helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

SQL Server 2005 Express. Skuteczne rozwiązania

Autor: Danuta Mendrala, Marcin Szeliga
ISBN: 978-83-246-1131-7
Format: B5, stron: 432



Zbiór praktycznych porad dla programistów i administratorów baz danych

- Instalacja i konfiguracja SQL Server 2005 Express
- Zabezpieczanie serwera i danych
- Projektowanie baz i tabel
- Tworzenie optymalnych zapytań

MS SQL Server to baza danych od dawna kojarzona ze stabilnością, wydajnością i... wysoką ceną. Jednak najnowsza edycja tej bazy, oznaczona symbolem 2005, ma także swoją nieodpłatną wersję. SQL Server 2005 Express dystrybuowany jest za darmo, a co więcej – licencja pozwala na wykorzystywanie go do celów komercyjnych. Oczywiście ta wersja posiada wiele ograniczeń w stosunku do wersji płatnej, mimo to jednak doskonale spełnia zadania zaplecza bazodanowego dla aplikacji i portali internetowych, zapasowej bazy danych dla rozbudowanych systemów informatycznych i uniwersalnego narzędzia do nauki.

Książka „SQL Server 2005 Express" to zbiór porad dotyczących instalacji i konfiguracji tej aplikacji oraz sposobów jej wykorzystania. Autor książki – doświadczony administrator i programista baz danych – umieścił tu rozwiązania niemal wszystkich typowych zadań, jakie stają przed użytkownikami bazy SQL Server 2005. Z tej książki dowiesz się, jak optymalnie skonfigurować bazę danych, zabezpieczyć serwer czy zaprojektować struktury baz i tabel. Nauczysz się korzystać z języka SQL, formułować i optymalizować zapytania, tworzyć kopie zapasowe danych i zarządzać transakcjami.

- Instalacja i konfiguracja SQL Server 2005 Express
- Zarządzanie bazami danych
- Tworzenie kopii zapasowych i odtwarzanie danych
- Zabezpieczanie serwera
- Konta i role użytkowników
- Typy danych
- Projektowanie baz i tabel
- Generowanie indeksów
- Tworzenie i wykorzystywanie procedur składowanych
- Pobieranie i modyfikowanie danych
- Transakcje i blokady

Usprawnij swoją pracę, korzystając z doświadczenia profesjonalistów

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Wstęp 11
SQL Server 2005 Express 11
O czym jest ta książka? 12
Konwencje i oznaczenia 13

Część I Administracja 15

Rozdział 1. Instalacja 17
Przygotowanie instalacji 17
Wymagania 17
Ograniczenia 19
Licencje 19
Instalacja 20
Wybór konta usługi serwera SQL 2005 20
Instalacja serwera 21
Instalacja narzędzi administracyjnych 23
Instalacja pomocy BOL 23
Weryfikacja instalacji 24
Pliki dziennika 24
Usługi serwera 25
Połączenie z bazą master i sprawdzenie wersji serwera 25
SQLDIAG 25
Uwaga na kontrolę konta użytkownika systemu Windows Vista 26
Definiowanie aliasów po stronie komputerów klienckich 26
Instalacja przykładowej bazy AdventureWorks 27
Zmiana nazwy serwera SQL 27
Usuwanie instalacji serwera SQL 2005 28

Rozdział 2. Aktualizacja 29
MSDE a SQL 2005 Express 29
Workload Governor 30
Porównanie funkcjonalności obu wersji serwera 30
SharePoint 30
WSS 2.0 i WSDE 30
WSS 3.0 i Windows Internal Database 31
MOSS 2007 i SQL 2005 Express 31
Windows Server Update Services 3.0 32

Aktualizacja serwera ... 32
Scenariusze aktualizacji ... 33
Aktualizacja baz danych ... 34
Upgrade Advisor ... 35
Aktualizacja bazy danych ... 36
Zadania po aktualizacji ... 40
Migracja do edycji płatnych ... 42
Dodatkowe funkcje edycji Standard ... 43
Dodatkowe funkcje edycji Enterprise ... 45

Rozdział 3. Konfiguracja serwera i baz danych ... 49
Architektura serwera ... 49
Aparat relacyjny ... 50
Aparat składowania danych ... 51
SQLOS ... 51
Konfiguracja serwera ... 52
Protokoły sieciowe ... 52
Usługi ... 56
Wybrane opcje serwera ... 56
Bazy danych ... 58
Systemowe bazy danych ... 59
Pliki bazodanowe ... 61
Naprawa błędów ... 68
Zarządzanie bazami danych ... 70
Tworzenie baz danych ... 70
Obiekty systemowe ... 70
Wybrane opcje bazy danych ... 72
Modyfikowanie baz danych ... 73
Skrypty administracyjne ... 75
Usuwanie baz danych ... 77
XML dla administratorów ... 77
Pliki konfiguracyjne ... 77
Plany wykonania zapytań ... 78
Zapisane plany wykonania instrukcji ... 79
Statystyki wykonania instrukcji ... 79
Dyrektywy optymalizatora ... 79
Wyniki funkcji EventData() ... 79
Pliki formatu programu BCP ... 80
Best Practices Analyzer 2.0 ... 80

Rozdział 4. Kopie zapasowe ... 81
Tworzenie kopii zapasowych ... 81
Urządzenia kopii zapasowych ... 81
Typy kopii zapasowych ... 83
Strategie ... 85
Automatyczne tworzenie kopii zapasowych ... 88
Sprawdzanie aktualności kopii ... 90
Odtwarzanie kopii zapasowych ... 91
Spójność bazy ... 91
Odtwarzanie kopii danych ... 92
Odtwarzanie kopii dzienników transakcyjnych ... 93
Odtwarzanie pojedynczych stron ... 94
Weryfikacja kopii poprzez SMO ... 95
Zmiana nazwy i lokalizacji bazy ... 96
Odtwarzanie serwera ... 96

Rozdział 5. Bezpieczeństwo ... 99
Model dogłębnej obrony ... 99
Zagrożenia ... 99
Zabezpieczenia na poziomie systemu Windows ... 100
Lista usług ... 101
Usługi serwera SQL ... 101
Protokoły sieciowe serwera SQL ... 102
Microsoft Baseline Security Analyzer ... 102
Zabezpieczenia na poziomie instancji ... 103
Model bezpieczeństwa serwera SQL 2005 ... 103
Uwierzytelnianie ... 104
Role serwera ... 108
Dodatkowe poświadczenia ... 109
Delegowanie poświadczeń ... 109
Dodatkowe funkcje serwera ... 110
Zabezpieczenia na poziomie bazy danych ... 111
Zaufane bazy danych ... 111
Konta użytkowników ... 111
Role bazy danych ... 112
Schematy ... 115
Monitorowanie ... 116
Dzienniki systemu Windows ... 116
Dzienniki serwera SQL ... 116

Część II Programowanie ... 119

Rozdział 6. Narzędzia, tabele, zmienne i typy danych ... 121
Narzędzia ... 121
Diagramy baz danych ... 122
Szablony instrukcji ... 123
SQLCMD ... 124
BCP i BULK INSERT ... 125
Typy danych ... 125
Wartość NULL ... 126
Typy daty i czasu ... 127
Typy znakowe ... 129
Duże obiekty ... 129
Typy CLR ... 130
XML ... 130
Zmienne ... 132
Inicjowanie zmiennych ... 132
Konwersja typów ... 133
Wartość czy wyrażenie? ... 133
Zmienne tabelaryczne ... 133
Tabele ... 134
Tworzenie tabel ... 134
Ograniczenia ... 137
Modyfikowanie tabeli ... 139
Tabele tymczasowe ... 142
Instancje użytkowników ... 143
Podłączenie pliku bazy danych ... 144
SQL Server Express Utility ... 145

Rozdział 7. Indeksy ... 147
Podstawowe struktury danych ... 147
Sterta ... 148
Indeks zgrupowany ... 148
Indeks niezgrupowany ... 151
Mechanizmy odczytywania i modyfikowania danych ... 152
Odczytywanie wierszy ... 152
Wstawianie wierszy ... 156
Usuwanie wierszy ... 157
Modyfikowanie wierszy ... 157
Tworzenie indeksów ... 158
Opcje indeksów ... 158
Indeksy złożone ... 159
Dodatkowe kolumny indeksów ... 160
Indeksowanie kolumn wyliczeniowych ... 160
Wielkość indeksów ... 161
Statystyki ... 162
Wyłączenie aktualizacji statystyki ... 163
Zarządzanie indeksami ... 164

Rozdział 8. Widoki ... 165
Tworzenie widoków ... 165
Restrykcje ... 165
Opcje widoków ... 167
Indeksowanie widoków ... 170
Korzystanie z indeksów ... 171
Indeksowanie wybranych wierszy tabeli ... 173
Partycjonowanie danych ... 174
Modyfikowanie widoku ... 175
Sprawdzanie zależności między obiektami ... 175
Aktualizacja metadanych ... 176
Łańcuchy własności ... 177

Rozdział 9. Funkcje użytkownika ... 179
Funkcje skalarne ... 179
T-SQL ... 180
CLR ... 182
Proste funkcje tabelaryczne ... 186
Ograniczenie dostępu do danych ... 187
Skrypty administracyjne ... 187
Modułowość ... 187
Złożone funkcje tabelaryczne ... 188

Rozdział 10. Procedury i wyzwalacze ... 191
Procedury składowane ... 191
Tworzenie procedur ... 192
Przetwarzanie procedur przez serwer SQL 2005 ... 193
Interfejs procedur ... 195
Wywoływanie procedur ... 199
Zmiana kontekstu wykonania ... 205
Wyzwalacze ... 206
Wyzwalacze DML ... 207
Wyzwalacze DDL ... 211
Wyzwalacze logowania ... 213

