Wydanie II

SQL

Praktyczny kurs

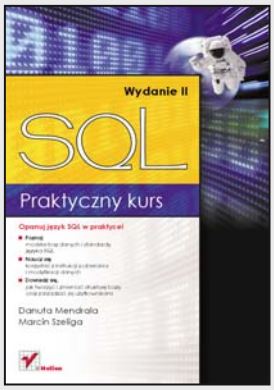
Opanuj język SQL w praktyce!

- **Poznaj**
  modele baz danych i standardy języka SQL
- **Naucz się**
  korzystać z instrukcji pobierania i modyfikacji danych
- **Dowiedz się,**
  jak tworzyć i zmieniać strukturę bazy oraz zarządzać jej użytkownikami

Danuta Mendrala
Marcin Szeliga

Helion

Praktyczny kurs SQL. Wydanie II



Autor: Danuta Mendrala, Marcin Szeliga
ISBN: 978-83-246-3373-9
Format: 158×235, stron: 304

Poznaj modele baz danych i standardy języka SQL

- Naucz się korzystać z instrukcji pobierania i modyfikacji danych
- Dowiedz się, jak tworzyć i zmieniać strukturę bazy oraz zarządzać jej użytkownikami

Opanuj język SQL w praktyce!

Bazy danych są dosłownie wszędzie. Trudno sobie dziś bez nich wyobrazić funkcjonowanie nowoczesnej biblioteki, choćby najmniejszego sklepu internetowego, biura rachunkowego czy nawet niewielkiego serwisu WWW. Użytkownicy korzystający z baz danych często nie mają nawet pojęcia, w jaki sposób odbywa się dostęp do informacji i jaki mechanizm jest za to odpowiedzialny. Na ignorancję tę nie mogą sobie jednak pozwolić osoby odpowiedzialne za tworzenie, zarządzanie i konserwowanie baz danych. Powinny one znać przynajmniej jeden z popularnych serwerów bazodanowych i sprawnie posługiwać się językiem SQL stanowiącym standardowe narzędzie komunikacji z relacyjnymi bazami.

Jeśli pragniesz dołączyć do ekskluzywnego grona administratorów baz danych lub chcesz zostać programistą aplikacji bazodanowych, lecz przeszkadza Ci brak znajomości SQL-a, sięgnij po książkę „Praktyczny kurs SQL. Wydanie II". W prosty i przystępny sposób prezentuje ona podstawowe pojęcia i zasady rządzące relacyjnym modelem baz danych, a także najważniejsze cechy i konstrukcje języka SQL oraz metody ich wykorzystywania. Lektura książki umożliwi Ci poznanie instrukcji odpowiedzialnych za odczytywanie danych z bazy i ich zapisywanie oraz modyfikację, jak również tworzenie baz i zmianę ich struktury. Poznasz też sposoby tworzenia ról i kont użytkowników oraz zarządzania ich uprawnieniami. Twoją wiedzę ugruntują praktyczne zadania kończące każdy rozdział, a zamieszczone na końcu książki rozwiązania pomogą skorygować ewentualne błędy.

- Teoretyczne podstawy funkcjonowania baz danych
- Historia języka SQL i obowiązujące standardy zapytań
- Odczytywanie, przeszukiwanie, łączenie i grupowanie danych
- Korzystanie z podzapytań
- Zapisywanie, modyfikacja i usuwanie danych
- Transakcje i równoległy dostęp do danych
- Tworzenie baz danych i modyfikacja ich struktury
- Korzystanie z widoków i indeksów
- Zarządzanie użytkownikami, rolami i prawami dostępu do baz danych

Dowiedz się, jak tworzyć relacyjną bazę danych i zarządzać nią za pomocą języka SQL

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

Spis treści

Rozdział 9.

Transakcje i współbieżność

- Czym są transakcje?
- Co oznacza skrót ACID?
- Jakie są zalety transakcyjnego przetwarzania danych?
- Na czym polega różnica pomiędzy transakcjami zagnieżdżonymi a zagnieżdżaniem transakcji?
- Co oznacza termin „współbieżność”?
- Po co serwery bazodanowe zakładają blokady?
- Kiedy dochodzi do zakleszczeń?
- Czy warto zmieniać domyślny poziom izolowania transakcji?
- W jakich sytuacjach optymistyczny model współbieżności jest lepszy niż pesymistyczny?

Właściwości transakcji

Transakcje gwarantują spójność modyfikowanych informacji. Typowym przykładem transakcyjnego przetwarzania danych jest przeniesienie pieniędzy z jednego konta na drugie. Taka operacja przebiega w dwóch etapach:

1. Zmniejszenie o pewną sumę stanu konta X.
2. Dodanie tej sumy do stanu konta Y.

Gdyby po wykonaniu pierwszej operacji wystąpił błąd uniemożliwiający wykonanie drugiej, z systemu zniknęłaby pewna suma pieniędzy. Równie nieprzyjemnym zaskoczeniem dla właściciela byłoby sprawdzenie przez niego stanu obu jego kont już po odjęciu danej sumy z pierwszego, ale przed jej dodaniem do drugiego konta.

Żeby temu zapobiec, transakcje muszą być:

1. Niepodzielne (ang. *Atomicity*). Niepodzielność oznacza, że zatwierdzane są wszystkie wchodzące w skład transakcji instrukcje albo nie jest zatwierdzana żadna z nich. Innymi słowy, wszystkie wchodzące w skład transakcji instrukcje muszą być wykonane poprawnie — jeżeli choć jedna z nich zgłosi błąd, wszystkie przeprowadzone w ramach transakcji zmiany zostaną wycofane.
2. Spójne (ang. *Consistency*). Ta cecha transakcji gwarantuje, że ich wykonanie nie doprowadzi, nawet w przypadku awarii serwera, do utraty spójności danych. Ponieważ wszystkie zmiany danych są wykonywane w ramach transakcji, przechowywane w bazach informacje zawsze będą spójne[1].
3. Izolowane (ang. *Isolation*). Izolowanie transakcji wymaga albo zablokowania modyfikowanych w ramach jednej z nich danych, albo utworzenia ich dodatkowej wersji. W zależności od obowiązującego w ramach serwera lub sesji klienckiej poziomu izolowania transakcji, może dojść do następujących sytuacji:
   a) Utrata aktualizacji (ang. *Lost update*) ma miejsce, gdy dwa procesy modyfikują jednocześnie te same dane. Przykładowo jeden użytkownik zmienia cenę towaru na 100 zł, a drugi — na 200. W takim przypadku jedna ze zmian zostanie utracona (zastąpiona drugą modyfikacją). **Domyślnie skonfigurowane serwery bazodanowe nie dopuszczają do utraty aktualizacji.**
   b) Brudne odczyty (ang. *Dirty reads*) — do takiej sytuacji dochodzi, gdy możliwe jest odczytanie zmian niezatwierdzonych jeszcze przez inny proces. Jeżeli proces odczytujący nie zażąda założenia blokady na odczytywanych danych, uzyska do nich dostęp nawet wtedy, kiedy właśnie będą modyfikowane. Gdyby proces modyfikujący wycofał wprowadzone zmiany, odczytane dane okazałyby się niespójne. **Domyślnie skonfigurowane serwery bazodanowe nie dopuszczają brudnych odczytów.**
   c) Niepowtarzalne odczyty (ang. *Non-repeatable reads*) mają miejsce, gdy powtórzenie w ramach transakcji tego samego odczytu daje inny wynik. Różnice w wynikach są spowodowane tym, że natychmiast po zakończeniu odczytu (a nie po zakończeniu całej transakcji) proces odczytujący zdejmuje blokady założone na odczytywane dane. Niezablokowane dane mogą być zmienione przez inny proces, a więc ich powtórne odczytanie da inny (niespójny) wynik. **Domyślnie skonfigurowane serwery bazodanowe dopuszczają niepowtarzalne odczyty.**
   d) Odczyty widma (ang. *Phantom reads*) — sytuacja taka ma miejsce, jeżeli pomiędzy dwoma wykonanymi w ramach transakcji odczytami zmieni się liczba odczytywanych wierszy. Jeśli np. podczas pierwszego odczytu w tabeli

[1] Przynajmniej w teorii. W praktyce bazy danych ulegają uszkodzeniu, choć bardzo rzadko z winy serwerów bazodanowych.

Produkty znajdowało się 100 produktów o cenach niższych niż 10 zł, instrukcja SELECT * FROM Produkty WHERE Cena <10 zwróciłaby 100 wierszy. W trakcie trwania transakcji możliwa jest jednak zmiana pozostałych wierszy tabeli, w tym obniżenie ceny jakiegoś produktu poniżej 10 zł. Możliwe jest również wstawienie do tej tabeli nowego produktu o cenie np. 7 zł. Z tego powodu drugie wykonanie tego samego zapytania zwróciłoby już 102 wiersze. **Domyślnie skonfigurowane serwery bazodanowe dopuszczają odczyty widma.**

4. Trwałe (ang. *Durability*). Trwałość transakcji gwarantuje, że efekty zatwierdzonych transakcji będą zapisane w bazie, nawet w przypadku awarii serwera baz danych. Do przywrócenia spójności danych serwery bazodanowe z reguły używają jakiejś formy dziennika transakcyjnego.

Pierwsze litery cech transakcji (A — *Atomicity*, C — *Consistency*, I — *Isolation*, D — *Durability*) tworzą skrót ACID, powszechnie używany do opisywania reguł przetwarzania danych, których muszą przestrzegać serwery bazodanowe, żeby mogły być nazwane transakcyjnymi lub relacyjnymi.

Transakcyjne przetwarzanie danych

Serwery bazodanowe mogą działać w trybie niejawnego zatwierdzania transakcji (w serwerze SQL 2011 taki tryb jest trybem domyślnym). Oznacza to, że użytkownicy nie muszą samodzielnie rozpoczynać transakcji, bo serwer robi to za nich.

W trybie niejawnego zatwierdzania transakcji wykonanie każdej instrukcji języka SQL składa się z trzech etapów:

1. Serwer bazodanowy automatycznie rozpoczyna nową transakcję.
2. Wykonywana jest pojedyncza instrukcja SQL.
3. Jeżeli instrukcja została wykonana z powodzeniem, transakcja jest zatwierdzana, w przeciwnym razie jest wycofywana.

Taki sposób działania oznacza, że użytkownicy nie mogą samodzielnie zatwierdzać lub wycofywać automatycznie rozpoczętych transakcji. Dlatego nazywa się on trybem niejawnego zatwierdzania transakcji.

Poniższy przykład ilustruje działanie trybu niejawnego zatwierdzania transakcji za pomocą funkcji systemowej @@TRANCOUNT zwracającej liczbę otwartych, aktywnych w danym momencie transakcji:

```
SELECT @@TRANCOUNT;
UPDATE dbo.Produkty
SET [Koszt standardowy]=3
WHERE [Kod produktu]='NWTC-82';
```

```
SELECT @@TRANCOUNT;
-----------------------------------------------------------
0
0
```

Jak widać, przed rozpoczęciem i po zakończeniu wykonywania instrukcji `UPDATE` nie było żadnych otwartych transakcji.

Tryb jawnego zatwierdzania transakcji

W niektórych serwerach bazodanowych (np. w serwerze Oracle) domyślnym trybem transakcyjnego przetwarzania danych jest tryb ich jawnego zatwierdzania. W tym trybie wykonanie każdej instrukcji języka SQL przebiega następująco:

1. Serwer bazodanowy automatycznie rozpoczyna nową transakcję.
2. Wykonywana jest pojedyncza instrukcja SQL.
3. Użytkownik samodzielnie musi zatwierdzić lub wycofać otwartą przez serwer transakcję.

Działanie tego trybu można zasymulować w serwerze SQL 2011, ustawiając opcję sesji `IMPLICIT_TRANSACTIONS`:

```
SET IMPLICIT_TRANSACTIONS ON;
SELECT @@TRANCOUNT;
UPDATE dbo.Produkty
SET [Koszt standardowy]=3
WHERE [Kod produktu]='NWTC-82';
SELECT @@TRANCOUNT;
-----------------------------------------------------------
0
1
```

Tym razem przed rozpoczęciem instrukcji `UPDATE` również nie było otwartych transakcji, ale niejawnie rozpoczęta transakcja nie została po jej wykonaniu automatycznie zamknięta. Musi to zrobić sam użytkownik — albo zatwierdzając wprowadzone zmiany, albo je wycofując.

