

Zdalne wykonywanie instrukcji języka Transact-SQL

Po połączeniu serwerów i skonfigurowaniu uwierzytelniania użytkowników możemy zdalnie wykonywać instrukcje języka Transact-SQL. Jednak próba bezpośredniego wykonania niektórych instrukcji zakończy się błędem:

SELECT ProductName, UnitPrice

FROM LOLEK_Kopia.Northwind.dbo.Products

WHERE CategoryID IN (2,3)

AND SupplierID=2

GO

ProductName UnitPrice

-- ----------------------

Chef Anton's Cajun Seasoning 22.0000

Chef Anton's Gumbo Mix 21.3500

Louisiana Fiery Hot Pepper Sauce 21.0500

Louisiana Hot Spiced Okra 17.0000

(4 row(s) affected)

TRUNCATE TABLE LOLEK_Kopia.Northwind.dbo.Products

```
GO
Server: Msg 117, Level 15, State 1, Line 1
The object name 'LOLEK_Kopia.Northwind.dbo.' contains more than the maximum number of
ªprefixes. The maximum is 2.
```

Rozwiązaniem tego problemu jest umożliwienie na połączonym serwerze wykonania zdalnych procedur (RPC) oraz wywołanie instrukcji języka Transact-SQL poprzez procedurę systemową sp_executesql.

Procedura sp_serveroption

Procedura umożliwia konfigurację zdalnych i połączonych serwerów.
Składnia:

```
sp_serveroption [@server =] 'serwer'
,[@optname =] 'parametr'
,[@optvalue =] 'wartość'
```

gdzie:
parametr określa nazwę konfigurowanej opcji i może przyjmować jedną z wartości wymienionych w tabeli 17.1.

Tabela 17.1. Lista dopuszczalnych parametrów procedury sp_serveroption

Parametr	Opis
collation compatible	Wartość TRUE oznacza, że zdalna i lokalna baza danych mają identyczny porządek sortowania i umożliwia SQL Serverowi odwoływanie się do lokalnych ustawień bazy podczas wykonywania zdalnych instrukcji
collation name	Określa nazwę porządku sortowania zdefiniowanego dla zdalnej bazy danych
connect timeout	Określa czas, którego przekroczenie spowoduje, że próba połączenia ze zdalnym serwerem zakończy się błędem
data access	Określa, czy zdalny serwer jest serwerem połączonym
dist	Określa, czy serwer pełni rolę dystrybutora
dpub	Określa, czy serwer jest wydawcą lokalnego dystrybutora
lazy schema validation	Określa, czy przed wykonaniem instrukcji ma zostać sprawdzona struktura obiektu bazodanowego. Wartość TRUE powoduje natychmiastowe wykonanie instrukcji
pub	Określa, czy serwer pełni rolę wydawcy
query timeout	Określa czas, którego przekroczenie spowoduje, że próba zdalnego wykonania instrukcji zakończy się błędem
rpc	Umożliwia zdalne wykonanie procedur
rpc out	Umożliwia zdalne wykonanie procedur przez zdalny serwer
sub	Określa, czy serwer pełni rolę subskrybenta
system	Zarezerwowane
use remote collation	Domyślna wartość FALSE powoduje, że do wykonywania rozproszonych zapytań wykorzystywany będzie lokalny porządek sortowania. Wartość TRUE (niezgodna z wersjami 7.0 i wcześniejszymi) do wykonania zapytań pobierających dane z kilku serwerów wykorzysta porządek sortowania zdalnej bazy danych

Przykład:

```
EXEC sp_serveroption @server = 'LOLEK_Kopia',
@optname = 'rpc',
@optvalue = 'TRUE'
EXEC sp_serveroption @server = 'LOLEK_Kopia',
@optname = 'rpc out',
@optvalue = 'TRUE'
```

```
GO
The command(s) completed successfully.
```

Procedura sp_addremotelogin

Wykonanie procedury spowoduje dodanie zdalnych loginów do lokalnego serwera, umożliwiając połączonemu serwerowi wykonywanie zdalnych procedur.
Składnia:

```
sp_addremotelogin [@remoteserver =] 'zdalny_serwer'
[, [@loginame =] 'login']
[, [@remotename =] 'zdalny_login']
```

gdzie:
wartością domyślną parametrów login i zdalny_login, oznaczającą wszystkie loginy, jest NULL.
Przykład:

```
EXEC sp_addremotelogin 'LOLEK_Kopia'
GO
(1 row(s) affected)
```

Procedura sp_helpserver

Zwraca informacje o połączonych lub biorących udział w replikacji serwerach.
Składnia:

```
sp_helpserver [[@server =] 'serwer']
[, [@optname =] 'parametr']
[, [@show_topology =] 'show_topology']
```

gdzie:
parametr może przyjmować jedną z poniższych wartości:

- collation compatible,
- data access,
- dist,
- dpub,
- lazy schema validation,
- pub,
- rpc,
- rpc out,
- sub,
- system,
- use remote collation.

Domyślna wartość NULL spowoduje zwrócenie wszystkich informacji o zdalnym serwerze.
Przykład:

```
sp_helpserver 'LOLEK_Kopia'
```

```
GO
name network_name status id
ªcollation_name connect_timeout query_timeout
---------- ------------ ------------------------------------------- --------------
ª--------------- ---------------- ----------------
LOLEK_Kopia NULL rpc,rpc out,data access,use remote collation 1 NULL
ª0 0
```

Procedura sp_executesql

Procedura umożliwia wykonanie (również wielokrotne) instrukcji języka Transact-SQL oraz przekazanie parametrów do wykonywanych instrukcji.
Składnia:

```
sp_executesql [@stmt =] instrukcja
[
{, [@params =] N'@parametr typ [,…n]'}
{, [@param1 =] 'wartość_1' [,…n]}
```

Przykład:

```
EXEC LOLEK_Kopia.master.dbo.sp_executesql
N'SELECT * INTO Northwind.dbo.p FROM Northwind.dbo.Products'
GO
EXEC LOLEK_Kopia.master.dbo.sp_executesql
N'TRUNCATE TABLE Northwind.dbo.p'
GO
EXEC LOLEK_Kopia.master.dbo.sp_executesql
N'SELECT ProductID, ProductName FROM Northwind.dbo.p'
GO
(77 row(s) affected)
ProductID ProductName
----------- ----------------------------------------
(0 row(s) affected)
```

Usuwanie powiązań pomiędzy serwerami

Procedura sp_droplinkedsrvlogin

Określone powiązania pomiędzy loginami połączonych serwerów możemy usunąć, wykonując procedurę sp_droplinkedsrvlogin. Usunięcie określonego powiązania spowoduje przywrócenie domyślnego ustawienia, w którym login użytkownika jest wykorzystywany również do uwierzytelniania przez zdalne serwery.
Składnia:

```
sp_droplinkedsrvlogin [@rmtsrvname =] 'zdalny_serwer',
[@locallogin =] 'login'
```

Przykład:

```
EXEC sp_droplinkedsrvlogin 'LOLEK_Kopia', NULL
GO
(1 row(s) affected)
```

Procedura sp_dropremotelogin

Wykonanie procedury usuwa powiązania pomiędzy zdalnymi loginami a loginami lokalnego serwera.
Składnia:

```
sp_dropremotelogin [@remoteserver =] 'zdalny_serwer'
[, [@loginame =] 'login']
[, [@remotename =] 'zdalny_login']
```

Przykład:

```
EXEC sp_dropremotelogin 'LOLEK_Kopia'
GO
(0 row(s) affected)
(1 row(s) affected)
```

Procedura sp_dropserver

Wykonanie procedury usuwa istniejące powiązanie między serwerami bazodanowymi.
Składnia:

```
sp_dropserver [@server =] 'serwer' [, [@droplogins =] {'droplogins' | NULL}]
```

gdzie:
droplogins określa, czy jednocześnie mają zostać usunięte wszystkie zdalne loginy (wartość 'droplogins'). Domyślną wartością jest NULL.
Przykład:

```
EXEC sp_dropserver 'LOLEK_Kopia', 'droplogins'
GO
(1 row(s) affected)
(1 row(s) affected)
```

Operatorzy

Operatorzy to osoby, które mogą zostać automatycznie powiadomione przez SQL Server o wykonanych zadaniach czy występujących błędach. Każdy z operatorów może zostać powiadomiony:

1. Poprzez wysłanie wiadomości elektronicznej.
2. Poprzez wysłanie komunikatu usługi systemowej Posłaniec.
3. Poprzez wysłanie komunikatu na pager, co jest możliwe, o ile serwer pocztowy umożliwia wysyłanie tego typu komunikatów.

Zarządzać operatorami mogą jedynie członkowie roli serwera sysadmin, którzy mają nadane uprawnienia do modyfikowania i pobierania danych przechowywanych w tabeli msdb..sysoperators.

Procedura sp_add_operator

Wykonanie procedury spowoduje dodanie nowego operatora.
Składnia:

```
sp_add_operator [@name =] 'operator'
[, [@enabled =] aktywny]
[, [@email_address =] 'addres_email']
[, [@pager_address =] 'adres_pagera']
[, [@weekday_pager_start_time =] w_tygodniu_dostępny_od]
[, [@weekday_pager_end_time =] w_tygodniu_dostępny_do]
[, [@saturday_pager_start_time =] w_sobotę_dostępny_od]
[, [@saturday_pager_end_time =] w_sobotę_dostępny_do]
[, [@sunday_pager_start_time =] w_niedzielę_dostępny_od]
[, [@sunday_pager_end_time =] w_niedzielę_dostępny_do]
[, [@pager_days =] dni_robocze]
[, [@netsend_address =] 'adres_komputera']
[, [@category_name =] 'kategoria']
```

gdzie:
opcje *…dostępny* dotyczą jedynie powiadamiania za pomocą pagera i oznaczają godziny pracy operatora, *dni_robocze* dotyczy jedynie powiadamiania za pomocą pagera i oznacza dni pracy operatora reprezentowane przez liczby: 1 — poniedziałek, 2 — wtorek, 4 — środa, 8 — czwartek, 16 — piątek, 32 — sobota, 64 — niedziela (możliwe są dowolne kombinacje ustawionych bitów).

Ponieważ SQL Server zgłosi błąd, jeżeli użyty alias adresu e-mail nie będzie jednoznaczny, definiując operatorów, należy posługiwać się pełnym adresem e-mail.

Przykład:

```
USE msdb
EXEC sp_add_operator @name = 'Karmazynowy',
@email_address = 'karmazynowy@firma.pl',
@netsend_address = 'LOLEK'
GO
The command(s) completed successfully.
```

Procedura sp_update_operator

Informacje o istniejącym operatorze możemy zmodyfikować, wykonując procedurę sp_update_operator.
Składnia:

```
sp_update_operator [@name =] 'operator'
[, [@new_name =] 'nowa_nazwa']
[, [@enabled =] aktywny]
[, [@email_address =] 'addres_email']
[, [@pager_address =] 'adres_pagera']
[, [@weekday_pager_start_time =] w_tygodniu_dostępny_od]
[, [@weekday_pager_end_time =] w_tygodniu_dostępny_do]
[, [@saturday_pager_start_time =] w_sobotę_dostępny_od]
[, [@saturday_pager_end_time =] w_sobotę_dostępny_do]
[, [@sunday_pager_start_time =] w_niedzielę_dostępny_od]
[, [@sunday_pager_end_time =] w_niedzielę_dostępny_do]
[, [@pager_days =] dni_robocze]
[, [@netsend_address =] 'adres_komputera']
[, [@category_name =] 'kategoria']
```

Przykład:

```
USE msdb
EXEC sp_update_operator 'karmazynowy', 'księgowy'
GO
The command(s) completed successfully.
```

Konfigurując usługę SQLServerAgent, możemy określić operatora awaryjnego (ang. Fail-safe opertator). Operator ten będzie automatycznie powiadamiany o zdarzeniach, które wystąpiły poza godzinami pracy pozostałych operatorów. W wersji 2000 SQL Servera operator awaryjny nie może zostać usunięty — aby usunąć tego operatora, trzeba najpierw pozbawić go funkcji operatora awaryjnego.

Procedura sp_help_operator

Procedura zwraca informacje o wszystkich operatorach lub o wybranym operatorze.
Składnia:

```
sp_help_operator [[@operator_name =] 'operator']
[, [@operator_id =] id_operatora]
```

Przykład:

```
USE msdb
EXEC sp_help_operator
GO
```

Procedura sp_delete_operator

Wykonanie procedury usuwa definicję wybranego operatora.
Składnia:

```
sp_delete_operator [@name =] 'operator'
```

```
[, [@reassign_to_operator =] 'istniejący_operator']
```

gdzie:
istniejący_operator określa nazwę operatora, który będzie powiadamiany zamiast usuwanego operatora.
Przykład:

```
USE msdb
EXEC sp_delete_operator 'zwolniony', 'księgowy'
GO
The command(s) completed successfully.
```

Zadania

Zadania (ang. Jobs) są podstawowym typem obiektów umożliwiającym automatyzację pracy administratora baz danych. Zdefiniowane zadania przechowywane są w tabeli systemowej msdb..sysjobs, która w celu zapewnienia maksymalnej wydajności i niezawodności pracy serwera jest w całości buforowana.

Tworzenie zadań

Każde utworzone zadanie składa się z określonej liczby kroków (ang. Steps). Poszczególne kroki mogą być wykonywane po zakończeniu poprzedniego kroku, po pomyślnym wykonaniu poprzedniego kroku lub w wypadku zakończenia poprzedniego kroku niepowodzeniem. W ramach kolejnych kroków mogą być wykonywane instrukcje języka Transact-SQL, skrypty języków VBS lub JS oraz polecenia powłoki. Domyślnie właścicielem zadania staje się użytkownik, który je utworzył.

Procedura sp_add_job

Wykonanie procedury spowoduje utworzenie nowego zadania.
Składnia:

```
sp_add_job [@job_name =] 'zadanie'
[, [@enabled =] aktywne]
[, [@description =] 'opis']
[, [@start_step_id =] krok]
[, [@category_name =] 'kategoria']
[, [@category_id =] id_kategorii]
[, [@owner_login_name =] 'login']
[, [@notify_level_eventlog =] zapis_w_dzienniku]
[, [@notify_level_email =] wysyłanie_wiadomości]
[, [@notify_level_netsend =] wysyłanie_komunikatów]
[, [@notify_level_page =] pager]
[, [@notify_email_operator_name =] 'adres_email']
[, [@notify_netsend_operator_name =] 'adres_komputera']
[, [@notify_page_operator_name =] 'adres_pagera']
[, [@delete_level =] usunięcie]
[, [@job_id =] id_zadania OUTPUT]
```

gdzie:

aktywne określa, czy zadanie będzie wykonywane zgodnie z zaplanowanym harmonogramem (domyślna wartość 1), czy konieczne będzie jego programowe wywołanie;
krok oznacza numer pierwszego kroku zadania (domyślnie pierwszy krok ma numer 1);
kategoria jest nazwą kategorii zadania;
id_kategorii jest niezależnym językowo identyfikatorem kategorii zadania;

Konsekwentne przypisywanie zadań do poszczególnych kategorii ułatwia wybieranie zadań i zarządzanie nimi.

login określa użytkownika, który będzie właścicielem zadania (domyślna wartość NULL oznacza bieżącego użytkownika);

zapis_w_dzienniku określa, kiedy fakt wykonania zadania zostanie odnotowany w dzienniku aplikacji systemu Windows NT/2000 i może przyjmować wartości:

- 0 — nigdy,
- 1 — przy pomyślnym wykonaniu,
- 2 (wartość domyślna) — przy wystąpieniu błędu,
- 3 — zawsze;

wysyłanie_wiadomości określa, kiedy informacja o wykonaniu zadania zostanie wysłana pocztą elektroniczną i może przyjmować te same wartości co opcja zapis_w_dzienniku (wartością domyślną jest 0);
wysyłanie_komunikatów określa, kiedy informacja o wykonaniu zadania zostanie wysłana za pomocą polecenia NET SEND i może przyjmować te same wartości co opcja zapis_w_dzienniku (wartością domyślną jest 0);
pager określa, kiedy informacja o wykonaniu zadania zostanie wysłana na pager i może przyjmować te same wartości co opcja zapis_w_dzienniku (wartością domyślną jest 0);
usunięcie określa, czy zadanie ma zostać po wykonaniu automatycznie usunięte (wartość 3). Usunięcie zadania spowoduje jednoczesne usunięcie istniejącej historii zadania. Domyślną wartością jest 0;
id_zadania jest automatycznie nadawanym przez SQL Server identyfikatorem pomyślnie utworzonych zadań.

Przykład:

```
USE msdb
EXEC sp_add_job @job_name = 'Przecena towarów',
@description = 'Obniżenie o 15% ceny zalegających towarów'
GO
The command(s) completed successfully.
```

Procedura sp_update_job

Zmodyfikować istniejące zadanie możemy za pomocą procedury sp_update_job.

Składnia:

```
sp_update_job [@job_name =] 'zadanie'
[, [@new_name =] 'nowa_nazwa']
[, [@enabled =] aktywne]
[, [@description =] 'opis']
[, [@start_step_id =] krok]
[, [@category_name =] 'kategoria']
[, [@category_id =] id_kategorii]
[, [@owner_login_name =] 'login']
[, [@notify_level_eventlog =] zapis_w_dzienniku]
[, [@notify_level_email =] wysyłanie_wiadomości]
[, [@notify_level_netsend =] wysyłanie_komunikatów]
[, [@notify_level_page =] pager]
[, [@notify_email_operator_name =] 'adres_email']
```

[, [@notify_netsend_operator_name =] 'adres_komputera']

[, [@notify_page_operator_name =] 'adres_pagera']

[, [@delete_level =] usunięcie]

[, [@automatic_post =] zarezerwowane]

Przykład:

USE msdb

EXEC sp_update_job @job_name = 'Przecena towarów',

@notify_level_email = 3,

@notify_level_netsend = 2,

@notify_email_operator_name = 'księgowy',

@notify_netsend_operator_name = 'księgowy'

GO

The command(s) completed successfully.

Procedura sp_help_job

Procedura zwraca określone informacje o wszystkich lub o wybranych zadaniach.
Składnia:

sp_help_job [[@job_id =] id_zadania]

[, [@job_name =] 'zadanie']

[, [@job_aspect =] 'zwracane_informacje']

[, [@job_type =] 'typ_zadania']

[, [@owner_login_name =] 'login']

[, [@subsystem =] 'podsystem']

[, [@category_name =] 'kategoria']

[, [@enabled =] aktywne]

[, [@execution_status =] status]

[, [@date_comparator =] 'porównanie']

[, [@date_created =] data_utworzenia]

[, [@date_last_modified =] data_zmodyfikowania]

[, [@description =] 'wzorzec_opisu']

gdzie:
zwracane_informacje określa liczbę zwróconych przez procedurę informacji i może przyjmować jedną z poniższych wartości:

- ALL — wszystkie dane o zadaniu,
- JOB — wyłącznie informacje o definicji zadania,
- SCHEDULES — informacje o harmonogramach wykonania zadania,
- STEPS — informacje o krokach zadania,
- TARGETS — informacje o docelowych serwerach.

typ_zadania określa, czy zadanie zostało zdefiniowane dla lokalnego czy dla wielu zdalnych serwerów i może przyjmować jedną z wartości LOCAL lub MULTI-SERVER,
status określa stan wykonania zadania i może przyjmować jedną z poniższych wartości:

- 0 — aktywne,
- 1 — trwa wykonywanie,
- 2 — oczekujące,
- 3 — ponownie wykonywane,
- 4 — bezczynne,
- 5 — przerwane,
- 7 — wykonane.

porównanie określa, czy data modyfikacji jest różna od daty utworzenia i może przyjmować jedną z poniższych wartości:

- = — daty są identyczne,
- > — data modyfikacji jest późniejsza od daty utworzenia.

Przykład:

```
USE msdb
EXEC sp_help_job @job_name = 'Przecena towarów',
@job_aspect = 'JOB'
GO
```

Tworzenie kroków zadania

Zadanie jest zbiorem powiązanych ze sobą kroków (czynności), które mogą zostać wykonane jak jedna operacja. Dodatkowo wykonanie zadania może wiązać się z powiadomieniem operatora i (co zostało opisane w dalszej części rozdziału) może zostać dowolnie zaplanowane. W ramach kolejnych kroków może być wykonywana operacja jednego typu, np. instrukcja języka Transact-SQL czy polecenie systemu operacyjnego.

Procedura sp_add_jobstep

Dodać nowy krok do istniejącego zadania możemy poprzez procedurę sp_add_jobstep. Definicje poszczególnych kroków zadań przechowywane są w tabeli msdb..sysjobsteps.
Składnia:

```
sp_add_jobstep [@job_id =] id_zadania | [@job_name =] 'zadanie'
[, [@step_id =] id_kroku]
{, [@step_name =] 'krok'}
[, [@subsystem =] 'podsystem']


[, [@command =] 'instrukcja']
[, [@additional_parameters =] 'parametry']
[, [@cmdexec_success_code =] raport]
[, [@on_success_action =] akcja_przy_powodzeniu]
[, [@on_success_step_id =] id_kroku_przy_powodzeniu]
[, [@on_fail_action =] akcja_przy_niepowodzeniu]
[, [@on_fail_step_id =] id_kroku_przy_niepowodzeniu]
```

[, [@server =] 'serwer']

[, [@database_name =] 'baza_danych']

[, [@database_user_name =] 'użytkownik']

[, [@retry_attempts =] liczba_powtórzeń]

[, [@retry_interval =] opóźnienie]

[, [@os_run_priority =] priorytet]

[, [@output_file_name =] 'plik']

[, [@flags =] flaga]

gdzie:

zadanie musi zostać określone albo poprzez identyfikator, albo poprzez nazwę.
podsystem określa typ operacji (podsystem, w ramach którego zostanie wykonana operacja) i może przyjmować jedną z poniższych wartości:

- ACTIVESCRIPTING —skrypt VBS lub JS,
- CMDEXEC — polecenie lub plik wykonywalny systemu operacyjnego,
- DISTRIBUTION — zadanie agenta dystrybucji,
- SNAPSHOT — zadanie agenta migawek,
- LOGREADER — zadanie agenta dziennika transakcyjnego,
- MERGE — zadanie agenta scalania,
- TSQL (wartość domyślna) — instrukcja języka Transact-SQL.

akcja_przy_powodzeniu określa akcję podejmowaną po pomyślnym wykonaniu operacji i może przyjmować jedną z poniższych wartości:

- 1 (wartość domyślna) — zakończ zadanie, raportując powodzenie.
- 2 — zakończ zadanie, raportując niepowodzenie.
- 3 — przejdź do następnego kroku.
- 4 — przejdź do kroku o określonym przez parametr id_kroku_przy_powodzeniu numerze.

akcja_przy_niepowodzeniu określa akcję podejmowaną po niepomyślnym wykonaniu operacji i może przyjmować jedną z poniższych wartości:

- 1— zakończ zadanie, raportując powodzenie.
- 2 (wartość domyślna) — zakończ zadanie, raportując niepowodzenie.
- 3 — przejdź do następnego kroku.
- 4 — przejdź do kroku o określonym przez parametr id_kroku_przy_niepowodzeniu numerze.

flaga określa, czy raport ma zostać dopisany do pliku wyjściowego (wartość 2), czy zawartość pliku ma zostać zastąpiona (wartość 4).

Definiując kroki różnych typów, należy uwzględnić poniższe wskazówki:

1. Jeżeli w ramach kroku wykonywana jest instrukcja języka Transact-SQL:
   - Należy jawnie określić docelową bazę danych.
   - W przypadku wykonywania procedury lub funkcji, które spodziewają się określonych parametrów, należy je przekazać do wywoływanego obiektu.
   - Niemożliwe jest przekazanie wyniku jednego kroku do innego kroku, a plik wyjściowy zawiera jedynie raport z wykonania operacji.
2. Jeżeli w ramach kroku wykonywane jest polecenie lub program wykonywalny systemu operacyjnego:
   - Należy jawnie określić wartość parametru raport sygnalizującą pomyślne wykonanie operacji.
   - Należy podać pełną ścieżkę do wykonywanego pliku.
3. Jeżeli w ramach kroku wykonywany jest skrypt języka Microsoft Visual Basic Scripting Edition lub Microsoft Jscript:
   - Należy określić język, w którym skrypt został napisany.
   - Po zainstalowaniu bibliotek innych języków skryptowych możliwe będzie uruchamianie skryptów napisanych w tych językach.
4. Kroki związane z replikacją traktowane są jako zadania określonych agentów replikacji.

Niezależnie od typu operacji, tworząc zadania składające się z wielu kroków, należy określić akcję wykonywaną po pomyślnym oraz niepomyślnym wykonaniu kolejnych kroków. Definiując powiązania pomiędzy poszczególnymi

krokami, należy pamiętać, że SQL Server umożliwia podejmowanie wielokrotnych, powtarzanych w określonych odstępach czasu, prób wykonania danego kroku, zanim zostanie on uznany za niepomyślnie wykonany. Niektóre operacje, szczególnie wymagające nawiązania połączenia ze zdalnym serwerem, powinny być powtórzone, zanim zostaną uznane za niewykonane.
Przykład:

```
USE msdb
EXEC sp_add_jobstep @job_name = 'Przecena towarów',
@step_name = 'Aktualizacja cen',
@database_name = 'Northwind',
@command = 'UPDATE dbo.Products
SET UnitPrice = UnitPrice * 0.75
WHERE UnitsInStock>100',
@on_success_action = 3
GO
The command(s) completed successfully.

USE msdb
EXEC sp_add_jobstep @job_name = 'Przecena towarów',
@step_name = 'generowanie raportów',
@database_name = 'Northwind',
@command = 'EXEC p_products',
@retry_attempts = 2,
@retry_interval = 1
GO
The command(s) completed successfully.
```

W ten sposób utworzyliśmy dwa kroki zadania. Po pomyślnym wykonaniu pierwszego kroku system podejmie trzykrotną próbę wykonania drugiego kroku. Ponieważ procedura o podanej nazwie nie istnieje, wykonanie całego zadania zakończy się niepowodzeniem, co spowoduje wysłanie, poprzez usługę Posłaniec i pocztę elektroniczną, wiadomości do operatora księgowy.

Procedura sp_update_jobstep

Istniejący krok zadania możemy zmodyfikować, wykonując procedurę sp_update_jobstep.
Składnia:

```
sp_update_jobstep [@job_id =] id_zadania | [@job_name =] 'zadanie'
[, [@step_id =] id_kroku]
{, [@step_name =] 'krok'}
[, [@subsystem =] 'podsystem']
[, [@command =] 'instrukcja']
[, [@additional_parameters =] 'parametry']
[, [@cmdexec_success_code =] raport]
[, [@on_success_action =] akcja_przy_powodzeniu]
[, [@on_success_step_id =] id_kroku_przy_powodzeniu]
[, [@on_fail_action =] akcja_przy_niepowodzeniu]
```

```
[, [@on_fail_step_id =] id_kroku_przy_niepowodzeniu]
[, [@server =] 'serwer']
[, [@database_name =] 'baza_danych']
[, [@database_user_name =] 'użytkownik']
[, [@retry_attempts =] liczba_powtórzeń]
[, [@retry_interval =] opóźnienie]
[, [@os_run_priority =] priorytet]
[, [@output_file_name =] 'plik']
[, [@flags =] flaga]
```

gdzie:
krok określa nową nazwę kroku identyfikowanego przez wartość parametru id_kroku.
Przykład:

```
USE msdb
EXEC sp_update_jobstep @job_name = 'Przecena towarów',
@step_id = 2,
@database_name = 'Northwind',
@additional_parameters = 'WHERE UnitsInStock>100'
GO
The command(s) completed successfully.
```

Procedura sp_help_jobstep

Procedura zwraca informacje o określonym kroku wybranego zadania.
Składnia:

```
sp_help_jobstep [@job_id =] 'id_zadania' |
[@job_name =] 'zadanie'
[, [@step_id =] id_kroku]
[, [@step_name =] 'krok']
[, [@suffix =] flaga]
```

gdzie:
flaga określa, czy opis zadania ma zostać dołączony do wyniku procedury (wartość 1). Domyślną wartością parametru jest 0.
Przykład:

```
USE msdb
EXEC sp_help_jobstep @job_name = 'Przecena towarów',
@step_id = 1
GO
```

Tworzenie harmonogramów wykonania zadań

Dla każdego zadania możliwe jest utworzenie wielu harmonogramów jego wykonania. Wynika z tego, że jeżeli planujemy automatyczne wykonywanie zadania w każdy poniedziałek i dodatkowo w pierwszą niedzielę nowego kwartału, wystarczy utworzyć jedno zadanie i zdefiniować dla niego dwa harmonogramy.

Procedura sp_add_jobschedule

Wykonanie procedury spowoduje utworzenie harmonogramu wykonania określonego zadania.
Składnia:

```
sp_add_jobschedule [@job_id =] id_zadania, | [@job_name =] 'zadanie',
[@name =] 'harmonogram'
[, [@enabled =] aktywny]
[, [@freq_type =] uruchamiane]
[, [@freq_interval =] uruchamiane_co]
[, [@freq_subday_type =] uruchamiane_co_minut]
[, [@freq_subday_interval =] uruchamiane_razy]
[, [@freq_relative_interval =] uruchamiene_co_miesiąc]
[, [@freq_recurrence_factor =] powtórzone_co]
[, [@active_start_date =] uruchamiane_od]
[, [@active_end_date =] uruchamiane_do]
[, [@active_start_time =] uruchamiane_od_godziny]
[, [@active_end_time =] uruchamiane_do_godziny]
```

gdzie:
uruchamiane określa, kiedy zadanie będzie uruchamiane i może przyjąć jedną z poniższych wartości:

- 1 — jednokrotnie,
- 4 — codziennie,
- 8 — cotygodniowo,
- 16 — comiesięcznie,
- 32 — comiesięcznie z uwzględnieniem parametru uruchamiane_co_miesiąc,
- 64 — podczas uruchamiania usługi SQLServerAgent,
- 128 — kiedy komputer będzie bezczynny.

Aby usługa SQLServerAgent mogła uruchamiać zadania podczas bezczynności komputera, trzeba, konfigurując tą usługę, określić procent użycia procesora i czas, w którym stopień jego użycia utrzymuje się poniżej tej wartości. Ponadto SQLServerAgent musi zostać uruchomiony w kontekście zabezpieczeń lokalnego administratora systemu operacyjnego.

uruchamiane_co określa, przy uwzględnieniu parametru uruchamiane, dzień uruchomienia zadania i może przyjmować wartości znajdujące się w tabeli 17.2.

Tabela 17.2. Zależności pomiędzy wartościami parametrów uruchamiane i uruchamiane_co

Wartość parametru uruchamiane	Wpływ na wartość parametru uruchamiane_co
1	Parametr nie jest uwzględniany
4	Każdy dzień
8	Każdy dzień określony przez parametr uruchamiane_co (aby uruchamiać zadanie kilka razy w tygodniu, należy użyć operatora OR) gdzie uruchamiane_co oznacza: 1 = niedziela, 2 = poniedziałek, 4 = wtorek, 8 = środa, 16 = czwartek, 32 = piątek, 64 = sobota
16	Określony dzień
32	uruchamiane_co przyjmuje jedną z poniższych wartości: 1 = niedziela, 2 = poniedziałek, 3 = wtorek, 4 = środa, 5 = czwartek, 6 = piątek, 7 = sobota, 8 = codziennie, 9 = w dni robocze,

	10 = w weekendy
64	Parametr nie jest uwzględniany
128	Parametr nie jest uwzględniany

uruchamiane_co_minut określa, co ile minut lub godzin będzie uruchamiane zadanie i może przyjmować poniższe wartości:

- 0 (wartość domyślna) — parametr nie jest uwzględniany.
- x — o określonej godzinie.
- x * 4 — co x minut.
- x * 8 — co x godzin.

uruchamiane_co_miesiąc określa, co który, określony poprzez parametr uruchamiane_co, dzień miesiąca zadanie będzie uruchamiane i może przyjmować jedną z poniższych wartości:

- 1 — pierwszy,
- 2 — drugi,
- 4 — trzeci,
- 8 — czwarty,
- 16 — ostatni.

powtórzone_co określa liczbę tygodni lub miesięcy, co które będzie uruchomiane zadanie.