Rozdział 11. Obsługa błędów ... 215
Komunikaty błędów ... 215
Kategorie błędów ... 216
Zasięg błędów ... 217
Niespójna obsługa błędów ... 218
Tworzenie własnych komunikatów błędów ... 219
Blok TRY … CATCH ... 220
Unikajmy błędów, zamiast je przechwytywać ... 220
Informacje o błędach ... 221
Błędy, których nie przechwycimy ... 221
Transakcje ... 222
Zgłaszanie błędów ... 225
Ponowne zgłoszenie błędu ... 226

Rozdział 12. Kryptografia ... 229
Algorytmy ... 229
Siła kryptografii ... 230
Algorytmy symetryczne ... 231
Algorytmy asymetryczne ... 234
Funkcje mieszania ... 235
Klucze ... 237
Główny klucz serwera ... 237
Główny klucz bazy danych ... 239
Klucze użytkowników ... 240
Podpisy cyfrowe ... 243
Sprawdzanie autentyczności danych ... 244
Podpisywanie modułów kodu ... 246
Szyfrowanie danych ... 248
Ograniczenie dostępu do klucza ... 249
Indeksowanie szyfrogramów ... 250

Część III Praca z danymi ... 253

Rozdział 13. Wykonywanie zapytań przez serwer SQL 2005 ... 255
Logiczny plan wykonania ... 256
Kolejność wykonywania klauzul instrukcji SELECT ... 256
Fizyczny plan wykonania ... 264
Optymalizator ... 264
Odczytywanie informacji o planach wykonań ... 266
Podstawowe operatory ... 269

Rozdział 14. Pobieranie danych ... 275
Automatyczne tworzenie zapytań ... 275
Graficzny konstruktor zapytań ... 275
Szablony zapytań ... 278
Wyszukiwanie danych ... 278
Klauzula WHERE ... 278
Wydajne wyszukiwanie danych ... 280
Łączenie źródeł danych ... 283
Złączenie krzyżowe ... 283
Złączenia naturalne ... 284
Złączenia zewnętrzne ... 285
Złączenia wielokrotne ... 285
Złączenia własne ... 287

Łączenie wyników zapytań ... 287
Operator APPLY ... 289
Porządkowanie zwracanych wierszy i ograniczanie ich liczby ... 289
Klauzula ORDER BY ... 289
Klauzula TOP ... 291
Klauzula TABLESAMPLE ... 292

Rozdział 15. Grupowanie danych ... 293
Funkcje grupujące ... 293
Klauzula GROUP BY ... 295
Operator CUBE ... 295
Operator ROLLUP ... 297
Funkcja GROUPING() ... 297
Klauzula HAVING ... 298
Klauzula OVER ... 298
Partycje ... 299
Funkcje rankingu ... 299
Operator PIVOT ... 303
Operator UNPIVOT ... 305

Rozdział 16. Podzapytania ... 307
Podzapytania jako zmienne ... 307
Podzapytania niepowiązane ... 308
Podzapytania powiązane ... 310
Predykat EXISTS ... 312
Podzapytania jako źródła danych ... 314
Tabele pochodne ... 314
Proste CTE ... 316
Rekurencyjne CTE ... 318

Rozdział 17. Modyfikowanie danych ... 323
Wstawianie danych ... 323
Wstawianie wyników zapytań ... 324
Wstawianie wyników procedur ... 326
Klauzula OUTPUT ... 326
Usuwanie danych ... 326
Instrukcja DELETE ... 327
Instrukcja TRUNCATE ... 330
Modyfikowanie danych ... 331
Klauzula OUTPUT ... 331
Eksport i import danych ... 332
Program BCP ... 332
Instrukcja BULK INSERT ... 332
Funkcja OPENROWSET() ... 333

Rozdział 18. Transakcje, blokady i wersjonowanie ... 335
Transakcje ... 336
Właściwości transakcji ... 336
Zagnieżdżanie transakcji ... 338
Blokady ... 339
Charakterystyka blokad ... 340
Jak serwer SQL 2005 zarządza blokadami? ... 344
Informacje o blokadach ... 345
Zakleszczenia ... 347
Modyfikowanie blokad zakładanych przez serwer ... 349

Poziomy izolowania transakcji ... 351
Read Uncommited ... 351
Read Commited ... 352
Repeatable Read ... 352
Serializable ... 353
Wersjonowanie wierszy ... 354
Read Commited Snapshot ... 354
Snapshot ... 355
Magazyn wersji ... 357

Część IV Optymalizacja ... 359

Rozdział 19. Mierzenie wydajności ... 361
Narzędzia ... 361
Monitor wydajności ... 361
Raporty konsoli SSMSE ... 364
Performance Dashboard Reports ... 366
Śledzenie aktywności użytkowników ... 368
Widoki dynamiczne ... 370
Linia bazowa ... 370
Próbkowanie danych zwracanych przez liczniki serwera SQL 2005 ... 371
Czyszczenie buforów i dokumentowanie wydajności zapytań ... 372
Wąskie gardła ... 373
Na co czeka serwer SQL 2005? ... 373
Procesor ... 376
Pamięć ... 378
Dysk ... 380
Sieć ... 382
Dziennik transakcyjny ... 382
Baza tempdb ... 383
Indeksy ... 384
Blokady i zatrzaski ... 387
Wolno wykonywane instrukcje ... 390

Rozdział 20. Poprawa wydajności ... 393
Optymalizacja baz danych ... 394
Struktura logiczna ... 394
Struktura fizyczna ... 397
Baza tempdb ... 399
Optymalizacja zapytań ... 400
Uwagi dotyczące optymalizacji zapytań ... 400
Refaktoryzacja kursorów ... 403
Indeksy ... 405
Użyteczne i nieużyteczne indeksy ... 406
Defragmentacja i przebudowa indeksów ... 407
Typowe błędy ... 408
Blokady ... 409
Rozbudowa serwera ... 410

Skorowidz ... 411

Rozdział 10.

Procedury i wyzwalacze

Procedury składowane i wyzwalacze to wykonujące określone zadania podprogramy składające się z dowolnych instrukcji T-SQL lub CLR. **W przeciwieństwie do funkcji użytkownika, procedury i wyzwalacze mogą modyfikować stan bazy danych**, dzięki czemu obiekty obu typów używane są do implementacji logiki aplikacyjnej po stronie bazy danych.

Procedury składowane

1. **Poprawiają bezpieczeństwo bazy danych**. Odbierając użytkownikom uprawnienia do tabel i zezwalając im na wywoływanie procedur, zyskujemy pewność, że nikt przypadkowo nie wykona błędnej instrukcji, np. instrukcji `UPDATE` bez klauzuli `WHERE`. **Procedury składowane są też jedyną skuteczną obroną przed atakami polegającymi na iniekcji kodu SQL**. Dodatkowo ukrywają one strukturę bazy danych przed użytkownikami.



> Iniekcja kodu SQL polega na podstawieniu pod zmienne dowolnych instrukcji języka SQL. Ponieważ SQL jest językiem interpretowanym, odczytująca wartość zmiennej instrukcja może zostać zmodyfikowana, a następnie wykonana. To zagrożenie można wyeliminować, przekazując wartości zmiennych jako parametry wywołania procedur. Drugie zagrożenie związane jest z wykonywaniem dynamicznie skonstruowanych instrukcji T-SQL. **Konstruowanie dynamicznych instrukcji w ciele procedury nie chroni przed iniekcją kodu**. Dlatego należy sprawdzać poprawność wszystkich zmiennych i uniemożliwiać ich błędną interpretację przez serwer SQL 2005. Jeżeli np. wartością zmiennej powinna być nazwa obiektu bazy danych, należy umieścić ją w wywołaniu funkcji `QUOTENAME()`.

2. **Tworzą, wraz z widokami i funkcjami użytkownika, warstwę abstrakcji oddzielającą użytkowników od tabel**. Jeżeli użytkownicy nie odwołują się bezpośrednio do tabel, zmiana ich struktury nie wymaga zmian aplikacji klienckich[1].

[1] Zmiana definicji procedur czy widoków, o ile tylko ich interfejs pozostanie taki sam, jest dla użytkowników niewidoczna.

3. **Zwiększają wydajność instrukcji**. Domyślnie, skompilowane plany wykonania procedur składowanych są buforowane i wykorzystywane do ich kolejnych wywołań. Ponadto wszystkie dane potrzebne do wykonania procedury użytkownik jednorazowo przesyła poprzez sieć w postaci nazwy procedury i parametrów jej wywołania.
4. **Zmniejszają zasięg ewentualnych błędów i umożliwiają ich spójną obsługę**. Wykorzystanie procedur do obsługi błędów zostało opisane w następnym rozdziale.