Przed przejściem do dalszych ćwiczeń zakończ transakcję i wyłącz omawiany tryb:

```
COMMIT TRAN;
SET IMPLICIT_TRANSACTIONS OFF;
```

> Tryb jawnego zatwierdzania transakcji pozwala wycofywać przypadkowe lub błędne modyfikacje, ale zatwierdzanie transakcji, która nie została przez nas rozpoczęta, jest mało intuicyjne.

Rozpoczynanie transakcji

Mechanizm transakcyjnego przetwarzania danych pokażemy, jawnie rozpoczynając i kończąc transakcje. Pozwoli nam to wykonać w ramach poszczególnych transakcji dowolną liczbę instrukcji oraz samodzielnie sterować czasem rozpoczęcia i zakończenia poszczególnych transakcji.

Żeby rozpocząć transakcję, należy wykonać instrukcję `BEGIN TRAN`[2]:

```
BEGIN TRAN;
SELECT @@TRANCOUNT;
-----------
1
```

Jeżeli teraz w ramach tej samej sesji (czyli w tym samym oknie edytora SQL) zaktualizujemy ceny wybranych towarów i sprawdzimy liczbę aktywnych transakcji, dowiemy się, że rozpoczęta przez nas transakcja nadal jest otwarta:

```
UPDATE dbo.Produkty
SET [Cena katalogowa]=1
WHERE Kategoria='Zupy'
SELECT @@TRANCOUNT;
-----------
1
```

Dopóki transakcja, w ramach której przeprowadziliśmy dowolne zmiany, jest otwarta, możemy je albo wycofać, albo zatwierdzić. Ponieważ serwer bazodanowy nie jest w stanie przewidzieć naszej decyzji, a jedną z cech transakcji jest jej odizolowanie, próba odczytania danych z tabeli `dbo.Produkty` w ramach tej samej sesji skończy się zupełnie inaczej niż ta sama próba wykonana przez innego użytkownika.

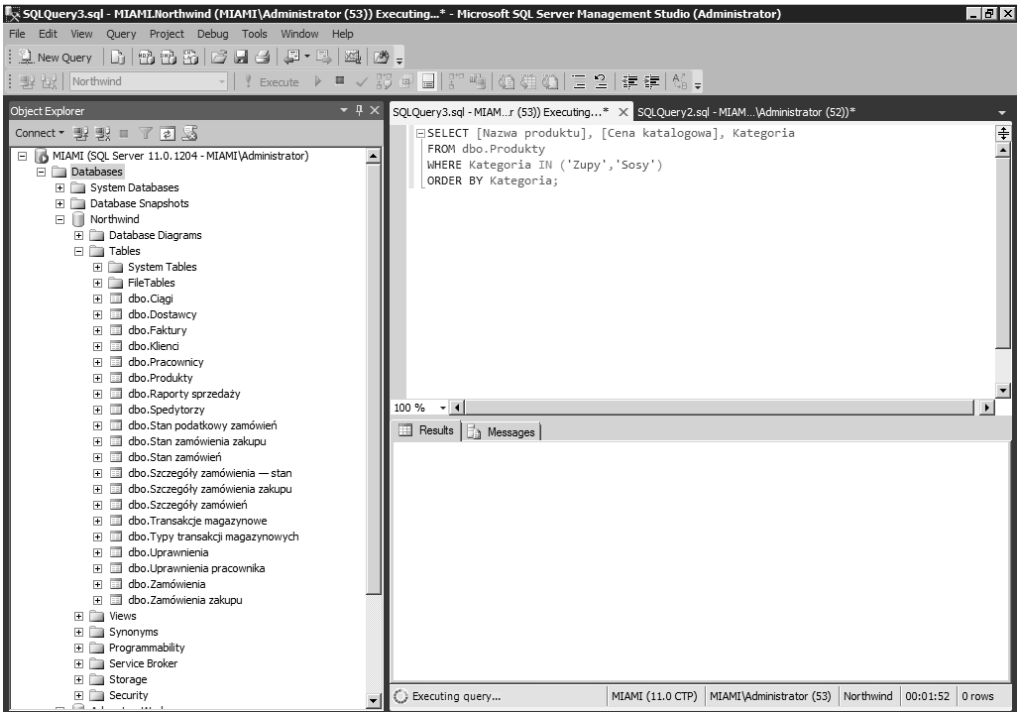
Żeby się o tym przekonać:

1. W tym samym oknie kodu SQL wykonaj zapytanie:

```
SELECT [Nazwa produktu], [Cena katalogowa], Kategoria
FROM dbo.Produkty
WHERE Kategoria IN ('Zupy','Sosy')
ORDER BY Kategoria;
------------------------------------------------------------
Northwind Traders Hot Pepper Sauce   21,05   Sosy
Northwind Traders Tomato Sauce       17,00   Sosy
Northwind Traders Curry Sauce        30,00   Sosy
Northwind Traders Clam Chowder        1,00   Zupy
Northwind Traders Vegetable Soup      1,00   Zupy
Northwind Traders Chicken Soup        1,00   Zupy
```

2. Zostało ono natychmiast wykonane, a cena każdej zupy wynosi 1.
3. Aby wykonać to samo zapytanie jako inny użytkownik, otwórz nowe okno edytora SQL[3] i skopiuj do niego powyższą instrukcję `SELECT` (rysunek 9.1).

[2] W niektórych serwerach bazodanowych transakcje rozpoczyna się instrukcjami `BEGIN TRANSACTION` lub `BEGIN WORK`.



Rysunek 9.1. *Zapytanie wykonuje się już prawie dwie minuty, ale wciąż nie zwróciło żadnych danych*

Transakcyjne przetwarzanie danych polega na takim realizowaniu żądań klientów przez serwery bazodanowe, żeby każdy klient miał wrażenie, iż jest jedynym użytkownikiem serwera. Wymaga to opisanego w dalszej części rozdziału blokowania obiektów, do których w danym momencie odwołują się inni użytkownicy serwera.