Przykłady:

Planujemy wykonywanie zadania w każdy poniedziałek o 19:35:

```
USE msdb
EXEC sp_add_jobschedule @job_name = 'Przecena towarów',
@name = 'poniedzialki',
@freq_type = 8,
@freq_interval = 2,
@freq_subday_type = 1,
@freq_recurrence_factor = 1,
@active_start_date = 20021205,
@active_start_time = 193500
GO
The command(s) completed successfully.
```

Komunikat SQLServerAgent is not currently running so it cannot be notified of this action świadczy wyłącznie o zatrzymaniu usługi SQLServerAgent — zdefiniowany harmonogram zadania został pomyślnie utworzony.

Planujemy wykonywanie zadania raz na trzy miesiące, w pierwszą niedzielę nowego kwartału o godzinie 15:20:

```
USE msdb
EXEC sp_add_jobschedule @job_name = 'Przecena towarów',
@name = 'kwartaly ',
@freq_type = 32,
@freq_interval = 1,
@freq_subday_type = 1,
@freq_relative_interval = 1,
@freq_recurrence_factor = 3,
@active_start_date = 20021205,
@active_start_time = 152000
GO
```

The command(s) completed successfully.

Procedura sp_update_jobschedule

Wykonanie procedury umożliwia zmianę istniejącego harmonogramu wykonania zadania.
Składnia:

```
sp_add_jobschedule [@job_id =] id_zadania, | [@job_name =] 'zadanie',
[@name =] 'harmonogram'
[, [@new_name =] 'nowa_nazwa'
[, [@enabled =] aktywny]
[, [@freq_type =] uruchamiane]
[, [@freq_interval =] uruchamiane_co]
[, [@freq_subday_type =] uruchamiane_co_minut]
[, [@freq_subday_interval =] uruchamiane_razy]
[, [@freq_relative_interval =] uruchamiene_co_miesiąc]
[, [@freq_recurrence_factor =] powtórzone_co]
[, [@active_start_date =] uruchamiane_od]
[, [@active_end_date =] uruchamiene_do]
[, [@active_start_time =] uruchamiane_od_godziny]
[, [@active_end_time =] uruchamiane_do_godziny]
```

Przykład:
SQL Server umożliwia czasowe wyłączenie harmonogramu, bez konieczności jego usuwania:
USE msdb EXEC sp_update_jobschedule @job_name = 'Przecena towarów', @name = 'kwartaly ', @new_name = 'kwartaly_ZABLOKOWANE', @enabled = 0 GO The command(s) completed successfully.

Procedura sp_help_jobschedule

Procedura zwraca informację o istniejącym harmonogramie wykonania zadania.
Składnia:

```
sp_help_jobschedule [@job_id =] id_zadania |
[@job_name =] 'zadanie'
[, [@schedule_name =] 'harmonogram'] |
[, [@schedule_id =] id_harmonogramu]
```

Przykład:

```
USE msdb
EXEC sp_help_jobschedule @job_name = 'Przecena towarów',
@schedule_name = 'kwartaly_ZABLOKOWANE'
GO
```

Wykonywanie zadań

Każdy użytkownik bazy danych może wykonać utworzone zadanie, które zostanie wykonane w kontekście zabezpieczeń wywołującego go użytkownika.

Jeżeli zadanie uruchomi użytkownik nienależący do roli sysadmin, zostanie ono pomyślnie wykonane tylko wtedy, gdy użytkownik posiadał wystarczające uprawnienie do wykonania poszczególnych kroków zadania. Kroki, w ramach których wykonywane są skrypty systemu operacyjnego lub pliki wykonywalne w wersji 2000 SQL Servera, domyślnie nie mogą być uruchamianie przez użytkowników nienależących do roli sysadmin. Aby użytkownicy mogli uruchamiać zadania tego typu, konieczne jest skonfigurowanie konta specjalnego użytkownika SQLAgentCmdExec. Z uprawnieniami tego użytkownika będą wykonywane zewnętrzne polecenia zadań uruchamianych przez użytkowników niebędących administratorami. Natomiast skrypty i polecenia systemowe uruchomione przez administratorów zostaną wykonane z uprawnieniami użytkownika, w którego kontekście zabezpieczeń została uruchomiona usługa SQL Server.

Procedura sp_start_job

Wykonanie procedury spowoduje natychmiastowe wykonanie zaplanowanego zadania.

Składnia:

```
sp_start_job [@job_name =] 'zadanie' | [@job_id =] id_zadania
[,[@error_flag =] zarezerwowane]
[,[@server_name =] 'serwer']
[,[@step_name =] 'krok']
[,[@output_flag =] zarezerwowane]
```

Przykład:

Jeżeli próba uruchomienia zadania zakończy się błędem, należy, nawet jeżeli nie zdefiniowana serwerów podrzędnych, jawnie wskazać (np. za pomocą konsoli Enterprise Manager lub opisanej w dalszej części rozdziału procedury sp_add_jobserver) serwer docelowy zadania.

```
USE msdb EXEC sp_start_job @job_name = 'Przecena towarów' GO Job 'Przecena towarów' started successfully.
```

Ponieważ wszystkie trzy próby wykonania drugiego kroku zakończą się błędem, po około dwóch minutach od uruchomienia zadania na ekranie komputera zostanie wyświetlony komunikat informujący o nieudanym wykonaniu zadania wysłany do operatora księgowy. Jednocześnie do tego operatora zostanie wysłana wiadomość elektroniczna.

Procedura sp_stop_job

Wykonanie procedury spowoduje natychmiastowe zatrzymanie uruchomionego zadania. Jeżeli wykonywany był właśnie skrypt, program czy polecenie systemu operacyjnego, ich wykonywanie zostanie przerwane, a dane zostaną utracone.

Składnia:

```
sp_stop_job [@job_name =] 'zadanie'
| [@job_id =] id_zadania
| [@originating_server =] 'serwer_nadrzędny'
| [@server_name =] 'serwer_podrzędny'
```

Przykład:

```
USE msdb
EXEC sp_stop_job @job_name = 'Przecena towarów'
GO
Job 'Przecena towarów' stopped successfully.
```

Historia wykonania zadań

Informacje o udanym lub nieudanym wykonaniu zadań i ich poszczególnych kroków przechowywane są w tabeli msdb..sysjobhistory. Domyślnie SQL Server przechowuje informacje o 1000 ostatnio wykonanych krokach, przy czym maksymalna liczba wierszy (jeden wiersz przechowuje informacje o wykonaniu pojedynczego kroku) przeznaczonych dla pojedynczego zadania nie może przekroczyć 100. Zmienić liczbę wierszy tabeli msdb..sysjobhistory oraz limit wierszy przeznaczonych dla pojedynczego zadania można m.in. za pomocą konsoli Enterprise Managera, konfigurując usługę SQLServerAgent.

Poniższe zapytanie zwraca podstawowe informacje o wykonanych zadaniach:

```
SELECT LEFT(step_name,25), run_status, run_time, run_duration, operator_id_netsent
FROM msdb.dbo.sysjobhistory
GO
run_status run_time run_duration operator_id_netsent
------------------------- ----------- ----------- ------------ -------------------
Aktualizacja cen 1 123336 0 0
generowanie raportów 2 123336 0 0
generowanie raportów 2 123436 0 0
generowanie raportów 0 123536 1 0
(Job outcome) 0 123336 201 2
(5 row(s) affected)
```

Widzimy, że aktualizacja cen została wykonana pomyślnie, następnie SQL Server trzykrotnie próbował wygenerować raporty, za każdym razem czekając minutę pomiędzy kolejnym próbami, po czym całe zadanie zostało zakończone z błędem, o czym został powiadomiony operator poprzez wysłanie wiadomości usługi Posłaniec.

Procedura sp_help_jobhistory

Procedura zwraca informacje o przebiegu wykonania określonych zadań.

Składnia:

```
sp_help_jobhistory [[@job_id =] id_zadania]
[, [@job_name =] 'zadanie']
[, [@step_id =] id_kroku]
[, [@sql_message_id =] id_komunikatu]
[, [@sql_severity =] poziom_błędu]
[, [@start_run_date =] data_uruchomienia]
[, [@end_run_date =] data_zatrzymania]
[, [@start_run_time =] czas_uruchomienia]
[, [@end_run_time =] czas_zatrzymania]
[, [@minimum_run_duration =] minimalny_czas_wykonywania]
[, [@run_status =] status]
[, [@minimum_retries =] minimalna_liczba_prób]
[, [@oldest_first =] porządek]
[, [@server =] 'serwer']
[, [@mode =] 'tryb']
```

gdzie:
status określa rezultat wykonania zadania bądź kroku i może przyjmować jedną z poniższych wartości:

- 0 — błąd,
- 1 — sukces,
- 2 — powtórzenie kroku,
- 3 — przerwane,
- 4 — trwa wykonywanie,
- 5 — nieznany.

porządek określa, czy jako pierwsze zostaną zwrócone informacje o ostatnio wykonanych zadaniach (domyślna wartość 0), czy też informacje o „najstarszych" zadaniach (wartość 1).
tryb określa, czy zwrócone będą podsumowania (domyślna wartość SUMMARY), czy wszystkie informacje (wartość FULL).
Przykład:

```
USE msdb
EXEC sp_help_jobhistory @minimum_run_duration = 2
GO
job_id job_name run_status run_date run_time
[a]run_duration operator_emailed operator_netsent operator_paged retries_attempted server
------------------------------------ ------------------ ------------ --------- -------------
[a]---------------- ------------ ---------------- --------------- ----------------- --------
7F88CD9B-2875-4FB6-94CC-CD574244779C Przecena towarów 0 20021205 123336
[a]201 NULL księgowy NULL 0 (local)
```

Procedura sp_purge_jobhistory

Wykonanie procedury spowoduje usunięcie historii wykonania wszystkich zadań lub określonego zadania. Tylko użytkownicy należący do roli sysadmin mogą usunąć całą zawartość tabeli msdb..sysjobhistory.
Składnia:

```
sp_purge_jobhistory [@job_name =] 'zadanie' | [@job_id =] id_zadania
```

Przykład:

```
USE msdb
EXEC sp_purge_jobhistory @job_name = 'Przecena towarów'
GO
5 history entries purged.
```

Zadania wykonywane na wielu serwerach

SQL Server umożliwia zdalne wykonanie zadań utworzonych na serwerze nadrzędnym (ang. Master Server) na wielu serwerach podrzędnych (ang. Target Server). Zadania utworzone na serwerze nadrzędnym zostaną przesłane do serwerów podrzędnych, a informacja o ich wykonaniu zostanie zapisana w tabelach msdb..sysjobs wszystkich serwerów. Dzięki temu administrator może zdefiniować hierarchię serwerów bazodanowych i zarządzać wieloma serwerami z jednej lokalizacji.

Procedura sp_msx_enlist

Aby możliwe było zdalne wykonywanie zaplanowanych zadań, serwer musi zostać określony jako serwer podrzędny wybranego serwera. Serwerem nadrzędnym może być wyłącznie SQL Server Serwer wersji 7.0 lub późniejszej, uruchomiony w systemie Windows NT lub 2000. Dodatkowo serwery muszą mieć możliwość wymiany danych poprzez potoki nazwane.

SQL Server w wersji 7.0 może zostać połączony jedynie z domyślną instancją SQL Serwera 2000. Wszystkie instancje nazwane nie będą dostępne dla wersji 7.0 serwerów bazodanowych.

Składnia:

```
sp_msx_enlist [@msx_server_name =] 'serwer_nadrzędny'
[, [@location =] 'lokalizacja']
```

Definiując hierarchię serwerów bazodanowych należy pamiętać, że:

- Serwer nadrzędny musi zostać określony jako serwer nadrzędny dla co najmniej jednego serwera podrzędnego.
- W tabeli msdb.dbo.systargetservers serwera nadrzędnego przechowywane są informacje o serwerach podrzędnych, a obecność przynajmniej jednego wiersza w tabeli określa serwer jako serwer nadrzędny.
- Serwer nadrzędny nie może zostać wskazany jako serwer podrzędny innego serwera.
- Podczas rejestrowania każdego serwera podrzędnego w serwerze nadrzędnym tworzony jest login o sufiksie _msx_probe, za pomocą którego serwer podrzędny będzie pobierał zaplanowane dla niego zadania.
- Zmiana hasła dla automatycznie utworzonego loginu spowoduje konieczność ponownego podłączenia serwera podrzędnego.
- Ponieważ login serwera podrzędnego jest uwierzytelniany przez SQL Server, oba serwery, nadrzędny i podrzędny, muszą zostać uruchomione w trybie mieszanym.
- Niemożliwe jest określenie dla tego samego serwera kilku serwerów nadrzędnych.
- Wszystkie serwery bazodanowe muszą znajdować się w obrębie tej samej domeny lub zaufanych domen systemu Windows NT/2000.
- Definiując hierarchię serwerów bazodanowych należy utworzyć operatora MSXOperator.

Przykład:

```
USE msdb
EXEC sp_msx_enlist 'LOLEK', 'Kopia uruchomiona na tym samym komputerze'
GO
The command(s) completed successfully.
```

Procedura sp_add_jobserver

Po określeniu hierarchii serwerów możliwe jest przypisanie zadań poszczególnym serwerom podrzędnym. Wykonanie na serwerze nadrzędnym procedury sp_add_jobserver spowoduje wykonanie określonego zadania na wybranym serwerze podrzędnym.

Tworząc zadania, które będą wykonywane na wielu serwerach, należy upewnić się, że lokalizacja plików lub skryptów, do których odwołuje się zadanie na wszystkich komputerach, jest taka sama — w przeciwnym wypadku próba wykonania zadania zakończy się niepowodzeniem.

Składnia:

```
sp_add_jobserver [@job_id =] id_zadania |
[@job_name =] 'zadanie'
[, [@server_name =] 'serwer']
```

Przykład:

```
USE msdb
EXEC sp_add_jobserver
@job_name = 'Przecena towarów',
@server_name = 'LOLEK\Kopia'
GO
The command(s) completed successfully.
```

Pobieranie zadań przez serwery podrzędne przebiega według następującego schematu:

1. Serwer nadrzędny wysyła wszystkie nowo utworzone zadania do serwerów wymienionych w tabeli msdb..sysdownloadlist.
2. Serwery podrzędne okresowo sprawdzają, czy nie zmieniła się definicja zaplanowanych dla nich zadań. Jeżeli tak — pobierają one nowe zadania.

Zmodyfikować zadanie wykonywane na wielu serwerach można jedynie na serwerze nadrzędnym, np. poprzez wykonanie procedury systemowej sp_update_job.

Procedura sp_help_jobserver

Procedura zwraca informacje o serwerach podrzędnych i wykonywanych na nich zadaniach. Szczegółowe informacje o wykonaniu poszczególnych zadań można uzyskać m.in. poprzez odczytanie zawartości tabeli msdb..sysjobhistory serwerów, na których wykonano zadania.
Składnia:

```
sp_help_jobserver [@job_id =] id_zadania |
[@job_name =] 'zadanie'
[, [@show_last_run_details =] szczegóły_wykonania_zadania]
```

gdzie:
szczegóły_wykonania_zadania określa, czy zostaną zwrócone również informacje o czasie rozpoczęcia, okresie w którym zadanie było wykonywane i komunikat wyświetlony przez ostatnie zadanie (wartość 1). Domyślną wartością jest 0.
Przykład:

```
EXEC sp_help_jobserver
@job_name ='Przecena towarów',
@show_last_run_details = 1
```

Procedura sp_delete_jobserver

Wykonanie procedury spowoduje usunięcie zadania z listy zadań przeznaczonych dla wybranego serwera podrzędnego.
Składnia:

```
sp_delete_jobserver [@job_id =] id_zadania |
[@job_name =] 'zadanie',
[@server_name =] 'serwer'
```

Przykład:

```
USE msdb
EXEC sp_delete_jobserver @job_name = 'Przecena towarów',
@server_name = 'LOLEK\Kopia'
GO
The command(s) completed successfully.
```

Procedura sp_msx_defect

Usunąć serwer z listy serwerów podrzędnych możemy za pomocą procedury sp_msx_defect.
Składnia:

```
sp_msx_defect [@forced_defection =] wymuś_odłączenie
```

gdzie:
wymuś_odłączenie przyjmuje wartości 0 (wartość domyślna) lub 1. Wartość 1 powinna być używana wyłącznie w celu odłączenia od serwera nadrzędnego, który jest trwale niedostępny. Użycie tej wartości spowoduje usunięcie wpisów w bazie msdb serwera podrzędnego. Aby zsynchronizować serwer nadrzędny, na którym wymuszono odłączenie serwera podrzędnego, należy na nim wykonać poniższą procedurę:

```
USE msdb
EXEC sp_delete_targetserver @server_name = 'serwer_podrzędny', @post_defection = 0
GO
```

Przykład:

```
EXEC msdb..sp_msx_defect
GO
Server defected from MSX 'LOLEK'. 1 Job(s) deleted.
```

Usuwanie zadań

Zadania i ich poszczególne harmonogramy mogą zostać zablokowane, co spowoduje, że nie będą one automatycznie wykonywane. Jeżeli jednak zadanie, określony harmonogram czy pewne kroki zadania nie są dłużej potrzebne, to powinny zostać usunięte.

Procedura sp_delete_jobschedule

Wykonanie procedury usuwa harmonogram wybranego zadania.
Składnia:

```
sp_delete_jobschedule [@job_id =] id_zadania, | [@job_name =] 'zadanie',
[@name =] 'harmonogram'
```

Przykład:

```
USE msdb
EXEC sp_delete_jobschedule @job_name = 'Przecena towarów',
@name = 'kwartaly_ZABLOKOWANE'
GO
```

```
The command(s) completed successfully.
```

Procedura sp_delete_jobste

Wykonanie procedury spowoduje usunięcie wybranego kroku zadania.
Składnia:

```
sp_delete_jobstep [@job_id =] id_zadania, | [@job_name =] 'zadanie',
[@step_id =] id_kroku
```

Przykład:

```
USE msdb
EXEC sp_delete_jobstep @job_name = 'Przecena towarów',
@step_id = 2
GO
The command(s) completed successfully.
```

Procedura sp_delete_job

Wykonanie procedury spowoduje usunięcie wybranego zadania.
Składnia:

```
sp_delete_job [@job_id =] id_zadania, | [@job_name =] 'zadanie'
[, [@originating_server =] 'serwer']
```

Przykład:

```
USE msdb
EXEC sp_delete_job @job_name = 'Przecena towarów'
GO
The command(s) completed successfully.
```

Alarmy

Bez możliwości ciągłego monitorowania wydajności serwera bazodanowego i automatycznego reagowania na pojawiające się w jego działaniu błędy, automatyzacja pracy administratora bazy danych byłaby niemożliwa. SQL Server umożliwia tworzenie alarmów dwóch typów:

1. Alarmy wywoływane wystąpieniem błędu (zarówno predefiniowanego, jak i zdefiniowanego przez użytkownika) o określonym numerze lub wystąpieniem błędu o określonym poziomie ważności.
2. Alarmy wywoływane bieżącą wydajnością serwera bazodanowego.

Ponieważ usługi MSSQLServer i SQLServerAgent wymieniają między sobą informację poprzez dziennik aplikacji systemu informacyjnego, aby możliwe było wywołanie alarmu przez usługę Server Agent, usługa SQL Server musi zapisać w nim komunikat błędu (ang. Message).

Wywołanie alarmu może spowodować wykonanie określonego zadania oraz powiadomienie wybranego operatora. Z reguły alarmy wywoływane wystąpieniem błędu systemowego (błędu o numerze od 1 do 50 000) reagują na typowe błędy, jak np. wypełnienie pliku dziennika transakcyjnego powyżej określonego poziomu. Natomiast alarmy wywoływane wystąpieniem błędu zdefiniowanego przez użytkownika muszą zostać wywołane za pomocą instrukcji RAISERROR i reagują na błędy typowe dla konkretnej bazy danych, np. próbę obniżenia cen towarów o ponad 20%.

Prawie całkowite zautomatyzowanie pracy SQL Servera możliwe jest dzięki jednoczesnemu wykorzystaniu wyzwalaczy, błędów użytkownika, alarmów i zadań. Dodatkowo o wykonaniu poszczególnych operacji może być powiadamiany wybrany operator. Automatyczna reakcja na niedozwoloną akcję użytkownika (np. zmianę ceny towaru) może przebiegać następująco:

1. Próba wykonania instrukcji UPDATE spowoduje wywołanie wyzwalacza, który sprawdza, czy nowa cena (cena z tabeli INSERTED) nie jest niższa o ponad 20% od ceny oryginalnej i jeżeli tak — wywołuje błąd użytkownika o numerze 50 001 na przykładowym poziome ważności 14 (na tym poziomie znajdują się komunikaty błędów związanych z niewystarczającymi uprawnieniami).
2. SQL Server zapisuje komunikat błędu użytkownika w dzienniku aplikacji.
3. SQLServerAgent odczytuje informacje o błędzie i sprawdza, czy w przechowywanej w buforze tabeli msdb..sysalerts nie znajduje się komunikat powiązany z błędem o tym numerze lub z błędami o poziome ważności 14 i jeżeli tak — wywołuje określony alarm.

> Wywołany zostanie wyłącznie jeden, bardziej specyficzny dla komunikatu błędu alarm. Jeżeli więc istnieją alarmy powiązane z błędami poziomu 14 i błędem 50 0001, zostanie wywołany alarm powiązany z błędem o określonym numerze.

4. Wywołany alarm wysyła do określonego operatora komunikat i jednocześnie uruchamia zadanie, w ramach którego np. blokowany jest dostęp do danych o towarze i generowany raport sprzedaży tego towaru w ciągu ostatniego dnia.

> Ponieważ SQL Server umożliwia przekazywanie alarmom i zadaniom parametrów, administrator może zostać powiadomiony m.in. o tym, który z użytkowników próbował zmienić cenę i jakiego towaru ta zmiana dotyczyła.

5. O pomyślnym lub niepomyślnym wykonaniu zadania ponownie informowany jest wybrany operator.

Komunikaty błędów

Użytkownicy należący do roli sysadmin lub serveradmin mogą tworzyć, modyfikować i usuwać komunikaty błędów.

Procedura sp_addmessage

Wykonanie procedury spowoduje dodanie własnego komunikatu błędu. Wszystkie komunikaty błędów przechowywane są w tabeli master.dbo.sysmessages.

Składnia:

```
sp_addmessage [@msgnum =] numer_komunikatu,
[@severity =] poziom_ważności,
[@msgtext =] 'komunikat'
[, [@lang =] 'język']
[, [@with_log =] 'logowalny']
[, [@replace =] 'zastępowany_komunikat']
```

gdzie:

poziom_ważności może przyjmować wartości z zakresu od 1 od 25, przy czym komunikaty błędów o poziomie ważności powyżej 18 są automatycznie zapisywane w dzienniku aplikacji i mogą być tworzone wyłącznie przez administratorów serwera,

język określa język komunikatu. Jeżeli komunikat zostanie utworzony w kilku językach, m.in. w języku określonym jako domyślny dla loginu użytkownika, komunikat zostanie wyświetlony użytkownikowi w jego języku domyślnym,

> Zanim zostaną utworzone komunikaty w innych językach, musi istnieć komunikat błędu o określonym numerze i tym samym poziomie ważności utworzony w języku angielskim.

logowalny określa, czy komunikat niezależnie od poziomu ważności zostanie zapisany w dzienniku aplikacji (wartość TRUE). Domyślną wartością jest FALSE,
zastępowany_komunikat określa komunikat, który zostanie zastąpiony nowo utworzonym komunikatem. Parametr musi zostać podany, jeżeli komunikat o określonym numerze już istnieje.
Przykład:

```
USE master
EXEC sp_addmessage 50001, 14, 'Price product"s %i was reduced by user %s',
'us_english','true'
EXEC sp_addmessage 50001, 14, 'Cena produktu %i została obniżona przez użytkownika %s',
'polish', 'true'
GO
(1 row(s) affected)
(1 row(s) affected)
```

Procedura sp_altermessage

Wykonanie procedury spowoduje zmianę istniejącego komunikatu błędu.
Składnia:

```
sp_altermessage [@message_id =] numer_komunikatu,
[@parameter =] 'logowalny',
[@parameter_value =] 'wartość'
```

gdzie:
logowalny określa, czy komunikat błędu będzie zapisany w dzienniku aplikacji (wartość WITH_LOG),
wartość określa, czy komunikat zawsze zostanie zapisany w dzienniku aplikacji (wartość TRUE), czy w zależności od metody wywołania błędu (wartość FALSE).
Przykład:

```
USE master
EXEC sp_altermessage 50001, 'WITH_LOG', 'false'
GO
(2 row(s) affected)
```

Procedura sp_dropmessage

Usunąć komunikat błędu użytkownika możemy poprzez wykonanie procedury systemowej sp_dropmessage.
Składnia:

```
sp_dropmessage [@msgnum =] numer_komunikatu
[, [@lang =] 'język']
```

Przykład:

```
USE master
```

```
EXEC sp_dropmessage 50001, 'polish'
GO
(1 row(s) affected)
```

Możliwe jest usunięcie komunikatu błędu powiązanego z określonym alarmem bez zrywania tego powiązania. Jeżeli zostanie utworzony inny komunikat błędu o tym samym numerze, jego wystąpienie spowoduje wywołanie alarmu.

Tworzenie alarmów

Wystąpienie błędu o określonym numerze lub błędu na określonym poziomie ważności może spowodować wywołanie alarmu. Definicje alarmów zapisane są w kolejnych wierszach tabeli msdb..sysalerts. SQL Server umożliwia utworzenie wielu alarmów wywoływanych wystąpieniem błędu o tym samym numerze, ale w tym wypadku konieczne jest jawne określenie bazy danych, dla której wystąpienie błędu spowoduje wywołanie alarmu (słowo kluczowe ALL oznacza wszystkie bazy danych).

Procedura sp_add_alert

Wykonanie procedury doda nowy alarm.
Składnia:

```
sp_add_alert [@name =] 'alarm'
[, [@message_id =] numer_komunikatu]
[, [@severity =] poziom_ważności]
[, [@enabled =] aktywny]
[, [@delay_between_responses =] opóźnienie_wykonania]
[, [@notification_message =] 'komunikat']
[, [@include_event_description_in =] dołącz_komunikat_błędu_do]
[, [@database_name =] 'baza_danych']
[, [@event_description_keyword =] 'wzorzec_tekstu_komunikatu']
[, {[@job_id =] id_zadania | [@job_name =] 'zadanie'}]
[, [@performance_condition =] 'monitor_wydajności']
[, [@category_name =] 'kategoria']
```

gdzie:
opóźnienie_wykonania określa liczbę sekund pomiędzy wywołaniem alarmu a uruchomieniem przez niego zadania czy powiadomieniem operatora. Zwiększenie domyślnej wartości 0 może zapobiec wielokrotnemu wykonaniu zadania przez występujący w krótkich odstępach czasu błąd, komunikat jest dodatkowym, opcjonalnym komunikatem przesłanym operatorowi,
dołącz_komunikat_błędu_do określa, czy opis błędu będzie dołączany do powiadomienia operatora, gdzie:

- 0 oznacza brak opisu,
- 1 — dołączanie opisu do wiadomości e-mail,
- 2 — dołączanie opisu do wiadomości wysłanej na pager,
- 4 — dołączanie opisu do wiadomości wysłanej poprzez usługę Posłaniec,
- 5 — dołączenie opisu do wiadomości e-mail i do wiadomości wysłanych pleceniem NET SEND (wartość domyślna).

Parametr może przyjmować wartość będącą dowolną alternatywą logiczną wymienionych wartości.
wzorzec_tekstu_komunikatu określa ciąg znaków, który musi wystąpić w tekście komunikatu błędu, aby alarm został wywołany,
monitor_wydajności jest wyrażeniem postaci licznik operator_porównania wartość określającym warunki związane z wydajnością SQL Servera, których wystąpienie spowoduje wywołanie alarmu.
Tworząc alarmy wywoływane, należy pamiętać, że:

- ♣ Błąd musi zostać zapisany w dzienniku aplikacji, aby możliwe było wywołanie alarmu.
- ♣ Tworząc alarmy wywoływane wystąpieniem błędu we wszystkich bazach danych, należy zadbać o to, aby komunikat przesłany operatorowi był maksymalnie dokładny, umieszczając w nim (co najmniej) nazwę bazy danych, w której wystąpił błąd.
- ♣ Błędy o poziomach od 19 do 25 są błędami krytycznymi i ich wystąpienie zawsze powinno spowodować powiadomienie operatora.
- ♣ SQL Server zawiera około 4000 predefiniowanych komunikatów błędów i przykładowe alarmy związane z błędami o poziomie ważności powyżej 18, które mogą być modyfikowane przez administratorów.
- ♣ Błędy użytkownika mają numery powyżej 50 000.

Przykład:

```
USE msdb
EXEC sp_add_alert @name = 'Zmiana ceny',
@message_id = 50001,
@database_name = 'Northwind'
GO
SQLServerAgent is not currently running so it cannot be notified of this action.
```

Procedura sp_add_notification

Administratorzy serwera mogą powiadomić wybranego operatora o wywołaniu alarmu, wykonując procedurę sp_add_notification.
Składnia:

```
sp_add_notification [@alert_name =] 'alarm',
[@operator_name =] 'operator',
[@notification_method =] sposób_powiadomienia
```

gdzie:
sposób_powiadomienia określa metodę powiadomienia operatora i może być dowolną kombinacją wartości: 1 (wysłanie wiadomości e-mail), 2 (powiadomienie na pager) i 4 (wysłanie wiadomości sieciowej).
Przykład:

```
USE msdb
EXEC sp_add_notification 'Zmiana ceny', 'księgowy', 4
GO
The command(s) completed successfully.
```

Procedura sp_help_notification

Procedura zwraca informacje o powiadamianych przez alarm operatorach lub o alarmach, których wywołanie spowoduje powiadomienie danego operatora.
Składnia:

```
sp_help_notification [@object_type =] 'typ_objektu',
[@name =] 'nazwa',
[@enum_type =] 'informacje',
[@notification_method =] powiadomienie
[, [@target_name =] 'obiekt_docelowy']
```

gdzie:
typ_objektu określa, czy zwracana będzie lista alarmów powiązanych z wybranym operatorem, czy też lista operatorów związanych z wybranym alarmem. Może przyjmować jedną z dwóch wartości: ALERTS lub OPERATORS,
informacje określa zakres zwracanych przez procedurę informacji i może przyjmować jedną z poniższych wartości:

- ACTUAL — zwraca listę obiektów typu typ_objektu o nazwie nazwa,
- ALL — zwraca listę wszystkich obiektów typu typ_objektu,
- TARGET zwraca listę obiektów typu typ_objektu powiązanych z obiektem obiekt_docelowy.

powiadomienie określa sposób powiadamiania i może przyjmować jedną z poniższych wartości:

- 1 — e-mail,
- 2 — pager,
- 4 — NET SEND,
- 7 — wszystkie sposoby.

obiekt_docelowy — określa obiekt typu typ_objektu powiązanego z powiadomieniem.
Przykład:

```
USE msdb
EXEC sp_help_notification 'OPERATORS', 'Zmiana ceny', 'ACTUAL', 7
GO
```

Procedura sp_update_alert

Definicje istniejącego alarmu możemy zmodyfikować, wykonując procedurę systemową sp_update_alert.
Składnia:

```
sp_update_alert [@name =] 'alarm'
[, [@new_name =] 'nowa_nazwa']
[, [@message_id =] numer_komunikatu]
[, [@severity =] poziom_ważności]
[, [@enabled =] aktywny]
[, [@delay_between_responses =] opóźnienie_wykonania]
[, [@notification_message =] 'komunikat']
[, [@include_event_description_in =] dołącz_komunikat_błędu_do]
[, [@database_name =] 'baza_danych']
[, [@event_description_keyword =] 'wzorzec_tekstu_komunikatu']
[, {[@job_id =] id_zadania | [@job_name =] 'zadanie'}]
[, [@occurrence_count =] licznik_wystąpień]
[, [@count_reset_date =] data_wyzerowania]
[, [@count_reset_time =] czas_wyzerowania]
[, [@last_occurrence_date =] data_ostatniego_wystąpienia]
[, [@last_occurrence_time =] czas_ostatniego_wystąpienia]
[, [@last_response_date =] data_ostatniego_uruchomienia]
[, [@last_response_time =] czas_ostatniego_uruchomienia]
[, [@performance_condition =] 'monitor_wydajności']
```

```
[, [@category_name =] 'kategoria']
```

gdzie:
licznik_wystapień, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie licznika wystąpień alarmu,
data_wyzerowania, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie daty ostatniego wyzerowania licznika wystąpień alarmu,
czas_wyzerowania, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie godziny ostatniego wyzerowania licznika wystąpień alarmu,
data_ostatniego_wystąpienia, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie daty ostatniego wystąpienia alarmu, czas_ostatniego_wystąpienia, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie godziny ostatniego wystąpienia alarmu,
data_ostatniego_uruchomienia, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie daty ostatniego uruchomienia przez alarm zadania lub powiadomienia operatora,
czas_ostatniego_uruchomienia, jeżeli przyjmuje wartość 0 (domyślną wartością jest NULL), spowoduje wyzerowanie daty ostatniego uruchomienia przez alarm zadania lub powiadomienia operatora.
Przykład:

```
USE msdb
EXEC sp_update_alert @name = 'Zmiana ceny',
@notification_message = 'Wystąpił błąd w bazie Northwind'
GO
The command(s) completed successfully.
```

Procedura sp_help_alert

Procedura zwraca informacje o wszystkich lub o wybranych alarmach.
Składnia:

```
sp_help_alert [[@alert_name =] 'alarm']
[, [@order_by =] 'uporządkuj_według']
[, [@alert_id =] id_alarmu]
[, [@category_name =] 'kategoria']
```

gdzie:
uporządkuj_według oznacza atrybut, według którego uporządkowane będą zwracane informacje. Domyślną wartością jest litera N oznaczająca nazwę alarmu.
Przykład:

```
USE msdb
EXEC sp_help_alert 'Zmiana ceny'
GO
```

Wywoływanie błędów użytkownika

Komunikaty błędów zdefiniowanych przez użytkownika, w przeciwieństwie do błędów systemowych, muszą zostać wywołane programowo. Z reguły błędy użytkownika wywoływane są poprzez wyzwalacze lub procedury składowane.