Tworzenie procedur

Procedury składowane:

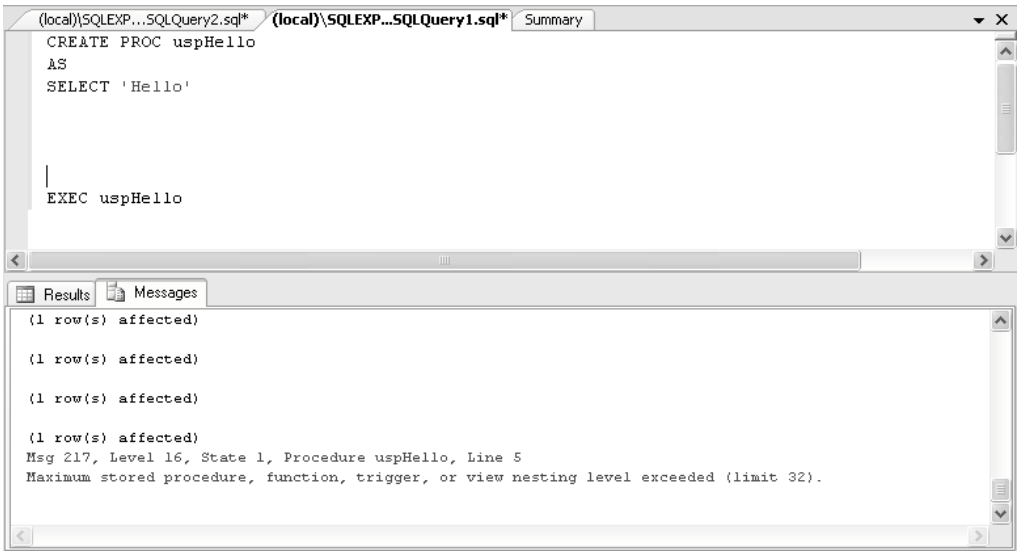
1. pozwalają na wywoływanie innych procedur lub rekurencyjne wywoływanie samej siebie; maksymalny poziom zagnieżdżenia procedur wynosi 32, po jego przekroczeniu serwer SQL 2005 zgłasza błąd i wycofuje wszystkie znajdujące się na stosie wywołań procedury,
2. akceptują do 2 100 parametrów wywołania; parametrami wywołania procedury nie mogą być dane tabelaryczne,
3. zwracają wartość statusu, domyślnie informującą o tym, czy procedura została wykonana bez błędów,
4. mogą zwracać wiele wartości w postaci parametrów wyjściowych (ang. *output*),
5. ich definicje mogą zostać zaszyfrowane za pomocą opcji `ENCRYPTION`,
6. umożliwiają wyłączenie buforowania planu ich wykonania za pomocą opcji `RECOMPILE` i zmiany kontekstu wykonania przy użyciu opcji `EXECUTE AS`. Te kwestie zostały opisane w dalszej części rozdziału.

Uwaga na dodatkowe instrukcje wsadu

Ciało procedury nie jest umieszczane w bloku `BEGIN ... END`. W rezultacie wszystkie instrukcje wsadu występujące po klauzuli `CREATE PROCEDURE ... AS` zostaną dodane do ciała procedury. Zwracamy na to uwagę, bo często podczas modyfikowania czy testowania procedur w tym samym oknie edytora umieszcza się inne instrukcje, np. wywołuje się testowaną procedurę (rysunek 10.1).



Rozwiązaniem tego problemu jest jawne kończenie ciała procedury dyrektywą `GO`. Najlepiej wyrobić sobie nawyk stawiania słowa `GO` zaraz po zadeklarowaniu nagłówka procedury, tak samo jak natychmiast po rozpoczęciu bloku `BEGIN` powinniśmy wpisywać kończącą go instrukcję `END`.



Rysunek 10.1. *Jeżeli spróbujemy wywołać procedurę, której wywołanie przypadkowo umieściliśmy w jej ciele, to próba zakończy się zgłoszeniem przez serwer błędu przekroczenia maksymalnego poziomu zagnieżdżenia procedur*

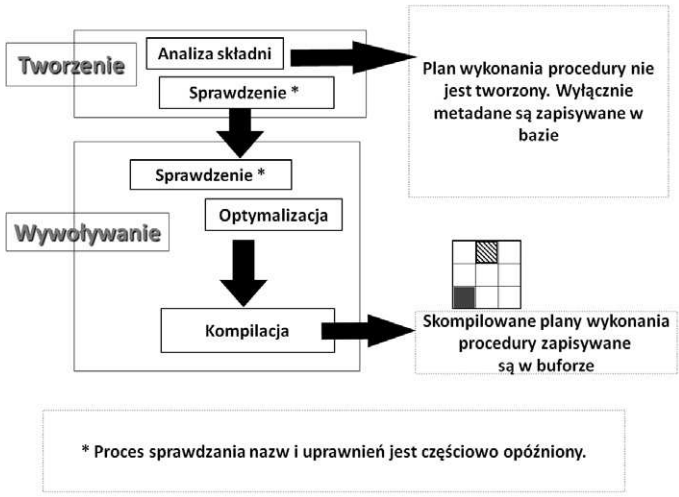
Przetwarzanie procedur przez serwer SQL 2005

Przetwarzanie procedury przebiega dwuetapowo.

1. Podczas tworzenia procedury sprawdzana jest poprawność syntaktyczna instrukcji CREATE PROCEDURE. Jeżeli instrukcja jest poprawna, procedura zostanie utworzona, a jej definicja i metadane zapisane w bazie danych.
2. Podczas wywoływania procedury wyszukiwane są obiekty, do których odwołuje się procedura. Jeżeli któryś z nich jest własnością innego użytkownika niż właściciel procedury, serwer SQL 2005 sprawdzi również, czy procedura ma nadane wystarczające uprawniania do wszystkich obiektów bazowych. Następnie dochodzi do optymalizacji procedury, w której wyniku najlepszy ze znalezionych plan jej wykonania jest kompilowany, wykonywany i zapisywany w buforze procedur (rysunek 10.2).



Jeżeli plan wykonania procedury znajduje się w buforze, serwer SQL 2005 będzie korzystał z niego przy kolejnych wywołaniach procedury, a więc będą one wymagały jedynie odczytania i wykonania wcześniej skompilowanego planu wykonania. Jeżeli jednak nazwa procedury musiała być sprawdzona, jej kolejne wywołania będą wymagały ponownego sprawdzenia, optymalizacji i kompilacji. Ponieważ serwer SQL 2005 zakłada, że procedury o nazwach zaczynających się od prefiksu sp_ znajdują się w bazie master, nie należy go używać w nazwach własnych procedur.



Rysunek 10.2. *Taki proces przetwarzania procedur powoduje, że ewentualne błędy, z wyjątkiem syntaktycznych, ujawnią się dopiero podczas pierwszego ich wywołania*

Opóźnione sprawdzanie nazw

Skoro podczas tworzenia procedury nie jest przygotowywany plan jej wykonania, możliwe jest utworzenie procedur odwołujących się do nieistniejących obiektów — tabel, widoków, funkcji czy innych procedur[2]:

```
USE AdventureWorks
GO
CREATE PROC uspTest
AS
SELECT kol1
FROM nieMaTakiejTabeli
------------------------------------------------------------
Command(s) completed successfully.
```

Oczywiście, próba wywołania procedury skończy się błędem:

```
EXEC uspTest
------------------------------------------------------------
Msg 208, Level 16, State 1, Procedure uspTest, Line 3
Invalid object name 'nieMaTakiejTabeli'.
```

W związku z tym, że serwer SQL 2005 sprawdza, czy obiekty, do których odwołuje się tworzona lub zmieniana procedura, istnieją, odwołania do nich muszą być poprawne:

```
ALTER PROC uspTest
AS
SELECT kol7
FROM tab1
------------------------------------------------------------
Msg 207, Level 16, State 1, Procedure uspTest, Line 3
Invalid column name 'kol7'.
```

[2] Jak pokazaliśmy we wcześniejszym przykładzie, możliwość odwołania się w ciele procedury do niej samej (a więc do nieistniejącej jeszcze procedury) pozwala na ograniczone stosowanie rekurencji.

Interfejs procedur

W nagłówku procedury należy określić nazwy i typy jej parametrów. Parametry procedury dzielą się na:

1. parametry wywołania, w ciele procedury będą one dostępne jako jej zmienne lokalne,
2. parametry wyjściowe; dodając po typie parametru słowo kluczowe `OUTPUT`, dodatkowo umożliwimy procedurze zwrócenie wartości takiego parametru.

W ramach procedur należy określić domyślną wartość parametrów i sprawdzić, czy podane przez użytkownika dane są prawidłowe. Poniższa procedura może być wywołana z jednym parametrem — nazwą kategorii. Jeżeli nie zostanie ona podana, procedura zwróci informacje o wszystkich produktach: nazwę kategorii, podkategorii i produktu uzupełnione o cenę i pozycję produktu w ramach podkategorii. Jeśli natomiast nazwa kategorii będzie zawierała niedozwolone znaki, działanie procedury zostanie przerwane[3]:

```
CREATE Procedure uspDaneProduktu (@Categoryname varchar(50) = '%')
AS
IF dbo.ufnLikeRegEx (@Categoryname,'^[a-zA-Z%\s\']*$') = 0
RETURN -1                                    -- nazwa nie składa się z samych liter i symbolu %
IF dbo.ufnLikeRegEx (@Categoryname,'/(\%27)|(\-\-)|(\%23)|(#)/ix') = 1
RETURN -1                                    -- nazwa zawiera zabronione znaki
SELECT c.Name AS Kategoria, sc.Name AS Podkategoria, p.Name Produkt, p.ListPrice,
DENSE_RANK() OVER                    -- funkcja DENSE_RANK() ponumeruje wiersze
(PARTITION BY sc.Name                -- w ramach poszczególnych podkategorii
ORDER BY p.ListPrice DESC) as Pozycja -- numerowane wiersze muszą być posortowane
FROM  Production.Product p
JOIN Production.ProductSubcategory sc ON p.ProductSubcategoryID =
sc.ProductSubcategoryID
JOIN Production.ProductCategory c ON sc.ProductCategoryID = c.ProductCategoryID
WHERE  c.Name LIKE @CategoryName
ORDER BY c.Name
RETURN @@ROWCOUNT -- zastępujemy systemowy status wykonania liczbą zwróconych wierszy
```

Parametry do procedury można przekazać na dwa sposoby.