Wycofywanie transakcji

Wycofanie transakcji oznacza przywrócenie danych do stanu sprzed jej rozpoczęcia i zdjęcie wszystkich założonych na potrzeby transakcji blokad. Jeżeli wrócimy do pierwszego okna edytora SQL (tego, w którym zapytanie zwróciło wyniki) i wykonamy w nim instrukcję `ROLLBACK TRAN`[4], a następnie przełączymy się do drugiego okna edytora SQL, przekonamy się, że zapytanie wreszcie zostało wykonane i w dodatku ceny produktów z pierwszej podkategorii wcale nie wynoszą 1. Spowodowane jest to wycofaniem transakcji, w ramach której ceny były zmienione, i zdjęciem założonych na jej potrzeby blokad:

[3] Można to zrobić, naciskając kombinację klawiszy *Ctrl+N* lub klikając przycisk *New Query*.

[4] W niektórych serwerach bazodanowych transakcje wycofuje się instrukcjami `ROLLBACK TRANSACTION` lub `ROLLBACK WORK`.

```
SELECT [Nazwa produktu], [Cena katalogowa], Kategoria
FROM dbo.Produkty
WHERE Kategoria IN ('Zupy','Sosy')
ORDER BY Kategoria;
------------------------------------------------------------
Northwind Traders Hot Pepper Sauce   21,05     Sosy
Northwind Traders Tomato Sauce       17,00     Sosy
Northwind Traders Curry Sauce        30,00     Sosy
Northwind Traders Clam Chowder        7,2375   Zupy
Northwind Traders Vegetable Soup      1,4175   Zupy
Northwind Traders Chicken Soup        1,4625   Zupy
```

Zatwierdzanie transakcji

Zatwierdzenie transakcji oznacza utrwalenie wprowadzonych w jej trakcie zmian i zdjęcie wszystkich założonych na potrzeby transakcji blokad. Wspomniany na początku rozdziału przykład przelania pieniędzy z jednego konta na drugie mógłby być zaimplementowany w taki sposób:

```
BEGIN TRAN;
EXEC uspDodajDoKonta '123-456-78-90', 500;
EXEC uspOdejmijOdKonta '231-645-87-09', 500;
IF @@ERROR=0
  COMMIT TRAN;
ELSE
  ROLLBACK TRAN;
```

Po jawnym rozpoczęciu transakcji następuje wywołanie dwóch (nieistniejących w przykładowej bazie danych) procedur. Jeżeli żadna z nich nie zgłosi błędu, cała transakcja będzie zatwierdzona (zatwierdzić transakcję możemy, wykonując instrukcję `COMMIT TRAN`[5]), w przeciwnym razie zostanie ona wycofana.

Zagnieżdżanie transakcji

Większość serwerów bazodanowych pozwala zagnieżdżać transakcje, czyli wykonać instrukcję `BEGIN TRAN` w ramach wcześniej rozpoczętej transakcji. **Wynikiem takiej operacji jest zwiększenie licznika otwartych transakcji, a nie rozpoczęcie nowej (atomowej, niepodzielnej, trwałej i spójnej) transakcji.**

Działanie mechanizmu zagnieżdżania transakcji ilustruje poniższy przykład: wykonanie instrukcji `BEGIN TRAN` powoduje zwiększenie o jeden licznika otwartych transakcji, wykonanie instrukcji `COMMIT TRAN` zmniejsza wartość tego licznika o jeden, ale wykonanie instrukcji `ROLLBACK TRAN` zamyka transakcje i zeruje licznik otwartych transakcji:

```
BEGIN TRAN;
SELECT @@TRANCOUNT;
```

[5] W niektórych serwerach bazodanowych transakcje zatwierdza się instrukcjami `COMMIT TRANSACTION` lub `COMMIT WORK`.

```
BEGIN TRAN;
SELECT @@TRANCOUNT;
BEGIN TRAN;
SELECT @@TRANCOUNT;
COMMIT TRAN;
SELECT @@TRANCOUNT;
ROLLBACK TRAN;
SELECT @@TRANCOUNT;
-----------------------------------------------------------
1
2
3
2
0
```

Punkty przywracania

Większość serwerów bazodanowych pozwala wycofać nie tylko całą transakcję, lecz także jej część. W tym celu należy w trakcie transakcji wykonać instrukcję `SAVE TRAN`[6], a następnie przywrócić ją do danego punktu:

```
BEGIN TRAN;
INSERT INTO dbo.Dostawcy(Firma)
VALUES ('TEST1');
SAVE TRAN PP1;
INSERT INTO dbo.Dostawcy(Firma)
VALUES ('TEST2');
SELECT @@TRANCOUNT;
ROLLBACK TRAN PP1;
SELECT @@TRANCOUNT;
-----------------------------------------------------------
1
1
```

Ponieważ przywrócenie stanu transakcji do określonego punktu nie powoduje jej zakończenia (liczba otwartych transakcji nadal wynosi 1), musimy ją zatwierdzić lub wycofać:

```
SELECT ID, Firma
FROM dbo.Dostawcy
WHERE Firma LIKE 'TEST_';
-----------------------------------------------------------
17   TEST1
```

Jako że druga instrukcja `INSERT` została wykonana po zdefiniowaniu punktu przywracania `PP1`, instrukcja `ROLLBACK TRAN PP1` przywróciła stan danych do momentu sprzed jej wykonania i w rezultacie tylko pierwszy wiersz został na trwałe wstawiony do tabeli.

[6] W niektórych serwerach bazodanowych punkty przywracania tworzy się instrukcjami `SAVE TRANSACTION` lub `SAVE WORK`.

Współbieżność

Współbieżność to zdolność systemu do jednoczesnego realizowania wielu operacji, z reguły uzyskiwana poprzez uruchomienie osobnych procesów (robotników) na potrzeby obsługi poszczególnych żądań.

> Współbieżność ma ogromny wpływ na skalowalność serwerów bazodanowych, czyli ich zdolność do coraz szybszego wykonywania transakcji dzięki rozbudowywaniu komputerów, na przykład zwiększaniu ich mocy obliczeniowej czy przepustowości dysków twardych.

Żeby każdy z kilkuset czy nawet kilku tysięcy jednoczesnych użytkowników serwera bazodanowego mógł pracować tak, jakby był jego jedynym użytkownikiem, konieczne jest odizolowanie od siebie poszczególnych transakcji. Umożliwiają to automatycznie zakładane blokady.