Instrukcja RAISERROR

Instrukcja zwraca zdefiniowany przez użytkownika komunikat błędu.
Składnia:

```
RAISERROR ({id_komunikatu | komunikat} {, poziom_ważności, stan}
[, argument [,…n]])
[WITH opcja [,…n]]
```

gdzie:
stan jest dowolną liczbą całkowitą z zakresu od 1 do 127 reprezentującą stan aplikacji wywołującej błąd,
argument jest argumentem przekazanym do komunikatu błędu. SQL Server umożliwia przekazanie argumentów typu integer, char, varchar, binary oraz varbinary,
opcja może przyjmować poniższe wartości:

- LOG — zapisanie komunikatu błędu w dzienniku SQL Servera i dzienniku aplikacji (maksymalna długość zapisywanych komunikatów wynosi 440 bajtów).
- NOWAIT — komunikat błędu zostanie natychmiast wysłany.
- SETERROR — wartość zmiennej @@ERROR zostanie ustawiona na numer_komunikatu lub 50 000.

Instrukcja RAISERROR umożliwia przekazanie wartości parametrów komunikatom błędów zdefiniowanych np. za pomocą procedury sp_addmessage. W ramach komunikatu można wyświetlić wartości zmiennych, których typ jest reprezentowany za pomocą symboli:

- %d lub %i — liczby całkowite,
- %o — liczby ósemkowe
- %p — wskaźniki,
- %s — ciągi znaków,
- %h — liczby szesnastkowe.

Przykład:

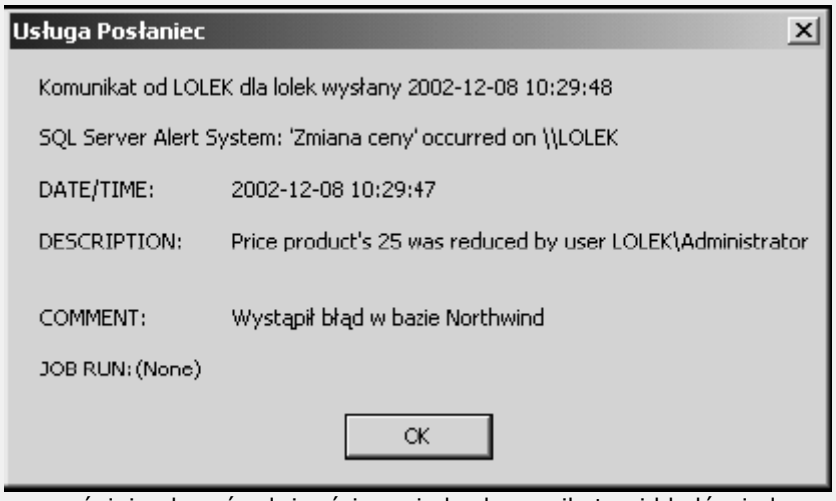
```
USE Northwind
GO
CREATE PROCEDURE p_ZmianaCeny
@ID_towaru int = NULL,
@cena money
AS
DECLARE @uzytkownik varchar(60)
SET @uzytkownik = suser_sname()
BEGIN TRANSACTION
UPDATE Northwind.dbo.Products
SET UnitPrice = @cena
WHERE ProductID = @ID_towaru
RAISERROR (50001, 14, 1, @ID_towaru, @uzytkownik)
WITH LOG
COMMIT TRANSACTION
GO
The command(s) completed successfully.
```

Zmiana ceny produktu poprzez procedurę spowoduje zwrócenie komunikatu błędu i wywołanie alarmu, co wywoła powiadomienie operatora księgowy:

```
USE Northwind
```

```
EXEC dbo.p_ZmianaCeny @ID_towaru = 25, @cena = 19
GO
(1 row(s) affected)
Server: Msg 50001, Level 14, State 1, Procedure p_ZmianaCeny, Line 11
Price product's 25 was reduced by user LOLEK\Administrator
```

Jeżeli tylko podczas wykonywania procedury usługa Server Agent była uruchomiona, po chwili na ekranie komputera zostanie wyświetlone okno dialogowe pokazane na rysunku 17.4.



Rysunek 17.4. Aby wyraźniej pokazać zależności pomiędzy komunikatami błędów i alarmami, poniższy alarm wywoływany jest przez procedurę, a nie przez wyzwalacz utworzony dla tabeli Products

Alarmy wywoływane bieżącą wydajnością

Oprócz alarmów reagujących na błędy systemowe lub błędy użytkownika, SQL Server umożliwia tworzenie alarmów wywoływanych bieżącymi wskazaniami liczników wydajności.

Podczas instalacji SQL Servera w systemach Windows NT/2000 rejestrowanych jest kilka dodatkowych obiektów monitora wydajności, z których każdy zawiera co najmniej kilka liczników. W rezultacie, za pomocą systemowego narzędzia Wydajność, możliwe jest dokładne monitorowanie wydajności poszczególnych aspektów pracy serwera bazodanowego. Każdy z tych liczników może zostać wykorzystany do wywołania alarmu.

Na przykład możliwe jest utworzenie alarmu, który zostanie wywołany w momencie, gdy wykorzystanie pliku dziennika transakcyjnego bazy Northwind przekroczy 70% lub gdy wielkość pliku dziennika transakcyjnego bazy pubs przekroczy 5 MB:

```
USE msdb
EXEC sp_add_alert @name = 'Northwind_dt',
@delay_between_responses = 60,
@performance_condition = 'SQLServer:Databases|Percent Log Used|Northwind|>|70'
GO
The command(s) completed successfully.
```

Należy pamiętać, że wartość liczników wydajności sprawdzana jest kilka razy na minutę. Tak więc, aby zadanie uruchomione przez alarm zapobiegło wystąpieniu błędu (w tym przypadku poprzez wykonanie kopii dziennika transakcyjnego, obcięcie jego nieaktywnej części lub zwiększenie rozmiaru pliku dziennika), alarm powinien zostać wywołany z pewnym wyprzedzeniem (np. po wypełnieniu dziennika transakcyjnego w 70 a nie 80 procentach). Z drugiej strony, aby uniknąć sytuacji, kiedy system, któremu zabrakło pewnych zasobów (np. pamięci operacyjnej, co spowodowało wielokrotne obniżenie wydajności), wielokrotnie uruchamia to samo zadanie, należy zwiększyć opóźnienie pomiędzy wywołaniem alarmu a jego reakcją.

Tematowi monitorowania pracy SQL Servera, w tym opisowi najważniejszych liczników programu Wydajność, poświecony jest rozdział 20.

Usuwanie alarmów

Niepotrzebne alarmy lub powiadomienia powinny zostać usunięte z bazy danych.

Procedura sp_delete_notification

Wykonanie procedury spowoduje usunięcie powiadomień wysyłanych do określonego operatora poprzez wybrany alarm.
Składnia:

```
sp_delete_notification [@alert_name =] 'alarm',
[@operator_name =] 'operator'
```

Przykład:

```
USE msdb
EXEC sp_delete_notification 'Zmiana ceny', 'księgowy'
GO
The command(s) completed successfully.
```

Procedura sp_delete_alert

Usunąć alarm możemy poprzez wykonanie procedury sp_delete_alert. Wszystkie zdefiniowane dla alarmu powiadomienia zostaną automatycznie usunięte.
Składnia:

```
sp_delete_alert [@name =] 'alarm'
```

Przykład:

```
USE msdb
EXEC sp_delete_alert 'Zmiana ceny'
GO
The command(s) completed successfully.
```

Kurs Transact SQL, część 18 (Tworzenie kopii zapasowych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Zabezpieczenie przed utratą danych jest jednym z podstawowych obowiązków administratora baz danych. Niezależnie od odporności na awarie wykorzystywanego sprzętu komputerowego (redundantne macierze dyskowe, zasilacze awaryjne itp.) jedynie posiadanie aktualnej kopii zapasowej gwarantuje odzyskanie danych utraconych w wyniku:

- Przypadkowego lub celowego wykonania instrukcji TRUNCATE albo DELETE.
- Przypadkowego lub celowego wykonania instrukcji UPDATE z brakującym albo błędnie sformułowanym warunkiem logicznym w klauzuli WHERE.
- Przypadkowego lub celowego zastąpienia istniejących danych bądź całej bazy danych importowanymi danymi albo odtwarzaną kopią zapasową.
- Kradzieży.
- Działania wirusów.

Ponieważ bazy danych z reguły zawierają niezbędne dla funkcjonowania firmy informacje, każdy administrator powinien przygotować i przetestować strategię wykonywania i odtwarzania kopii zapasowych. W ramach strategii należy określić m.in.:

- Dopuszczalny koszt wykonywania kopii zapasowych, z uwzględnieniem czasu potrzebnego na wykonanie kopii i kosztu nośników niezbędnych do przechowywania danych archiwalnych.
- Dopuszczalny okres czasu potrzebny do przywrócenia sprawności systemu po awarii.
- Czy dopuszczalna jest utrata danych (np. danych z ostatniego dnia roboczego), a jeżeli tak, to ilu?

Na podstawie odpowiedzi na powyższe pytania należy określić model odtwarzania baz danych. SQL Server 2000 umożliwia wybór jednego z trzech trybów dla każdej bazy danych, zarówno dla baz systemowych, jak i baz użytkownika:

1. Kompletny model odtwarzania (ang. Full Recovery Model) — domyślny model odtwarzania baz użytkownika pozwalający na odtworzenie bazy danych do dowolnego momentu w czasie na podstawie wykonanych kopii plików danych oraz kopii dziennika transakcyjnego. W tym modelu do dziennika transakcyjnego zapisywane są wszystkie operacje, włącznie z tworzeniem indeksów i operacjami nielogowalnymi, takimi jak wykonanie instrukcji BULK INSERT. W rezultacie, na podstawie wpisów dziennika, możliwe jest odtworzenie wszystkich informacji przechowywanych w bazie danych w dowolnym, określonym czasie. Ponieważ wszystkie instrukcje zostają zapisane w dzienniku transakcyjnym, a automatyczne usuwanie jego nieaktywnej części jest niemożliwe, należy liczyć się z koniecznością systematycznego wykonywania kopii dziennika transakcyjnego zapobiegającej jego przepełnieniu oraz z koniecznością zarezerwowania dla pliku dziennika minimum 30% wielkości plików bazodanowych.

   SQL Server 2000 umożliwia oznaczanie transakcji za pomocą instrukcji BEGIN TRANSACTION WITH MARK, a następnie odtworzenie bazy danych do stanu sprzed transakcji lub bezpośrednio po wykonaniu transakcji.
2. Pośredni model odtwarzania (ang. Bulk_Logged Recovery Model) — podobnie jak poprzedni model umożliwia on wykonywanie kopii zapasowych zarówno plików danych, jak i dziennika transakcyjnego. W przeciwieństwie do modelu kompletnego, w dzienniku transakcyjnym nie są zapisane wszystkie informacje potrzebne do odtworzenia wykonania instrukcji nielogowalnych (np. BULK INSERT, SELECT INTO, WRITETEXT). Zamiast tego w dzienniku znajduje się jedynie wpis potwierdzający wykonanie tych instrukcji. W rezultacie plik dziennika transakcyjnego może być mniejszy i nie ulegnie tak szybko przepełnieniu, ale ponieważ nie jest możliwe usunięcie nieaktywnej części dziennika, nadal konieczne jest systematyczne wykonywanie jego kopii zapasowych. Ograniczeniem tego trybu jest niemożność odtworzenia bazy do dowolnego momentu w czasie, a zaletą — szybsze wykonywanie operacji nielogowalnych.
3. Uproszczony model odtwarzania (ang. Simple Recovery Model) — domyślny model odtwarzania baz systemowych umożliwia wykonanie kopii zapasowych wyłącznie plików danych i jest zalecany dla niewielkich baz danych oraz baz danych przechowujących rzadko modyfikowane dane. W tym modelu podczas wykonywania punktu kontrolnego nieaktywna część dziennika transakcyjnego zostaje automatycznie usunięta, co zapobiega przepełnieniu czy wzrostowi pliku dziennika transakcyjnego, ale uniemożliwia

wykonywanie jego kopii (ponieważ część danych z dziennika zostaje bezpowrotnie utracona, dziennik nie zawiera ciągłej historii wykonywanych instrukcji).

W poprzednich wersjach SQL Servera odpowiednikiem wyboru uproszczonego modelu odtwarzania jest ustawienie opcji trunc. log on chkpt (obetnij dziennik transakcyjny podczas punktu kontrolnego).

Zmienić model odtwarzania bazy danych możemy za pomocą instrukcji ALTER DATABASE SET RECOVERY:

```
ALTER DATABASE Northwind
SET RECOVERY FULL
GO
ALTER DATABASE msdb
SET RECOVERY BULK_LOGGED
GO
The command(s) completed successfully.
```

Po wybraniu modelu odtwarzania należy określić typ strategii tworzenia kopii zapasowych i na jego podstawie zaplanować wykonywanie kopii odpowiedniego typu w zamierzonym czasie. Ponieważ operacje wykonania kopii zapasowych silnie obciążają serwer, należy zaplanować ich wykonywanie w czasie niskiej aktywności użytkowników systemu.

Kopie zapasowe

SQL Server umożliwia wykonanie kopii zapasowych bez konieczności zatrzymywania serwera czy blokowania użytkownikom dostępu do danych. Kopia zapasowa bazy danych zawiera:

- ♣ Logiczną i fizyczną strukturę bazy danych.
- ♣ Dane.
- ♣ Część dziennika transakcyjnego przechowującą informacje o instrukcjach wykonanych od momentu rozpoczęcia do momentu zakończenia wykonywania kopii zapasowej.

Natomiast w kopii dziennika transakcyjnego znajdują się wyłącznie uporządkowane wpisy z dziennika (instrukcje języka Transact-SQL). Ponieważ w dzienniku przechowywana jest informacja o czasie wykonania każdej operacji wyłącznie na podstawie kopii dziennika, możliwe jest odtworzenie bazy do określonego momentu w czasie czy prześledzenie historii zmian danych.

Podczas tworzenia kopii zapasowych SQL Server:

1. Wykonuje punkt kontrolny i przypisuje ostatniej aktywnej transakcji numer LSN (ang. Log Sequence Number).
2. Zapisuje przechowywane na dysku strony bazy danych bezpośrednio do urządzenia kopii zapasowej (zapisywana jest zawartość plików bazodanowych z pominięciem bufora pamięci podręcznej).

> Bezpośredni zapis stron z dysku do urządzenia kopii zapasowych powoduje, że podczas jej wykonywania niemożliwe jest wykonanie instrukcji modyfikujących fizyczną strukturę bazy danych.

3. Zapisuje wszystkie transakcje z dziennika transakcyjnego o numerach wyższych od numeru LSN.

Kopie zapasowe mogą być wykonywane na tymczasowych lub trwałych urządzeniach kopii zapasowych, z których każde może być:

- ♣ Plikiem znajdującym się na dysku twardym (nie musi to być lokalny dysk twardy komputera, na którym uruchomiono SQL Server).
- ♣ Lokalnie podłączonym urządzeniem taśmowym zgodnym ze standardem Microsoft Tape.
- ♣ Dowolnym urządzeniem wyprodukowanym przez inne firmy połączonym z SQL Serverem poprzez potoki nazwane.

Ponieważ SQL Server tworzy kopie zapasowe, bezpośrednio odczytując zawartość plików bazy danych, podczas ich wykonywania niemożliwe jest:

1. Zmienianie struktury bazy danych (wykonywanie instrukcji ALTER DATABASE).
2. Zwiększanie lub zmniejszanie rozmiaru plików bazy danych.
3. Tworzenie indeksów.
4. Wykonywanie operacji nielogowalnych.

SQL Server nie dopuści do wykonania żadnej z powyższych operacji, informując użytkownika o tym, że trwa wykonywanie kopii zapasowej.

Kiedy tworzyć kopie zapasowe?

Choć odpowiedź na to pytanie zależy przede wszystkim od konkretnych wymogów i możliwości użytkownika bazy danych, istnieje kilka ogólnych wskazówek ułatwiających zaplanowanie wykonywania kopii zapasowych zarówno baz użytkownika, jak i baz systemowych.

Baza master

Baza master przechowuje katalog serwera — zbiór metainformacji niezbędnych do uruchomienia i działania SQL Servera. Lokalizacja plików tej bazy danych musi zostać podana jako parametr startowy usługi MSSQLServer, a ich brak lub uszkodzenie uniemożliwia uruchomienie serwera. Ponieważ brak aktualnej kopii zapasowej tej bazy może wielokrotnie wydłużyć czas potrzebny na przywrócenie sprawności systemu, kopie bazy master należy wykonać zawsze, gdy:

- ♣ Utworzono lub usunięto bazę użytkownika (wykonano instrukcję CREATE DATABASE lub DROP DATABASE).
- ♣ Zmodyfikowano strukturę jakiejkolwiek bazy danych (wykonano instrukcję ALTER DATABASE),
- ♣ Utworzono, usunięto lub zmodyfikowano loginy użytkowników.
- ♣ Dodano lub usunięto loginy użytkowników z predefiniowanych ról serwera.
- ♣ Utworzono, usunięto lub zmodyfikowano urządzenie kopii zapasowej.
- ♣ Połączono, odłączono lub zmodyfikowano połączenie serwerów bazodanowych.
- ♣ Utworzono, usunięto lub zmodyfikowano komunikat błędu.

Na płycie instalacyjnej SQL Servera znajdują się oryginalne pliki wszystkich systemowych baz danych oraz przykładowych baz Northwind i pubs. Znajduje się na niej również, opisany w rozdziale 19., program Rebuildm.exe pozwalający na odtworzenie po awarii serwera baz systemowych, w tym bazy master.

Baza msdb

W bazie msdb przechowywane są definicje alarmów, operatorów, zadań oraz ich wzajemnych powiązań. W tej bazie znajduje się również historia wykonania poszczególnych zadań, wywołania alarmów oraz powiadomienia operatorów. Ponieważ brak aktualnej kopii bazy spowoduje konieczność ponownego zdefiniowania obiektów tych typów, kopię bazy msdb należy wykonywać zawsze, gdy:

- ♣ Utworzono, usunięto lub zmodyfikowano definicję alarmu.
- ♣ Utworzono, usunięto lub zmodyfikowano definicję operatora.
- ♣ Utworzono, usunięto lub zmodyfikowano definicję zadania.

Dodatkowo w celu zabezpieczenia danych historycznych, należy co pewien okres czasu (np. co miesiąc) wykonać kopię bazy msdb, nawet jeżeli nie zmodyfikowano obiektu żadnego z powyższych typów.

Baza model

Baza model jest wzorcem (szablonem) dla wszystkich nowo utworzonych baz użytkownika. Ponieważ system nie modyfikuje automatycznie zawartości tej bazy, jej kopie należy wykonać po każdym zmodyfikowaniu jej zawartości przez administratora.

Bazy użytkownika

Niezależnie od przyjętej strategii wykonywania kopii zapasowych, należy wykonać kopię zapasową bazy danych zawsze, gdy:

- ♣ Utworzono lub zmodyfikowano strukturę bazy danych. Posiadanie pełnej kopii bazy danych jest szczególnie ważne w przypadku wdrożenia strategii zakładającej wykonywanie kopii dziennika transakcyjnego — w tym

wypadku, aby możliwe było odtworzenie kopii dziennika, trzeba dysponować pełną kopią zawierającą aktualny schemat bazy danych.

- ♣ Utworzono indeks — chociaż indeks może zostać odtworzony na podstawie wpisów z dziennika transakcyjnego, to operacja taka wymaga ponownego utworzenia indeksu i może być operacją bardzo kosztowną, podczas gdy odtworzenie pełnej kopii bazy danych będzie wiązać się z prostym skopiowaniem istniejącego w niej indeksu.
- ♣ Usunięto nieaktywną część dziennika transakcyjnego (wykonano instrukcje BACKUP LOG WITH TRUNCATE_ONLY lub BACKUP LOG WITH NO LOG) — po takiej operacji dziennik nie zawiera już informacji o zmianach odwzorowanych w plikach bazodanowych, a więc ich utrata spowoduje niemożność odtworzenia danych na podstawie samego dziennika.
- ♣ Wykonano operacje nielogowalne, takie jak masowe ładowanie danych czy tworzenie kopii istniejących tabeli za pomocą instrukcji SELECT … INTO — jedynie kompletny model odtwarzania umożliwia odzyskanie danych zapisanych w wyniku tych operacji, ale nawet w tym przypadku ich odtworzenie na podstawie wpisów dziennika będzie czynnością bardzo kosztowną.

Urządzenia kopii zapasowych

SQL Server przechowuje kopie zapasowe w trwałych lub tymczasowych urządzeniach kopii zapasowych. Wynika z tego, że zanim będzie możliwe wykonanie kopii, należy utworzyć takie urządzenie.

Trwałe urządzenia kopii zapasowych

Trwałe urządzenie kopii zapasowych musimy utworzyć, o ile planujemy automatyzację cyklicznego wykonywania kopii zapasowych. Informacje o lokalizacji (nazwa fizyczna) oraz nazwie (nazwa logiczna) urządzenia przechowywane są w tabeli systemowej master.dbo.sysdevices.

SQL Server 2000 pozwala na utworzenie maksymalnie 64 trwałych urządzeń kopii zapasowych.

Procedura sp_addumpdevice

Użytkownicy należący do ról serwera sysadmin lub diskadmin mogą dodawać nowe urządzenia kopii zapasowych.
Składnia:

```
sp_addumpdevice [@devtype =] 'typ',
[@logicalname =] 'nazwa_logiczna',
[@physicalname =] 'nazwa_fizyczna'
[, {[@cntrltype =] typ_kontrolera
| [@devstatus =] 'status'
}
]
```

gdzie:
typ określa typ urządzenia i może przyjmować jedną z trzech wartości:

- ♣ DISK — plik na dysku twardym,
- ♣ TAPE — lokalny napęd taśmowy,
- ♣ PIPE — urządzenie komunikujące się poprzez potoki nazwane.

typ_kontrolera jest parametrem ignorowanym przez SQL Server,
status określa, czy możliwe jest odczytywanie etykiety urządzenia (domyślna wartość NOSKIP), czy też ewentualna etykieta będzie ignorowana (wartość SKIP).
Przykład:

```
sp_addumpdevice 'disk', 'Northwind_BD',
'D:\Program Files\Microsoft SQL Server\Mssql\Backup\Northwind.BAK'
GO
```

(1 row(s) affected)

'Disk' device added.

Procedura sp_helpdevice

Procedura zwraca informacje o istniejących urządzeniach kopii zapasowych.
Składnia:

sp_helpdevice [[@devname=] 'nazwa']

Przykład:

sp_helpdevice 'Northwind_BD'

GO

device_name physical_name description

status cntrltype size

---------- -- --------------------

Northwind_BD D:\Program Files\Microsoft SQL Server\Mssql\Ba disk, backup device

ª16 2 0

(1 row(s) affected)

Procedura sp_dropdevice

Wykonanie procedury spowoduje usunięcie trwałego urządzenia kopii zapasowych.
Składnia:

sp_dropdevice [@logicalname =] 'nazwa'

[, [@delfile =] 'delfile']

gdzie:
delfile określa, czy z dysku zostanie usunięty plik (wartość delfile).
Przykład:

EXEC sp_dropdevice 'Northwind_BD', 'delfile'

GO

Device dropped.

Tymczasowe urządzenia kopii zapasowych

SQL Server umożliwia również zapisanie kopii zapasowych w urządzeniach tymczasowych. W tym przypadku, wykonując instrukcję BACKUP, należy określić typ i fizyczną nazwę urządzenia (w przypadku pliku będzie to jego pełna nazwa, tj. nazwa uzupełniona o ścieżkę dostępu).

Rozwiązanie to powinno być stosowane podczas wykonywania kopii zapasowych bazy master (skoro definicje urządzeń zapisane są właśnie w tej bazie, jej utrata spowoduje również utratę informacji o utworzonych trwałych urządzeniach kopii zapasowych) oraz wtedy, gdy nie planujemy ponownego wykorzystania pliku kopii zapasowych (np. podczas przenoszenia bazy na inny serwer).

Wykonywanie kopii zapasowych

Kopie zapasowe bazy danych mogą utworzyć użytkownicy należący do roli serwera sysadmin, właściciel bazy danych oraz użytkownicy należący do roli bazy danych db_backupoperator. SQL Server umożliwia wykonanie kopii zapasowej całej bazy danych, wybranych plików lub grup plików bazy danych oraz kopii dziennika transakcyjnego. Możliwe jest również równoczesne zapisywanie kopii zapasowej do wielu urządzeń kopii zapasowych — w tym przypadku możliwe jest skrócenie czasu potrzebnego do wykonania kopii zapasowej. Kopię zapasową każdego typu można utworzyć, wykonując instrukcję języka Transact-SQL BACKUP:

Instrukcja BACKUP DATABASE

Kompletna składnia instrukcji BACKUP DATABSE wygląda następująco:

```
BACKUP DATABASE {baza_danych | @zmienna_określająca_baze_danych}
<plik_lub_grupa_plików> [,…n]
TO <urządzenia_kopii_zapasowej> [,…n]
[WITH
[BLOCKSIZE = {blok | @zmienna_określająca_wielkość_bloku}]
[[,] DESCRIPTION = {'opis' | @zmienna_określająca_opis}]
[[,] DIFFERENTIAL]
[[,] EXPIREDATE = {data | @zmienna_określająca_datę}
| RETAINDAYS = {dni | @zmienna_określająca_dni}]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] FORMAT | NOFORMAT]
[[,] {INIT | NOINIT}]
[[,] MEDIADESCRIPTION = {'opis_nośnika' | @zmienna_określająca_opis_nośnika}]
[[,] MEDIANAME = {nośnik | | @zmienna_określająca_nośnik}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] NAME = {zestaw | @zmienna_określająca_zestaw}]
[[,] {NOSKIP | SKIP}]
[[,] {NOREWIND | REWIND}]
[[,] {NOUNLOAD | UNLOAD}]
[[,] RESTART]
[[,] STATS [= postęp]]
]
```

gdzie:

blok określa wielkość pojedynczego bloku danych zapisanego w urządzeniu kopii zapasowych (w większości wypadków wybrana przez SQL Server wielkość bloku jest optymalna dla danego typu urządzenia),
opis jest dowolnym, nieprzekraczającym 255 znaków, opisem kopii zapasowej,
DIFFERENTIAL określa, że zostanie wykonana kopia przyrostowa i może zostać użyty wyłącznie w przypadku tworzenia kopii całej bazy danych, data określa datę, po której plik kopii zapasowej będzie mógł zostać nadpisany,
dni określa liczbę dni, w których plik kopii zapasowej nie będzie mógł zostać nadpisany,
hasło określa hasło, które będzie musiało zostać podane przed przywróceniem danych zapisanych w pliku kopii zapasowej,
FORMAT określa, że urządzenie kopii zapasowej ma zostać ponownie sformatowane, co wiąże się z utratą wszystkich przechowywanych w nim plików kopii zapasowych oraz z usunięciem nagłówka i opisu nośnika,
INIT określa, że istniejące w urządzeniu pliki kopii zapasowych zostaną nadpisane, o ile czas ważności wszystkich plików został przekroczony, nowa nazwa nośnika odpowiada istniejącej nazwie, a nadpisywane pliki nie należą do zestawu kopii zapasowej zapisanego na kilku urządzeniach,
SKIP określa, że nie zostanie odczytany (zawierający m.in. informacje o dacie wygaśnięcia) nagłówek pliku kopii zapasowej,
REWIND określa, że SQL Server po zapisaniu danych przewinie taśmę,

UNLOAD określa, że SQL Server po zapisaniu danych przewinie i wysunie taśmę,
RESTART umożliwia wznowienie przerwanego procesu wykonywania kopii zapasowych na wielu urządzeniach taśmowych,
postęp określa, jak często informacja o postępie wykonywania kopii będzie wyświetlana użytkownikowi (domyślną wartością jest 10%).

Metody wykonywania kopii zapasowych różnych typów zostały przedstawione w dalszej części rozdziału.

Instrukcja BACKUP LOG

SQL Server umożliwia odrębne wykonanie kopii dziennika transakcyjnego wybranej bazy danych. Pełna składnia instrukcji BACKUP LOG wygląda następująco:

```
BACKUP LOG {baza_danych | @zmienna_określająca_bazę_danych}
{
TO <urządzenia_kopii_zapasowej> [,…n]
[WITH
[BLOCKSIZE = {blok | @zmienna_określająca_wielkość_bloku}]
[[,] DESCRIPTION = {'opis' | @zmienna_określająca_opis}]
[[,] EXPIREDATE = {data | @zmienna_określająca_datę}
| RETAINDAYS = {dni | @zmienna_określająca_dni}]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] FORMAT | NOFORMAT]
[[,] {INIT | NOINIT}]
[[,] MEDIADESCRIPTION = {{'opis_nośnika' | @zmienna_określająca_opis_nośnika}]
[[,] MEDIANAME = {nośnik | @zmienna_określająca_nośnik}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] NAME = {zestaw | @zmienna_określająca_zestaw}]
[[,] NO_TRUNCATE]
[[,] {NORECOVERY | STANDBY = plik_tymczasowy}]
[[,] {NOREWIND | REWIND}]
[[,] {NOSKIP | SKIP}]
[[,] {NOUNLOAD | UNLOAD}]
[[,] RESTART]
[[,] STATS [= postęp]]
]
}
```

gdzie:
NO_TRUNCATE określa, że pomimo wykonania kopii, nieaktywna część dziennika transakcyjnego nie zostanie usunięta.
NORECOVERY określa, że SQL Server nie wykona procesu odtwarzania spójności (ang. Recovery).
STANDBY określa, że SQL Server nie wykona procesu odtwarzania spójności, ale dzięki utworzeniu pliku_tymczasowego baza będzie dostępna dla użytkowników w trybie tylko do odczytu.

Pełna kopia bazy danych

Podstawowym typem kopii zapasowej jest pełna kopia bazy danych, zawierająca wszystkie informacje zapisane zarówno w plikach bazy danych (strukturę obiektów bazodanowych oraz dane tabel i indeksów), jak i w plikach dziennika transakcyjnego, przy czym zostaną również skopiowane wszystkie dane z aktywnej części dziennika.