1. Podając rozdzielone przecinkami wartości. W takim przypadku kolejność wartości musi odpowiadać kolejności parametrów, a dwa przecinki nie mogą wystąpić bezpośrednio obok siebie. Oznacza to, że nawet parametry, dla których zdefiniowano wartości domyślne, muszą być podane, chyba że występują na końcu listy parametrów.
2. Podając nazwę parametru i przypisując mu wartość.

W poniższym przykładzie pokazujemy, jak wywołać utworzoną w poprzednim punkcie procedurę bez parametrów i z błędnym parametrem:

```
EXEC uspDaneProduktu
EXEC uspDaneProduktu 'a''OR 1=1--'
```

[3] Funkcja `ufnLikeRegEx` została utworzona w poprzednim rozdziale.

oraz z podaniem nazwy kategorii i sprawdzeniem zwróconego przez procedurę statusu:

```
DECLARE @l INT
EXEC @l = uspDaneProduktu 'Accessories'
PRINT @l
```

Ponieważ procedury mogą modyfikować stan bazy, często są używane do zautomatyzowania zadań administracyjnych, np. nadawania lub odbierania uprawnień. W poniższym przykładzie pokazujemy rozbudowaną procedurę administracyjną, której działanie polega na nadawaniu wskazanemu użytkownikowi uprawnień do wywoływania wszystkich procedur bazy danych:

```
CREATE PROCEDURE uspGrantExec @user SYSNAME
AS
DECLARE @cSQL varchar(8000)
DECLARE @Obj varchar(8000)
DECLARE @Cur  CURSOR
BEGIN TRY                          -- blok TRY... CATCH opisaliśmy w rozdziale 11.
SET @Cur = CURSOR FOR SELECT QUOTENAME(routine_schema) + '.' +
QUOTENAME(routine_name)
FROM information_schema.routines
WHERE routine_name NOT LIKE 'dt_%' AND routine_type = 'PROCEDURE'
OPEN @Cur
FETCH NEXT FROM @Cur INTO @Obj
WHILE @@FETCH_STATUS = 0
BEGIN
  SET @cSQL = 'GRANT EXEC ON ' + @Obj + ' TO ' + @user
  EXEC (@cSQL)
  FETCH NEXT FROM @Cur INTO @Obj
END
DEALLOCATE @Cur
END TRY
BEGIN CATCH
RETURN -1                          -- w rozdziale 11. wyjaśniliśmy też problemy związane z taką
                                   -- obsługą wyjątków
END CATCH
```

Żeby zezwolić użytkownikowi na wywoływanie wszystkich procedur, wystarczy wykonać poniższy skrypt:

```
DECLARE @w INT
EXEC @w = uspGrantExec 'blabla'
IF @w <0  PRINT 'Błąd'
-------------------------------------------------------------
Błąd

DECLARE @w INT
EXEC @w = uspGrantExec 'Danka'
IF @w <0  PRINT 'Błąd'
-------------------------------------------------------------
Command(s) completed successfully.
```

Wstawanie wyniku procedury do tabeli

Procedura może być wywołana w ramach instrukcji `INSERT INTO`. Takie wywołanie procedury spowoduje wstawienie zwracanych przez nią wyników do tabeli. Liczba i typ kolumn tabeli muszą odpowiadać strukturze wyniku działania procedury:

```
CREATE PROCEDURE uspPrac @Nazwisko nvarchar(50) = '%', @Imie nvarchar(50) = '%'
AS
SELECT FirstName, LastName, JobTitle
FROM HumanResources.vEmployeeDepartment
WHERE FirstName LIKE @Imie AND LastName LIKE @Nazwisko
GO
CREATE TABLE #Pracownicy
(Imie nvarchar(50),
Nazwisko nvarchar(50),
Stanowisko nvarchar(50))
GO
INSERT INTO #Pracownicy
EXEC uspPrac 'H%', 'D%'

SELECT *
FROM #Pracownicy
------------------------------------------------------------
Doris    Hartwig        Production Technician - WC10
Douglas  Hite           Production Technician - WC45
David    Hamilton       Production Supervisor - WC40
Don      Hall           Production Technician - WC50
```

Gdybyśmy w ramach procedury utworzyli tabelę tymczasową, zostałaby automatycznie zniszczona po zakończeniu jej działania, a więc wywołanie procedury powiodłoby się, ale próba odczytania wyników skończyłaby się błędem:

```
ALTER PROCEDURE uspPrac @Nazwisko nvarchar(50) = '%', @Imie nvarchar(50) = '%'
AS
CREATE TABLE #tmpPracownicy
(Imie nvarchar(50),
Nazwisko nvarchar(50),
Stanowisko nvarchar(50))

INSERT INTO #tmpPracownicy
SELECT FirstName, LastName, JobTitle
FROM HumanResources.vEmployeeDepartment
WHERE FirstName LIKE @Imie AND LastName LIKE @Nazwisko
GO
EXEC uspPrac 'H%', 'D%'
SELECT *
FROM #tmpPracownicy
------------------------------------------------------------
(4 row(s) affected)
Msg 208, Level 16, State 0, Line 1
Invalid object name '#tmpPracownicy'.
```

Parametry wyjściowe

Parametry wyjściowe w rzeczywistości działają jak znane z obiektowych języków programowania parametry przekazywane przez referencje. Możemy za ich pomocą przekazać wartość do procedury, która dowolnie ją zmodyfikuje i odeśle wynik do programu klienckiego:

```
CREATE PROCEDURE uspParamOut @l1 INT, @l2 INT OUTPUT
AS
SET @l2= @l1*@l2
```

```
GO

DECLARE @w INT
SET @w=10
EXEC uspParamOut 5,@w OUTPUT
SELECT @w
------------------------------------------------------------
50
```

Parametry tabelaryczne

Możliwość przekazywania jako parametru wywołania procedur tabel, pozwoliłaby m.in. na tworzenie procedur wywoływanych z dowolną liczbą parametrów (jednym parametrem byłoby jedno pole tabeli) lub na wykonanie dla każdego wiersza tabeli w ramach procedury określonej operacji. Niestety, parametrem procedury nie mogą być dane typu `TABLE`.

W zamian za to serwer SQL 2005 obsługuje standard XML, pozwalający na wywoływanie procedur z parametrami typu XML. Skoro dokument XML może zawierać wiele (do 2 GB) danych, rozwiązanie problemu sprowadza się do przekazania procedurze dokumentu XML, który następnie w ramach procedury będzie przeanalizowany, a odczytane z niego dane użyte w założonym przez nas celu:

```
CREATE PROCEDURE uspParamTab @doc XML
AS
--możemy skorzystać z funkcji OPENXML
DECLARE @h INT
EXEC sp_xml_preparedocument @h OUTPUT, @doc
SELECT *
FROM OPENXML(@h, 'Towary/Towar')
WITH (id INT '@id',        -- klauzula WITH pozwala sformatować wynik
  nazwa VARCHAR(30) '@nazwa',
  cena DECIMAL '@cena')
EXEC sp_xml_removedocument @h
--albo z metod typu XML
SELECT @doc.value ('(Towary/Towar/@nazwa)[1]','VARCHAR(30)')
```

Przykładowa procedura jest bardzo prosta, a jej działanie sprowadza się do odczytania przekazanego jako parametr wywołania dokumentu XML:

```
DECLARE @Towary XML
SET @Towary =' <Towary>
 <Towar id="1" nazwa="Syrop" cena="1000" />
 <Towar id="4" nazwa="Zioła" cena="200" />
 </Towary>'

EXEC uspParamTab @Towary
------------------------------------------------------------
1        Syrop          1000
4        Zioła          200
Syrop
```

Wywoływanie procedur

Procedury należy wywoływać za pomocą dyrektywy `EXEC`. W przeciwnym razie, jeżeli nazwa procedury nie będzie pierwszym wyrażeniem wsadu, serwer SQL 2005 nie będzie mógł poprawnie jej zinterpretować[4]:

```
sp_who
sp_who2
------------------------------------------------------------
Msg 15007, Level 16, State 1, Procedure sp_who, Line 59
'sp_who2' is not a valid login or you do not have permission.
```

Automatyczne wywołanie procedury przy uruchomieniu serwera SQL 2005

Podczas uruchamiania serwer SQL 2005 sprawdza, które ze znajdujących się w bazie master procedur składowanych mają być automatycznie wywołane. Procedurę jako startową możemy oznaczyć za pomocą procedury systemowej `sp_procoption`, ustawiając wartość parametru `startup` na `True`[5].

Plany wykonania procedur

Optymalizacja i kompilacja są procesami silnie obciążającymi procesor. Żeby zmniejszyć to obciążenie, serwer SQL 2005 automatycznie buforuje i wielokrotnie wykorzystuje raz obliczone optymalne plany wykonań procedur. Ten sam plan wykonania procedury jest ponownie używany, chyba że:

1. serwer SQL 2005 zostanie zatrzymany i ponownie uruchomiony,
2. bufor procedur zostanie wyczyszczony; taką operację można przeprowadzić:
   a) na poziomie serwera, wykonując instrukcję `DBCC FREEPROCCACHE`;
   b) na poziomie wybranej bazy danych, wykonując instrukcję `DBCC FLUSHPROCINDB()`;
3. plan zostanie usunięty z bufora z powodu długiego okresu jego nieużywania albo z powodu braku dostępnej pamięci RAM,
4. plan zostanie uznany za przestarzały z powodu zmiany struktury obiektów bazowych procedury lub z powodu odświeżenia statystyk opisujących przechowywane w indeksach tych tabel dane.