Blokady

Pomijając analizy wewnętrznych mechanizmów działania różnych serwerów bazodanowych, blokady można podzielić ze względu na ich tryb (sposób blokowania) i zakres (typ blokowanych zasobów).

Tryby blokad

Tryb blokady decyduje o tym, czy możliwe będzie jej założenie na zasobie wcześniej zablokowanym przez inny proces:

1. **Blokady współdzielone S** (ang. *Shared*) są domyślnie zakładane na odczytywanych obiektach, takich jak tabele czy wiersze. Na obiekt zablokowany w trybie S inne procesy też mogą założyć blokadę S, czyli **odczytujący nie blokują innych odczytujących**. Blokady S domyślnie zakładane są tylko na czas wykonywania zapytania, a nie całej transakcji.
2. **Blokady wyłączne X** (ang. *eXclusive*) są zakładane na modyfikowanych obiektach. Blokady X są niekompatybilne z innymi blokadami, czyli modyfikujący blokują innych użytkowników. W przeciwieństwie do blokad współdzielonych blokady wyłączne domyślnie są utrzymywane do zakończenia całej transakcji, a nie pojedynczej operacji.

Zakresy blokad

Blokady mogą być zakładane na poziomie poszczególnych wierszy, kluczy indeksów, stron, zakresów lub całych tabel. Te obiekty tworzą naturalną hierarchię: tabela składa się z wielu stron, na każdej stronie zapisanych jest wiele wierszy itd. Z tego powodu serwery bazodanowe muszą analizować wszystkie istniejące blokady, zanim założą nową — jeżeli choć jeden wiersz tabeli jest zablokowany w trybie X, nie można na całej tabeli założyć innej blokady.

Im większe obiekty są blokowane, tym mniejsza współbieżność (bo użytkownicy muszą dłużej czekać na dostęp do zablokowanych zasobów), ale również tym mniejsza liczba blokad, którymi musi zarządzać serwer bazodanowy (zostanie założona jedna blokada na całej tabeli zamiast miliona blokad na poszczególnych wierszach).

Zakleszczenia

Zakleszczenie (ang. *DeadLock*) ma miejsce, gdy różne procesy blokują się nawzajem w taki sposób, że żaden z nich nie jest w stanie założyć blokad wymaganych do ukończenia już rozpoczętych operacji.

Najczęściej występują dwa typy zakleszczeń:

1. Zakleszczenia cykliczne, wynikające z tego, że dwa procesy w różnych kolejnościach próbują uzyskać dostęp do tych samych zasobów.
2. Zakleszczenia konwersji blokad, związane ze zmianą wcześniej założonej blokady współdzielonej (wiele procesów może jednocześnie zablokować ten sam zasób w trybie S) na blokadę wyłączną (tylko jeden proces może założyć na tym samym obiekcie blokadę X).

Serwery bazodanowe automatycznie wykrywają zakleszczenia i przerywają działanie jednego procesu. Na ofiarę zakleszczenia wybierany jest proces o niższym priorytecie, a jeżeli oba procesy mają ten sam priorytet, ofiarą zakleszczenia zostaje ten, którego wycofanie jest mniej kosztowne.

Mechanizm wykrywania i usuwania zakleszczeń pokazuje poniższy przykład:

Pierwszy użytkownik w ramach jawnie rozpoczętej transakcji modyfikuje kilka danych w tabeli `dbo.Dostawcy`:

```
BEGIN TRAN;
UPDATE dbo.Dostawcy
SET Firma = UPPER(Firma)
WHERE ID<5;;
------------------------------------------------------------
 (4 row(s) affected)
```

Następnie inny użytkownik w ramach jawnie rozpoczętej przez siebie transakcji modyfikuje znacznie więcej danych w tabeli `Transakcje magazynowe`[7]:

```
BEGIN TRAN;
UPDATE dbo.[Transakcje magazynowe]
SET Ilość += 1
WHERE [ID transakcji] <135;
------------------------------------------------------------
 (100 row(s) affected)
```

[7] Możemy zasymulować jednoczesną pracę dwóch użytkowników, otwierając nowe okno edytora SQL — każde z okien nawiązuje własną sesję z bazą danych.

W dalszej kolejności pierwszy użytkownik próbuje odczytać zawartość tabeli zablokowanej już przez drugą sesję (okno wyników może pokazać kilka wierszy, ale i tak użytkownik będzie musiał czekać na możliwość zablokowania w trybie S pozostałych wierszy tabeli `Transakcje magazynowe`):

```
SELECT *
FROM dbo.[Transakcje magazynowe];
```

W tym momencie nie wystąpiło jeszcze zakleszczenie — wystarczyłoby, żeby drugi użytkownik zakończył swoją transakcję. Ale jeżeli w ramach drugiej sesji użytkownik spróbuje odczytać zawartość tabeli zmodyfikowanej przez pierwszego użytkownika, oba procesy się zakleszczą:

```
SELECT ID, Firma, [Tytuł zawodowy]
FROM dbo.Dostawcy;
------------------------------------------------------------
1      Dostawca A   Kierownik ds. sprzedaży
2      Dostawca B   Kierownik ds. sprzedaży
3      Dostawca C   Przedstawiciel handlowy
4      Dostawca D   Kierownik ds. marketingu
5      Dostawca E   Kierownik ds. sprzedaży
…
```

Po chwili drugie zapytanie zostało jednak wykonane, co więcej — nazwy firm nie zostały przekonwertowane na wielkie litery. Żeby przekonać się, dlaczego tak się stało, wystarczy przełączyć się do okienka pierwszej sesji. Znajdziemy w nim poniższy komunikat błędu:

```
Transaction (Process ID 52) was deadlocked on lock resources with another process
↪and has been chosen as the deadlock victim. Rerun the transaction.
```

Jeżeli sprawdzimy liczbę otwartych w ramach pierwszej sesji transakcji, okaże się, że jawnie rozpoczęta przez pierwszego użytkownika transakcja została — zgodnie z komunikatem błędu — wycofana:

```
SELECT @@TRANCOUNT;
------------------------------------------------------------
0
```

Ponieważ wycofanie transakcji wiąże się ze zdjęciem założonych na jej potrzeby blokad, druga sesja mogła z powodzeniem zakończyć operacje i odczytać tabelę `dbo.Dostawcy`. Liczba transakcji otwartych w ramach drugiej sesji nadal wynosi 1 — żeby zakończyć ćwiczenie i wycofać zmiany, należy wykonać w tym oknie edytora SQL instrukcję `ROLLBACK TRAN`.