Pełna kopia bazy danych jest wymagana do odtworzenia kopii przyrostowej lub kopii dziennika transakcyjnego bazy danych.

Na przykład, aby utworzyć pełną kopię bazy master, należy wykonać poniższą instrukcję:

```
BACKUP DATABASE master
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\master.bak'
GO
Processed 1312 pages for database 'master', file 'master' on file 1.
Processed 1 pages for database 'master', file 'mastlog' on file 1.
BACKUP DATABASE successfully processed 1313 pages in 0.732 seconds (14.685 MB/sec).
```

Wykonanie kolejnego skryptu zapisze pełną kopię bazy Northwind do zestawu dwóch trwałych urządzeń kopii zapasowej:

```
EXEC sp_addumpdevice 'disk', 'Northwind1',
'D:\Program Files\Microsoft SQL Server\Mssql\Backup\Northwind1.BAK'
EXEC sp_addumpdevice 'disk', 'Northwind2',
'E:\Backup\Northwind2.BAK'
GO
BACKUP DATABASE Northwind
TO Northwind1, Northwind2
WITH RETAINDAYS = 7
,MEDIANAME = 'Pełna kopia bazy Northwind zapisana w dwóch trwałych urządzeniach kopii
ªzapasowej'
,STATS = 5
GO
(1 row(s) affected)
'Disk' device added.
(1 row(s) affected)
'Disk' device added.
31 percent backed up.
49 percent backed up.
81 percent backed up.
97 percent backed up.
Processed 384 pages for database 'Northwind', file 'Northwind' on file 1.
100 percent backed up.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP DATABASE successfully processed 385 pages in 0.354 seconds (8.892 MB/sec).
```

Tworząc zestaw kopii zapasowych, należy uwzględnić poniższe wskazówki:

1. Wszystkie jednocześnie wykorzystane urządzenia kopii zapasowych muszą być tego samego typu.
2. Możliwe, choć nie zalecane, jest jednoczesne wykorzystanie trwałych i tymczasowych urządzeń kopii zapasowych.
3. Pliki wchodzące w skład zestawu kopii zapasowych muszą być wykorzystane wspólnie, a utrata jednego z nich powoduje, że pozostałe stają się bezużyteczne.

4. Po zapisaniu w urządzeniu pliku należącego do zestawu kopii zapasowych niemożliwe będzie dopisanie do niego innych plików, chyba że zostanie użyta opcja WITH FORMAT.
5. Tworząc zestawy kopii zapasowych, należy nadawać im nazwy — w ten sposób ułatwiamy sobie rozpoznanie plików kopii zapasowych należących do danego zestawu.

Przyrostowa kopia bazy danych

Tworzenie przyrostowych kopii baz danych pozwala skrócić czas potrzebny na odtworzenie bazy z plików kopii dziennika transakcyjnego oraz skraca czas potrzebny na wykonanie kopii plików bazodanowych.

Do pliku przyrostowej kopii bazy danych zostaną zapisane wszystkie dane, które zostały zmodyfikowane od czasu wykonania ostatniej pełnej kopii bazy danych. Podczas wykonywania kopii tego typu SQL Server porównuje numer LSN transakcji zapisanych w danym zakresie z numerem LSN zapisanym w pełnej kopii bazy danych i jeżeli zakres zawiera choć jedną transakcje późniejszą, to zostanie on dodany do pliku kopii przyrostowej.

Aby zmodyfikować, a następnie utworzyć przyrostową kopię bazy Northwind, należy wykonać poniższy skrypt:

```
USE Northwind
UPDATE dbo.Products
SET UnitPrice = UnitPrice * RAND()
GO
EXEC sp_addumpdevice 'disk', 'Northwind_diff',
'D:\Program Files\Microsoft SQL Server\Mssql\Backup\Northwind_diff.BAK'
GO
BACKUP DATABASE Northwind
TO Northwind_diff
WITH DIFFERENTIAL
GO
(77 row(s) affected)
(1 row(s) affected)
'Disk' device added.
Processed 24 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 25 pages in 0.086 seconds
ª(2.298 MB/sec).
```

Jeżeli ta sama informacja zmieniała się kilkakrotnie od czasu wykonania pełnej kopii zapasowej (np. pensja pracownika została zwiększona, a następnie ponownie obniżona), w pliku kopii przyrostowej zostaną zapisane tylko aktualne dane.

Kopia dziennika transakcyjnego

Jeżeli baza danych nie działa w uproszczonym modelu odtwarzania, konieczne jest systematyczne wykonywanie kopii dziennika transakcyjnego, w innym przypadku albo plik dziennika będzie się stale rozrastał, albo zostanie on zapełniony i dostęp użytkowników do bazy zostanie ograniczony (alternatywnym rozwiązaniem jest systematyczne, programowe usuwanie nieaktywnej części dziennika, ale takie rozwiązanie, przypominające uproszczony model odtwarzania, w praktyce nie jest stosowane).

Plik kopii dziennika transakcyjnego jest bezużyteczny bez aktualnego pliku pełnej kopii tej bazy danych.

SQL Server do pliku kopii dziennika transakcyjnego zapisuje wszystkie wpisy z dziennika danej bazy danych, począwszy od pierwszego wpisu, który nie został zapisany do poprzednio wykonanego pliku kopii dziennika transakcyjnego. Następnie domyślną operacją jest usunięcie nieaktywnej części dziennika.

Poniższy skrypt zmienia domyślny model odtwarzania bazy Northwind, zapełnia jej dziennik transakcyjny, a następnie wykonuje jego kopię — ponieważ plik dziennika transakcyjnego tej bazy może być automatycznie zwiększany, proszę

porównać jego wielkość i procent wykorzystania przed uruchomieniem skryptu, po aktualizacji tabeli Products i po wykonaniu kopii dziennika (z wyniku usunięto niektóre nieistotne informacje):

```
DBCC SQLPERF (LOGSPACE)
USE Northwind
DECLARE @I INT
SET @I = 1
WHILE @I < 50
BEGIN
UPDATE dbo.Orders
SET OrderDate = OrderDate
SET @I = @I +1
END
DBCC SQLPERF (LOGSPACE)
EXEC sp_addumpdevice 'disk', 'Northwind_log',
'D:\Program Files\Microsoft SQL Server\Mssql\Backup\Northwind_log.BAK'
GO
BACKUP LOG Northwind
TO Northwind_log
DBCC SQLPERF (LOGSPACE)
GO
Database Name Log Size (MB) Log Space Used (%) Status
------------ ----------------------- ------------------------ --------------
Northwind 0.9921875 42.322834 0
(6 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system
ªadministrator.
(830 row(s) affected)
.
.
.
(830 row(s) affected)
Database Name Log Size (MB) Log Space Used (%) Status
------------ ----------------------- ------------------------ --------------
Northwind 6.7421875 93.597916 0
(6 row(s) affected)
DBCC execution completed. If DBCC printed error messages, contact your system
ªadministrator.
(1 row(s) affected)
'Disk' device added.
Processed 742 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP LOG successfully processed 742 pages in 0.795 seconds (7.645 MB/sec).
Database Name Log Size (MB) Log Space Used (%) Status
------------ ----------------------- ------------------------ --------------
```

```
Northwind 6.7421875 27.158169 0
(6 row(s) affected)
```

Klauzula NO_TRUNCATE

Użycie klauzuli NO_TRUNCATE spowoduje nieusuwanie nieaktywnej części dziennika transakcyjnego po wykonaniu jego kopii. W ten sposób możliwe jest utworzenie kopii dziennika transakcyjnego uszkodzonej lub utraconej bazy danych — wystarczy, aby plik dziennika transakcyjnego (z reguły przechowywany na osobnym dysku twardym) pozostał nieuszkodzony.

O ile jest to tylko możliwe, przed odtworzeniem bazy danych z kopii zapasowej należy wykonać kopię dziennika transakcyjnego, używając klauzuli NO_TRUNCATE — w ten sposób możliwe będzie przywrócenie bazy do stanu z dowolnego momentu przed awarią.

Przykład:

```
ALTER DATABASE pubs
SET RECOVERY BULK_LOGGED
BACKUP LOG pubs
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\pubs_log.bak'
WITH NO_TRUNCATE
GO
Processed 1 pages for database 'pubs', file 'pubs_log' on file 1.
BACKUP LOG successfully processed 1 pages in 0.026 seconds (0.236 MB/sec).
```

Klauzule TRUNCATE_ONLY oraz NO_LOG

Obie klauzule są swoimi funkcjonalnymi odpowiednikami — użycie dowolnej z nich spowoduje usunięcie nieaktywnej części dziennika transakcyjnego bez wykonania jego kopii. Ponieważ cześć informacji zostaje w ten sposób nieodwracalnie utracona, po obcięciu dziennika transakcyjnego należy natychmiast wykonać pełną kopię bazy danych.

W praktyce administratorzy usuwają nieaktywną część dziennika transakcyjnego przed wykonaniem jej pełnej kopii zapasowej — w ten sposób plik wynikowy jest mniejszy, wykonanie kopii zajmuje mniej czasu, a w razie awarii możliwe jest odtworzenie wszystkich przechowywanych w bazie danych.

Przykład:

```
BACKUP LOG pubs
WITH NO_LOG
BACKUP DATABASE pubs
TO DISK = 'E:\Backup\pubs.bak'
GO
Processed 208 pages for database 'pubs', file 'pubs' on file 1.
Processed 1 pages for database 'pubs', file 'pubs_log' on file 1.
BACKUP DATABASE successfully processed 209 pages in 0.164 seconds (10.396 MB/sec).
```

Kopia plików lub grup plików

W przypadku bardzo dużych baz danych (VLDBS) codzienne wykonywanie kopii całej bazy danych jest niepraktyczne lub niemożliwe. W tym wypadku, zamiast poświęcać osiem godzin dziennie na wykonanie kopii zapasowej całej bazy danych, administrator może zdecydować się na utworzenie kopii wybranego pliku lub grupy plików.

Na przykład, aby utworzyć kopię pliku pubs bazy danych pubs, należy wykonać poniższy skrypt:

```
BACKUP DATABASE pubs
FILE = 'pubs' TO DISK = 'D:\Program Files\Microsoft SQL
[a]Server\Mssql\Backup\pubs_file.BAK'
BACKUP LOG pubs
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\pubs_file.BAK'
WITH NOINIT
GO
Processed 208 pages for database 'pubs', file 'pubs' on file 1.
BACKUP DATABASE…FILE=<name> successfully processed 208 pages in 0.131 seconds
[a](13.007 MB/sec).
Processed 1 pages for database 'pubs', file 'pubs_log' on file 2.
BACKUP LOG successfully processed 1 pages in 0.032 seconds (0.112 MB/sec).
```

Tworząc kopię wybranych plików lub grup plików należy pamiętać, że:

- ♣ Po wykonaniu kopii niektórych plików danych należy natychmiast wykonać kopię dzielnika transakcyjnego tej bazy — jej odtworzenie będzie konieczne w celu zsynchronizowania kopii z pozostałymi plikami bazy danych.
- ♣ Możliwe jest utworzenie maksymalnie 16 plików lub grup plików dla bazy danych.
- ♣ Poszczególne pliki lub grupy plików powinny być archiwizowane cyklicznie, według opracowanego harmonogramu.
- ♣ SQL Server nie dopuści do wykonania kopii plików zawierających tabelę bez jednoczesnego wykonania kopii plików zawierających indeksy powiązane z tą tabelą.

Strategie wykonywania kopii zapasowych

Znając dostępne modele odtwarzania i metody wykonywania kopii zapasowych różnych typów, należy zaplanować spełniającą wymogi użytkowników bazy danych strategię wykonywania kopii zapasowych.

Strategia pełnych kopii bazy danych

W przypadku niewielkich lub rzadko modyfikowanych baz danych najlepszą strategią może okazać się systematyczne wykonywanie pełnych kopii bazy danych. W takim przypadku należy uruchomić bazę danych w uproszczonym trybie odzyskiwania albo systematycznie usuwać nieaktywną część dziennika transakcyjnego.

W ramach strategii okresowo (np. codziennie o 21:00) wykonywana jest pełna kopia zapasowa bazy danych.

Zaletami tej strategii są:

- ♣ Uproszczenie i maksymalne skrócenie procesu odtwarzania bazy danych.
- ♣ Zminimalizowanie liczby potrzebnych do przechowywania kopii nośników — w zasadzie wystarczy przechowywać jedynie ostatnią pełną kopię bazy danych.

Wady:

- ♣ Konieczność częstego (co najmniej codziennego) wykonywania pełnej kopii bazy danych.
- ♣ Ryzyko utraty wszystkich danych zmodyfikowanych od czasu wykonania ostatniej pełnej kopii zapasowej.

Proces odtwarzania bazy danych w tym przypadku powinien przebiegać następująco:

1. O ile jest to możliwe, przed przywróceniem kopii bazy danych należy wykonać kopię dziennika transakcyjnego bez jego obcinania (operacja ta będzie niedostępna dla baz działających w uproszczonym trybie odzyskiwania).

2. Odtworzyć uszkodzoną bazę z ostatniego pliku pełnej kopii zapasowej, w ten sposób nadpisując uszkodzoną bazę jej poprawną wersją.
3. Odtworzyć, o ile został wykonany punkt 1., plik kopii dziennika transakcyjnego. W razie potrzeby należy określić czas wykonania ostatniej odtwarzanej transakcji.

Strategia pełnych kopii bazy danych i kopii dziennika transakcyjnego

Strategia zalecana w przypadku niewielkich i często modyfikowanych baz danych. Ponieważ zakłada ona tworzenie kopii dziennika transakcyjnego, baza danych nie może pracować w uproszczonym modelu odzyskiwania.

W ramach strategii okresowo (np. codziennie o 21:00) wykonywana jest pełna kopia zapasowa bazy danych., a ponadto co np. 4 godziny wykonywana jest kopia dziennika transakcyjnego.

Zaletami tej strategii są:

- ♣ Zapewnienie możliwości odtworzenia danych z dowolnego momentu sprzed awarii.
- ♣ Stosunkowo prosty do przeprowadzenia proces odtwarzania bazy danych.
- ♣ Ograniczenie liczby potrzebnych do przechowywania kopii nośników — w zasadzie wystarczy przechowywać jedynie ostatnią pełną kopię bazy danych i wykonane danego dnia kopie dziennika transakcyjnego.

Wady:

- ♣ Konieczność częstego (co najmniej codziennego) wykonania pełnej kopii bazy danych.
- ♣ Konieczność kolejnego odtwarzania kilku dzienników transakcyjnych, co zdecydowanie wydłuża czas potrzebny na przywrócenie sprawności systemu.

Proces odtwarzania bazy danych w tym przypadku powinien przebiegać następująco:

1. O ile jest to możliwe, przed przywróceniem kopii bazy danych należy wykonać kopię dziennika transakcyjnego bez jego obcinania (zakładamy, że awaria nastąpiła o godzinie 15:32, a kopie dziennika transakcyjnego wykonywane były od godziny 10:00 co 4 godziny).
2. Odtworzyć uszkodzoną bazę z ostatniego pliku pełnej kopii zapasowej, nadpisując w ten sposób uszkodzoną bazę jej poprawną wersją.
3. Kolejno odtworzyć kopie dziennika transakcyjnego utworzone o godzinie 10 i 14.
4. Odtworzyć, o ile został wykonany punkt 1., plik kopii dziennika transakcyjnego. W razie potrzeby należy określić czas ostatniej odtwarzanej transakcji.

Strategia przyrostowych kopii bazy danych

Strategia zalecana w przypadku średniej wielkości baz danych. W ramach strategii okresowo (np. co tydzień) wykonywana jest pełna kopia bazy danych, a w międzyczasie (np. co dziennie) tworzone są kopie przyrostowe. Ponieważ każda kopia przyrostowa zawiera wszystkie dane zmodyfikowane od czasu wykonania ostatniej pełnej kopii, każdy kolejny plik kopii przyrostowej będzie większy od poprzedniego, aż do momentu ponownego wykonania pełnej kopii bazy danych.

Możliwe jest uzupełnienie strategii o wykonywanie kopii dziennika transakcyjnego lub, po uruchomieniu bazy w uproszczonym modelu odzyskiwania, ograniczenie się do wykonywania kopii plików danych.

Zaletami tej strategii są:

- ♣ Skrócenie czasu potrzebnego na wykonanie kopii zapasowych.
- ♣ Skrócenie, w porównaniu do poprzedniej strategii, czasu potrzebnego na odtworzenie bazy danych.
- ♣ W przypadku tworzenia kopii dziennika transakcyjnego zapewnienie możliwości odtworzenia danych z dowolnego momentu sprzed awarii.

Wady:

- ♣ Skomplikowany proces odtwarzania bazy danych.
- ♣ Konieczność przechowywania ostatnio utworzonych plików kopii pełnej i przyrostowej kopii zapasowej oraz wszystkich wykonanych od czasu wykonania ostatniej kopii przyrostowej plików kopii dziennika transakcyjnego.

Proces odtwarzania bazy danych w tym przypadku powinien przebiegać następująco:

1. O ile jest to możliwe, przed przywróceniem kopii bazy danych należy wykonać kopię dziennika transakcyjnego bez jego obcinania (zakładamy, że awaria nastąpiła w środę o godzinie 19:12, pełna kopia wykonywana jest co niedzielę, kopie przyrostowe codziennie o 19:00, a kopie dziennika transakcyjnego wykonywane są codziennie od godziny 10:00 co 4 godziny).
2. Odtworzyć uszkodzoną bazę z ostatniego pliku pełnej kopii zapasowej, nadpisując w ten sposób uszkodzoną bazę jej poprawną wersją,
3. Odtworzyć plik przyrostowej kopii zapasowej wykonanej we wtorek wieczorem.
4. Kolejno odtworzyć kopie dziennika transakcyjnego utworzone o godzinie 10, 14 i 18.
5. Odtworzyć, o ile został wykonany punkt 1., plik kopii dziennika transakcyjnego. W razie potrzeby należy określić czas ostatniej odtwarzanej transakcji.

Strategia kopii plików bazy danych

Strategia zalecana w przypadku dużych i bardzo dużych baz danych. Ponieważ odtworzenie bazy danych z kopii zawierającej jedynie niektóre pliki danych wymaga odtworzenia kopii dziennika transakcyjnego, baza danych nie może pracować w uproszczonym modelu odzyskiwania.

Zaletami tej strategii są:

- ♣ Skrócenie czasu potrzebnego na wykonanie kopii zapasowych.
- ♣ Ograniczenie liczby potrzebnych do przechowywania kopii nośników.

Wada:

- ♣ Skomplikowany proces odtwarzania bazy danych.

Proces odtwarzania bazy danych w tym przypadku powinien przebiegać następująco:

1. O ile jest to możliwe, przed przywróceniem kopii bazy danych należy wykonać kopię dziennika transakcyjnego bez jego obcinania (zakładamy, że awaria nośnika, na którym znajdowały się pliki należące do grupy G3, nastąpiła w piątek o godzinie 13:13). Strategia zakłada, że:
    - ♣ Baza danych jest podzielona na trzy grupy plików (G1, G2 i G3).
    - ♣ Pełna kopia bazy danych wykonywana jest co niedzielę.
    - ♣ Kopia grupy G1 tworzona jest w poniedziałek i czwartek o godzinie 23:00.
    - ♣ Kopia grupy G2 tworzona jest we wtorek i piątek o godzinie 23:00.
    - ♣ Kopia grupy G3 tworzona jest w środę i sobotę o godzinie 23:00.
    - ♣ Codziennie o godzinie 12:00 i 22:00 tworzona jest kopia dziennika transakcyjnego.
2. Odtworzyć plik kopii zapasowej grupy G3 wykonanej w środę.
3. Kolejno odtworzyć obie kopie dzienników transakcyjnych z czwartku, a następnie kopię dziennika transakcyjnego wykonaną w piątek o godzinie 12:00.
4. Odtworzyć, o ile został wykonany punkt 1., plik kopii dziennika transakcyjnego. W razie potrzeby należy określić czas ostatniej odtwarzanej transakcji.

Kurs Transact-SQL, część 19 (Odtwarzanie kopii zapasowych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Awaria i spowodowana nią przerwa w pracy serwera bazodanowego wcześniej czy później przytrafi się każdemu administratorowi. Jedynie znajomość procesu odtwarzania bazi danych przez SQL Server oraz przećwiczony plan ich odtwarzania pozwolą mu wtedy zachować spokój i szybko przywrócić pełną funkcjonalność systemu bazodanowego.

Proces odtwarzania spójności bazy danych

Proces odtwarzania spójności bazy danych (ang. Recovery) jest wewnętrznym mechanizmem SQL Servera, który zapewnia spójność wszystkich przechowywanych w nim danych. Proces ten jest uruchamiany automatycznie podczas uruchamiania SQL Servera, ale może być wywołany przez użytkownika podczas odtwarzania danych z kopii zapasowej.

W ramach procesu odtwarzania spójności SQL Server:

1. Analizuje zawartość dziennika transakcyjnego, począwszy od transakcji oznaczonej numerem LSN, aż do końca, czyli do transakcji, w trakcie której zatrzymano bazę danych lub w trakcie której wystąpiła awaria.
2. Wyniki wszystkich zatwierdzonych transakcji zostają zapisane w plikach danych (następuje „przewinięcie w przód" dziennika transakcyjnego).
3. Wszystkie niezatwierdzone (wycofane lub przerwane) transakcje zostają usunięte z dziennika transakcyjnego („przewinięcie w tył" dziennika transakcyjnego), co powoduje, że dziennik transakcyjny nie zawiera już ciągłej historii wszystkich przeprowadzonych operacji.

W rezultacie baza danych zawiera spójne dane, ale utracona została ciągłość dziennika transakcyjnego i niemożliwe będzie odtworzenie jego kolejnych kopii zapasowych. Natomiast w przypadku odtworzenia kopii bez wykonania procesu odtwarzania, ciągłość dziennika zostaje zachowana, ale baza znajduje się w niespójnym stanie i jest niedostępna dla użytkowników, chyba że zamiast usuwać niezatwierdzone transakcje SQL Server zapisał je do pliku tymczasowego — w takim przypadku baza będzie dostępna dla użytkowników w trybie tylko do odczytu.

Przygotowanie do odtworzenia kopii zapasowej

SQL Server umożliwia sprawdzenie poprawności kopii zapasowej bez jej przywracania. Ponadto zabezpiecza nas przed utratą danych w wyniku przywrócenia niewłaściwego lub niekompletnego zestawu kopii zapasowej. W przeciwieństwie do wykonywania kopii zapasowej, jej przywracanie wymaga odłączenia od bazy wszystkich użytkowników i uruchomienia jej w trybie ograniczonego dostępu.

Weryfikacja kopii zapasowej

Zweryfikować poprawność kopii zapasowej możemy, wykonując instrukcję języka Transact- SQL RESTORE.

Instrukcja RESTORE HEADERONLY

Wykonując instrukcję RESTORE HEADERONLY poznamy m.in.:

- Nazwę i opis zarówno urządzenia, jak i przechowywanych w nim plików kopii zapasowych.
- Typ urządzenia i przechowywanych w nim kopii zapasowych.
- Czas wykonania poszczególnych kopii i ich wielkość.
- Numer każdego pliku kopii przechowywanej w urządzeniu.

Możliwe jest odczytanie informacji zarówno o trwałych, jak i tymczasowych urządzeniach kopii zapasowych.

Składnia:

```
RESTORE HEADERONLY
FROM <urządzenie_kopii_zapasowej>
[WITH {NOUNLOAD | UNLOAD}
[[,] FILE = numer_pliku]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
]
```

Przykłady:

```
RESTORE HEADERONLY
FROM Northwind_log
GO

RESTORE HEADERONLY
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\northwind_file.BAK'
GO
```

Instrukcja RESTORE FILELISTONLY

Instrukcja zawraca informacje o oryginalnych plikach bazy danych, której kopię wykonano. Wykonując instrukcję RESTORE FILELISTONLY, poznamy m.in.:

- ♣ Fizyczną i logiczną nazwę plików danych i plików dziennika transakcyjnego.
- ♣ Bieżącą i maksymalną wielkość plików bazy danych.

Składnia:

```
RESTORE FILELISTONLY
FROM <urządzenie_kopii_zapasowej>
[WITH
[FILE = numer_pliku]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] {NOUNLOAD | UNLOAD}]
]
```

Przykład:

```
RESTORE FILELISTONLY
FROM Northwind1
```

```
GO
```

Instrukcja RESTORE LABELONLY

Instrukcja zwraca informacje o urządzeniu kopii zapasowej.
Składnia:

```
RESTORE LABELONLY
FROM <urządzenie_kopii_zapasowej>
[WITH {NOUNLOAD | UNLOAD}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
```

Przykład:

```
RESTORE LABELONLY
FROM Northwind2
GO
```

Instrukcja RESTORE VERIFYONLY

Poprawność pliku kopii zapasowej bez jej odtwarzania możemy sprawdzić, wykonując instrukcję RESTORE VERIFYONLY. Instrukcja nie sprawdza logicznej poprawności kopii, a jedynie jej kompletność i poprawność odczytu pliku.
Składnia:

```
RESTORE VERIFYONLY
FROM <urządzenie_kopii_zapasowej> [,…n]
[WITH
[FILE = numer_pliku]
[[,] {NOUNLOAD | UNLOAD}]
[[,] LOADHISTORY]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] {NOREWIND | REWIND}]
]
```

Przykłady:

```
RESTORE VERIFYONLY
FROM Northwind2
GO
Server: Msg 3132, Level 16, State 1, Line 1
The media set for database '' has 2 family members but only 1 are provided. All
```

```
[a]members must be provided.
Server: Msg 3013, Level 16, State 1, Line 1
VERIFY DATABASE terminating abnormally.

RESTORE VERIFYONLY
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\master.bak'
GO
The backup set is valid.
```

Ograniczenie dostępu do bazy danych

Zanim będzie możliwe odtworzenie kopii zapasowej, konieczne jest ograniczenie dostępu użytkowników do odtwarzanej bazy danych. Dostęp użytkowników można ograniczyć np. poprzez ustawienie opcji dbo use only (dotyczy wszystkich wersji SQL Servera) albo poprzez klauzulę WITH RESTRICTED_USER (dotyczy wersji 2000):

```
USE master
EXEC sp_dboption 'Northwind', 'dbo use only', 'on'
GO
sp_dboption command succeeded.
```

Zagadnieniu monitorowania bieżącej aktywności użytkowników, w tym kończeniu sesji użytkowników, poświęcony jest rozdział 20.

Odtwarzanie kopii zapasowych

Po sprawdzeniu poprawności kopii i ograniczeniu dostępu do bazy danych, odtworzenie kopii zapasowej sprowadza się do wydania właściwej instrukcji RESTORE. SQL Server przed zstąpieniem istniejących danych informacjami odtwarzanymi z kopii sprawdza, czy podana nazwa istniejącej bazy danych zgada się z nazwą zapisaną w pliku kopii zapasowej, czy fizyczna i logiczna nazwa plików bazodanowych odpowiada nazwie plików zapisanych w kopii oraz czy dostępne są wszystkie części zestawu kopii zapasowych i jeżeli wszystkie warunki są spełnione instrukcja RESTORE jest wykonywana.

Tak więc żadna z poniższych instrukcji nie zostanie pomyślnie wykonana:

```
USE master
RESTORE DATABASE pubs
FROM Northwind_diff
WITH RESTRICTED_USER
GO
Server: Msg 3154, Level 16, State 1, Line 2
The backup set holds a backup of a database other than the existing 'pubs' database.
Server: Msg 3013, Level 16, State 1, Line 2
RESTORE DATABASE terminating abnormally.
USE master
EXEC sp_dboption 'Northwind', 'dbo use only', 'on'
GO
RESTORE DATABASE Northwind
```

```
FROM Northwind2
GO
sp_dboption command succeeded.
Server: Msg 3132, Level 16, State 1, Line 1
The media set for database 'Northwind' has 2 family members but only 1 are provided.
[a]All members must be provided.
Server: Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE terminating abnormally.
```

Instrukcja RESTORE DATABASE

Bazę danych z pliku pełnej lub przyrostowej kopii bazy danych możemy odtworzyć, wykonując instrukcję RESTORE DATABSE. Kompletna składnia instrukcji wygląda następująco:

```
RESTORE DATABASE {baza_danych | @zmienna_określająca_bazę_danych}
[<plik_lub_grupa_plików> [,…n]]
[FROM <urządzenie_kopii_zapasowej> [,…n]]
[WITH
[RESTRICTED_USER]
[[,] FILE = numer_pliku]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] MEDIANAME = {nośnik | @zmienna_określająca_nośnik}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] MOVE 'plik_bazy_danych' TO 'fizyczna_nazwa_pliku']
[,…n]
[[,] KEEP_REPLICATION]
[[,] {NORECOVERY | RECOVERY | STANDBY = plik_tymczasowy}]
[[,] NORECOVERY]
[[,] {NOREWIND | REWIND}]
[[,] {NOUNLOAD | UNLOAD}]
[[,] REPLACE]
[[,] RESTART]
[[,] STATS [= postęp]]
]
```

gdzie:
MOVE umożliwia odtworzenie bazy danych do innej lokalizacji niż oryginalna,
KEEP_REPLICATION spowoduje zachowanie istniejących ustawień replikacji odtwarzanej bazy danych,
NORECOVERY spowoduje niewykonywanie procesu odtwarzania spójności bazy danych,
RECOVERY (klauzula domyślna) spowoduje wykonanie procesu odtwarzania, w wyniku którego baza będzie gotowa do użycia, ale niemożliwe będzie odtworzenie kolejnych kopii zapasowych,
STANDBY spowoduje, że wycofywane w wyniku procesu odtwarzania transakcje nie zostaną usunięte, ale zapisane do pliku tymczasowego. W rezultacie możliwe będą: odtworzenie kolejnych kopii zapasowych i odczyt danych przez użytkowników,
REPLACE wymusi odtworzenie bazy z kopii zapasowej, nawet jeżeli istnieje baza danych o innej nazwie czy innym zestawie plików,
RESTART wznawia przerwany proces odtwarzania bazy danych z taśmowych urządzeń kopii zapasowych lub z zestawu plików kopii zapasowej.

Instrukcja RESTORE LOG

Kompletna instrukcja, dzięki której możemy odtworzyć kopię dziennika transakcyjnego bazy danych, wygląda następująco:

```
RESTORE LOG {baza_danych | @zmienna_określająca_bazę_danych}
[FROM <urządzenie_kopii_zapasowej> [,…n]]
[WITH
[RESTRICTED_USER]
[[,] FILE = numer_pliku]
[[,] PASSWORD = {hasło | @zmienna_określająca_hasło}]
[[,] MOVE 'plik_bazy_danych' TO 'fizyczna_nazwa_pliku']
[,…n]
[[,] MEDIANAME = {nośnik | @zmienna_określająca_nośnik}]
[[,] MEDIAPASSWORD = {hasło_nośnika | @zmienna_określająca_hasło_nośnika}]
[[,] KEEP_REPLICATION]
[[,] {NORECOVERY | RECOVERY | STANDBY = plik_tymczasowy}]
[[,] {NOREWIND | REWIND}]
[[,] {NOUNLOAD | UNLOAD}]
[[,] RESTART]
[[,] STATS [= postęp]]
[[,] STOPAT = {data_i_godzina | @zmienna_określająca_datę_i_godzinę}
| [,] STOPATMARK = 'transakcja' [AFTER data_i_godzina]
| [,] STOPBEFOREMARK = 'transakcja' [AFTER data_i_godzina]
]
]
```

gdzie:
STOPAT spowoduje przerwanie odtwarzania bazy danych po odtworzeniu transakcji z określonego dnia i godziny,
STOPATMARK spowoduje przerwanie odtwarzania bazy danych po odtworzeniu określonej transakcji. Jeżeli zostanie użyte słowo kluczowe AFTER, odtwarzanie zostanie przerwane po odtworzeniu pierwszej transakcji o podanej nazwie występującej po dacie i czasie,
STOPBEFOREMARK spowoduje przerwanie odtwarzania bazy danych przed odtworzeniem określonej transakcji. Jeżeli zostanie użyte słowo kluczowe AFTER, odtwarzanie zostanie przerwane przed odtworzeniem pierwszej transakcji o podanej nazwie występującej po dacie i czasie.