Żeby przekonać się o tym, jak działa mechanizm buforowania planów wykonania procedur, utworzymy procedurę, która zwraca informacje o zamówieniach z podanego przedziału czasu:

---

[4] W języku T-SQL znak końca wiersza jest ignorowany. Nawet umieszczenie po nazwach obu procedur średnika nie umożliwi ich poprawnej interpretacji.

[5] Przykład wykorzystania procedury startowej do utworzenia tabeli przechowującej stan serwera znajduje się w rozdziale 6.

```
CREATE PROC uspZamowienia @dataOd DATETIME = '20010101', @dataDo DATETIME = '20020101'
AS
SELECT SalesOrderID, OrderDate, Status
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @dataOd AND @dataDo
```

Następnie wyczyścimy bufor procedur bazy AdventureWorks:

```
DECLARE @dbid SMALLINT
SELECT @dbid = DB_ID('AdventureWorks')
DBCC FLUSHPROCINDB(@dbid)
```

I wywołamy naszą procedurę:

```
EXEC uspZamowienia '20010702','20010702'
```

Jeżeli teraz odczytamy dane o zbuforowanych planach wykonania naszej procedury, dowiemy się, że jest jeden plan jej wykonania i plan ten był raz użyty:

```
SELECT st.text, qs.execution_count, p.query_plan
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) p
WHERE st.text like '%uspZamowienia%' AND st.text NOT LIKE '%cache%'
```

Ostatnia kolumna wyniku zawiera plan, według którego serwer SQL 2005 wykonał procedurę w postaci dokumentu XML. Po dwukrotnym kliknięciu znajdującego się w niej odnośnika plan wykonania zostanie wyświetlony w oknie edytora XML. Ostatnią sekcją tego dokumentu będzie sekcja `<ParameterList>`, w której znajdziemy wartości parametrów wywołania procedury użyte do zoptymalizowania i skompilowania planu:

```
<ParameterList>
              <ColumnReference Column="@dataDo" ParameterCompiledValue=
               "'2001-07-02 00:00:00.000'" />
              <ColumnReference Column="@dataOd" ParameterCompiledValue=
               "'2001-07-02 00:00:00.000'" />
            </ParameterList>
```

Jeżeli ponownie wywołamy naszą procedurę, podając inne wartości parametrów jej wywołania:

```
EXEC uspZamowienia
GO
EXEC uspZamowienia '20010701','20010703'
```

a następnie raz jeszcze odczytamy informacje o zbuforowanych planach wykonania naszej procedury, przekonamy się, że **w buforze nadal znajduje się jeden, trzykrotnie użyty, plan jej wykonania**. Jeżeli popatrzymy na wartości skompilowanych parametrów, okaże się, że nie zmieniły się one i nadal są takie, jak podczas pierwszego wywołania procedury.

Problem pierwszego wywołania procedury

Serwer SQL 2005 tworzy plan wykonania procedury podczas jej pierwszego wywołania, używając do tego celu przekazanych w podczas tego wywołania wartości parametrów. Przekonaliśmy się też, że raz zbuforowany plan wykonania procedury

będzie używany do jej kolejnych wywołań, o ile nie zajdzie jedna z czterech wcześniej opisanych sytuacji. Dzięki czemu ponowne wywoływanie procedur nie wymaga ich optymalizacji i kompilacji. Jednakże oznacza to, że **kolejne wywołania tej samej procedury z różnymi parametrami zostaną przeprowadzone według tego samego planu**.

Jaki może to mieć wpływ na wydajność procedury, przekonamy się, analizując graficzny plan wykonania naszej procedury pod kątem użycia indeksów do zmniejszenia liczby odczytanych stron. Najpierw należy utworzyć indeks, na którego podstawie nasza procedura będzie mogła wybierać zamówienia:

```
CREATE INDEX IX_SalesOrderHeader_OrderDate
ON Sales.SalesOrderHeader(OrderDate)
```

Ponieważ dodanie indeksu (inaczej niż jego usunięcie) nie zawsze powoduje automatyczne przedawnienie planów wykonań, należy samodzielnie usunąć zbuforowane plany wykonań:

```
DECLARE @dbid SMALLINT
SELECT @dbid = DB_ID('AdventureWorks')
DBCC FLUSHPROCINDB(@dbid)
```

Po włączeniu statystyk wejścia-wyjścia (włączeniu opcji sesji `STATISTICS IO`) i wyświetleniu planów wykonań zapytań (przycisk *Include Actual Execution Plan*) raz jeszcze wywołamy naszą procedurę:

```
EXEC uspZamowienia '20010702','20010702'
------------------------------------------------------------
(4 row(s) affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 14, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

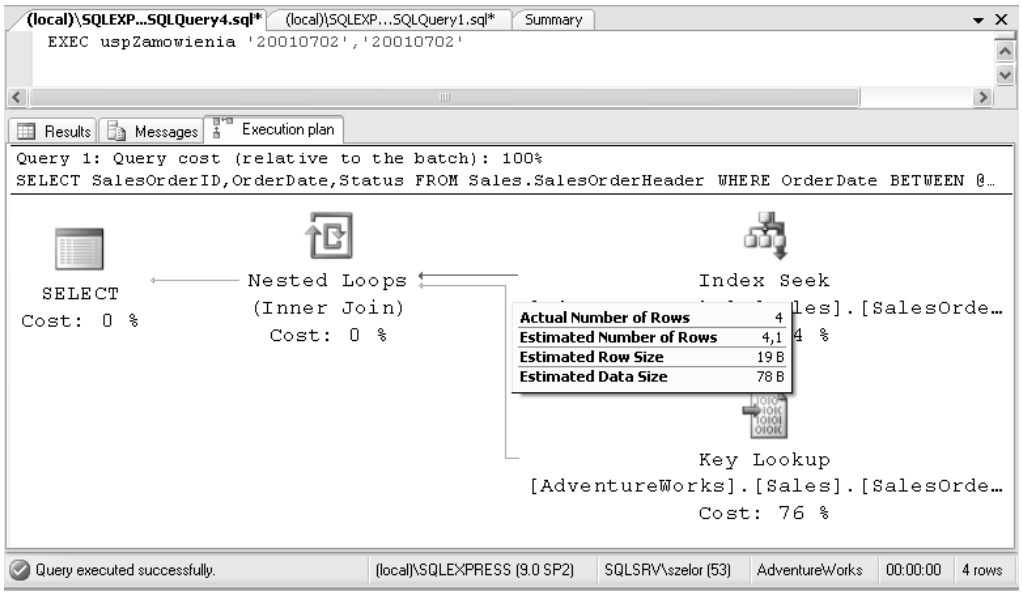
Do jej wykonania wystarczyło odczytanie 14 stron (ang. *logical reads 14*). Plan wykonania procedury został pokazany na rysunku 10.3.

Ponownie wywołamy naszą procedurę, ale tym razem podamy szeroki zakres dat zamówień. Na rysunku 10.4 możemy zobaczyć, że procedura nadal jest wykonywana według tego samego planu.

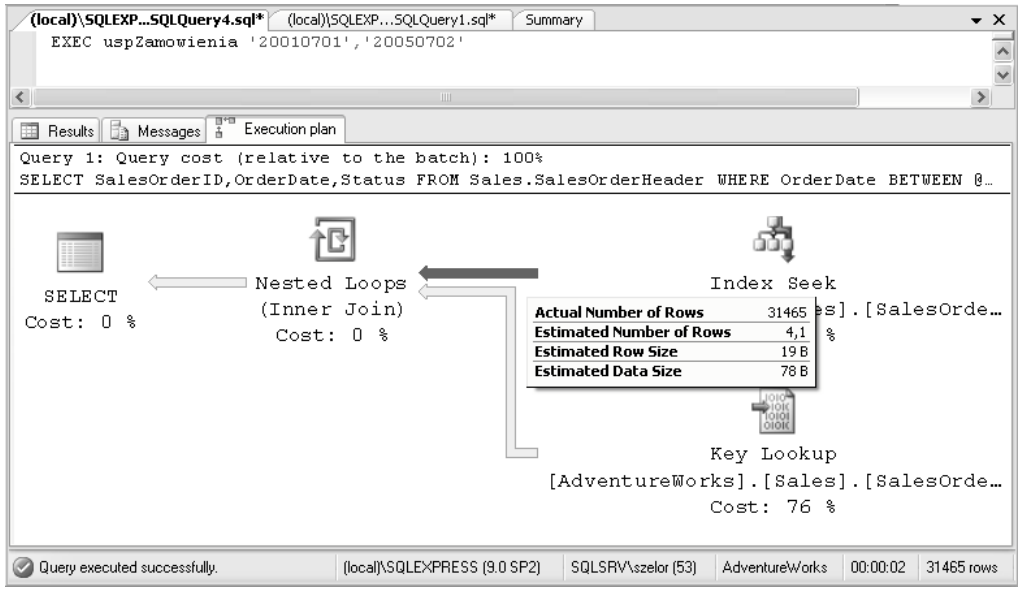
Jak można się było spodziewać, w tym przypadku wykonanie procedury według tego samego planu wymagało odczytania ogromnej liczby stron. Serwer SQL 2005 **odczytał 94 456 stron, chociaż cała tabela Sales.SalesOrderHeader mieści się na 703 stronach**[6].

W tej sytuacji ponowna optymalizacja, kompilacja i wykonanie procedury według nowego planu byłyby mniej kosztowne niż jej wykonanie według kompletnie nieefektywnego planu. Kolejne podpunkty opisują różne rozwiązania tego problemu — wybór właściwego zależy od konkretnej sytuacji.

[6] Opis wewnętrznych mechanizmów odczytywania danych z indeksów oraz zapytanie zwracające liczbę stron tabeli znajdują się w rozdziale 6.



Rysunek 10.3. *Ponieważ w tym dniu zrealizowane były tylko cztery zamówienia, serwer SQL 2005 zoptymalizował wykonanie procedury poprzez wyszukanie właściwych zamówień w indeksie niezgrupowanym i odczytanie odpowiednich stron z indeksu zgrupowanego*



Rysunek 10.4. *Chociaż selektywność takiego wywołania procedury jest znacznie niższa (tym razem zapytanie zwraca 31 465, a nie 4 wiersze, jak założył serwer SQL), zostało ono wykonane w ten sam sposób — poprzez przeszukanie indeksu niezgrupowanego i odczytanie wszystkich spełniających podany warunek stron z indeksu zgrupowanego*

Dyrektywa OPTIMIZE FOR

Jeżeli procedura jest często wywoływana z tymi samymi, typowymi dla niej wartościami parametrów, należy poinformować serwer SQL 2005, że plan jej wykonania ma być zoptymalizowany właśnie dla tych wartości[7]. Umożliwia to dyrektywa optymalizatora OPTIMIZE FOR:

```
ALTER PROC uspZamowienia @dataOd DATETIME = '20010101', @dataDo DATETIME = '20020101'
AS
SELECT SalesOrderID,OrderDate,Status
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @dataOd AND @dataDo
OPTION (OPTIMIZE FOR (@dataOd='20010701', @dataDo='20010902'))
```

Od teraz każde wywołanie procedury spowoduje jej wykonanie według tego samego, optymalnego dla wskazanych wartości parametrów planu. Jeżeli odczytamy zbuforowany plan wykonania, przekonamy się, że przekazane wartości parametrów zostały zastąpione wskazanymi w dyrektywie:

```
EXEC uspZamowienia
GO
SELECT p.query_plan.query
('declare default element namespace
"http://schemas.microsoft.com/sqlserver/2004/07/showplan";
/ShowPlanXML/BatchSequence/Batch/Statements/StmtSimple/QueryPlan/ParameterList')
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) p
WHERE st.text like '%uspZamowienia%' AND st.text NOT LIKE '%cache%'
AND st.text NOT LIKE 'SELECT%'
------------------------------------------------------------
<ParameterList xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan">
  <ColumnReference Column="@dataDo" ParameterCompiledValue=
   "'2001-09-02 00:00:00.000'" />
  <ColumnReference Column="@dataOd" ParameterCompiledValue=
   "'2001-07-01 00:00:00.000'" />
</ParameterList>
```

Użycie zmiennych lokalnych w ciele procedury

Drugie rozwiązanie problemu ustalenia planu wykonania procedury podczas jej pierwszego wywołania polega na zadeklarowaniu w ramach procedury zmiennych, przypisaniu im wartości parametrów wywołania procedury i użyciu tych zmiennych w ciele procedury. Ponieważ optymalizator nie będzie znał wartości tych zmiennych, wybierze optymalny plan wykonania, zakładając, że selektywność zapytań, w których zostały one użyte, wynosi 30%. **Przyjęcie tak niskiej selektywności w praktyce oznacza, że serwer SQL 2005 nie skorzysta z indeksów niezgrupowanych**.

Możemy się o tym przekonać, modyfikując naszą procedurę:

```
ALTER PROC uspZamowienia @dataOd DATETIME = '20010101', @dataDo DATETIME = '20020101'
AS
DECLARE @doD DATETIME, @dDo DATETIME
```

[7] Ważna jest typowa selektywność, a nie faktyczne wartości wzorcowych danych.

```
SET @doD=@dataOd
SET @dDo=@dataDo
SELECT SalesOrderID,OrderDate,Status
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @dOD AND @dDo
```

A teraz wywołamy ją z parametrami, o których wiemy, że optymalny plan ich wykonania polega na użyciu indeksu niezgrupowanego założonego na kolumnie `OrderDate`:

```
EXEC uspZamowienia '20010702','20010702'
------------------------------------------------------------
(4 row(s) affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 703, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Liczba odczytanych stron (703) świadczy o tym, że zapytanie zostało wykonane poprzez odczytanie całej tabeli `Sales.SalesOrderHeader`.

Rekompilacja procedury

Jeżeli mamy możliwość zmiany instrukcji po stronie aplikacji klienckiej, możemy poinformować serwer SQL 2005, że wywołujemy procedurę z nietypowymi parametrami. Użycie dyrektywy `RECOMPLIE` spowoduje, że:

1. zbuforowany plan wykonania procedury zostanie uznany za przedawniony,
2. do zoptymalizowania bieżącego wywołania procedury zostaną użyte aktualne wartości parametrów; ten plan nie zostanie zbuforowany,
3. kolejne wywołanie procedury spowoduje optymalizację, kompilację i zapisanie w buforze nowego planu wykonania procedury.

Żeby się o tym przekonać, musimy przywrócić oryginalną postać procedury:

```
ALTER PROC uspZamowienia @dataOd DATETIME = '20010101', @dataDo DATETIME = '20020101'
AS
SELECT SalesOrderID,OrderDate,Status
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @dataOd AND @dataDo
```

A następnie wywołać procedurę — najpierw z wysoce selektywnymi wartościami parametrów, a następnie bez parametrów, ale z dyrektywą `RECOMPLIE`:

```
EXEC uspZamowienia '20010702','20010702'
------------------------------------------------------------
(4 row(s) affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 14, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

EXEC uspZamowienia
WITH RECOMPILE
------------------------------------------------------------
(1424 row(s) affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 703, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Wyłączenie buforowania planu wykonań (części) procedury

Serwer SQL 2005, w przeciwieństwie do wcześniejszych wersji, rekompiluje, jeżeli zachodzi taka potrzeba, plany wykonania poszczególnych instrukcji tworzących ciało procedury, a nie plan wykonania całej procedury. W procedurach składających się z jednej instrukcji, takich jak procedura `uspZamowienia`, różnica jest nieistotna, ale dla rozbudowanych procedur jest już zauważalna.

Ta cecha serwera SQL 2005 pozwala użyć w ciele procedury dyrektywy `RECOMPILE` i oznaczyć w ten sposób te instrukcje, których plany wykonania w ogóle nie mają być buforowane:

```
ALTER PROC uspZamowienia @dataOd DATETIME = '20010101', @dataDo DATETIME = '20020101'
AS
SELECT SalesOrderID,OrderDate,Status
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @dataOd AND @dataDo
OPTION (RECOMPILE)
```

W ten sposób każde wywołanie procedury, nawet z tymi samymi parametrami, spowoduje jej ponowną optymalizację i rekompilację, co wiąże się ze zwiększeniem obciążenia procesora. W efekcie plan wykonania będzie dostosowany do aktualnej wartości parametrów wywołania procedury.

Oznaczenie planów wykonań jako przedawnionych

Procedura systemów `sp_recompile` pozwala oznaczyć jako przedawniony plan wykonania procedury lub wszystkich procedur związanych z tabelą lub widokiem o podanej nazwie. Dzięki czemu pierwsze wywołanie procedury spowoduje optymalizację i kompilację planu jej wykonania (wywołanie procedury `sp_recompile` nie oznacza natychmiastowej rekompilacji).

Defragmentacja indeksów, inaczej niż ich przebudowa, nie powoduje przedawnienia planów wykonań instrukcji odwołujących się do tych indeksów. Dodając wywołanie tej procedury do skryptów administracyjnych zawierających instrukcję `ALTER INDEX … REORGANIZE`, zagwarantujemy uaktualnienie planów wykonania procedur:

```
EXEC sp_recompile 'Sales.SalesOrderHeader'
```

Zmiana kontekstu wykonania

Serwer SQL 2005 pozwala na wykonanie procedury[8] w kontekście konta użytkownika innego niż ten, który ją wywołał. Umożliwia to klauzula `EXECUTE AS` z jedną z czterech opcji.

1. Domyślna opcja `CALLER` oznacza, że procedura zostanie wykonana z uprawnieniami użytkownika, który ją wywołał.
2. Opcja `SELF` spowoduje wykonanie procedury w kontekście konta jej twórcy albo użytkownika, który tę procedurę jako ostatni zmodyfikował.

[8] Oraz funkcji i wsadu.

3. Opcja OWNER oznacza, że procedura zostanie wykonana w kontekście jej właściciela.
4. Możliwe jest również podanie nazwy konta użytkownika, w którego kontekście procedura ma być wykonana.


Wskazówka

Zmiana kontekstu wykonania procedury pozwala na odebranie użytkownikom dodatkowych uprawnień nawet wtedy, kiedy muszą oni wywoływać procedury, które wykonują instrukcje DDL (takie jak ALTER czy DROP), DCL (takie jak GRANT czy DENY) czy zawierają dynamicznie konstruowany kod języka T-SQL. Jednak **ten sam efekt można uzyskać w bezpieczniejszy**, opisany w rozdziale 12., **sposób — poprzez podpisanie procedury**.

Poniższa procedura obcina jedną z utworzonych wcześniej tabel (co wymaga uprawnienia ALTER TABLE) i wyświetla informacje o identyfikatorach użytkowników i ról, których tożsamościami może się posłużyć:

```
CREATE PROC uspExecAs
WITH EXECUTE AS OWNER
AS
SELECT principal_id, name, type
FROM sys.user_token
TRUNCATE TABLE dbo.tab2
```

Jeżeli teraz nadamy uprawnienia do wywoływania tej procedury standardowemu użytkownikowi, to, mimo że bezpośrednio nie może on obciąć tabeli tab2:

```
GRANT EXEC ON uspExecAs TO Danka
GO
EXECUTE AS LOGIN = 'Danka'
TRUNCATE TABLE dbo.tab2
------------------------------------------------------------
Msg 1088, Level 16, State 7, Line 1
Cannot find the object "tab2" because it does not exist or you do not have
permissions.
```

Próba wykonania przez tego użytkownika procedury skończy się powodzeniem — teraz instrukcja TRUNCATE TABLE zostanie wykonana w kontekście konta dbo:

```
EXECUTE AS LOGIN = 'Danka'
EXEC uspExecAs
------------------------------------------------------------
1        dbo     SQL USER
0        public  ROLE
(2 row(s) affected)
```

Wyzwalacze

W przeciwieństwie do procedur składowanych, wyzwalacze:

1. nie mogą być bezpośrednio wywoływane,
2. nie mają interfejsu; wyzwalacze nie mogą być wywoływane z parametrami i nie powinny zwracać do użytkownika żadnych danych; po włączeniu opcji

serwera `disallow results from triggers` próba zwrócenia poprzez wyzwalacz danych skończy się błędem,

3. zawsze są synchronicznie wykonywane w ramach transakcji użytkownika; jeżeli nawet użytkownik nie rozpoczął jawnie transakcji, kod wyzwalacza będzie wykonany w ramach transakcji niejawnie rozpoczętej przez niego, co oznacza, że wyzwalacz może wycofać instrukcje użytkownika oraz do momentu zakończenia jego działania serwer SQL 2005 utrzyma blokady założone przez oryginalną instrukcję,
4. mogą być wyłączane instrukcją `DISABLE TRIGGER` i włączane instrukcją `ENABLE TRIGGER`,
5. są wywoływane w nieokreślonej kolejności. Procedura `sp_settriggerorder` pozwala jedynie wskazać nazwę wyzwalacza wywoływanego jako pierwszego (ang. *first*) i jako ostatniego (ang. *last*) z listy wyzwalaczy uruchamianych za pomocą tej samej instrukcji.


Wskazówka

Wycofywanie transakcji jest znacznie kosztowniejsze niż ich zatwierdzanie. Jeżeli taką samą funkcjonalność można uzyskać innymi metodami (np. spójność danych można zagwarantować za pomocą ograniczeń), należy zrezygnować z wyzwalaczy. W pozostałych przypadkach decyzję o ewentualnym wycofaniu transakcji należy podjąć jak najszybciej.

Wyzwalacze DML

Wyzwalacze DML wywoływane są w momencie wstawiania, usuwania lub modyfikowania danych w powiązanych z nimi tabelach lub widokach. **Ich wywołanie ma miejsce tylko raz dla instrukcji użytkownika, a nie dla każdego wiersza zmodyfikowanego w ramach tej instrukcji**.

Wirtualne tabele INSERTED i DELETED

Wyzwalacze DML mają dostęp do specjalnych tabel `INSERTED` i `DELETED`.

1. Wyzwalacze wywoływane instrukcją `DELETE` mają dostęp do zawierającej wcześniejszą wersję wierszy tabeli `DELETED`.
2. Wyzwalacze wywoływane instrukcją `INSERT` mają dostęp do zawierającej proponowaną wersję wierszy tabeli `INSERTED`.
3. Wyzwalacze wywoływane instrukcją `UPDATE` mają dostęp do obu tabel[9].


Wskazówka

Gdy chcemy sprawdzić, jaka instrukcja spowodowała wywołanie wyzwalacza, możemy policzyć wiersze wirtualnych tabel `INSERTED` i `DELETED`.

[9] W rozdziale 7. wyjaśniliśmy wewnętrzny mechanizm modyfikowania danych, m.in. sytuacje, w których instrukcja `UPDATE` jest zastępowana instrukcjami `DELETE` i `INSERT`.

Struktura obu tych tabel jest identyczna i odpowiada strukturze tabeli, z którą wyzwalacz został powiązany. Wirtualne tabele są przechowywane wyłącznie w pamięci operacyjnej, a ich zawartość może być jedynie odczytywana.

Poniższy przykład służy tylko do zademonstrowania wirtualnych tabel INSERTED i DELETED — w rzeczywistości użytkownicy nie spodziewają się, że po wykonaniu instrukcji UPDATE zostaną im zwrócone tego typu dane:

```
CREATE TRIGGER trWirtualneTabele
ON HumanResources.Department
AFTER UPDATE
AS
SELECT @@ROWCOUNT
SELECT *
FROM DELETED
SELECT *
FROM INSERTED
GO
UPDATE HumanResources.Department
SET Name=UPPER(Name)
WHERE DepartmentID<3
------------------------------------------------------------
2

2 Tool design   Research and Development      1998-06-01 00:00:00.000
1 Engineering   Research and Development      1998-06-01 00:00:00.000

2 TOOL DESIGN   Research and Development      1998-06-01 00:00:00.000
1 ENGINEERING   Research and Development      1998-06-01 00:00:00.000
```

Żeby zapobiec sytuacjom, w których wykonanie instrukcji UPDATE kończy się zwróceniem danych, należy włączyć zaawansowaną opcję serwera:

```
EXEC sp_configure 'disallow results from triggers',1
RECONFIGURE
```

Rozmiar wirtualnych tabel ma zasadniczy wpływ na wydajność wyzwalaczy. Jeżeli są one duże, wydajność wyzwalacza, a tym samym wydajność oryginalnej instrukcji użytkownika, będzie znacznie mniejsza. W takim przypadku warto rozważyć skopiowanie zawartości tabeli INSERTED lub DELETED do tabeli tymczasowej i poindeksowanie tej tabeli (niemożliwe jest założenie indeksów na tabele wirtualne). Rozwiązanie to powinno być stosowane tylko wtedy, gdy:

1. wirtualne tabele zawierają kilkaset lub więcej wierszy,
2. w ramach wyzwalacza wirtualne tabele są wielokrotnie przeszukiwane lub sortowane:

```
ALTER TRIGGER HumanResources.trWirtualneTabele
ON HumanResources.Department
AFTER UPDATE
AS
SELECT *
INTO #tblINSERTED
FROM INSERTED
CREATE UNIQUE CLUSTERED INDEX IDX_tblI
ON #tblINSERTED(DepartmentID)
--Dalsze operacje wykonywane na tabeli tblINSERTED
```

Wyzwalacze typu AFTER

Wyzwalacze typu AFTER mogą być powiązane wyłącznie z trwałymi tabelami, a nie z widokami lub tabelami tymczasowymi. Wyzwalacze, tak jak ograniczenia, umożliwiają zapewnienie spójności danych, ale:

1. wywołanie wyzwalaczy następuje po wykonaniu instrukcji użytkownika[10]; jeżeli wykonanie oryginalnej instrukcji zostanie przerwane (np. przez ograniczenia), wyzwalacz w ogóle nie zostanie wywołany,
2. w ramach wyzwalaczy możliwe jest odwoływanie się i modyfikowanie zawartości innych tabel,
3. wyzwalacze umożliwiają porównanie wcześniejszej i bieżącej wartości modyfikowanych danych,
4. wyzwalacze pozwalają na zgłoszenie własnego komunikatu błędu.

Poniższy wyzwalacz wstawia aktualną datę zmodyfikowania opinii o produkcie:

```
CREATE TRIGGER trOpisyProd ON Production.ProductReview
AFTER UPDATE
AS
UPDATE Production.ProductReview
SET Production.ProductReview.ModifiedDate = GETDATE()
FROM INSERTED I --złączenie naturalne z tabelą INSERTED wyeliminuje te wiersze, które nie były zmieniane
JOIN Production.ProductReview p ON I.ProductReviewID = p.ProductReviewID
```

Sprawdzanie zmodyfikowanej kolumny

W ciele wyzwalaczy wywoływanych za pomocą instrukcji UPDATE można sprawdzić, która kolumna tabeli została zmodyfikowana przez oryginalną instrukcję.

1. Predykat UPDATE zwraca prawdę, jeżeli kolumna o podanej nazwie została zmodyfikowana (również wtedy, kiedy modyfikacja zakończyła się niepowodzeniem):

```
CREATE TRIGGER trNazwaDoc
ON Production.Document
AFTER UPDATE
AS
IF UPDATE (Title)
BEGIN
RAISERROR ('Nie można zmienić nazwy dokumentu', 16, 10)
ROLLBACK
END
GO

UPDATE Production.Document
SET Title = 'X'
---------------------------------------------------------------
```

[10] Serwer SQL 2005 nie umożliwia tworzenia wyzwalaczy wywoływanych przed wykonaniem oryginalnej instrukcji użytkownika.

```
Msg 50000, Level 16, State 10, Procedure trNazwaDoc, Line 7
Nie można zmienić nazwy dokumentu
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

2. Funkcja `COLUMN_UPDATED()` zwraca jeden bit dla każdej kolumny tabeli, z którą powiązany jest dany wyzwalacz. Wartość `1` oznacza zmodyfikowaną kolumnę, `0` — kolumnę, która nie była modyfikowana przez oryginalną instrukcję. Najstarszy bajt wyniku zawiera informacje o pierwszych ośmiu kolumnach tabeli, najmłodszy o ośmiu ostatnich kolumnach. Natomiast w ramach poszczególnych bajtów kolejność bitów jest odwrotna — najmłodszy bit reprezentuje pierwszą z ośmiu kolumn, a najstarszy ostatnią. Taka organizacja wyniku funkcji pozwala jednorazowo sprawdzić, za pomocą operatora `&`, które kolumny zostały zmodyfikowane[11], bez konieczności tworzenia zagnieżdżonych instrukcji warunkowych `IF UPDATE`.