Poziomy izolowania transakcji

Możemy wpływać na sposób zakładania blokad przez serwery bazodanowe, zmieniając poziom izolowania transakcji. Większość serwerów pozwala ustawić (na poziomie serwera, bazy danych lub poszczególnych sesji) jeden z czterech poziomów izolowania transakcji, przedstawionych przez nas od najmniej restrykcyjnego, w którym maksymalna współbieżność jest okupiona występowaniem największej liczby typów niespójności danych, do najbardziej restrykcyjnego, który kosztem ograniczenia współbieżności gwarantuje najwyższy poziom spójności danych.

Read Uncommitted

W trybie niezatwierdzonego odczytu (ang. *Read Uncommitted*) odczyt danych nie powoduje założenia blokady współdzielonej. **Na tym poziomie występują brudne odczyty, niepowtarzalne odczyty i odczyty widma (jedynym niekorzystnym zjawiskiem niewystępującym na tym poziomie jest utrata aktualizacji).**

Żeby się o tym przekonać:

1. W jednej sesji (oknie edytora SQL) rozpoczniemy transakcję i zaktualizujemy adres klienta:

```
BEGIN TRAN;
UPDATE dbo.Klienci
SET Adres = 'ZmianaWToku'
WHERE ID=1;
------------------------------------------------------------
 (1 row(s) affected)
```

2. W drugiej sesji zmienimy poziom izolowania transakcji na `Read Uncommitted` i spróbujemy odczytać modyfikowane przez innego użytkownika dane:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
GO
SELECT Adres
FROM dbo.Klienci
WHERE ID=1;
------------------------------------------------------------
ZmianaWToku
```

Udało nam się odczytać dane, pomimo że osoba, która je zmieniała, nie zatwierdziła jeszcze transakcji, a więc w każdej chwili może ją wycofać. **W tym trybie** (często wymuszanym na poziomie poszczególnych instrukcji za pomocą specyficznych dla danego serwera bazodanowego dyrektyw optymalizatora) **można odczytywać dane, o których wiemy, że nie będą w tym samym czasie modyfikowane.**

Kończąc ćwiczenie, zamknij bez zatwierdzania otwartej transakcji i na nowo otwórz oba okna edytora SQL — w ten sposób kolejne ćwiczenie rozpoczniemy, pracując w domyślnym trybie izolowania transakcji.

Read Committed

Tryb odczytu zatwierdzonego (ang. *Read Committed*) **jest domyślnym poziomem izolowania transakcji.** Na tym poziomie odczyt danych wymaga założenia na nich blokady współdzielonej. Ponieważ zakładana na czas zmiany blokada X jest niekompatybilna z innymi blokadami, w tym z blokadą S, eliminuje to brudne odczyty. Jednak **na tym poziomie nadal występują niepowtarzalne odczyty i odczyty widma**.

Zjawisko niepowtarzalnego odczytu pokazuje poniższy przykład:

1. W pierwszym oknie edytora SQL ustawiamy tryb odczytów zatwierdzonych[8], jawnie rozpoczynamy transakcję i odczytujemy adres wybranego klienta:

[8] Ponieważ ten tryb jest trybem domyślnym, instrukcja `SET` jest dodana tylko w celach demonstracyjnych.

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRAN;
SELECT Adres
FROM dbo.Klienci
WHERE ID=1;
------------------------------------------------------------
ul. Jasna 123
```

2. W tym momencie transakcja jest nadal otwarta, a my w drugim oknie edytora SQL zmienimy adres tego klienta:

```
UPDATE dbo.Klienci
SET Adres = 'OdczytWToku'
WHERE ID=1;
------------------------------------------------------------
 (1 row(s) affected)
```

3. Jeżeli pierwszy użytkownik w ramach tej samej transakcji ponownie odczyta adres tego klienta, uzyska inny wynik:

```
SELECT Adres
FROM dbo.Klienci
WHERE ID=1;
COMMIT TRAN;
------------------------------------------------------------
OdczytWToku
```

Repeatable Read

W trybie powtarzalnego odczytu (ang. *Repeatable Read*) blokady współdzielone typu S są utrzymywane do czasu zakończenia całej transakcji. Dzięki temu inny proces nie może zmodyfikować odczytywanych w jej ramach danych, co eliminuje niepowtarzalne odczyty. Z niekorzystnych zjawisk związanych z izolowaniem transakcji **na tym poziomie występują tylko odczyty widma**.

Zjawisko odczytu widma pokazuje poniższy przykład:

1. W ramach pierwszej sesji zmienimy poziom izolowania transakcji na `Repeatable Read` i w ramach jawnie rozpoczętej transakcji odczytamy nazwy towarów o kodach kończących się cyfrą `6`:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRAN;
SELECT [Nazwa produktu]
FROM dbo.Produkty
WHERE [Kod produktu] LIKE '%6';
------------------------------------------------------------
Northwind Traders Boysenberry Spread
Northwind Traders Marmalade
Northwind Traders Gnocchi
Northwind Traders Tomato Sauce
Northwind Traders Cake Mix
Northwind Traders Smoked Salmon
```

2. Podczas gdy pierwsza transakcja jest wciąż otwarta, w drugim oknie edytora SQL zmienimy kod jednego z pozostałych, niezwróconych przez pierwsze zapytanie produktu na `NWTCO-6`, a więc na kod spełniający warunki pierwszego zapytania:

```
UPDATE dbo.Produkty
SET [Kod produktu] = 'NWTCO-6'
WHERE [Kod produktu] ='NWTCO-3';
------------------------------------------------------------
 (1 row(s) affected)
```

3. Jeżeli pierwszy użytkownik raz jeszcze wykona, w ramach tej samej transakcji, to samo zapytanie, tym razem jego wynik będzie liczył więcej wierszy — pojawi się w nim wiersz widmo:

```
SELECT [Nazwa produktu]
FROM dbo.Produkty
WHERE [Kod produktu] LIKE '%6';
------------------------------------------------------------
Northwind Traders Syrup
Northwind Traders Boysenberry Spread
Northwind Traders Marmalade
Northwind Traders Gnocchi
Northwind Traders Tomato Sauce
Northwind Traders Cake Mix
Northwind Traders Smoked Salmon
```

4. Jeśli jednak w ramach drugiej sesji spróbujemy zmienić dane odczytywane w ramach nadal otwartej pierwszej transakcji (czyli doprowadzić do niepowtarzalnego odczytu), instrukcja będzie oczekiwać, aż pierwsza transakcja zostanie zakończona, a założone dla niej blokady zdjęte:

```
UPDATE dbo.Produkty
SET [Kod produktu] = 'NWTCO-1'
WHERE [Kod produktu] ='NWTJP-6';
```

5. Żeby powyższa aktualizacja została wykonana, w pierwszym oknie edytora SQL wykonaj instrukcję `COMMIT TRAN`.

W trybie `Repeatable Read` **należy odczytywać te dane, które w ramach transakcji są odczytywane kilkukrotnie i mogą być zmieniane w tym samym czasie przez innych użytkowników.** Sytuacja taka ma miejsce np. w różnego rodzaju zestawieniach i raportach zbiorczych, w których odczytując te same dane, za każdym razem musimy otrzymać te same wyniki, inaczej zestawienie lub raport będą niespójne.

Serializable

W trybie szeregowania transakcje odwołujące się do tych samych tabel są wykonywane jedna po drugiej. Blokowanie całych obiektów, a nie tylko odczytywanych danych, na czas trwania transakcji pozwala wyeliminować odczyty widma, ale powoduje, że odczytując nawet jeden wiersz tabeli, możemy uniemożliwić pozostałym użytkownikom zmodyfikowanie przechowywanych w niej danych.

Żeby się o tym przekonać:

1. W pierwszym oknie edytora SQL przełączymy się do trybu szeregowania, jawnie rozpoczniemy transakcję i odczytamy informacje o wybranym towarze:

```
SELECT [Nazwa produktu]
FROM dbo.Produkty
```

```
WHERE [Kod produktu] LIKE '%6';
------------------------------------------------------------
Northwind Traders Syrup
Northwind Traders Gnocchi
Northwind Traders Tomato Sauce
Northwind Traders Cake Mix
Northwind Traders Smoked Salmon
```

2. Jeżeli teraz w drugim oknie edytora SQL spróbujemy zmienić dane dowolnego, również niezwróconego przez pierwsze zapytanie produktu, okaże się, że aktualizacja została zablokowana i będzie wykonana dopiero po zakończeniu pierwszej transakcji:

```
UPDATE dbo.Produkty
SET [Kod produktu] = 'NWTCA-48'
WHERE [Kod produktu] ='NWTCA-49';
```

3. Kończąc ćwiczenie, zamknij oba okna edytora SQL bez zatwierdzania rozpoczętej w jednym z nich transakcji.

W trybie `Serializable` mamy gwarancję, że odczytywane w ramach transakcji dane zawsze będą takie same — serwer bazodanowy nie dopuści nie tylko do ich zmiany, lecz także do pojawienia się nowych danych. Jednak przez ten czas pozostali użytkownicy nie będą mogli modyfikować zablokowanych tabel. W większości przypadków powoduje to tak znaczne wydłużenie czasu reakcji serwera, że lepiej jest skopiować odczytywane dane[9], a jeżeli zmian nie jest zbyt dużo, przełączyć się do modelu optymistycznego.

Model optymistyczny

W modelu optymistycznym tylko modyfikujący blokują innych modyfikujących, czyli różni użytkownicy mogą jednocześnie modyfikować i odczytywać te same dane.

Serwery bazodanowe zapewniają spójność modyfikowanych w tym modelu danych poprzez ich wersjonowanie. Zakładając (optymistycznie), że w czasie gdy jeden użytkownik odczytuje dane, inni raczej nie będą ich modyfikować, serwery te są w stanie na bieżąco zarządzać dodatkowymi wersjami danych.

Jeżeli to założenie jest prawdziwe, czyli jeżeli jednoczesne modyfikacje i odczyty tych samych danych nie zachodzą zbyt często, możemy znacznie skrócić czas reakcji serwera[10], przełączając bazę do optymistycznego modelu współbieżności. Żeby się o tym przekonać:

1. W pierwszym oknie edytora SQL wykonamy poniższe instrukcje, przełączając bazę *Northwind* do modelu optymistycznego:

```
USE master;
ALTER DATABASE Northwind
SET READ_COMMITTED_SNAPSHOT ON
```

[9] Niektóre serwery bazodanowe pozwalają utworzyć migawkę (ang. *Snapshot*) danych.

[10] Niektóre serwery bazodanowe, np. serwer Oracle, domyślnie działają w optymistycznym modelu współbieżności.

```
WITH ROLLBACK IMMEDIATE;
----------------------------------------------------------
Command(s) completed successfully.
```

2. W tym samym oknie edytora SQL połączymy się z bazą *Northwind* i w ramach jawnie rozpoczętej transakcji zmienimy dane dwóch pracowników:

```
USE Northwind;
BEGIN TRAN;
UPDATE dbo.Pracownicy
SET Nazwisko = 'X'
WHERE ID <3;
----------------------------------------------------------
 (2 row(s) affected)
```

3. W nowym oknie edytora SQL odczytamy dane o kilku pracownikach:

```
SELECT ID, Nazwisko
FROM dbo.Pracownicy
WHERE ID <3;
----------------------------------------------------------
1      Ciesielska
2      Czupta
```

4. Okazuje się, że tym razem zapytanie zostało wykonane natychmiast, ale z zachowaniem wymogów domyślnego trybu izolowania transakcji, czyli trybu `Read Committed` — **pozostali użytkownicy serwera odczytają ostatnią zatwierdzoną wersję danych**. Gdyby rozpoczęta w ramach pierwszej sesji transakcja została zatwierdzona, to ponowne wykonanie tego samego zapytania zwróciłoby najnowszą, zatwierdzoną wersję ze zmienionymi danymi dwóch pierwszych pracowników.

Model pesymistyczny

W modelu pesymistycznym odczytujący są blokowani przez modyfikujących (serwer będzie czekał z założeniem blokady S, aż zdjęta zostanie blokada X)**, a modyfikujący przez odczytujących** (założenie blokady X wymaga zdjęcia blokady S).

Ponieważ koszt zarządzania wieloma wersjami tych samych danych rośnie wraz ze wzrostem wersjonowanych danych, w tym modelu zakłada się (pesymistycznie), że odczytywane dane będą w tym samym czasie regularnie modyfikowane.

Żeby przywrócić pesymistyczny (domyślny) model współbieżności bazy *Northwind*, należy wykonać poniższe instrukcje:

```
USE master;
ALTER DATABASE Northwind
SET READ_COMMITTED_SNAPSHOT OFF
WITH ROLLBACK IMMEDIATE;
----------------------------------------------------------
Nonqualified transactions are being rolled back. Estimated rollback completion:
↪100%.
```

Podsumowanie

- Serwery bazodanowe przeprowadzają wszystkie zmiany danych w ramach jawnie lub niejawnie rozpoczętych transakcji.
- Transakcje powinny być otwierane jak najpóźniej i zamykane jak najwcześniej.
- Transakcje powinny zawierać tylko powiązane ze sobą instrukcje.
- Przerwane (czy to z powodu awarii klienta, czy też serwera) transakcje będą wycofane.
- Na czas trwania transakcji pewne obiekty bazy danych są automatycznie blokowane.
- Serwery bazodanowe automatycznie wykrywają zakleszczenia i usuwają je poprzez wycofanie jednej z zakleszczonych transakcji.
- Odizolowanie, jedną z czterech cech ACID transakcji, uzyskuje się za pomocą automatycznie zakładanych i zwalnianych blokad.
- Można sterować sposobem zakładania i czasem trwania blokad, zmieniając poziom izolowania transakcji.
- W modelu optymistycznym serwery bazodanowe wersjonują dane, co poprawia współbieżność kosztem większego obciążenia serwera.

Zadania

1. Twoim zadaniem jest przygotowanie raportu podsumowującego roczną sprzedaż. Wyliczając sumy i średnie wartości sprzedaży produktów, kilkukrotnie musisz odczytać tabelę `dbo.Produkty`. Jak zagwarantujesz poprawność wyników raportu?
2. Po przerwie na lunch użytkownicy zgłaszają, że próby dalszej pracy z bazą danych kończą się chwilowym „zawieszeniem" programu i wreszcie komunikatem błędu mówiącym, iż serwer bazodanowy jest niedostępny. Po sprawdzeniu okazuje się, że serwer i sieć działają normalnie, a baza nie została uszkodzona. Co jest najbardziej prawdopodobną przyczyną problemu?
3. W ramach tworzonej procedury modyfikujesz duże ilości danych zapisanych w kilkunastu tabelach oraz wstawiasz jeden wiersz, informujący o wykonaniu wszystkich operacji, do tabeli znajdującej się w bazie danych na zdalnym serwerze. Połączenie między serwerami jest mocno obciążone i zdarza się, że czas nawiązywania sesji i przesyłania danych pomiędzy serwerami wielokrotnie się wydłuża. Co zrobić, aby w przypadku zgłoszenia przez procedurę błędu braku połączenia ze zdalnym serwerem nie trzeba było ponownie wykonywać długotrwałych modyfikacji danych?

Skorowidz

I

J

K

L

Ł

M

N

O

P

R

S

T

U

W

Z

Praktyczny kurs SQL

Wydanie II

Bazy danych są dosłownie wszędzie. Trudno sobie dziś bez nich wyobrazić funkcjonowanie nowoczesnej biblioteki, choćby najmniejszego sklepu internetowego, biura rachunkowego czy nawet niewielkiego serwisu WWW. Użytkownicy korzystający z baz danych często nie mają nawet pojęcia, w jaki sposób odbywa się dostęp do informacji i jaki mechanizm jest za to odpowiedzialny. Na ignorancję tę nie mogą sobie jednak pozwolić osoby odpowiedzialne za tworzenie i konserwowanie baz danych oraz zarządzanie nimi. Powinny one znać przynajmniej jeden z popularnych serwerów bazodanowych i sprawnie posługiwać się językiem SQL stanowiącym standardowe narzędzie komunikacji z relacyjnymi bazami.

Jeśli pragniesz dołączyć do ekskluzywnego grona administratorów baz danych lub chcesz zostać programistą aplikacji bazodanowych, lecz przeszkadza Ci brak znajomości SQL-a, sięgnij po książkę „Praktyczny kurs SQL. Wydanie II". W prosty i przystępny sposób prezentuje ona podstawowe pojęcia i zasady rządzące relacyjnym modelem baz danych, a także najważniejsze cechy i konstrukcje języka SQL oraz metody ich wykorzystywania. Lektura książki umożliwi Ci poznanie instrukcji odpowiedzialnych za odczytywanie danych z bazy i ich zapisywanie oraz modyfikację, jak również tworzenie baz i zmianę ich struktury. Poznasz też sposoby tworzenia ról i kont użytkowników oraz zarządzania ich uprawnieniami. Twoją wiedzę ugruntują praktyczne zadania kończące każdy rozdział, a zamieszczone na końcu książki rozwiązania pomogą skorygować ewentualne błędy.

- **Teoretyczne podstawy funkcjonowania baz danych**
- **Historia języka SQL i obowiązujące standardy zapytań**
- **Odczytywanie, przeszukiwanie, łączenie i grupowanie danych**
- **Korzystanie z podzapytań**
- **Zapisywanie, modyfikacja i usuwanie danych**
- **Transakcje i równoległy dostęp do danych**
- **Tworzenie baz danych i modyfikacja ich struktury**
- **Korzystanie z widoków i indeksów**
- **Zarządzanie użytkownikami, rolami i prawami dostępu do baz danych**

Dowiedz się, jak tworzyć relacyjną bazę danych i zarządzać nią za pomocą języka SQL.

helion.pl
księgarnia internetowa

Nr katalogowy: 6224

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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

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

Cena 47,00 zł

ISBN 978-83-246-3373-9

Informatyka w najlepszym wydaniu