Inicjowanie procesu odtwarzania spójność bazy danych

Domyślnie podczas odtwarzania każdego pliku kopii zapasowej SQL Server odtworzy spójność bazy danych. Ponieważ nie zawsze jest to pożądane, należy zawsze jawnie określać, poprzez klauzulę RECOVERY lub NORECOVERY, czy odtworzenie danego pliku kopii zapasowej ma zakończyć się zainicjowaniem procesu odtwarzania spójności bazy danych.

Odtwarzając kopie zapasowe, należy użyć klauzuli NORECOVERY podczas odtwarzania wszystkich, z wyjątkiem ostatniego, plików kopii zapasowych tej bazy. Innymi słowy, jeżeli planujemy odtworzyć kolejne dane z posiadanego pliku kopii przyrostowej lub dziennika transakcyjnego, nie możemy inicjalizować procesu odtwarzania spójności.

Odtwarzanie pełnych kopii baz danych

Bazę danych z reguły odtwarza się z pełnej kopii zapasowej, jeżeli nośnik przechowujący pliki bazy danych został utracony lub uszkodzony, baza danych została usunięta z serwera lub struktura bazy danych została zniszczona (podczas przenoszenia bazy danych na inny serwer albo do innej lokalizacji).

W większości wypadków przed odtworzeniem bazy danych z pełnej kopii należy spróbować wykonać kopię dziennika transakcyjnego i w ten sposób odzyskać dane zmodyfikowane po wykonaniu kopii. Jeżeli tylko uda się wykonać kopię dziennika, pełną kopię bazy danych należy odtwarzać bez inicjowania procesu odzyskiwania spójności.

Poniższy skrypt wykonuje kopię dziennika transakcyjnego bazy Northwind, a następnie przywraca tę bazę z pliku pełnej kopii zapasowej wykonanego w poprzednim rozdziale i z nowo utworzonego pliku kopii dziennika transakcyjnego. W rezultacie, jeżeli tylko od czasu wykonania pełnej kopii zapasowej nie został obcięty (np. poprzez wykonanie kopii) dziennik transakcyjny, odtworzona baza będzie zawierała wszystkie dane znajdujące się w utraconej bazie danych.

```
USE master
-- symulujemy utworzenie poprzedniego dnia pełnej kopii bazy danych
BACKUP DATABASE pubs
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\p.BAK'
-- próba utworzenia kopii dziennika transakcyjnego uszkodzonej bazy
BACKUP LOG pubs
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\p.BAK'
WITH NO_TRUNCATE
GO
EXEC sp_dboption 'pubs', 'dbo use only', 'on'
GO
RESTORE DATABASE pubs
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\p.BAK'
WITH FILE= 1
,NORECOVERY
RESTORE LOG pubs
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\p.BAK'
WITH FILE= 2
,RECOVERY
GO
Processed 208 pages for database 'pubs', file 'pubs' on file 1.
Processed 1 pages for database 'pubs', file 'pubs_log' on file 1.
BACKUP DATABASE successfully processed 209 pages in 0.206 seconds (8.276 MB/sec).
Processed 1 pages for database 'pubs', file 'pubs_log' on file 2.
BACKUP LOG successfully processed 1 pages in 0.029 seconds (0.158 MB/sec).
sp_dboption command succeeded.
Processed 208 pages for database 'pubs', file 'pubs' on file 1.
Processed 1 pages for database 'pubs', file 'pubs_log' on file 1.
RESTORE DATABASE successfully processed 209 pages in 0.191 seconds (8.926 MB/sec).
Processed 1 pages for database 'pubs', file 'pubs_log' on file 2.
RESTORE LOG successfully processed 1 pages in 0.042 seconds (0.024 MB/sec).
```

Odtwarzanie przyrostowych kopii baz danych

Podczas odtwarzania przyrostowej kopii bazy danych, SQL Server zastępuje jedynie te zakresy danych, które przechowują informacje zmodyfikowane po wykonaniu pełnej kopii zapasowej. W rezultacie odtworzenie pliku kopii zajmuje mniej czasu niż odtworzenie pełnej kopii bazy danych.

Składnia instrukcji RESTORE służącej do odtworzenia kopii przyrostowej jest identyczne ze składnią instrukcji odtwarzającej pełną kopę bazy danych — należy jedynie wskazać plik zawierający kopię przyrostową. W przypadku utraty lub uszkodzenia struktury bazy danych przed przywróceniem kopii przyrostowej należy odtworzyć pełną kopię bazy danych.

W poniższym przykładzie ponownie skorzystamy z utworzonych w poprzednim rozdziale plików kopii bazy Northwind:

```
USE master
-- odtwarzamy pełną kopię zapisaną w dwóch urządzeniach
RESTORE DATABASE Northwind
FROM Northwind1, Northwind2
WITH NORECOVERY
, RESTRICTED_USER
-- odtwarzamy kopię przyrostową
RESTORE DATABASE Northwind
FROM Northwind_diff
WITH RECOVERY
, RESTRICTED_USER
GO
Processed 384 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 385 pages in 0.355 seconds (8.866 MB/sec).
Processed 24 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 25 pages in 0.142 seconds (1.391 MB/sec).
```

Odtwarzanie kopii dziennika transakcyjnego

Odtwarzanie kopii dziennika transakcyjnego, w przeciwieństwie do odtwarzania kopii plików danych, nie polega na zastępowaniu zakresów zawierających oryginalne dane zakresami zapisanymi w plikach kopii. W tym wypadku wszystkie transakcje zapisane w kopii dziennika zostają ponownie wykonane, a ich wyniki zapisane w plikach danych. Tak więc odtworzenie kopii dziennika może być wielokrotnie dłuższe niż odtworzenie kopii danych.

W poniższym przykładzie „złośliwie” zmodyfikujemy zawartość tabeli Categories, a następnie odtworzymy, korzystając m.in. z utworzonego w rozdziale 18. pliku kopii dziennika, bazę Northwind do stanu sprzed modyfikacji:

```
-- użytkownik modyfikuje tabelę o określonej godzinie
USE Northwind
UPDATE dbo.Categories
SET Description = 'I kto się śmieje ostatni?'
PRINT GETDATE ()
SELECT Description
FROM dbo.Categories
GO
-- a na to administrator
USE master
```

```
BACKUP LOG Northwind
TO DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\nw_log.BAK'
WITH NO_TRUNCATE
EXEC sp_dboption 'Northwind', 'dbo use only', 'on'
RESTORE DATABASE Northwind
FROM Northwind1, Northwind2
WITH NORECOVERY
RESTORE DATABASE Northwind
FROM Northwind_diff
WITH NORECOVERY
-- wreszcie możemy odtworzyć wszystkie kopie dziennika transakcyjnego
RESTORE LOG Northwind
FROM Northwind_log
WITH NORECOVERY
GO
RESTORE LOG Northwind
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\Backup\nw_log.BAK'
WITH RECOVERY
,STOPAT = 'Dec 12, 2002 9:02 AM'
GO
SELECT Description
FROM Northwind.dbo.Categories
GO
8 row(s) affected)
Dec 12 2002 9:04AM
Description
------------------------------------------------------------
I kto się śmieje ostatni?
...
I kto się śmieje ostatni?
(8 row(s) affected)
Processed 25 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP LOG successfully processed 25 pages in 0.100 seconds (2.048 MB/sec).
sp_dboption command succeeded.
Processed 384 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 385 pages in 0.638 seconds (4.932 MB/sec).
Processed 24 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 25 pages in 0.288 seconds (0.686 MB/sec).
Processed 744 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE LOG successfully processed 744 pages in 0.716 seconds (8.503 MB/sec).
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
```

```
RESTORE LOG successfully processed 1 pages in 0.008 seconds (0.448 MB/sec).
Description
-----------------------------------------------------------------------
Soft drinks, coffees, teas, beers, and ales
Sweet and savory sauces, relishes, spreads, and seasonings
Desserts, candies, and sweet breads
Cheeses
Breads, crackers, pasta, and cereal
Prepared meats
Dried fruit and bean curd
Seaweed and fish
(8 row(s) affected)
```

Niemożliwe jest użycie klauzul STOPAT, STOPATMARK oraz STOPBEFOREMARK bez zainicjowania procesu odtwarzania spójności bazy danych.

Odtwarzanie systemowych baz danych

Jeżeli uszkodzeniu ulegnie baza master, serwer bazodanowy nie zostanie uruchomiony. W takim przypadku przed odtworzeniem logicznych kopii baz systemowych trzeba uruchomić SQL Server.

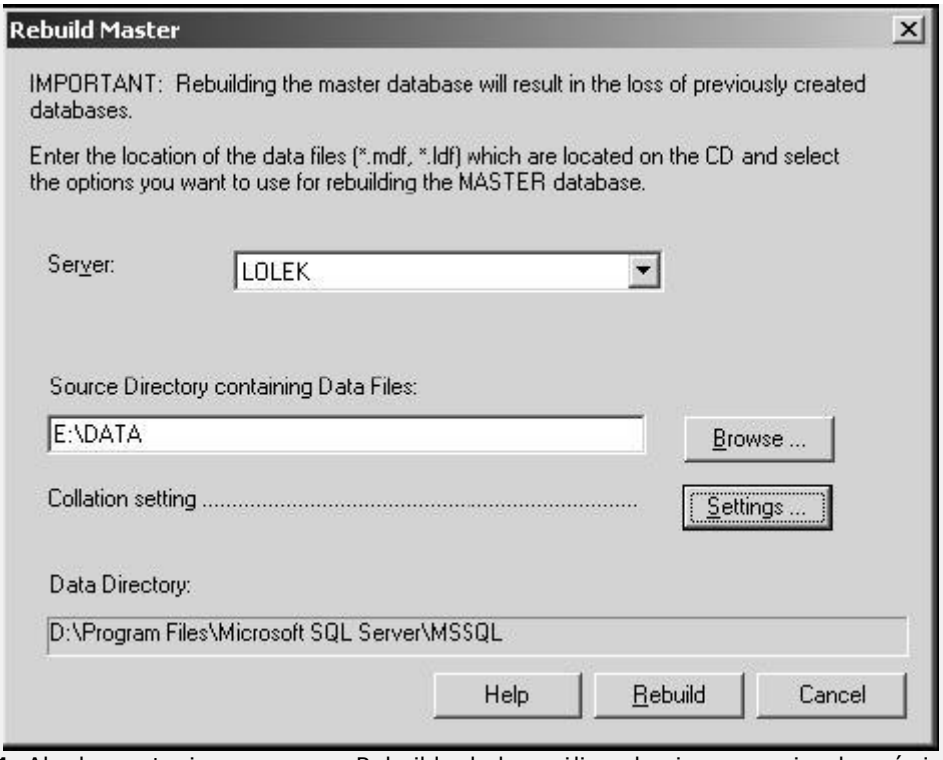
Ponieważ baza master nie jest zbyt często modyfikowana, rozwiązaniem, które skraca czas potrzebny na uruchomienie SQL Servera jest zatrzymanie usługi MSSQLServer i wykonanie aktualnej kopii plików master.mdf i mastlog.ldf. W razie awarii uruchomienie serwera będzie wymagało jedynie skopiowania plików do ich oryginalnej lokalizacji.

Na płycie instalacyjnej serwera znajduje się narzędzie Rebuildm.exe umożliwiające odtworzenie wszystkich (wbrew nazwie) baz systemowych. Chociaż program miał służyć do szybkiego odtwarzania serwera po awarii, w praktyce jego użycie może wymagać wykonania trzech dodatkowych czynności:

1. Po pierwsze, należy skopiować na dysk twardy zawartość folderów Data (zawierającego kopie plików systemowych baz danych) oraz binn (zawierającego wszystkie narzędzia, w tym program Rebuildm).
2. Następnie należy upewnić się, że wszystkie skopiowane pliki bazodanowe nie mają ustawionego atrybutu tylko do odczytu i w razie konieczności skasować ten atrybut.
3. Ostatnią czynnością będzie upewnienie się, czy w folderze binn znajduje się podfolder o nazwie resources. Jeżeli nie — należy zmienić nazwę folderu res na resources.

Teraz możemy już uruchomić program Rebuildm. Wyświetlone zostanie okno dialogowe pokazane na rysunku 19.1. Po wybraniu odtwarzanej instancji należy wskazać lokalizację, do której skopiowano pliki systemowych baz danych, a następnie określić porządek sortowania dla odtwarzanego serwera.

Porządek sortowania musi odpowiadać porządkowi ustalonemu dla utraconej bazy master.



Rysunek 19.1. Aby korzystanie z programu Rebuildm było możliwe, konieczne może okazać się dostosowanie zawartości płyty instalacyjnej do ustawień programu

Po kliknięciu przycisku Rebuild wszystkie systemowe bazy danych zostaną zastąpione wersjami oryginalnymi. Proces tworzenia nowych ustawień trwa nie więcej niż kilkadziesiąt sekund — jeżeli na tym etapie wykonanie programu zastanie zatrzymane (przesuwający się pasek postępu nie świadczy o tym, że program działa), należy zakończyć jego działanie i sprawdzić, czy pliki bazodanowe nie są chronione przed zapisem.

Po zakończeniu odtwarzania baz systemowych należy uruchomić usługę MSSQLServer. Dalsze postępowanie zależy od tego, czy dysponujemy aktualną kopią baz systemowych.

Co zrobić w przypadku braku kopii baz systemowych?

Najczarniejszy scenariusz zakłada brak kopii jakiejkolwiek bazy systemowej — w tym przypadku, po przywróceniu sprawności serwera bazodanowego, musimy kolejno:

1. Odtworzyć zawartość bazy master.
2. Odtworzyć zawartość bazy msdb.
3. Odtworzyć zawartość bazy model.

Odtwarzanie zawartości bazy master

Brak aktualnego schematu katalogu powoduje, że korzystanie z systemu bazodanowego przez użytkowników jest praktycznie niemożliwe. Tak więc jak najszybciej należy:

1. Podłączyć bazy użytkowników — jeżeli nośnik, na którym znajdują się pliki baz danych użytkowników, pozostał nieuszkodzony, wystarczy podłączyć bazy, wykonując procedurę systemową sp_attach_db lub sp_attach_single_file_db:

2.
3. USE master
4. EXEC sp_attach_db @dbname = N'Wydawnictwo',
5. @filename1 = N'E:\kopia\wydawnictwo\wydawnictwo_dane.mdf',
6. @filename2 = N'E:\kopia\wydawnictwo\wydawnictwo_dziennik.ldf'
7. GO
8. Successfully attached database 'Wydawnictwo'.

9.

Jeżeli natomiast nośnik lub same pliki baz danych uległy uszkodzeniu, należy odtworzyć bazy z aktualnej kopii zapasowej. Ponieważ utracone zostały również przechowywane w bazie master definicje trwałych urządzeń kopii zapasowych i informacje o znajdujących się w nich plikach, tym razem będziemy musieli użyć fizycznych nazw plików kopii zapasowych.

10. Umożliwić użytkownikom połączenie z SQL Serverem i korzystanie z baz danych — w tym celu musimy odtworzyć wszystkie loginy użytkowników i ponownie powiązać je z kontami użytkowników w poszczególnych bazach danych.
11. Odtworzyć komunikaty błędów o numerach większych niż 50 000.

Odtwarzanie zawartości bazy msdb

Po odtworzeniu schematu katalogu użytkownicy mogą korzystać z usług serwera bazodanowego. Co prawda brak m.in. alarmów, zadań i operatorów ogranicza funkcjonalność systemu, ale na tym etapie administrator może rozpocząć prawdopodobnie całonocne prace związane z usuwaniem dalszych skutków awarii.

W pierwszej kolejności należy odtworzyć definicje alarmów, następnie operatorów i zadań. Ostatnim etapem będzie odtworzenie powiązań pomiędzy obiektami tych typów.

Odtwarzanie zawartości bazy model

Ponieważ program Rebuildm zastępuje wszystkie systemowe i przykładowe bazy danych ich wersją instalacyjną, po jego uruchomieniu może okazać się konieczne odtworzenie zawartości bazy model. Jednak w tym przypadku przechowywane w niej informacje nie zostały bezpowrotnie utracone — skoro baza model jest szablonem baz użytkowników, wystarczy zapisać definicję wybranych (modelowych) obiektów z dowolnej bazy użytkownika i odtworzyć je w bazie model.

Co zrobić w przypadku posiadania aktualnych kopii baz systemowych?

Jeżeli przezornie wykonaliśmy kopię baz systemowych (w przypadku baz systemowych w praktyce stosuje się wyłącznie strategie pełnych kopii baz danych), aby przywrócić pełną funkcjonalność systemu bazodanowego, wystarczy odtworzyć bazy z kopii. Jako pierwszą należy przywrócić bazę master.

Odtwarzanie kopii bazy master

Przed przywróceniem kopii zapasowej trzeba ograniczyć dostęp użytkowników do danej bazy danych. W przypadku odtwarzania bazy master należy ograniczyć dostęp użytkowników do SQL Servera, uruchamiając usługę MSSQLServer w trybie pojedynczego użytkownika (opcja –m). Następnie odtwarzamy aktualną kopię zapasową bazy master:

```
USE master
RESTORE DATABASE master
FROM DISK = 'D:\Program Files\Microsoft SQL Server\Mssql\backup\master.bak'
GO
The master database has been successfully restored. Shutting down SQL Server.
SQL Server is terminating this process.
```

Ponieważ po pomyślnym odtworzeniu bazy master usługa MSSQLServer zostanie automatycznie zatrzymana, nie należy odtwarzać kopii tej bazy za pomocą graficznej konsoli Enterprise Managera — nagłe zatrzymanie serwera spowoduje, że „zawieszona” konsola będzie wyświetlać mylące informacje o trwającym postępie przywracania kopii zapasowej.

Po ponownym uruchomieniu serwera należy przywrócić kopie baz msdb i model.

Jeżeli opcja –m została wpisana w oknie parametrów startowych konsoli Enterprise Manager, należy usunąć ten parametr i ponownie uruchomić SQL Server. W trybie pojedynczego użytkownika, jeżeli tylko automatycznie uruchamiana jest usługa SQLServerAgent, to usługa ta, jako pierwszy (i jedyny) użytkownik nawiąże połączenie z bazą msdb.

Konfigurowanie serwera zapasowego

Wersje Enterprise i Developer SQL Servera 2000 umożliwiają prawie automatyczne, przeprowadzone za pomocą graficznego kreatora konsoli Enterprise Manager, skonfigurowanie serwera zapasowego. Mechanizm kopiowania dziennika transakcyjnego (ang. Log Shipping) pozwala na zwiększenie niezawodności systemu bazodanowego poprzez utworzenie dokładnej kopii bazy danych użytkownika na oddzielnym serwerze bazodanowym. Dodatkowo możliwe jest udostępnienie w trybie tylko do odczytu kopii bazy danych użytkownikom i w rezultacie zmniejszenie obciążenia podstawowego serwera bazodanowego.

Przed utworzeniem serwera awaryjnego należy:

1. Umożliwić, konfigurując uprawnienia użytkowników, automatyczne wykonywanie kopii dziennika transakcyjnego na serwerze podstawowym i ich odtwarzanie na serwerze zapasowym.
2. Udostępnić na serwerze podstawowym folder, w którym zapisywane będą piki kopii dziennika transakcyjnego.
3. Udostępnić na serwerze zapasowym folder, w którym zapisywane będą pliki tymczasowe umożliwiające użytkownikom odczyt danych i ewentualną zamianę funkcji serwerów.
4. Upewnić się, że na obu (a właściwie na wszystkich trzech, bo kreator skonfiguruje również jeden serwer jako serwer monitorujący przebieg kopiowania dzienników transakcyjnych) serwerach usługa SQLServerAgent jest automatycznie uruchamiana.

Następnie uruchamiamy kreatora planu zarządzania bazami danych (ang. Database Maintenance Plan Wizard) i postępujemy według jego wskazówek (należy zaznaczyć pole wyboru Ship the transaction logs to other SQL Server (Log Shipping)).

Informacje o konfiguracji serwera zapasowego przechowywane są w siedmiu tabelach bazy msdb:

- ♣ W tabeli log_shipping_databases zapisane są nazwy baz danych i identyfikatory zadań kopiowania dzienników transakcyjnych.
- ♣ W tabeli log_shipping_monitor znajdują się nazwy serwerów monitorujących kopiowanie oraz informacje o typie logowania do tych serwerów.
- ♣ Tabela log_shipping_plan_databases przechowuje definicje zadania kopiowania dzienników transakcyjnych dla baz danych.
- ♣ W tabeli log_shipping_plan_history przechowywane są informacje historyczne.
- ♣ Tabela log_shipping_plans przechowuje definicje wszystkich zadań kopiowania dzienników.
- ♣ Tabela log_shipping_primaries przechowuje informacje o serwerach podstawowych.
- ♣ Tabela log_shipping_secondaries przechowuje informacje o serwerach zapasowych.

Jeżeli z jakiegoś powodu skonfigurowanie za pomocą kreatora serwera zapasowego się nie powiodło (bo, np. zmiany nie są replikowane), a podejrzewamy, że podczas konfiguracji popełniliśmy błąd, należy usunąć z obu serwerów utworzone w związku z konfiguracją definicje zadań, a następnie usunąć wszystkie dane z wymienionych tabel (przed usunięciem rekordów z tabeli log_shipping_primaries należy usunąć powiązane rekordy z tabeli log_shipping_secondaries) i ponownie uruchomić kreatora.

Tak jak konfiguracja serwera zapasowego jest możliwa właściwe wyłącznie poprzez konsolę Enterprise Manager, zamiana ról serwera podstawowego i zapasowego możliwa jest wyłącznie z poziomu języka Transact-SQL.

Zamiana ról serwerów bazodanowych będzie możliwa, o ile podczas konfiguracji zostało zaznaczone pole wyboru Allow database to assume primary role.

Aby zastąpić serwer podstawowy serwerem zapasowym, należy:

1. Skopiować wszystkie loginy z serwera podstawowego do serwera zapasowego. Zadanie może być wykonane za pomocą zadania kopiowania loginów (ang. Transfer Logins Task) pakietu DTS.
2. Pozbawić serwer podstawowy jego roli, wykonując procedurę systemową msdb.dbo. sp_change_primary_role.
3. Wypromować serwer zapasowy, wykonując procedurę systemową msdb.dbo. sp_change_secondary_role.
4. Poinformować serwer monitorujący kopiowanie dzienników transakcyjnych o dokonanej zamianie, wykonując procedurę msd.dbo.sp_change_monitor_role.
5. Powiązać loginy użytkowników skopiowane z serwera podstawowego z bazą danych działającą na serwerze zapasowym, wykonując procedurę msdb.dbo.sp_resolve_logins.

Kurs Transact-SQL, część 20 (Monitoring i optymalizacja pracy SQL Server)

- **Data publikacji:** 2006-02-15 10:00 |
- **Odsłon:** 552 |
- **Dodał:** Wojciech Kowasz
- ::

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Nawet najlepiej skonfigurowany SQL Server wymaga ciągłej uwagi administratora. Fakt wystąpienia większości, jeżeli nie wszystkich, błędów może być automatycznie raportowany operatorom. Natomiast obowiązkiem administratora jest:

- Sprawdzanie, czy niepożądane osoby nie uzyskały dostępu do SQL Servera lub danych w nim przechowywanych (ponieważ hakerzy znają standardowe metody wykorzystywane przez SQL Server i system Microsoft Windows do informowania o włamaniach, a ponadto potrafią je blokować, nie można na podstawie braku wpisów w dzienniku zabezpieczeń czy dzienniku SQL Servera wnosić o braku udanych prób przejęcia kontroli nad systemem).
- Optymalizacja wydajności systemu bazodanowego.
- Bieżące rozwiązywanie problemów wynikających z codziennej aktywności użytkowników, np. kaskadowego blokowania transakcji czy zablokowania konta użytkownika.

Poniższy rozdział zawiera opis technik pozwalających na zrealizowanie dwóch ostatnich celów.

Optymalizacja wydajności systemu bazodanowego

Wydajność jest terminem względnym — system, który dla niektórych użytkowników jest bardzo wolny, dla innych może być wzorem wydajności. Tak więc, aby móc porównywać wydajniejsze i mniej wydajne rozwiązania, należy przyjąć pewną konwencję, w ramach której wydajność mierzona jest dwoma wskaźnikami:

1. Liczbą operacji przeprowadzanych na sekundę („przepustowość” systemu).
2. Czasem potrzebnym na zwrócenie klientowi pierwszego bajta wyniku („czas reakcji” systemu).

O ile wartość pierwszego wskaźnika zależy przede wszystkim od dostępnych dla SQL Servera zasobów systemowych, o tyle wartość drugiego zależy głównie od logicznej i fizycznej struktury bazy danych i aplikacji klienckich. Istotny wpływ na oba wskaźniki ma również wykorzystywana przez serwer i klientów metoda pobierania i przetwarzania danych.

Ponadto, aby możliwe było ocenianie wpływu działań podejmowanych przez administratora na wydajność systemu, konieczne jest utworzenie linii bazowej zawierającej zastane wartości najważniejszych liczników wydajności.

> Odrębne linie bazowe powinny zostać utworzone w czasie dużego obciążenia i w czasie względnej bezczynności systemu — porównanie wartości liczników często jest najszybszą metodą wykrycia najsłabszego ogniwa systemu.

Na końcową wydajność systemu mają wpływ jego wszystkie składniki, począwszy od komputera i systemu operacyjnego, poprzez SQL Server, fizyczną i logiczną strukturę bazy danych, skończywszy na aplikacji klienckiej. Co więcej, niewydajność jednego z tych elementów może negatywnie wpłynąć na wyniki monitorowania wydajności innego. Właśnie dlatego strojenie (optymalizacja) baz danych jest umiejętnością w dużej mierze praktyczną, nabywaną poprzez zdobywanie indywidualnych doświadczeń, a nie wyuczoną.

Aby uniknąć wielogodzinnych poszukiwań odpowiedzi na pytania typu „Dlaczego wykonanie raportu X zajmuje tyle czasu, skoro wywołuje on jedynie Y procedur systemowych i zwraca średnio Z danych?”, monitorowanie wydajności należy przeprowadzać, zaczynając od monitorowania wydajności komputera, poprzez śledzenie pracy systemu operacyjnego, analizę wykorzystania zasobów systemowych przez SQL Server i sprawdzenie struktury bazy danych, a

kończąc na monitorowaniu wydajności aplikacji klienckiej i poszczególnych uruchomionych w niej raportów czy zestawień.

Zasoby komputera

Wydajność SQL Servera, jak każdego serwera bazodanowego, zależy przede wszystkim od czterech czynników:

1. Pamięci operacyjnej.
2. Mocy obliczeniowej.
3. Wydajności systemu wejścia-wyjścia.
4. Przepustowości lokalnej sieci komputerowej.

Brak któregokolwiek z tych zasobów ma zgubny wpływ na wykorzystanie pozostałych i wydajność całego systemu bazodanowego. Na przykład niewystarczająca ilość pamięci operacyjnej spowoduje coraz częstsze odwoływanie się do danych zapisanych na dysku, a w rezultacie przeciążenie systemu wejścia-wyjścia, które dla systemów Microsoft Windows natychmiast skutkuje obniżeniem wydajności samego systemu, co powoduje powstanie kolejki procesów (w tym procesów SQL Servera) czekających na dostęp do procesora, a to z kolei niemożność bieżącego odbierania i wysyłania danych do klientów itd..

Ponieważ SQL Server działa jedynie w środowisku systemów operacyjnych Microsoft Windows, a w praktyce produkcyjne bazy danych uruchamiane są wyłącznie w środowisku serwerów tej firmy, wszystkie cztery zasoby możemy monitorować, korzystając z systemowego narzędzia Wydajność zawierającego dwie konsole: Monitor systemu oraz Dzienniki wydajności i alerty. Interesuje nas raczej zapisanie wyników do pliku niż ich bieżąca analiza na ekranie, utworzymy więc dziennik liczników.

Poniższe punkty zawierają krótki opis architektury SQL Servera i sposobów wykorzystania przez niego poszczególnych zasobów komputera.

Pamięć operacyjna

Wirtualna pamięć przydzielona SQL Serwerowi wykorzystywana jest do przechowywania zarówno procesów serwera (ang. Executables), jak i danych. Maksymalna ilość pamięci przydzielonej SQL serwerowi zależy od wersji systemu operacyjnego i na przykład dla systemu Windows 2000 Datacenter Server wynosi 32 GB.

SQL Server dostępną pamięć przydziela poszczególnym procesom realizującym żądania klienckie. Domyślnie ilość fizycznej pamięci operacyjnej dostępnej SQL Serverowi jest konfigurowana dynamicznie, co sprowadza się do tego, że o ile tylko system posiada co najmniej 5 MB dostępnej pamięci fizycznej, to kolejne żądanie zarezerwowania jej dla SQL Servera zostanie zrealizowane. Możliwe jest także przydzielenie minimalnej i maksymalnej ilości dostępnej dla SQL Servera fizycznej pamięci. Opcje te możemy ustawić albo konfigurując właściwości serwera poprzez konsolę Enterprise Manager, albo wykonując procedurę sp_configure. Większość opcji tej procedury dostępna jest dopiero po wyświetleniu opcji zaawansowanych:

```
EXEC sp_configure
GO
name minimum maximum config_value run_value
---------------------------------- ----------- ----------- ------------ -----------
allow updates 0 1 0 0
default language 0 9999 0 0
max text repl size (B) 0 2147483647 65536 65536
nested triggers 0 1 1 1
remote access 0 1 1 1
remote login timeout (s) 0 2147483647 20 20
remote proc trans 0 1 0 0
remote query timeout (s) 0 2147483647 0 0
show advanced options 0 1 0 0
user options 0 16383 0 0
EXEC sp_configure 'show advanced option','1'
RECONFIGURE
```

```
EXEC sp_configure
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE
[a]statement to install.
name minimum maximum config_value run_value
---------------------------------- ----------- ----------- ------------ -----------
affinity mask 0 2147483647 0 0
allow updates 0 1 0 0
awe enabled 0 1 0 0
c2 audit mode 0 1 0 0
cost threshold for parallelism 0 32767 5 5
cursor threshold -1 2147483647 -1 -1
default full-text language 0 2147483647 1033 1033
default language 0 9999 0 0
fill factor (%) 0 100 0 0
index create memory (KB) 704 2147483647 0 0
lightweight pooling 0 1 0 0
locks 5000 2147483647 0 0
max degree of parallelism 0 32 0 0
max server memory (MB) 4 2147483647 2147483647 2147483647
max text repl size (B) 0 2147483647 65536 65536
max worker threads 10 32767 255 255
media retention 0 365 0 0
min memory per query (KB) 512 2147483647 1024 1024
min server memory (MB) 0 2147483647 0 0
nested triggers 0 1 1 1
network packet size (B) 512 65536 4096 4096
open objects 0 2147483647 0 0
priority boost 0 1 0 0
query governor cost limit 0 2147483647 0 0
query wait (s) -1 2147483647 -1 -1
recovery interval (min) 0 32767 0 0
remote access 0 1 1 1
remote login timeout (s) 0 2147483647 20 20
remote proc trans 0 1 0 0
remote query timeout (s) 0 2147483647 0 0
scan for startup procs 0 1 0 0
set working set size 0 1 0 0
show advanced options 0 1 1 1
two digit year cutoff 1753 9999 2049 2049
user connections 0 32767 0 0
```

user options 0 16383 0 0

Aby ustalić ilość pamięci, w której możliwe będzie buforowanie danych (procesom serwera przydzielana jest zawsze taka sama ilość pamięci operacyjnej) należy określić wartości opcji min server memory i max server memory:

```
EXEC sp_configure 'max server memory','192'
EXEC sp_configure 'min server memory', '128'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'max server memory (MB)' changed from 255 to 192. Run the
[a]RECONFIGURE statement to install.
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'min server memory (MB)' changed from 0 to 128. Run the
[a]RECONFIGURE statement to install.
```

Zarezerwowanie minimalnej ilości pamięci operacyjnej na potrzeby SQL Servera poprawi wydajność systemu bazodanowego, jeżeli usługa serwera okresowo jest bezczynna (system usuwa takie usługi z pamięci) lub jeżeli na tym samym komputerze uruchomiony jest inny serwer, który może „zawłaszczyć sobie" całą dostępną pamięć operacyjną.