Zagnieżdżanie wyzwalaczy

Domyślnie możliwe jest wykonanie przez jeden wyzwalacz operacji, które spowodują wywołanie innego wyzwalacza. Jeżeli np. usunięcie wiersza z tabeli `tab1` uruchomi wyzwalacz `tr1`, który zmodyfikuje dane w tabeli `tab2`, z jaką był powiązany wyzwalacz `tr2` wywoływany instrukcją `UPDATE`, on także zostanie uruchomiony. Dopiero przekroczenie 32 poziomów zagnieżdżenia spowoduje wystąpienie błędu i wycofanie całej transakcji. Gdy chcemy zabronić zagnieżdżania wyzwalaczy, możemy na poziomie serwera SQL 2005 wyłączyć domyślnie ustawioną opcję `nested triggers`:

```
EXEC sp_configure 'nested triggers'
```

Rekurencyjne wywołania wyzwalaczy

Rekurencyjne wywoływanie wyzwalaczy jest domyślnie zablokowane. Dotyczy to sytuacji, w których wyzwalacz modyfikuje dane w ten sposób, że w wyniku własnego działania wymusza własne wywołanie. Dotyczy też pośrednich rekurencyjnych wywołań, które mają miejsce, gdy pierwszy wyzwalacz zmienia dane w tabeli, z jaką powiązany jest inny wyzwalacz, a ten z kolei zmienia dane w tabeli powiązanej z pierwszym wyzwalaczem. Po włączeniu opcji bazy danych `RECURSIVE_TRIGGERS` przykładowy wyzwalacz `trOpisyProd` przestanie działać:

```
ALTER DATABASE AdventureWorks
SET RECURSIVE_TRIGGERS ON
GO
UPDATE Production.ProductReview
SET ReviewerName=ReviewerName
WHERE ProductReviewID=1
------------------------------------------------------------
Msg 217, Level 16, State 1, Procedure trOpisyProd, Line 3
Maximum stored procedure, function, trigger, or view nesting level exceeded (limit 32).
```

[11] Pozycję bitu reprezentującą kolumnę @n obliczymy następująco: *SUBSTRING(COLUMNS_UPDATED(), ➥(@n-1)/8+1,1)*. Natomiast jego wartość ze wzoru *POWER(2,(@n-1)%8)*.

Wyzwalacze typu INSTEAD OF

Możemy uniezależnić działanie naszych wyzwalaczy od zmian opcji baz danych wprowadzanych przez administratorów, zastępując wyzwalacze typu AFTER wyzwalaczami typu INSTEAD OF. Oto cechy wyzwalaczy tego typu.

1. Wykonywane są zamiast oryginalnej instrukcji, a nie po niej. Oznacza to, że:
   a) w ramach wyzwalacza należy ponownie wykonać oryginalną instrukcję użytkownika,
   b) kod wyzwalacza będzie wykonany, zanim nastąpi sprawdzenie ograniczeń tabeli,
   c) można powiązać takie wyzwalacze z widokami.
2. Wywoływane są tylko raz.

W pierwszym przykładzie pokazujemy, jak zastąpić wyzwalacz trOpisyProd wyzwalaczem typu INSTEAD OF:

```
ALTER TRIGGER Production.trOpisyProd
ON Production.ProductReview
INSTEAD OF UPDATE
AS
UPDATE Production.ProductReview
SET Production.ProductReview.ModifiedDate = GETDATE(),
Production.ProductReview.ProductID = I.ProductID,
Production.ProductReview.ReviewerName = I.ReviewerName,
Production.ProductReview.ReviewDate = I.ReviewDate,
Production.ProductReview.EmailAddress = I.EmailAddress,
Production.ProductReview.Rating = I.Rating,
Production.ProductReview.Comments = I.Comments
FROM INSERTED I --tu znajdziemy nowe wartości
JOIN Production.ProductReview p ON I.ProductReviewID = p.ProductReviewID
```

Wyzwalacze DDL

Wyzwalacze DDL są uruchamiane w momencie wykonywania przez użytkownika instrukcji CREATE, ALTER, DROP, GRANT, DENY, REVOKE, lub UPDATE STATISTICS. Wyzwalacze tego typu mogą być tworzone na poziomie serwera lub bazy danych.

Funkcja EVENTDATA()

Informacje o zdarzeniu, które spowodowało wywołanie wyzwalacza, zwraca funkcja EventData(). Funkcja ta może wystąpić tylko w ramach wyzwalacza, a struktura wynikowego dokumentu XML zależy od typu przechwyconej instrukcji.

W poniższym przykładzie pokazujemy, jak utworzyć uniwersalny wyzwalacz, którego działanie polega na zapisywaniu w tabeli dziennika informacji o wszystkich wykonywanych przez użytkowników instrukcjach DDL:

```
CREATE TABLE master.dbo.logDDL
(id INT IDENTITY PRIMARY KEY,
uzytkownik SYSNAME,
czas DATETIME,
instrukcja SYSNAME,
obiekt SYSNAME,
opis XML)
GO
CREATE TRIGGER trLogDDL
ON ALL SERVER
FOR CREATE_DATABASE,ALTER_DATABASE, DROP_DATABASE,DDL_LOGIN_EVENTS
AS
DECLARE @doc AS XML
SET @doc = EVENTDATA()
INSERT INTO master.dbo.logDDL (uzytkownik, czas,instrukcja, obiekt, opis)
VALUES (
CAST(@doc.query ('data(//LoginName)') AS SYSNAME),
CAST(@doc.query ('data(//PostTime)') AS VARCHAR(23)),
CAST(@doc.query ('data(//EventType)') AS SYSNAME),
CAST(@doc.query ('data(//ObjectName)') AS SYSNAME),
@doc )
```

Żeby sprawdzić działanie wyzwalacza, wystarczy wykonać poniższy skrypt:

```
CREATE DATABASE test
DROP DATABASE test
ALTER LOGIN danka
WITH DEFAULT_DATABASE=AdventureWorks, CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF

SELECT uzytkownik, czas, instrukcja, obiekt
FROM master.dbo.logDDL
------------------------------------------------------------
SQLSRV\szelor   2007-05-23 16:18:46.967      CREATE_DATABASE
SQLSRV\szelor   2007-05-23 16:19:31.650      DROP_DATABASE
SQLSRV\szelor   2007-05-23 16:20:37.707      ALTER_LOGIN         Danka
```

Wyzwalacze DDL w roli dodatkowego mechanizmu zabezpieczeń

W ramach wyzwalacza można wycofać oryginalną instrukcję użytkownika, a więc może on stanowić dodatkowy, niezależny od uwierzytelniania i autoryzacji, mechanizm bezpieczeństwa. Ponieważ wyzwalacz zostanie uruchomiony zawsze, niezależnie od tego, kto wykonuje daną instrukcję, po utworzeniu poniższego wyzwalacza nikt, nawet administrator, nie wykreuje w bazie tabeli bez klucza podstawowego:

```
CREATE TRIGGER trgTblPK
ON DATABASE
FOR CREATE_TABLE
AS
DECLARE @eventdata AS XML, @objectname AS NVARCHAR(257), @msg AS NVARCHAR(500)
SET @eventdata = EVENTDATA()
SET @objectname = QUOTENAME(@eventdata.value('(/EVENT_INSTANCE/SchemaName)[1]',
'sysname')) + N'.' + QUOTENAME(@eventdata.value('(/EVENT_INSTANCE/ObjectName)[1]',
'sysname'))
IF COALESCE( OBJECTPROPERTY(OBJECT_ID(@objectname), 'TableHasPrimaryKey'), 0) = 0
BEGIN
  SET @msg = N'Tabela ' + @objectname + ' nie ma klucza podstawowego.'
```

```
  RAISERROR(@msg, 16, 1)
  ROLLBACK
  RETURN
END
GO

CREATE TABLE tab6(col1 INT NOT NULL)
GO
CREATE TABLE tab6(col1 INT NOT NULL PRIMARY KEY)
------------------------------------------------------------
(1 row(s) affected)
Msg 50000, Level 16, State 1, Procedure trgTblPK, Line 11
Tabela [dbo].[tab6] nie ma klucza podstawowego.
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger. The batch has been aborted.

(1 row(s) affected)
```

Wyzwalacze logowania

Wyzwalacze logowania są wywoływane w momencie próby zalogowania się użytkownika do serwera SQL 2005. Tak jak pozostałe typy wyzwalaczy, umożliwiają one wycofanie transakcji, co w ich przypadku oznacza odrzucenie próby zalogowania. Informacje o przechwyconym zdarzeniu kategorii `AUDIT_LOGIN` zwraca funkcja `EVENTDATA()`.



> Jeżeli w trakcie wykonywania wyzwalacza logowania wystąpi błąd krytyczny albo wyzwalacz spróbuje wykonać operację, do której przeprowadzenia dany użytkownik nie miał wystarczających uprawnień, próba połączenia z serwerem SQL zakończy się niepowodzeniem. Dopóki wyzwalacz logowania nie zostanie dokładnie przetestowany, nie należy zamykać połączenia administracyjnego z serwerem SQL 2005.

Poniższy wyzwalacz pozwoli wskazanemu użytkownikowi zalogować się wyłącznie z komputera o podanej nazwie:

```
CREATE TRIGGER trLimitSesji
ON ALL SERVER
WITH EXECUTE AS 'sa'
FOR LOGON
AS
IF ORIGINAL_LOGIN()= 'SQLSRV\szelor' AND HOST_NAME() <> 'SQLSRV'
ROLLBACK
```