Możliwe jest również przydzielenie na stałe określonej części pamięci operacyjnej SQL Serwerowi. W tym wypadku należy ustawić wartość obu poprzednich opcji na tą samą wartość, a następnie ustawić wartość opcji set working set size na 1:

```
EXEC sp_configure 'max server memory','128'
EXEC sp_configure 'min server memory', '128'
EXEC sp_configure 'set working set size', '1'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'max server memory (MB)' changed from 192 to 128. Run the
[a]RECONFIGURE statement to install.
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'min server memory (MB)' changed from 128 to 128. Run the
[a]RECONFIGURE statement to install.
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'set working set size' changed from 1 to 1. Run the RECONFIGURE
[a]statement to install.
```

SQL Server minimalizuje liczbę operacji wejścia-wyjścia, starając się buforować wszystkie odczytane z dysku strony danych. Jedynie brak dostępnej pamięci operacyjnej zmusi go do przerwania buforowania kolejnych danych.

Kolejna opcją, której ustawienie może poprawić wykorzystanie pamięci przez SQL Server, jest opcja max worker threads. Określa ona liczbę wątków, które mogą zostać uruchomione na żądanie klientów. Domyślna wartość 255 jest optymalna dla serwerów obsługujących wiele żądań jednocześnie (SQL Server pozwala na nawiązanie jednoczesnych połączeń przez 32 767 klientów). Ponieważ każdy wątek rezerwuje 512 KB pamięci operacyjnej, w przypadku systemów obsługujących mniejszą liczbę klientów, zmniejszenie tej liczby może poprawić wydajność serwera:

```
EXEC sp_configure 'max worker threads','100'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'max worker threads' changed from 255 to 100. Run the RECONFIGURE
[a]statement to install.
```

Liczbę połączeń klienckich poznamy, odczytując zawartość licznika SQL Server: General Statistics \User Connections.

Oprócz danych, pamięć operacyjna przydzielana jest również procesom serwera:

- Podczas uruchamiana usługa SQL Server rezerwuje dla siebie ok. 4 MB pamięci RAM.
- Każde połączenie sieciowe klienta rezerwuje 24 KB pamięci.

Tworząc linię bazową dostępnych zasobów komputera, należy dodać do dziennika następujące liczniki związane z pamięcią:

1. Pamięć\Dostępna pamięć (MB) — wartość tego licznika nie powinna spaść poniżej 5.
2. Pamięć\Strony/s — wartość licznika nie powinna utrzymywać się stale powyżej 0.
3. Pamięć\Odczyty stron/s — wartość licznika nie powinna przekraczać 5.
4. SQL Server: Buffer Manager\Buffer Cache Hit Ratio — wartość licznika nie powinna spadać poniżej 95.

Aby uzyskać niezafałszowane wyniki, monitor wydajności należy uruchomić zdalnie.

Procesor

SQL Server jest programem wielowątkowym — poszczególne zadania mogą być realizowane przez osobne wątki programu. Większość wątków SQL Servera działa w trybie użytkownika systemu Windows NT/2000, ale kilka (wątek przeprowadzający punkt kontrolny, wątek odpowiedzialny za logowanie użytkowników oraz wątek realizujący obsługę bibliotek sieciowych serwera) uruchomianych jest w trybie chronionym.

Poszczególne żądania klientów realizowane są według następującego schematu:

1. Jeżeli wątek jest już uruchomiony, SQL Server przydziela żądaniu realizujący je wątek.
2. Jeżeli wątek nie jest uruchomiony, a liczba aktywnych wątków jest mniejsza od maksymalnej, uruchomiony zostaje nowy watek.
3. Jeżeli wątek nie jest uruchomiony, a usługa ODS wykorzystuje już wszystkie dostępne wątki, żądanie klienta czeka, aż któryś z aktywnych wątków zakończy swoje działanie.

Ponadto niektóre zadania (szczególnie w wersji Enterprise) realizowane są równolegle przez klika wątków. Do takich operacji należą m.in. wykonywanie i odtwarzanie kopii zapasowych, pobieranie danych z kilku tabel w ramach jednego zapytania czy równoległe odczytywanie kilku indeksów.

Taka architektura sprawia, że SQL Server potrafi (i taka jest jego domyślna konfiguracja) wykorzystać wszystkie dostępne procesory komputera, na którym został zainstalowany. Liczbę procesorów, które będą wykorzystywane przez wątki SQL Servera możemy programowo określić, ustawiając wartość opcji affinity mask:

```
sp_configure 'affinity mask', '15'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
```

```
[a]administrator.
Configuration option 'affinity mask' changed from 0 to 15. Run the RECONFIGURE
[a]statement to install.
```

Powyższy skrypt umożliwił SQL Serverowi korzystanie z pierwszych czterech procesorów (binarną reprezentacją liczby 15 jest 00001111). Ograniczenie liczby dostępnych procesorów może wpłynąć na wydajność SQL Servera poprzez ograniczenie liczby przełączeń pomiędzy wątkami uruchomionymi w trybie użytkownika a wątkami działającymi w trybie chronionym, dzięki pozostawieniu procesorów dostępnych systemowi operacyjnemu.

Ustawiając wartość opcji priority boost na 1, możemy uruchomić wątki SQL Servera z priorytetem 13 (domyślnym priorytetem jest 7):

```
sp_configure 'priority boost', '1'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'priority boost' changed from 0 to 1. Run the RECONFIGURE
[a]statement to install.
```

> Zwiększenie priorytetu w komputerach jednoprocesorowych lub dla głównego procesora w komputerach wieloprocesorowych niekorzystnie wpłynie na wydajność i stabilność całego systemu.

Ustawiając wartość opcji lightweight pooling na 1, spowodujemy, że zadania wielowątkowe nie będą realizowane poprze kilka wątków, a poprzez uruchomienie w ramach pojedynczego wątku kilku włókien (ang. Fiber). Ponieważ przełączanie pomiędzy włóknami jest szybsze i wymaga mniej zasobów niż przełączanie pomiędzy wątkami, ustawienie tej opcji jest zalecane dla systemów wieloprocesorowych, w których wykorzystanie procesorów utrzymuje się na wysokim poziomie, a liczba przełączeń pomiędzy trybem użytkownika i chronionym jest duża:

```
sp_configure 'lightweight pooling', '1'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'lightweight pooling' changed from 0 to 1. Run the RECONFIGURE
[a]statement to install.
```

Tworząc linię bazową dostępnych zasobów komputera, należy dodać do dziennika następujące liczniki związane z procesorem:

1. Procesor\Czas procesora (%) — wartość licznika nie powinna stale przekraczać 90.
2. Procesor\Czas uprzywilejowany (%) — wartość licznika nie powinna stale przekraczać 10.
3. System\Długość kolejki procesora — wartość licznika nie powinna przekraczać 5, przy czym średnia wartość nie powinna przekraczać 3.
4. System\Przełączanie kontekstu/s — przy wartości powyżej 7000 należy przełączyć SQL Server w tryb włókien.

Dysk twardy

SQL Server jest programem wielowątkowym — poszczególne zadania mogą być realizowane przez osobne wątki programu. Większość wątków SQL Servera działa w trybie użytkownika systemu Windows NT/2000, ale kilka (wątek przeprowadzający punkt kontrolny, wątek odpowiedzialny za logowanie użytkowników oraz wątek realizujący obsługę bibliotek sieciowych serwera) uruchomianych jest w trybie chronionym.

Poszczególne żądania klientów realizowane są według następującego schematu:

1. Jeżeli wątek jest już uruchomiony, SQL Server przydziela żądaniu realizujący je wątek.

2. Jeżeli wątek nie jest uruchomiony, a liczba aktywnych wątków jest mniejsza od maksymalnej, uruchomiony zostaje nowy watek.
3. Jeżeli wątek nie jest uruchomiony, a usługa ODS wykorzystuje już wszystkie dostępne wątki, żądanie klienta czeka, aż któryś z aktywnych wątków zakończy swoje działanie.

Ponadto niektóre zadania (szczególnie w wersji Enterprise) realizowane są równolegle przez klika wątków. Do takich operacji należą m.in. wykonywanie i odtwarzanie kopii zapasowych, pobieranie danych z kilku tabel w ramach jednego zapytania czy równoległe odczytywanie kilku indeksów.

Taka architektura sprawia, że SQL Server potrafi (i taka jest jego domyślna konfiguracja) wykorzystać wszystkie dostępne procesory komputera, na którym został zainstalowany. Liczbę procesorów, które będą wykorzystywane przez wątki SQL Servera możemy programowo określić, ustawiając wartość opcji affinity mask:

```
sp_configure 'affinity mask', '15'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'affinity mask' changed from 0 to 15. Run the RECONFIGURE
[a]statement to install.
```

Powyższy skrypt umożliwił SQL Serverowi korzystanie z pierwszych czterech procesorów (binarną reprezentacją liczby 15 jest 00001111). Ograniczenie liczby dostępnych procesorów może wpłynąć na wydajność SQL Servera poprzez ograniczenie liczby przełączeń pomiędzy wątkami uruchomionymi w trybie użytkownika a wątkami działającymi w trybie chronionym, dzięki pozostawieniu procesorów dostępnych systemowi operacyjnemu.

Ustawiając wartość opcji priority boost na 1, możemy uruchomić wątki SQL Servera z priorytetem 13 (domyślnym priorytetem jest 7):

```
sp_configure 'priority boost', '1'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'priority boost' changed from 0 to 1. Run the RECONFIGURE
[a]statement to install.
```

Zwiększenie priorytetu w komputerach jednoprocesorowych lub dla głównego procesora w komputerach wieloprocesorowych niekorzystnie wpłynie na wydajność i stabilność całego systemu.

Ustawiając wartość opcji lightweight pooling na 1, spowodujemy, że zadania wielowątkowe nie będą realizowane poprze kilka wątków, a poprzez uruchomienie w ramach pojedynczego wątku kilku włókien (ang. Fiber). Ponieważ przełączanie pomiędzy włóknami jest szybsze i wymaga mniej zasobów niż przełączanie pomiędzy wątkami, ustawienie tej opcji jest zalecane dla systemów wieloprocesorowych, w których wykorzystanie procesorów utrzymuje się na wysokim poziomie, a liczba przełączeń pomiędzy trybem użytkownika i chronionym jest duża:

```
sp_configure 'lightweight pooling', '1'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'lightweight pooling' changed from 0 to 1. Run the RECONFIGURE
[a]statement to install.
```

Tworząc linię bazową dostępnych zasobów komputera, należy dodać do dziennika następujące liczniki związane z procesorem:

1. Procesor\Czas procesora (%) — wartość licznika nie powinna stale przekraczać 90.
2. Procesor\Czas uprzywilejowany (%) — wartość licznika nie powinna stale przekraczać 10.
3. System\Długość kolejki procesora — wartość licznika nie powinna przekraczać 5, przy czym średnia wartość nie powinna przekraczać 3.
4. System\Przełączanie kontekstu/s — przy wartości powyżej 7000 należy przełączyć SQL Server w tryb włókien.

Dysk twardy

SQL Server odczytuje i zapisuje strony i zakresy danych, wywołując podsystem wejścia- -wyjścia systemu operacyjnego. W przypadku systemu Windows 2000 dane zawsze odczytywane i zapisywane są w blokach o wielkości 64 KB, odpowiadających zakresom danych.

Ponieważ wykonanie większości instrukcji języka Transact-SQL wymaga odczytania o wiele większej liczby danych niż liczba danych zwróconych użytkownikowi (np. odczytanie kilku tabel i indeksów), SQL Server stosuje technikę odczytu z wyprzedzeniem (informacje o odczytanych z wyprzedzeniem stronach poznamy, włączając statystyki IO w programie Query Analyzer). Technika ta stosowana jest w trzech przypadkach:

1. Podczas odczytu danych z dużych tabel — menadżer buforów, zamiast odczytywać zapisane w różnych jednostkach alokacji dane sekwencyjne, odczytuje maksymalnie 32 zakresy zapisane w sąsiednich jednostkach alokacji i umieszcza pobrane dane w buforze.
2. Podczas odczytu i przeszukiwania indeksów.
3. Podczas wyszukiwania danych w indeksach niegrupujących.

Strony przechowywane w buforze są okresowo odczytywane przez SQL Server. Ponieważ operacja ta wymaga odczytania danych bezpośrednio z pamięci operacyjnej, nie zwiększa ona obciążenia systemu wejścia-wyjścia. Podczas takiego odczytywania wszystkie strony przechowujące dane niezapisane na dysku twardym oznaczane są jako brudne (ang. dirty). Ponadto sprawdzane jest, ile razy w ciągu określonego czasu wystąpiło żądanie pobrania danych zapisanych na danej stronie. Jeżeli dane nie były pobierane, strona zostaje usunięta z bufora, przy czym strony brudne, przed skasowaniem zostają zapisane na dysku twardym. Niezależnie od procesu skanowania i usuwania stron zapisanych w buforze, podczas wykonywania punktu kontrolnego wszystkie brudne strony zostają zapisane na dysku twardym.

W wersji 2000 konfiguracja systemu wejścia-wyjścia serwera jest w dużym stopniu zautomatyzowana i z poziomu języka Transact-SQL możemy jedynie określić maksymalny czas potrzebny serwerowi do wykonania procesu odtwarzania spójności danych (proces ten uruchamiany jest podczas startu SQL Servera). Zmieniając domyślną wartość (0) opcji recovery interval (min), określimy, ile minut może zająć odtworzenie spójnego obrazu danych, a więc jak często brudne strony muszą zostać zapisane na dysku (domyślna wartość spowoduje dostosowanie wartości parametru do rzeczywistego, okresowo mierzonego obciążenia serwera):

```
sp_configure 'recovery interval (min)', '2'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'recovery interval (min)' changed from 0 to 2. Run the
[a]RECONFIGURE statement to install.
```

Zmiana domyślnej wartości może poprawić wydajność wysoce bezawaryjnego, wyposażonego w wydajne zasilacze awaryjne systemu, poprzez zmniejszenie częstotliwości występowania silnie obciążającego serwer (a szczególne układ wejścia-wyjścia) procesu tworzenia punktu kontrolnego.

Tworząc linię bazową dostępnych zasobów komputera, należy dodać do dziennika następujące liczniki związane z dyskiem twardym:

1. Dysk fizyczny\Czas dysku (%) — wartość licznika nie powinna stale przekraczać 85.
2. Dysk fizyczny\Średnia długość kolejki dysku — wartość licznika nie powinna przekraczać 4, przy czym średnia wartość nie powinna przekraczać 2.
3. Dysk fizyczny\Bajty odczytu dysku/s — wartość licznika nie powinna przekraczać 90% maksymalnej, określonej przez producenta urządzenia, liczby odczytywanych bajtów na sekundę.
4. Dysk fizyczny\Bajty zapisu dysku/s — wartość licznika nie powinna przekraczać 90% maksymalnej, określonej przez producenta urządzenia, liczby zapisywanych bajtów na sekundę.

Sieć

W przeciwieństwie do poprzednich zasobów, zasoby sieciowe, które mają bezpośredni wpływ na wydajność systemu bazodanowego, nie są wyłącznie zasobami komputera, na którym uruchomiono SQL Server. Głównymi czynnikami, które wpływają na wydajność sieci komputerowej są:

- ♣ Karta sieciowa komputera, na którym uruchomiono SQL Server.
- ♣ Umieszczenie komputera w silnie obciążonym segmencie sieci.
- ♣ Przepustowość lokalnej sieci komputerowej.
- ♣ Liczba danych przesyłanych pomiędzy serwerem a klientami.

Ponieważ z poziomu języka Transact-SQL na żaden z tych czynników nie mamy wpływu, pozostaje nam jedynie (o ile administrator serwera bazowanego nie jest jednocześnie administratorem sieci) monitorować dostępność zasobów sieciowych.

Tworząc linię bazową dostępnych zasobów komputera, do dziennika należy dodać następujące liczniki związane z siecią:

- ♣ Interfejs sieciowy\bajty odebrane/s — wartość licznika nie powinna przekraczać 90% maksymalnej, określonej przez producenta urządzenia, liczby odbieranych bajtów na sekundę.
- ♣ Interfejs sieciowy\bajty wysłane/s — wartość licznika nie powinna przekraczać 90% maksymalnej, określonej przez producenta urządzenia, liczby wysyłanych bajtów na sekundę.

W ten sposób utworzyliśmy dziennik zawierający 14 liczników. Dziennik ten może zostać wykorzystany do utworzenia linii bazowej — w takim wypadku należy uruchomić go w godzinach dużego i niewielkiego obciążenia systemu (np. o 11 rano i 19 po południu), zapisać wyniki, a następnie okresowo (np. co miesiąc) uruchamiać go w tych godzinach i porównywać otrzymane wyniki z poprzednimi.

Jeżeli wartości liczników nie odpowiadają podanym powyżej, należy rozbudować system komputerowy o brakujący zasób lub zasoby.

System operacyjny

Źle skonfigurowany system operacyjny nawet na najszybszym komputerze będzie działał wolno i niestabilnie. Po upewnieniu się, że powodem niskiej wydajności systemu bazodanowego nie jest po prostu brak pewnych zasobów sprzętowych, należy upewnić się, czy wina nie leży po stronie systemu operacyjnego komputera, na którym uruchomiono SQL Server.

Ponieważ tematem tej książki nie jest konfiguracja i optymalizacja systemów operacyjnych firmy Microsoft, poniżej znajdują się jedynie ogólne wskazówki, które mogą być przydatne administratorom baz danych.

Błędy konfiguracji systemu operacyjnego

Systemy Windows NT\2000 zawierają narzędzie pomagające wykrywać i diagnozować błędy w konfiguracji samego systemu operacyjnego i zainstalowanych programów. Uruchamiając narzędzie Podgląd zdarzeń, mamy możliwość prześledzenia obu interesujących nas w tym momencie dzienników:

- ♣ Dziennik systemu zawiera informacje, ostrzeżenia i błędy zgłoszone przez procesy systemu operacyjnego.
- ♣ Dziennik aplikacji zawiera analogiczne dane zgłoszone przez procesy klienckie, w tym SQL Server.

Ponieważ systemy NT i 2000 kilkakrotnie próbują uruchomić nieodpowiadające usługi, odnotowane w dzienniku systemu błędy w ich konfiguracji mogą wielokrotnie spowolnić działanie całego systemu. Ta sama uwaga dotyczy większości programów klienckich — chociaż to system zarządza ich dostępem do zasobów, miałem okazję przekonać się, że wydajność specjalistycznego serwera spadła o 90% z powodu uruchamianego programu antywirusowego, który próbował sprawdzić zawartość pozostawionej w napędzie CD „porysowanej” płyty.

Przed przystąpieniem do monitorowania i optymalizacji pracy SQL Severa należy naprawić wszystkie błędy systemu i aplikacji (wyjątkiem jest np. błąd spowodowany niemożnością zsynchronizowania zegara komputera z witryną www.microsoft.com).

Monitorowanie wydajności systemu operacyjnego

Niektóre z liczników umożliwiających monitorowanie wydajności systemu operacyjnego zostały już dodane do dziennika podczas monitorowania zasobów sprzętowych (brak któregoś z wcześniej wymienionych zasobów wpływa nie tylko na obniżenie wydajności SQL Servera, ale także wydajności systemu operacyjnego). Teraz do tej listy dodamy kolejne liczniki:

1. Plik stronicowania\Użycie (%) — jeżeli wartość licznika przekracza 85% zadeklarowanej wielkości pliku stronicowania, należy zwiększyć jego rozmiar początkowy.
2. Plik stronicowania\Użycie szczytowe (%) — jeżeli wartość licznika przekracza 95% zadeklarowanej wielkości pliku stronicowania. należy zwiększyć jego rozmiar początkowy.

   Plik pagefile.sys tworzony jest podczas uruchamiania systemu operacyjnego i jeżeli w trakcie jego pracy zostanie zapełniony — zostaje powiększony (domyślna konfiguracja systemu). W rezultacie następuje fragmentowanie tego pliku, co nieskorzystanie wpływa na wydajność systemu operacyjnego. Wielkość początkowa i maksymalna pliku powinny być takie same, wyznaczone na podstawie odczytów dwóch powyższych liczników.
3. Procesor\Przerwania/s — gwałtowny wzrost wartości licznika świadczy o źle skonfigurowanym lub uszkodzonym urządzeniu podłączonym do komputera. Średnia wartość licznika zależy od liczby i rodzaju zainstalowanych urządzeń i dla większości typowych konfiguracji nie powinna przekraczać 400.
4. Dysk fizyczny\Średnia liczba bajtów dysku\Transfer — ponieważ zapis i odczyt dużych bloków danych jest bardziej efektywny, średnia wartość licznika powinna być jak najwyższa (powyżej 50 000). W przeciwnym razie należy określić, która usługa lub aplikacja często zapisują lub odczytują małe ilości danych, a następnie właściwie skonfigurować taką usługę (aplikację).
5. Dysk fizyczny\Transfery dysku/s — o ile wartość maksymalna powinna być jak najwyższa, o tyle wysoka wartość średnia świadczy o ponawianych, z powodu kolejkowania operacji wejścia-wyjścia, próbach pobrania danych z dysku, o jego złej konfiguracji lub uszkodzeniu.
6. Interfejs sieciowy\Długość kolejki wyjściowej — wartość licznika nie powinna przekraczać 5, przy czym średnia wartość nie powinna przekraczać 3.
7. Interfejs sieciowy\Usunięte pakiety wychodzące — wysoka wartość licznika świadczy o osiągnięciu takiego stopnia obciążenia sieci, że poprawne pakiety, pomimo przechowywania ich przez pewien czas w buforze, zostają usunięte z powodu braku dostępu do nośnika.

SQL Server

Kolejnym etapem, po zoptymalizowaniu pracy sytemu operacyjnego, jest optymalizacja pracy SQL Servera. W tym wypadku, oprócz narzędzia Wydajność, można skorzystać z konsoli Enterprise Manager (sekcje Managment \SQL Server Agent\Display Error Log, Managment\SQL Server Logs, Managment\Current Activity), instrukcji języka Transact- -SQL, statystyk zwracanych przez program Query Analyzer oraz narzędzia SQL Profiler.

Aby tworzone linie bazowe zawierały wiarygodne dane, przed ich utworzeniem należy usunąć z bufora strony danych oraz prekompilowane plany wykonania instrukcji języka Transact-SQL:

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
```

Wykorzystanie zasobów sprzętowych przez SQL Server

Domyślna konfiguracja SQL Servera umożliwia mu dynamiczne przydzielanie sobie dostępnych zasobów sprzętowych i w większości przypadków nie ma potrzeby jej zmieniania. Jeżeli jednak konfiguracja systemu nie odpowiada konfiguracji zalecanej przez firmę Microsoft (np. komputer, na którym uruchomiono SQL Server pełni równocześnie funkcję serwera terminali czy kontrolera domeny), zmiana domyślnych ustawień może poprawić wydajność serwera bazodanowego. Stopień wykorzystania przez SQL Server zasobów komputera poznamy, dodając do dziennika następujące liczniki (kolejno zostały przedstawione liczniki związane z dostępną pamięcią i procesorem):

1. SQL Server: Buffer Manager\Total pages — niska, w stosunku do wielkości baz danych, liczba świadczy o braku dostępnej pamięci operacyjnej.
2. SQL Server: Memory Manager: Total Server Memory — jeżeli wartość licznika stale przekracza 90% zainstalowanej w komputerze pamięci operacyjnej, świadczy to o jej braku.
3. Proces\Błędy stron/s\sqlservr — stale utrzymująca się powyżej 0 wartość licznika świadczy o odwoływaniu się przez SQL Server do niedostępnych danych (danych, które nie zostały zapisane w buforze lub zostały zablokowane przez inny proces).
4. Proces\Bajty prywatne\sqlservr — wartość licznika nie powinna spadać poniżej 5 MB.
5. Procesor\Czas użytkownika (%) — stałe utrzymywanie się wysokich wartości licznika może świadczyć o konkurowaniu SQL Servera z innymi procesami użytkownika o dostęp do procesora.

6. Proces\Czas procesora (%)\sqlservr — porównując wskazania tego licznika i poprzedniego możemy sprawdzić, czy to SQL Server wygrywa konkurencję z innymi procesami użytkownika o dostęp do procesora.

Błędy SQL Servera

Tak jak błędy w konfiguracji systemu operacyjnego nieskorzystanie wpływają na wydajność systemu i wszystkich uruchomionych programów, błędy w konfiguracji i działaniu SQL Servera nieskorzystanie wpływają na wydajność systemu bazodanowego.

Z poziomu języka Transact-SQL możemy, wywołując dwie nieudokumentowane procedury systemowe, przeglądać dziennik błędów SQL Servera. Pierwsza procedura, sp_enumerrorlogs, zwraca informacje o istniejących dziennikach błędów (nowy dziennik tworzony jest automatycznie podczas uruchamiania serwera):

```
USE master
EXEC sp_enumerrorlogs
GO
Archive # Date Log File Size (Byte)
---------- ------------------ -------------------------
0 12/15/2002 12:51 2072
1 12/15/2002 10:06 2425
2 12/14/2002 19:17 2294
3 12/14/2002 18:58 2294
4 12/14/2002 18:53 2438
5 12/14/2002 10:54 2274
6 12/14/2002 10:50 2274
(7 row(s) affected)
```

Aktualny dziennik ma numer 0, ostatni (przechowywanych jest 7 dzienników) — numer 6. Kolejne uruchomienie SQL Servera spowoduje dodanie do numerów dzienników 1 i usunięcie najstarszego pliku dziennika.

Zawartość wybranego dziennika błędów możemy odczytać, wykonując procedurę sp_readerrorlog:

```
USE master
EXEC sp_readerrorlog 4
GO
ERRORLOG.4
ContinuationRow
-------------------------------------------------------------------------------- ---
2002-12-14 16:30:02.96 server Microsoft SQL Server 2000 - 8.00.100 (Intel X86) 0
Apr 18 2000 01:19:00 0
Copyright (c) 1988-2000 Microsoft Corporation 0
Enterprise Edition on Windows NT 5.0 (Build 2195:) 0
2002-12-14 16:30:03.01 server Copyright (C) 1988-2000 Microsoft Corporation. 0
2002-12-14 16:30:03.01 server All rights reserved. 0
2002-12-14 16:30:03.01 server Server Process ID is 688. 0
2002-12-14 16:30:03.01 server Logging SQL Server messages in file 'D:\Program
aFiles\Microsoft SQL Server\MSSQL\log\ERRORLOG'. 0
2002-12-14 16:30:03.14 server SQL Server is starting at priority class 'normal'(1 CPU
adetected). 0
```

```
2002-12-14 16:30:03.78 server Working Set size set to 129664 kilobytes. 0
2002-12-14 16:30:04.00 server User Mode Scheduler configured for fiber processing 0
2002-12-14 16:30:04.04 server Using dynamic lock allocation. [2500] Lock Blocks,
[a][5000] Lock Owner Blocks 0
2002-12-14 16:30:04.10 server Attempting to initialize Distributed Transaction
[a]Coordinator. 0
2002-12-14 16:30:06.50 spid3 Starting up database 'master'. 0
.
.
.
2002-12-14 18:53:31.48 spid3 SQL Server is terminating due to 'stop' request from
[a]Service Control Manager. 0
(31 row(s) affected)
```

Baza danych

Niektórzy uważają, że optymalizacja (strojenie) bazy danych zaczyna się właśnie na tym etapie — monitorowania i poprawy wydajności pracy konkretnej bazy danych użytkownika. Zakładają tym samym, że komputer posiada niezbędne zasoby sprzętowe, a system operacyjny i serwer bazodanowy są poprawnie skonfigurowane i bezbłędnie działające. Niestety, nie zawsze jest to prawdą. Wynika z tego, że zanim przeprowadzimy optymalizację bazy danych, musimy upewnić się, że żaden z tych trzech elementów nie jest wąskim gardłem systemu — w przeciwnym razie praca poświecona na strojenie bazy danych pójdzie na marne.

> Na bieżącą wydajność bazy danych znaczący wpływ ma wykrywanie i rozwiązywanie problemów związanych z aktywnością użytkowników, takich jak wzajemne blokowanie dostępu do zasobów przez procesy różnych użytkowników. Zagadnienie te zostały opisane w dalszej części rozdziału.

Fizyczna struktura bazy danych

Wielkość, uporządkowanie i operacje wykonywane na plikach bazodanowych mają istotny wpływ na wydajność bazy danych. Aby zapewnić maksymalną wydajność. należy:

> Fizyczna struktura plików bazodanowych została opisana w rozdziale 8.

1. Umieścić pliki bazy danych na odrębnym dysku twardym lub macierzy dyskowej, najlepiej połączonej poprzez własny port wejścia-wyjścia.
2. Umieścić pliki dziennika transakcyjnego na odrębnym, wydajnym dysku twardym lub macierzy dyskowej połączonej poprzez własny port wejścia-wyjścia.
3. Utworzyć kilka plików bazy danych i umieścić je na odrębnych dyskach twardych.
4. Tworząc grupy plików, zapisać na odrębnych dyskach dane tabeli i powiązane z tą tabelą indeksy.
5. Uporządkować fizyczną strukturę plików bazy danych poprzez regularne defragmentowanie dysków.
6. Uporządkować logiczną strukturę plików bazy danych, regularnie wykonując instrukcje: DBCC CHECKALLOC, DBCC CHECKDB, DBCC CHECKFILEGROUP, DBCC CLEANTABLE, DBCC INDEXDEFRAG.
7. Stale monitorować procent wykorzystania plików bazodanowych, w razie potrzeby zwiększając lub zmniejszając ich wielkość.

Podczas instalowania SQL Servera w systemie rejestrowany jest m.in. obiekt narzędzia Wydajność — SQL Server:Databases. Odczytując wartość liczników tego obiektu, poznamy m.in.:

1. Łączną wielkość wszystkich plików danych wybranej bazy (SQL Server:Databases\ Data File(s) Size (KB)\baza_danych).
2. Łączną wielkość wszystkich plików dziennika transakcyjnego wybranej bazy danych (SQL Server:Databases\Log File(s) Size (KB)\baza_danych).
3. Stopień wykorzystania plików dziennika transakcyjnego wybranej bazy danych (SQL Server: Databases\Percent Log Used\baza_danych) — jeżeli wartość licznika przekracza 80, należy zwiększyć rozmiar plików, zmienić model odzyskiwania bazy danych albo zwiększyć częstotliwość wykonywania kopii zapasowej dziennika.

4. Liczbę operacji zwiększania wielkości plików dziennika transakcyjnego wybranej bazy danych (SQL Server: Databases\Log Growths\baza_danych) — jeżeli wartość licznika stale rośnie, należy podjąć kroki zmierzające do ograniczenia liczby kosztownych operacji zwiększania rozmiaru plików.
5. Liczbę operacji zmniejszania rozmiaru plików dziennika transakcyjnego wybranej bazy danych (SQL Server: Databases\Log Shrinks\baza_danych) — jeżeli wartość licznika stale rośnie, należy podjąć kroki zmierzające do ograniczenia liczby kosztownych operacji zmniejszania rozmiaru plików.
6. Liczbę aktywnych transakcji (SQL Server: Databases\Active Transactions\ baza_danych).
7. Liczbę transakcji realizowanych w ciągu sekundy (SQL Server:Databases\ Transactions/sec\baza_danych).

Logiczna struktura bazy danych

Projektując wydajną bazę danych, należy uwzględnić konfigurację systemu, w którym baza będzie działała. Szczegółowe wskazówki dotyczące projektowania bazy danych, tworzenia tabel, tworzenia i roli indeksów (indeksy mają decydujący wpływ na wydajność bazy danych), procedur składowanych, funkcji użytkownika i wyzwalaczy znajdują się w drugiej części książki. Oddzielnym problemem jest zoptymalizowanie bazy danych replikowanej pomiędzy serwerami bazodanowymi (podstawowe informacje o replikacji baz danych znajdują się w rozdziale 21.).

Ponieważ nawet najlepiej zaprojektowana i zaprogramowana baza danych może okazać się niewydajna, należy okresowo sprawdzać wartości liczników:

1. SQL Server: Access Methods\Full Scans/sec — wysoka wartość licznika świadczy albo o braku indeksów, albo o niewłaściwie sformułowanych zapytaniach (zapytaniach, których wykonanie powoduje odczyt całego indeksu). Metody optymalizacji zapytań i zalety wynikające ze stosowania argumentów typu SARG w zapytaniach zostały opisane w rozdziale 4.
2. SQL Server: Access Methods\Page Split/sec — wysoka wartość licznika świadczy o źle dobranym współczynniku wypełnienia indeksów.
3. SQL Server: Locks\Locks Timeouts/sec — wysoka wartość licznika świadczy o źle zaprojektowanych tabelach lub niewłaściwej strategii blokowania dostępu do danych.
4. SQL Server: Locks\Number of Deadlocks/sec — wysoka wartość licznika świadczy o źle zaprojektowanych tabelach lub niewłaściwej strategii blokowania dostępu do danych.
5. SQL Server: SQL Statistics\SQL Compilations/sec — wysoka wartość licznika świadczy o braku procedur lub funkcji systemowych lub o niewłaściwym odwoływaniu się do obiektów (korzyści wynikające ze stosowania pełnych nazw obiektów zostały opisane w rozdziale 12.).

SQL Profiler

SQL Profiler jest instalowanym wraz z SQL Serverem programem monitorującym bieżącą aktywność bazy danych. Za jego pomocą możemy m.in. znaleźć najdłużej wykonywaną instrukcję języka Transact-SQL, poznać liczbę otwieranych kursorów itd. Plik śledzenia (dziennik programu Profiler) możemy również zdefiniować, wywołując procedury systemowe SQL Servera:

Jedynie użytkownicy należący do roli sysadmin mogą monitorować bieżącą aktywność SQL Servera.

1. Aby utworzyć nowy plik śledzenia, należy wywołać procedurę sp_trace_create. Poniżej znajduje się pełna definicja procedury:

2.
3. sp_trace_create [@traceid =] id_pliku OUTPUT
4. , [@options =] opcja
5. , [@tracefile =] 'plik'
6. [, [@maxfilesize =] wielkość]
7. [, [@stoptime =] 'czas_zatrzymania']
8.

gdzie:

opcja może przyjmować wartości: 0 (tworzony jest plik śledzenia), 2 (po osiągnięciu maksymalnej wielkości automatycznie tworzony jest nowy plik dziennika), 4 (po wystąpieniu błędu, który uniemożliwił zapisanie zdarzenia do pliku, SQL Server zostanie zatrzymany) lub 6 (obie opcje zostaną ustawione).

Przykład:

```
DECLARE @tid as INT
SET @tid = 51
EXEC sp_trace_create @traceid = @tid OUTPUT,
@options = 0,
@tracefile = N'D:\logowania'
GO
The command(s) completed successfully.
```

9. Dodać do pliku zdarzenie, którego wystąpienie zostanie zanotowane, możemy poprzez wykonanie procedury sp_trace_setevent:

```
10.
11. sp_trace_setevent [@traceid =] id_pliku
12. , [@eventid =] id_zdarzenia
13. , [@columnid =] id_kolumny
14. , [@on =] aktywne
15.
```

gdzie:

id_pliku określa plik śledzenia (aby można było dodać zdarzenia, śledzenie musi być zatrzymane),
id_zdarzenia określa obserwowane zdarzenie i może przyjmować wartości z zakresu od 10 do 117. Opis poszczególnych zdarzeń znajduje się w dokumentacji BOL,
id_kolumny dookreśla obserwowane zdarzenie i może przyjmować wartości z zakresu od 1 do 44. Opis poszczególnych zdarzeń znajduje się w dokumentacji BOL,
Aktywne określa, czy wystąpienie zdarzenia będzie dodane do pliku (1) czy nie(0).

Na przykład, aby dodać do pliku śledzenia zdarzenia logowania użytkownika i zanotować nazwę użytkownika systemu operacyjnego, który połączył się z SQL Serverem, napiszemy:

```
DECLARE @wl as bit
SET @wl = 1
EXEC sp_trace_setevent @traceid = 51,
@eventid = 14,
@columnid = 6,
@on = @wl
GO
The command(s) completed successfully.
```

16. Wybrać zdarzenia określonego typu, których wystąpienie będzie lub nie będzie dodane do pliku śledzenia, możemy poprzez procedurę:

```
17.
18. sp_trace_setfilter [@traceid =] id_pliku
19. , [@columnid =] id_kolumny
```

```
, [@logical_operator =] operator
, [@comparison_operator =] operator_porównania
, [@value =] wartość
```

gdzie:

operator określa operator koniunkcji (1) lub alternatywy (0),
operator_porównania może przyjmować jedną z wartości wymienionych w tabeli 20.1.

Tabela 20.1. Dopuszczalne operatory porównania procedury sp_trace_setfilter

Wartość	Opis
0	Równy
1	Różny
2	Większy niż
3	Mniejszy niż
4	Większy lub równy
5	Mniejszy lub równy
6	Zgodny ze wzorcem
7	Niezgodny ze wzorcem

Aby ograniczyć zapisywane do pliku śledzenia zdarzenia logowania wyłącznie do logowania administratora systemu, napiszemy:

```
sp_trace_setfilter @traceid = 51
, @columnid = 6
, @logical_operator = 1
, @comparison_operator = 6
, @value = N'Admin%'
GO
The command(s) completed successfully.
```

Po utworzeniu pliku śledzenia, zdefiniowaniu monitorowanych zdarzeń i ewentualnym dodaniu filtrów możemy uruchomić zapisywanie zdarzeń do pliku, wykonując procedurę sp_trace_setstatus:

```
sp_trace_setstatus [@traceid =] id_pliku
, [@status =] status
```

gdzie:

status może przyjmować jedną z trzech wartości: 0 — zatrzymanie śledzenia, 1 — uruchomienie śledzenia, 2 — zatrzymanie śledzenia i usunięcie definicji pliku śledzenia.

Aby uruchomić przykładowy plik śledzenia, napiszmy:

```
sp_trace_setstatus 51,1
GO
The command(s) completed successfully.
```

29. Informacje o wszystkich lub o określonym pliku śledzenia możemy poznać, wywołując funkcję systemową fn_trace_getinfo:

```
30.
31. SELECT *
32. FROM ::fn_trace_getinfo(default)
33. GO
34. traceid property value
35. ----------- ----------- ----------------------------
36. 51 1 0
37. 51 2 D:\PerfLogs\logowania
38. 51 3 NULL
39. 51 4 NULL
40. 51 5 0
41. (5 row(s) affected)
42.
```

gdzie:

1 — zwraca wartość atrybutu @options pliku śledzenia,
2 — nazwę pliku,
3 — rozmiar maksymalny,
4 — czas zatrzymania śledzenia,
5 — aktualny status śledzenia.

Program kliencki

Cały wysiłek administratora bazy danych poświęcony na optymalizację wydajności bazy danych może pójść na marne, jeśli programista stworzył niewydajny program kliencki. W takim wypadku nawet wart kilkadziesiąt tysięcy serwer skonfigurowany przez najlepszych specjalistów nie poprawi wydajności całego systemu.

Czynnikiem związanym programem klienckim, który ma zasadniczy wpływ na wydajność systemu bazodanowego jest sposób pobierania i przetwarzania danych. Jeżeli dane przetwarzane są wyłącznie po stronie serwera (lub poprzez serwer bazodanowy i dodatkowy serwer, który implementuje logikę biznesową), należy się liczyć z koniecznością przesyłania wielu informacji poprzez sieć (w takim wypadku np. posortowanie dopiero co pobranego zbioru rekordów według wartości innego argumentu spowoduje wysłanie zapytania do serwera, który jeszcze raz wyśle klientowi jego wynik — te same dane, ale inaczej posortowane). Zaletą tego rozwiązania jest wykonywanie operacji na zbiorach poprzez SQL Server, a nie program kliencki — optymalizator SQL Servera wielokrotnie szybciej sortuje czy filtruje zbiory danych niż program odwołujący się do obiektu ADO Recordset. Rozwiązanie to wymaga jednak wydajnego systemu bazodanowego. Jeżeli natomiast przetwarzanie raz pobranych danych odbywa się po stronie klienta, należy bardzo dokładnie rozpatrzyć kwestie związane z blokowaniem dostępu do danych — np. technologia ADO pozwala na porównanie pobranych danych z aktualnymi danymi zapisanymi w bazie przed podjęciem próby zapisania ich w bazie.

Możemy ocenić, czy wąskim gardłem systemu nie jest właśnie program kliencki, odczytując zawartość poniższych liczników:

1. SQL Server:General Statistics\User Connections — jeżeli wydajność systemu bazodanowego spada proporcjonalnie do liczby połączonych klientów, a poprzednio upewniono się, SQL Serwerowi nie brakuje żadnych zasobów systemowych, może świadczyć to o tym, że wina leży właśnie po stronie programu klienckiego.

2. SQL Server:SQL Statistics\Batch Requests/sec — duża liczba aktualizacji wsadowych może być powodem blokowania dostępu do zasobów i spadku wydajności systemu bazodanowego.

SQL Server umożliwia ograniczenie maksymalnego kosztu wykonania zapytania. Ustawiając wartość opcji query governor cost limit, uniemożliwimy klientom wykonywanie zapytań, których czas realizacji przekracza określoną liczbę sekund:

```
sp_configure 'query governor cost limit', '30'
RECONFIGURE
GO
DBCC execution completed. If DBCC printed error messages, contact your system
[a]administrator.
Configuration option 'query governor cost limit' changed from 0 to 30. Run the
[a]RECONFIGURE statement to install.
```

Monitorowanie bieżącej aktywności użytkowników

Aby zapewnić spójność danych, SQL Server blokuje dostęp użytkownikom do modyfikowanych w określonym czasie danych. W zależności od zastosowanej metody blokowania dostęp może być ograniczony do pojedynczych pól, wierszy, kolumn lub tabel. Tak czy inaczej, każda, nawet najbardziej liberalna polityka blokowania dostępu do modyfikowania danych może spowodować wystąpienie zakleszczeń oraz blokad kaskadowych.

Zakleszczenie występuje wtedy, gdy użytkownicy po zablokowaniu dostępu do pewnych zasobów, do momentu zakończenia transakcji (a więc zwolnienia zablokowanych zasobów) wymagają uzyskania dostępu do zasobów zablokowanych przez drugiego użytkownika. Taki problem jest automatycznie wykrywany i rozwiązywany przez SQL Server poprzez wycofanie mniej kosztownej transakcji i zatwierdzenie transakcji, która została wybrana na „zwycięzcę".

Blokada kaskadowa występuje wtedy, gdy zasoby zostały zablokowane przez pierwszego użytkownika, drugi użytkownik, po zablokowaniu innych zasobów, oczekuje na zakończenie transakcji pierwszego użytkownika i dopóki to nie nastąpi, blokuje inne zasoby, trzeci użytkownik do zakończenia transakcji wymaga dostępu do zasobów zablokowanych przez drugiego itd. Ponieważ taki problem nie jest automatycznie rozwiązywany przez SQL Server, administrator powinien stale monitorować aktywność użytkowników.

Procedura sp_who

Procedura zwraca informacje o procesach wszystkich lub wybranych użytkowników.

Składnia:

```
[@login_name =] 'login'
```

gdzie:

login określa użytkownika. Użycie słowa Active spowoduje wyświetlenie informacji wyłącznie o aktywnych procesach wszystkich użytkowników.

```
sp_who
GO
spid ecid status loginname hostname blk dbname cmd
------ ------ ------------ ------------------- --------- ---- -------- --------
[a]---------
1 0 background sa 0 NULL LAZY
[a]WRITER
2 0 sleeping sa 0 NULL LOG
[a]WRITER
3 0 background sa 0 master
```

```
ªSIGNAL HANDLER
.
.
.
53 0 sleeping LOLEK\Administrator LOLEK 0 master
ªAWAITING COMMAND
54 0 sleeping jacek LOLEK 0 Northwind
ªAWAITING COMMAND
55 0 sleeping agatka LOLEK 0 Northwind
ªAWAITING COMMAND
(15 row(s) affected)
```

Procedura sp_who2

Nieudokumentowana procedura sp_who2 zwraca dodatkowe informacje o procesach użytkowników, takie jak czas, w którym proces użytkownika miał dostęp do procesora, czas wykonania ostatniej instrukcji i nawa programu klienckiego:

```
sp_who2 'active'
GO
```

Procedura sp_lock

Listę wszystkich blokad lub blokad założonych przez wybranego klienta możemy poznać, wywołując procedurę sp_lock.

Składnia:

```
sp_lock [[@spid1 =] 'spid1'] [,[@spid2 =] 'spid2']
```

gdzie:

spid1 jest numerem procesu klienckiego. Numer procesu danego klienta możemy poznać albo odczytując zawartość tabeli master.dbo.sysprocesses, albo wywołując procedurę sp_who.

Poniższy przykład pokazuje sytuację, w której dostęp do wszystkich wierszy tabeli Categories został zablokowany przez użytkownika Agatka (proces o numerze spid = 55) i w rezultacie zapytanie użytkownika Jacek (spid = 54) odwołujące się do tej tabeli oczekuje na zwolnienie zasobów (nazwę obiektu możemy poznać, wywołując funkcję OBJECT_NAME):

```
sp_lock
GO
spid dbid ObjId IndId Type Resource Mode Status
------ ------ ----------- ------ ---- ---------------- -------- ------
51 1 85575343 0 TAB IS GRANT
52 6 0 0 DB S GRANT
54 6 0 0 DB S GRANT
54 6 2041058307 1 PAG 1:100 IS GRANT
54 6 2041058307 1 KEY (010086470766) S WAIT
54 6 2041058307 0 TAB IS GRANT
55 6 2041058307 2 KEY (6f02aca3cb50) X GRANT
55 6 2041058307 2 KEY (a001cca9fb2e) X GRANT
```

55 6 2041058307 2 KEY (f50154104fc9) X GRANT

55 6 2041058307 0 TAB IX GRANT

55 6 2041058307 2 KEY (050255028615) X GRANT

55 6 2041058307 1 KEY (03000d8f0ecc) X GRANT

55 6 2041058307 1 KEY (06003f7fd0fb) X GRANT

55 6 2041058307 1 KEY (0500d1d065e9) X GRANT

55 6 2041058307 2 KEY (d20213c0d76f) X GRANT

55 6 2041058307 1 KEY (010086470766) X GRANT

55 6 2041058307 2 KEY (8f01c7be1f0c) X GRANT

55 6 2041058307 1 KEY (08000c080f1b) X GRANT

55 6 2041058307 1 KEY (0400b4b7d951) X GRANT

55 6 2041058307 1 KEY (020068e8b274) X GRANT

55 6 2041058307 1 KEY (07005a186c43) X GRANT

55 6 2041058307 2 KEY (3002356af06e) X GRANT

55 6 2041058307 1 PAG 1:100 IX GRANT

55 6 2041058307 1 PAG 1:101 IX GRANT

55 6 2041058307 2 KEY (8002f2ea92dc) X GRANT

55 6 0 0 DB S GRANT

Instrukcja KILL

Określony proces użytkownika możemy zakończyć, wykonując instrukcję KILL.
Składnia:

KILL spid | UOW | UOW WITH {COMMIT | ABORT} [STATUSONLY]

gdzie:

UOW określa transakcję rozproszoną.
COMMIT spowoduje zatwierdzenie transakcji rozpoczętej przez dany proces.
ABORT spowoduje wycofanie transakcji rozpoczętej przez dany proces.
STATUSONLY zwraca informację o liczbie transakcji, które muszą zostać wycofane z powodu zakończenia procesu i może zostać użyty wyłącznie w przypadku zakończenia procesu o statusie ROLLBACK.
Na przykład, aby zakończyć proces Agatki i pozwolić Jackowi na odczytanie danych z tabeli Categories, napiszemy:

kill 55

GO

The command(s) completed successfully.

Kurs Transact SQL, część 21 (Replikacja baz danych)

Autor: Marcin Szeliga

Spis treści

Podstawą kursu jest wydana przez Helion książka *Transact-SQL. Czarna księga*.

Wszystkie ćwiczenia przygotowane są dla SQL Server 2000 i korzystają z przykładowych baz danych Northwind lub Pubs.

Replikacja polega na kopiowaniu i przesyłaniu danych lub obiektów bazodanowych pomiędzy wieloma serwerami oraz synchronizowaniu danych znajdujących się na biorących udział w replikacji serwerach bazodanowych.

Dzięki replikacji możemy: poprawić niezawodność systemu bazodanowego, umiejscowić informacje zapisane w bazach danych jak najbliżej użytkowników końcowych, a w rezultacie poprawić wydajność systemu bazodanowego.

Replikując dane:

- Umożliwiamy lokalny dostęp do danych użytkownikom z oddalonych oddziałów firmy (zamiast nawiązywać połączenia poprzez wolne, zawodne i niebezpieczne sieci WAN, użytkownicy będą łączyć się z lokalnym serwerem bazodanowym).
- Pozwalamy na częściową niezależność serwerów bazodanowych, które mogą cały dzień przechowywać kopie danych, a jedynie nocą, podczas niskiego obciążenia systemu, połączą się z dystrybutorem i zsynchronizują informacje.
- Możemy podzielić dane w sposób odpowiadający strukturze firmy.
- Możemy fizycznie rozdzielić serwery bazodanowe realizujące różne zadania w oparciu o te same dane (np. przenieść serwer OLAP na inny fizycznie SQL Server).

Wstęp do replikacji

Model wydawca — dystrybutor — subskrybent

Logiczny model replikacji firmy Microsoft opisuje trzy role, które mogą zostać przypisane serwerom bazodanowym: serwer może zostać skonfigurowany jako dystrybutor, wydawca oraz subskrybent.

1. Zadaniami serwera pełniącego rolę wydawcy są:
   - Utrzymywanie wzorcowej wersji danych.
   - Udostępnianie danych źródłowych innym serwerom.
   - Monitorowanie zmian w replikowanych danych i informowanie o nich serwer pełniący role dystrybutora.
2. Dane udostępnione przez wydawcę zostają przesłane na zarządzający replikacją serwer pełniący rolę dystrybutora. Zadaniami tego serwera są:
   - Utrzymywanie systemowej bazy danych distribution (baza zostanie automatycznie utworzona podczas przypisywania serwerowi tej roli).
   - Przechowywanie i zarządzanie metainformacjami związanymi z replikacją danych, takimi jak historia zmian danych, transakcje przeprowadzone na poszczególnych serwerach, konfiguracja serwerów biorących udział w replikacji itd.
3. Subskrybent jest serwerem bazodanowym, który:
   - Odbiera zmiany replikowanych danych przeprowadzone na innych serwerach.
   - Przechowuje lokalną kopię tych danych. Dane mogą zostać udostępnione użytkownikom tylko do odczytu lub do odczytu i modyfikacji.

Publikacje

Dane replikowane pomiędzy serwerami bazodanowymi składają się z publikacji. Publikacją jest dowolny zbiór artykułów pobranych z tej samej bazy danych (zakresem publikacji jest pojedyncza baza danych).

Artykuły mogą zawierać część danych przechowywanych w tabeli (w przypadku replikacji transakcyjnej niemożliwa jest projekcja danych) lub mogą składać się z takich obiektów jak: tabela, widok, procedura składowana czy funkcja.

Dowolna liczba artykułów, po przypisaniu ich do określonej publikacji, może zostać replikowana pomiędzy serwerami bazodanowym. Natomiast niemożliwe jest niezależne skonfigurowanie (np. określenie innego harmonogramu replikacji) pojedynczych artykułów.

Model replikacji zaimplementowany przez firmę Microsoft nie umożliwia replikowania obiektów lub danych powiązanych z innymi obiektami czy danymi bez jednoczesnego replikowania obiektów źródłowych — jeżeli dodaliśmy do publikacji procedurę składowaną odwołującą się do tabeli, tabela ta również musi zostać dodana do publikacji.

Tworząc kilka publikacji zawierających jedynie wybrane wiersze (selekcja) lub kolumny (projekcja), możemy wielokrotnie zmniejszyć liczbę danych przesyłanych podczas synchronizacji. Na przykład większość tabel oprócz kolumn często aktualizowanych zawiera kolumny aktualizowane bardzo rzadko — w takim wypadku wystarczy utworzyć odrębne publikacje: jedną synchronizowaną co godzinę, a drugą co 24 godziny, żeby bez zauważalnego wydłużenia opóźnienia zminimalizować liczbę przesyłanych danych.

Synchronizowanie danych

Zarówno dystrybutor, jak i subskrybent mogą zainicjować proces synchronizacji replikowanych danych. W pierwszym wypadku mówimy o replikacji wypychanej (ang. push), a w przypadku zainicjowania synchronizacji przez subskrybenta — o replikacji ściąganej (ang. pull).

Replikacja wypychana jest zalecana w przypadku synchronizowania poufnych danych pomiędzy niewielką liczbą serwerów i wymaga wydajnego komputera, na którym zainstalowano serwer pełniący rolę dystrybutora. Obciążająca subskrybentów replikacja ściągana zalecana jest w przypadku synchronizowania danych pomiędzy dużą liczbą serwerów.

Dystrybutor może zezwolić na pobranie replikowanych danych anonimowym serwerom (w takim wypadku SQL Server musi jedynie nawiązać połączenie z dystrybutorem, czyli wystarczy, że podamy login i hasło dystrybutora) lub ograniczyć listę subskrybentów do serwerów określonych podczas tworzenia publikacji.

Typy replikacji

Dane wchodzące w skład publikacji mogą być synchronizowane poprzez wysyłanie migawki bazy danych (replikację migawkową), kopię danych zmodyfikowanych od czasu ostatniej synchronizacji (replikację scalaną) lub poprzez przesyłanie aktywnej części dziennika transakcyjnego (replikację transakcyjną). Każdy typ replikacji cechuje się różnym stopniem niezależności serwerów (ich autonomią) oraz czasem, po którym dane na wszystkich serwerach są spójne (opóźnieniem).

Zerowe opóźnienie (ale za to również zerową autonomię) gwarantuje jedynie implementacja transakcji rozproszonych.

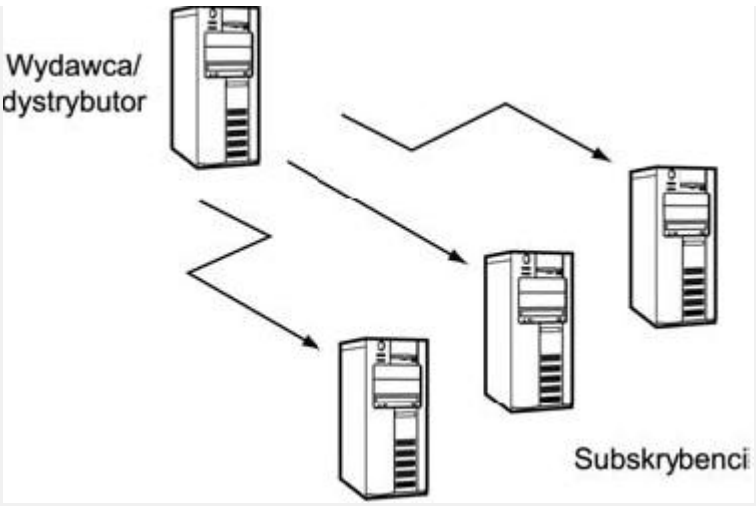
1. Replikacja transakcyjna (ang. Transactional replication) — dane zmodyfikowane w bazie wydawcy są przesyłane do subskrybentów w takiej kolejności, w jakiej zostały wprowadzone. W rezultacie serwery subskrybentów cechuje bardzo mała autonomia i niewielki czas opóźnienia. Ponieważ zmiany są zatwierdzane w kolejności ich wprowadzania, w tym typie replikacji nie występują konflikty (sytuacje, w których te same dane zostały zmodyfikowane na różnych serwerach) i podczas synchronizacji tylko jedna wartość będzie musiała zostać wprowadzona do wszystkich baz danych. Replikacja transakcyjna wymaga szybkiego i niezawodnego połączenia wszystkich serwerów bazodanowych i w praktyce nie jest stosowana poza sieciami lokalnymi (brak połączenia z jednym serwerem spowoduje niemożność obcięcia dziennika transakcyjnego bazy distribution i w krótkim czasie doprowadzi do zatrzymania pracy całego systemu).
2. Replikacja scalana (ang. Merge replication) — dane zmodyfikowane na jednym serwerze bazodanowym po określonym czasie są przesyłane do dystrybutora, który rozwiązuje konflikty i wysyła spójny obraz danych do wszystkich serwerów bazodanowych. Konflikty rozwiązywane są na podstawie priorytetów przyznanych poszczególnym serwerom. Ten typ replikacji cechuje pewna autonomia i związane z nią średnie opóźnienie.
3. Replikacja migawkowa (ang. Snapshot replication) — obraz bazy danych wydawcy poprzez dystrybutora systematycznie przesyłany jest subskrybentom. Ponieważ poszczególni subskrybenci przechowują te same dane, konflikty nie występują. Ten typ replikacji charakteryzuje najwyższa autonomia i największe opóźnienie.

Każdy z wymienionych typów replikacji został opisany w dalszej części rozdziału.

Fizyczne modele replikacji

Replikacja dowolnego typu może zostać zaimplementowana w jednym z trzech fizycznych modeli replikacji: centralnego wydawcy (dystrybutora), centralnego subskrybenta lub w modelu równorzędnym.

Domyślny model centralny wydawca (dystrybutor) zakłada, że jeden serwer pełni rolę wydawcy (dystrybutora) replikującego dane pomiędzy dowolną liczbą subskrybentów (rysunek 21.1).

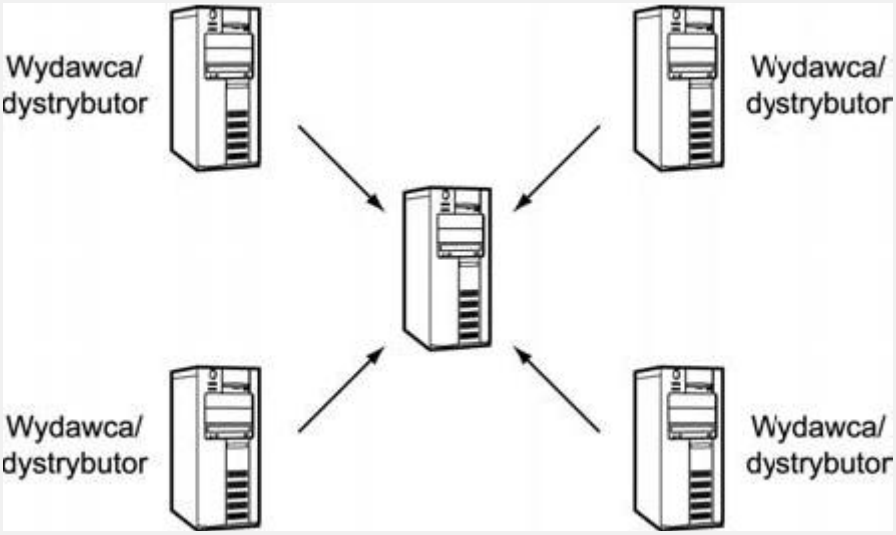


Rysunek 21.1. Model centralnego wydawcy (dystrybutora)

Role wydawcy i dystrybutora mogą zostać przypisane temu samemu lub odrębnym serwerom bazodanowym. W tym drugim przypadku większość operacji związanych z replikacją będzie wykonywanych przez dystrybutora, co odciąży serwer pełniący rolę wydawcy.

Typowym przykładem tego modelu jest replikacja danych w centralnie zarządzanej firmie, w której decyzje (zmiany) podejmowane są w centrali, a następnie udostępniane oddziałom — subskrybentom.

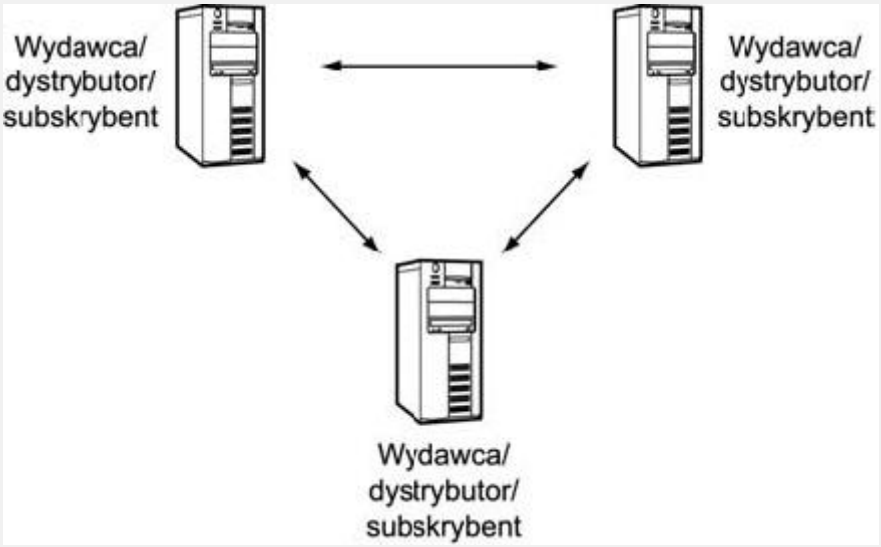
Model centralnego subskrybenta zakłada, że dowolna liczba wydawców (dystrybutorów) przesyła dane do jednego subskrybenta (rysunek 21.2).



Rysunek 21.2. Model centralnego subskrybenta

W tym przypadku dane modyfikowane jednocześnie na wielu serwerach (oddziałach) zostają przesłane do centrali (subskrybenta), gdzie są analizowane.

Model równorzędny zakłada, że w replikacji bierze udział wielu dystrybutorów (wydawców) i wielu subskrybentów (rysunek 21.3).



Rysunek 21.3. Model równorzędny

Implementacja modelu centralnego subskrybenta wymaga zmiany logicznej struktury bazy danych polegającej na zmianie kluczy głównych na klucze kompozytowe jednoznacznie identyfikujące wiersz i serwer (oddział), w którym zmodyfikowano wiersz lub filtrowanie danych (projekcji) wybranych do publikacji. W przeciwnym wypadku liczba konfliktów uniemożliwi wydajną pracę systemu.

Implementacja tego modelu jest najtrudniejsza i wymaga dokładnego zaplanowania struktury replikowanej bazy danych i zawartości poszczególnych publikacji — w innym wypadku utrzymanie spójnego obrazu danych będzie odbywało się kosztem albo ograniczenia funkcjonalności poszczególnych serwerów, albo wycofywania podczas synchronizacji sporej części transakcji użytkowników.

Konfiguracja replikacji pomiędzy serwerami bazodanowymi

Replikacja danych pomiędzy serwerami bazodanowym wymaga:

1. Uruchomienia usług SQL Server i SQLServerAgent w kontekście zabezpieczeń konta użytkownika domenowego (do replikacji nie jest wymagane nadanie tym użytkownikom uprawnień administratora komputera lub domeny).
2. Nadania uprawnień użytkownikowi, w kontekście którego działa usługa SQLServerAgent, do folderu, w którym zapisywane są dane publikacji.
3. Umożliwienia usłudze SQLServerAgent nawiązania połączenia z pozostałymi serwerami bazodanowymi replikującymi dane.

   Chociaż replikacja realizowana jest przez agentów replikacji, ich działanie wymaga uruchomienia usługi SQLServerAgent na wszystkich komputerach replikujących dane.
4. Umożliwienia obu głównym usługom SQL Servera zapis i odczyt kluczy rejestru związanych z SQL Serverem.
5. Ujednolicenia zestawu znaków wszystkich baz danych biorących udział w replikacji.

Wybór dystrybutora

Jako pierwszy należy wybrać serwer bazodanowy, który będzie pełnił rolę dystrybutora. To on będzie koordynował synchronizację danych i przechowywał (w bazie distribution oraz na lokalnych dyskach twardych) replikowane dane. Dystrybutorem może być ten sam serwer, który będzie pełnił rolę wydawcy lub osobny serwer bazodanowy. Zaletami tego drugiego rozwiązania są:

- ♣ Wzrost wydajności systemu bazodanowego osiągnięty dzięki przydzieleniu dystrybutorowi i wydawcy własnych zasobów sprzętowych.
- ♣ Zrównoważenie obciążenia serwerów.

Rozwiązanie to ma również wady:

- ♣ Wzrost kosztów.
- ♣ Utworzenie dodatkowego punktu awarii systemu.
- ♣ Znaczne zwiększenie obciążenia sieci w segmencie, w którym znajdują się wydawca i dystrybutor.

Rolę dystrybutora powinien pełnić SQL Server w najwyższej ze wszystkich biorących udział w replikacji serwerów wersji.

Wybór wydawcy i subskrybenta

Po określeniu, który serwer będzie pełnił rolę dystrybutora, wybór wydawcy i subskrybenta zależy wyłącznie od wybranego modelu replikacji i od lokalizacji istniejących serwerów bazodanowych. W przypadku replikacji transakcyjnych wydawcą nie może być SQL Server zainstalowany w środowisku systemów Windows 9x.

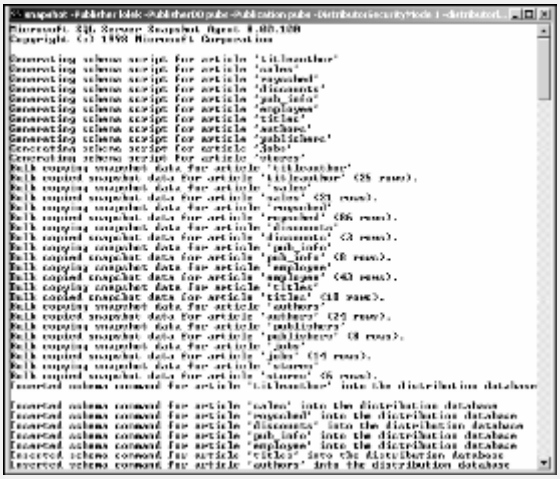
Agenci replikacji

Za replikację danych (tworzenie kopii obiektów bazodanowych i danych, wykrywanie modyfikacji, przesyłanie publikacji i wcielanie ich do baz subskrybentów) odpowiedzialne są usługi replikacji nazywane agentami.

Agent migawki

Agent migawki (ang. Snapshot Agent) — używany w replikacjach każdego typu, odpowiada za wykonywanie pełnego obrazu publikacji, włącznie z definicją wszystkich replikowanych obiektów bazodanowych. Tak utworzony obraz (migawka) jest podstawą wszystkich replikacji i służy do wstępnego zsynchronizowania baz danych. Jeden agent migawki odpowiada za jedną publikację.

Zdiagnozować ewentualne problemy w pracy agenta możemy poprzez wywołanie go z wiersza polecenia oraz analizę zwracanych komunikatów (rysunek 21.4). Plik wykonywalny agenta, snapshot.exe, znajduje się w folderze ProgramFiles%\Microsoft SQL Server\80\COM:



Rysunek 21.4. Wynik uruchomienia agenta migawki z wiersza polecenia

D:\Program Files\Microsoft SQL Server\80\COM\snapshot -Publisher LOLEK –PublisherDB

ªpubs -Publication pubs –DistributorSecurityMode 1 –DistributionLogin E

Agent transakcji

Agent transakcji (ang. Log Reader Agent) — używany wyłącznie w replikacji transakcyjnej, odpowiada za przesyłanie transakcji oznaczonych do replikacji z dystrybutora do subskrybentów. Jeden agent transakcji odpowiada za replikowanie jednej bazy danych.

Ewentualne problemy w pracy agenta możemy zdiagnozować, wywołując go z wiersza polecenia i analizując zwracane komunikaty (rysunek 21.5). Plik wykonywalny agenta, logread.exe, znajduje się w folderze %ProgramFiles%\Microsoft SQL Server\80\COM:

D:\Program Files\Microsoft SQL Server\80\COM\logread -Publisher lolek -PublisherDB

ªpubs -DistributorSecurityMode 1

Rysunek 21.5. Ponieważ poprawnie skonfigurowany agent transakcji jest automatycznie uruchamiany, kolejna próba jego uruchomienia, bez zatrzymania działającego dla bazy danych agenta, zakończy się wyświetleniem komunikatu błędu 18 752 i wszystkich błędów będących jego konsekwencją

Informacje wyświetlone na rysunku 21.5 świadczą o poprawnej pracy agenta — w bazie wydawcy nie ma żadnych transakcji, które należałby wysłać do subskrybentów.

Agent scalania

Agent scalania (ang. Merge Agent) — używany wyłącznie w replikacji scalanej, odpowiada za wcielenie początkowej migawki do bazy subskrybenta oraz za przesyłanie i wcielanie danych zmodyfikowanych w replikowanych bazach. Jeden agent migawki odpowiada za jedną publikację scalaną. W przypadku replikacji wypychanej agent działa po stronie dystrybutora, w przypadku replikacji ściąganej — po stronie subskrybenta.

Ewentualne problemy w pracy agenta możemy zdiagnozować, wywołując go z wiersza polecenia i analizując zwracane komunikaty (rysunek 21.6). Plik wykonywalny agenta, replmerg.exe, znajduje się w folderze %ProgramFiles%\Microsoft SQL Server\80\COM:

Rysunek 21.6. Wynik uruchomienia agenta scalania z wiersza polecenia

D:\Program Files\Microsoft SQL Server\80\COM\replmerg -Publisher lolek -PublisherDB

[a]pubs -Publication pubs2 -Subscriber lolek\kopia -SubscriberDB p_3 -SubscriptionType

[a]0 -SubscriberSecurityMode 1 -Distributor lolek -DistributorLogin sa

Agent dystrybucji

Agent dystrybucji (ang. Distribution Agent) — używany w replikacjach migawkowej i transakcyjnej, odpowiada za przesyłanie pliku migawki i transakcji z dystrybutora do subskrybentów. W przypadku replikacji wypychanej agent działa po stronie dystrybutora, w przypadku replikacji ściąganej — po stronie subskrybenta.

Ewentualne problemy w pracy agenta możemy zdiagnozować, wywołując go z wiersza polecenia i analizując zwracane komunikaty (rysunek 21.7). Plik wykonywalny agenta, distrib.exe, znajduje się w folderze %ProgramFiles%\Microsoft SQL Server\80\COM:

D:\Program Files\Microsoft SQL Server\80\COM\distrib -Subscriber lolek\kopia

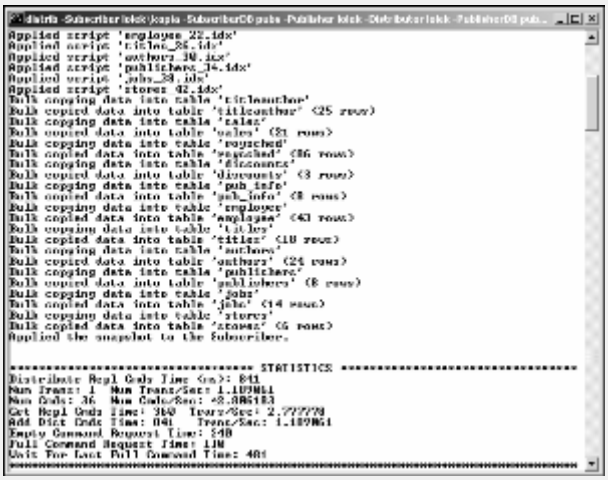
[a]-SubscriberDB pubs -Publisher lolek -Distributor lolek -PublisherDB pubs

[a]-DistributorSecurityMode 1 -distributorlogin E

Agent kolejkowania

Agent kolejkowania (ang. Queue Reader Agent) — używany w replikacjach migawkowej i transakcyjnej, odpowiada za realizowanie określonych harmonogramów synchronizowania danych.

Ewentualne problemy w pracy agenta możemy zdiagnozować, wywołując go z wiersza polecenia i analizując zwracane komunikaty (rysunek 21.8). Plik wykonywalny agenta, qrdrsvc.exe, znajduje się w folderze %ProgramFiles%\Microsoft SQL Server\80\COM:



Rysunek 21.7. Wynik uruchomienia agenta dystrybucji z wiersza polecenia

Rysunek 21.8. Wynik uruchomienia agenta kolejkowania z wiersza polecenia

```
D:\Program Files\Microsoft SQL Server\80\COM>"D:\Program Files\Microsoft SQL Server\
a80\COM\qrdrsvc.exe
```

Rolę dystrybutora i wydawcy możemy przypisać serwerom, uruchamiając kreatora publikacji i dystrybucji (ang. Publishing and Distribution Wizard).

Replikacja migawkowa

Replikacja migawkowa polega na wykonywaniu kopii danych i obiektów wydawcy i przesyłaniu ich do subskrybentów. Za przebieg replikacji odpowiadają uruchomiony na serwerze pełniącym rolę dystrybutora agent migawki oraz agent dystrybucji.

Synchronizując dane, agent migawki kolejno:

1. Nawiązuje połączenie z wydawcą i zakłada współdzieloną blokadę na wszystkich tabelach wchodzących w skład publikacji.
2. Odczytuje strukturę wybranych tabel i zapisuje je do plików .sch. Dodatkowo, jeżeli publikacja zawiera obiekty tego typu, zostaną utworzone pliki .idx zawierające definicję indeksów, .trg — wyzwalaczy i .dri — zawężeń.

   Domyślnie pliki migawki zapisywane są w podfolderze folderu \ProgramFiles\ Microsoft SQL Server\MSSQL\REPLDATA, ale ze względów bezpieczeństwa i w celu poprawy wydajności powinny zostać zapisane na odrębnym dysku, chyba że w replikacji biorą udział serwery w wersji 7.0 lub wcześniejszej.
3. Odczytuje wybrane dane i zapisuje je do plików .bcp.
4. Zapisuje historię wykonanych operacji do tabeli distribution..MSrepl_ transactions oraz distribution..MSrepl_commands i zwalnia założone przez siebie blokady.

Agent dystrybucji, aby przesłać dane migawki subskrybentom, kolejno:

1. Nawiązuje połączenie z dystrybutorem i odczytuje zawartość tabel. distribution..MSrepl_transactions oraz distribution..MSrepl_commands.
2. Pobiera pliki migawki i przesyła je do komputera, na którym został uruchomiony serwer pełniący rolę subskrybenta.
3. Jeżeli zachodzi taka potrzeba (subskrybentem jest inny serwer bazodanowy lub SQL Server w innej wersji), konwertuje instrukcje języka Transact-SQL tworzące obiekty bazodanowe.
4. Tworzy obiekty bazodanowe.
5. Wypełnia obiekty bazodanowe danymi.
6. Zapisuje historię swoich operacji w bazie distribution.

Utworzyć i skonfigurować replikację migawkową możemy poprzez uruchomienie kreatora publikacji (ang. Publication Wizard).

Replikowane dane są zapisywane do plików, a następnie importowane do docelowych baz danych za pomocą narzędzia bcp (ang. Bulk Copy Program). Jeżeli wszystkie bazy danych replikujące dane pracują w środowisku SQL Servera 2000, dane zapisywane są w charakterystycznym dla tego serwera formacie, w przeciwnym wypadku dane zostaną zapisane w postaci pliku tekstowego, co wiąże się z obniżeniem wydajności programu.

Poprawić wydajność replikacji możemy poprzez modyfikację wartości poniższych opcji agenta migawki:

1. MaxBcpThreads — domyślnie import i eksport danych realizowany jest przez pojedynczy wątek. W systemach wieloprocesorowych, w których dane zapisane są na kilku dyskach twardych, zwiększenie liczby wątków może skrócić czas potrzebny na kopiowanie danych.
2. BcpBatchSize — domyślna wartość (100 000 wierszy) powinna zostać zwiększona w systemach, w których replikowana jest duża liczba danych, a połączenie pomiędzy serwerami cechuje się dużą niezawodnością. W

przypadku awaryjnych połączeń zmniejszenie domyślnej wartości spowoduje ponowienie wysyłania mniejszych pakietów.

Replikacja transakcyjna

Utworzyć i skonfigurować replikację transakcyjną możemy poprzez uruchomienie kreatora publikacji (ang. Publication Wizard).

Replikacja transakcyjna polega na kopiowaniu transakcji przeprowadzonych w bazie danych wydawcy do baz subskrybentów i przebiega w trzech etapach:

1. Początkowa migawka bazy wydawcy zostaje wcielona do bazy subskrybentów w celu zsynchronizowania baz danych.
2. Agent transakcji systematycznie sprawdza, czy w dzienniku transakcyjnym bazy wydawcy nie zostały zapisane transakcje, które należy skopiować do subskrybentów i jeżeli je znajdzie — kopiuje je do dystrybutora. Agent transakcji kolejno:
    - ♣ Zapisuje do tabeli distribution.dbo.MSlogreader_history numer LSN ostatnio przesłanej transakcji.
    - ♣ Łączy się z wydawcą i odczytuje dziennik transakcyjny replikowanej bazy, sprawdzając, czy zostały w nim zapisane transakcje o wyższym numerze LSN (aby odczytać dziennik, agent zdalnie wywołuje procedurę sp_replcmds),
    - ♣ Kopiuje transakcje tworzące publikacje do dystrybutora.
    - ♣ Wykonuje procedurę systemową sp_repldone oznaczającą transakcje jako odczytane.
    - ♣ Podczas najbliższego obcinania dziennika transakcyjnego usuwa transakcje o numerach mniejszych niż LSN.
3. Agent dystrybucji przesyła i wciela transakcje do baz danych subskrybentów.

Ponieważ transakcje oznaczone jako nieprzeczytane nie mogą zostać usunięte, niewłaściwa praca agenta transakcji może być przyczyną ciągłego zwiększania rozmiaru pliku dziennika transakcyjnego bazy distribution.

Ponieważ replikacja transakcyjna polega na kopiowaniu instrukcji języka Transact-SQL, a nie danych, niemożliwe jest dołączenie do publikacji tabel, które nie mają zdefiniowanego klucza głównego. Spójność replikowanych danych administratorzy z reguły wymuszają, dodając do definicji zawężeń i wyzwalaczy klauzulę NOT FOR REPLICATION — w takim wypadku zawężenie zostaje sprawdzone, a wyzwalacz uruchomiony, o ile dane zostały zmodyfikowane w lokalnym serwerze, a nie w wyniku replikacji.

Przed skonfigurowaniem replikacji transakcyjnej należy upewnić się, że bazy danych pracują w pełnym modelu odtwarzania, w przeciwnym wypadku operacje nielogowalne nie zostaną zreplikowane.

Aktywni subskrybenci

W replikacji transakcyjnej (tak jak w migawkowej, ale w praktyce nie stosuje się replikacji migawkowej z aktywnymi subskrybentami) można zezwolić subskrybentom na zmianę replikowanych danych. Aby jednak było to możliwe, należy spełnić poniższe warunki:

1. Dane nie mogą być dodatkowo replikowane przez scalanie.
2. Niemożliwe jest modyfikowanie po stronie subskrybentów danych typu text, ntext oraz image.
3. Bazy subskrybentów nie mogą zawierać innych danych niż dane wcielone poprzez migawkę początkową.
4. Niemożliwe będzie takie pionowe filtrowanie danych, w wyniku którego artykuły nie będą zawierały wartości automatycznie dodanej kolumny MSrepl_tran_version.
5. Dodatkowym ograniczeniem jest wymóg określania kolumn instrukcji INSERT podczas wstawiania wierszy po stronie subskrybentów.

Jeżeli dane zostały zmodyfikowane po stronie subskrybenta, uruchomiona zostaje usługa MS DTC i transakcja zostaje, w dwóch fazach, zatwierdzona zarówno na serwerze pełniącym rolę wydawcy, jak i na serwerze pełniącym rolę subskrybenta. Od tego momentu modyfikacja jest traktowana tak, jakby została przeprowadzona po stronie wydawcy i zgodnie z harmonogramem replikacji zostanie wysłana do pozostałych subskrybentów.

Monitorować replikację transakcyjna możemy z poziomu języka Transact-SQL za pomocą poniższych procedur systemowych:

Procedura sp_browsereplcmds

Procedura zwraca listę instrukcji wykonanych w związku z replikacją danych.

Składnia:

```
sp_browsereplcmds [[@xact_seqno_start =] 'początkowa_instrukcja']
[,[@xact_seqno_end =] 'końcowa_instrukcja']
[,[@originator_id =] 'id_programu']
[,[@publisher_database_id =] 'id_wydawcy']
[,[@article_id =] 'id_artykułu']
[,[@command_id =] 'id_instrukcji']
```

Przykład:

```
USE distribution
EXEC sp_browsereplcmds @article_id =13
GO
```

Procedura sp_replshowcmds

Procedura zwraca listę instrukcji oczekujących na wykonanie, odpowiadającą zmianom, które wystąpiły po ostatnim wywołaniu procedury sp_repldone. Przed jej wywołaniem należy zatrzymać agenta transakcji.

Składnia:

```
sp_replshowcmds [@maxtrans =] największa_liczba_transakcji
```

Przykład:

```
USE pubs
EXEC sp_replshowcmds
GO
xact_seqno originator_id publisher_database_id article_id type command
---------------------- ------------- ---------------------- ----------- ---
0x00000007000000DF0008 0 0 14 30 {CALL
ªsp_MSdel_authors ('998-72-3563')}
(1 row(s) affected)
```

Procedura sp_repltrans

Procedura zwraca listę transakcji przeznaczonych do replikacji, ale niedostarczonych do dystrybutora. Duża liczba takich transakcji może wymagać skrócenia czasu, po upływie którego następuje synchronizacja danych.

Składnia:

```
sp_repltrans
```

Przykład:

```
USE pubs
EXEC sp_repltrans
GO
xdesid xact_seqno
```

```
---------------------- ----------------------
0x00000007000000DF0001 0x00000007000000DF0008
0x00000007000000E10001 0x00000007000000E10005
(2 row(s) affected)
```

Procedura sp_replflush

Wykonanie procedury usuwa artykuły czekające na skopiowanie do subskrybentów oraz wyrejestrowuje uruchomionego dla bazy danych agenta transakcji.

Składnia:

```
sp_replflush
```

Przykład:

```
USE pubs
EXEC sp_replflush
GO
The command(s) completed successfully.
```

W przypadku replikacji transakcyjnej duży wpływ na wydajność całego systemu ma serwer pełniący rolę subskrybenta — silnie obciążająca system replikacja transakcyjna nakłada dodatkowe wymagania co do zasobów tego komputera.

Poprawić wydajność replikacji możemy poprzez modyfikację wartości poniższych opcji agenta transakcji:

1. PollingInterval — zmieniając domyślną wartość (10 sekund) określimy, jak często agent ma sprawdzać, czy w dzienniku transakcyjnym bazy wydawcy nie pojawiły się nowe transakcje (opcja dostępna jest również dla agenta dystrybucji).
2. ReadBatchSize — ponieważ agent odczytuje jednokrotnie określoną liczbę transakcji, zwiększenie tego parametru poprawi wydajność w sytuacji, gdy niewielki procent transakcji był przeznaczony do replikacji.
3. MaxCmdsInTran — transakcje składające się z wielu instrukcji mogą zostać przez agenta podzielone na kilka mniejszych, co np. w przypadku problemów z połączeniem sieciowym może poprawić wydajność replikacji. Wartość parametru określa maksymalną liczbę instrukcji składających się na pojedynczą transakcję.

Replikacja scalana

Replikacja scalana polega na systematycznym wprowadzaniu zmian dokonanych na jednym z serwerów (na wydawcy lub którymś z subskrybentów) do pozostałych serwerów biorących udział w replikacji. Ponieważ dane mogą być jednocześnie modyfikowane na różnych serwerach, będą występowały rozwiązywane przez dystrybutora konflikty. Za replikację tego typu odpowiadają agent migawki i agent scalania.

Utworzyć i skonfigurować replikację scalaną możemy poprzez uruchomienie kreatora publikacji (ang. Publication Wizard).

W ramach każdej publikacji replikacji scalanej agent migawki kolejno:

1. Tworzy migawkę zawierającą definicje obiektów i dane oraz zapisuje ją na dystrybutorze.
2. Tworzy cyklicznie uruchamiane zdanie, w ramach którego synchronizowane będą dane i rozwiązywane ewentualne konflikty.
3. Dodaje do każdej tabeli wchodzącej w skład publikacji kolumnę typu uniqueidentifier o nazwie rowguid.
4. Tworzy niegrupujący, unikalny indeks powiązany z kolumną rowguid o nazwie typu index_liczba.
5. Dodaje dwie tabele systemowe. Tabela MSmerge_contents umożliwia replikację zmienionych i dodanych wierszy, tabela MSmerge_tombstone — replikację usuniętych wierszy.
6. Dla każdej tabeli wchodzącej w skład publikacji tworzy wyzwalacze wywoływane wykonaniem instrukcji INSERT, UPDATE i DELETE o nazwach typu ins_losowyGUID, del_losowyGUID, upd_losowyGUID. Wyzwalacze te zapisują modyfikowane dane w uprzednio utworzonych tabelach systemowych.
7. Zapisuje historię swoich operacji w bazie distribution.

Zadaniem agenta scalania jest:

1. Odtworzenie tabeli MSmerge_contents we wszystkich serwerach pełniących rolę subskrybenta.

2. Odczyt zawartości tej tabeli we wszystkich serwerach biorących udział w replikacji.
3. Odzwierciedlenie zmian wprowadzonych po stronie subskrybenta na serwerze pełniącym rolę wydawcy.
4. Po skopiowaniu wszystkich wierszy do wydawcy odzwierciedlenie zmian wprowadzonych po stronie wydawcy na serwerach pełniących rolę subskrybenta.
5. Rozwiązanie konfliktów, jeżeli wiersz o tej samej wartości rowguid znajdował się w kilku bazach danych.

W przeciwieństwie do dwóch poprzednich typów replikacji, które do wymiany danych wykorzystywały zasoby komputera, na którym uruchomiono serwer pełniący rolę dystrybutora i samą bazę distribution, replikacja scalana dodaje niezbędne tabele systemowe do replikowanych baz danych. W tym przypadku rola dystrybutora sprowadza się do:

- ♣ Identyfikowania serwerów biorących udział w replikacji i ich ról.
- ♣ Przechowywania informacji o lokalizacji plików migawki.
- ♣ Przechowywaniu informacji o czasie ostatniej synchronizacji.

Domyślnie zmiany danych przechowywane są przez 14 dni, co oznacza, że jeżeli któryś z subskrybentów przez tak długi okres czasu będzie niedostępny i nie scali zmian, będzie to wymagać ponownej synchronizacji wykonanej za pomocą migawki.

Rozwiązywanie konfliktów

Aby możliwe było rozwiązywanie konfliktów, dane nie mogą być równocześnie synchronizowane za pomocą kilku replikacji scalanych.

Konflikty w replikacji scalanej mogą być wykrywane i rozwiązywane na poziomie całych wierszy lub poszczególnych kolumn tabeli. W pierwszym przypadku jakakolwiek modyfikacja wiersza przeprowadzona w kilku bazach danych powoduje powstanie konfliktu, w drugim — jedynie zmiana danych przechowywanych w tych samych kolumnach. Domyślnie SQL Server wykrywa konflikty na poziomie kolumn. Zaletą wykrywania konfliktów na poziome kolumn jest zminimalizowanie ich liczby i ograniczenie do minimum danych, które muszą być przesyłane podczas synchronizacji baz danych. Wadą tego rozwiązania jest wysoki koszt związany z koniecznością śledzenia zmian zachodzących w poszczególnych kolumnach.

Niezależnie od poziomu blokowania konflikty rozwiązywane są na podstawie priorytetów nadanych poszczególnym serwerom. Serwerem z najwyższym priorytetem jest zawsze wydawca — zmiany wprowadzone na tym serwerze zastąpią zmiany wprowadzone na jakimkolwiek serwerze pełniącym rolę subskrybenta. Konflikty zawsze mogą zostać rozwiązane, ponieważ zmodyfikowane dane są przechowywane w poszczególnych bazach danych w automatycznie utworzonych tabelach o nazwach typu conflict_Publikacja_ Artykuł. Na przykład, po utworzeniu publikacji pubs2 obejmującej wszystkie tabele bazy pubs, w bazie zostaną utworzone poniższe tabele:

```
USE pubs
SELECT LEFT(name,30)
FROM dbo.sysobjects
WHERE name LIKE 'conflict%'
GO
------------------------------
conflict_pubs2_authors
conflict_pubs2_discounts
conflict_pubs2_employee
conflict_pubs2_jobs
conflict_pubs2_pub_info
conflict_pubs2_publishers
conflict_pubs2_roysched
conflict_pubs2_sales
conflict_pubs2_stores
conflict_pubs2_titleauthor
conflict_pubs2_titles
(11 row(s) affected)
```

Należy także poświecić uwagę automatycznie generowanym wartościom kluczy głównych. Ponieważ będą one generowane w kilku serwerach, aby zapewnić im niepowtarzalność, należy przypisać zakresy wartości nadawanych przez poszczególne serwery. Z poziomu języka Transact-SQL możemy zrobić to, wywołując procedurę sp_adjustpublisheridentityrange.

Zarządzanie replikacjami

Implementacja wybranego typu i modelu replikacji wpływa na wykonywanie typowych zadań administracyjnych, takich jak zmiana definicji tabeli. Należy też pamiętać o wprowadzonych automatycznie przez SQL Server zmianach do definicji niektórych obiektów.

Replikowanie definicji tabel

Dodając do publikacji dane przechowywane w tabeli, SQL Server modyfikuje jej strukturę poprzez dodanie kolumn umożliwiających jednoznaczne określenie wiersza w skali wszystkich baz danych uruchomionych na serwerach biorących udział w replikacji.

O ile zmiana nazwy tabeli i poszczególnych kolumn oraz zmiana typu danych mogą zostać przeprowadzone bez żadnych dodatkowych ograniczeń, o tyle usunięcie lub dodanie kolumny należy wykonać, wywołując procedury systemowe sp_repladdcolumn lub sp_repldropcolumn, a nie wykonując instrukcję ALTER TABLE. Tak więc:

```
USE pubs
EXEC sp_repladdcolumn @source_object = 'dbo.authors'
,@column = 'sex'
,@typetext = 'char (1) NOT NULL DEFAULT "M"'
,@publication_to_add = 'all'
GO
```

będzie funkcjonalnym odpowiednikiem instrukcji:

```
USE pubs
ALTER TABLE dbo.authors
ADD sex char(1) NOT NULL DEFAULT 'M'
GO
```

Jeżeli próba wykonania procedury sp_repladdcolumn zakończyła się wyświetleniem komunikatu błędu o numerze 21 230, oznacza to, że wybrana tabela nie jest związana z jakąkolwiek publikacją.

Procedura sp_repldropcolumn umożliwia usunięcie kolumny, o ile nie przechowuje ona unikalnych identyfikatorów wiersza, do których zalicza się: klucz główny, kolumny o zawężeniu UNIQUE jednoznacznie identyfikujące wiersz w ramach publikacji oraz kolumny typu UNIQUEIDENTIFIER.

Replikowanie definicji widoków, funkcji i procedur

Replikując definicje obiektów bazodanowych, należy pamiętać, że:

1. Definicje obiektów nie powinny wchodzić w skład tej samej publikacji co dane.
2. Replikując widoki, procedury lub funkcje należy dołączyć do replikacji tabele, do których odwołują się te obiekty — SQL Server automatycznie dołączy do replikacji jedynie strukturę tabel źródłowych, bez przechowywanych w nich danych.
3. Uprawnienia użytkowników nie są replikowane wraz z obiektami — aby umożliwić użytkownikom korzystanie z replikowanych obiektów na wszystkich serwerach. należy nadać do nich odpowiednie uprawnienia.

Sprawdzanie replikacji danych

Podstawowym narzędziem do monitorowania replikacji jest konsola Enterprise Manager — dzięki monitorowi replikacji (ang. Replication Monitor) możemy poznać dane o bieżącej konfiguracji wszystkich agentów oraz historie wykonywanych przez nich zadań.
Jednak sprawdzić konfigurację replikacji oraz jej wynik, a więc poprawność danych wchodzących w skład publikacji i artykułów, możemy z poziomu języka Transact-SQL.

Procedura sp_helpsubscription

Zwraca informacje o subskrypcjach istniejących publikacji.
Składnia:

```
sp_helpsubscription [[@publication =] 'publikacja']
[,[@article =] 'artykuł']
[,[@subscriber =] 'subskrybent']
[,[@destination_db =] 'docelowa_baz_danych']
```

Przykład:

```
USE pubs
EXEC sp_helpsubscription
GO
```

Procedura sp_helparticle

Procedura zwraca szczegółowe informacje o określonym artykule.
Składnia:

```
sp_helparticle [@publication =] 'publikacja' [,[@article =] 'artykuł']
[,[@returnfilter =] filtr]
```

gdzie:
filtr określa, czy procedura ma zwrócić informacje o filtrowaniu poziomym danych przeprowadzonym za pomocą klauzuli FOR REPLICATION (domyślna wartość 1), czy dane te zostaną pominięte (wartość 0).
Przykład:

```
USE pubs
EXEC sp_helparticle @publication = 'pubs'
,@article = 'authors'
,@returnfilter = 0
GO
```

Procedura sp_helppublication

Procedura zwraca informacje o publikacji.
Składnia:

sp_helppublication [[@publication =] 'publikacja']

Przykład:

```
USE pubs
EXEC sp_helppublication @publication = 'pubs'
GO
```

Procedura sp_table_validation

Procedura zwraca liczbę wierszy oraz sumę kontrolną danych przechowywanych w tabeli lub widoku powiązanym z indeksem. Procedura może zostać wykorzystana w replikacji każdego typu.

Składnia:

```
sp_table_validation [@table =] 'tabela'
[, [@expected_rowcount =] spodziewana_liczba_wierszy]
[, [@expected_checksum =] spodziewana_suma_kontrolna]
[, [@rowcount_only =] tylko_zliczanie_wierszy]
[, [@owner =] 'właściciel']
[, [@full_or_fast =] metoda]
[, [@shutdown_agent =] agent]
[, [@table_name =] tabela_wyjściowa]
```

gdzie:

tylko_zliczanie_wierszy określa, czy ma zostać policzona jedynie liczba wierszy (domyślna wartość 1), czy również suma kontrolna danych (wartość 0),
metoda określa, czy liczba wierszy ma zostać zliczona na podstawie wyniku funkcji COUNT(*) (wartość 0), na podstawie zawartości tabeli sysindexes.rows (wartość 1). czy SQL Server ma najpierw spróbować oszacować liczbę wierszy na podstawie wpisów w tabeli systemowej i w razie niepowodzenia wykonać pełne skanowanie tabeli (domyślna wartość 2),
agent określa, czy agent dystrybucji ma zostać zatrzymany po zakończeniu sprawdzania (wartość 1), czy nie (domyślna wartość 0).

Przykład:

```
USE pubs
EXEC sp_table_validation @table = 'authors'
,@rowcount_only =0
GO
Conditional Fast Rowcount method requested without specifying an expected count.
[a]Fast method will be used.
Generated expected rowcount value of 23 and expected checksum value of 10644740958
[a]for authors.
```

Procedura sp_article_validation

Procedura sprawdza poprawność danych wchodzących w skład określonego artykułu. Procedura może być wykorzystywana w replikacjach transakcyjnych i migawkowych.

Składnia:

```
sp_article_validation [@publication =] 'publikacja' [, [@article =] 'artykuł']
[, [@rowcount_only =] tylko_zliczanie_wierszy]
[, [@full_or_fast =] metoda]
[, [@shutdown_agent =] agent]
```

Przykład:

```
USE pubs
EXEC sp_article_validation @publication = 'pubs'
,@article = 'authors'
GO
Server: Msg 20523, Level 16, State 1, Procedure sp_article_validation, Line 104
Could not validate the article 'authors'. It is not activated.
```

Procedura sp_publication_validation

Procedura sprawdza poprawność wszystkich artykułów wchodzących w skład publikacji. Procedura może być wykorzystywana w replikacjach transakcyjnych i migawkowych.
Składnia:

```
sp_publication_validation [@publication =] 'publikacja'
[, [@rowcount_only =] tylko_zliczanie_wierszy]
[, [@full_or_fast =] metoda]
[, [@shutdown_agent =] agent]
```

Przykład:

```
USE pubs
EXEC sp_publication_validation @publication = 'pubs'
GO
The command(s) completed successfully.
```