AUTOMATYKA • 2008 • Tom 12 • Zeszyt 1

Mirosław Kasper*

Zastosowanie blokad, poziomów izolacji oraz protokołów zarządzania konkurencyjnością do zwiększania wydajności i skalowalności systemów baz danych

1. Wprowadzenie

Wzrost ilości danych gromadzonych, przechowywanych i przetwarzanych we współczesnych systemach informatycznych wymusił konieczność stosowania coraz bardziej rozbudowanych i złożonych systemów. Składają się one najczęściej z baz centralnych oraz współpracujących z nimi aplikacji. Stosowane w obrębie jednego przedsiębiorstwa bazy mogą działać na różnych platformach sprzętowych, jak również mogą pochodzić od wielu dostawców (Oracle, IBM DB2, MS SQL Server, Postgres, …), podczas gdy aplikacje w takim systemie wymagają nie tylko danych znajdujących się w bazach bezpośrednio z nimi powiązanych, ale muszą również korzystać z aktualnych i spójnych danych pochodzących z baz obsługiwanych przez inne aplikacje całego systemu. Powoduje to konieczność stosowania różnych mechanizmów wymiany danych pomiędzy poszczególnymi aplikacjami całego systemu. W niniejszym artykule przedstawione zostały trudności występujące podczas realizacji złożonych systemów przetwarzania danych współpracujących z relacyjnymi bazami danych OLTP (*Online Transaction Processing*), zarówno w systemach wykorzystujących bazy centralne, jak i w tych z bazami rozproszonymi. Znajomość zagadnień opisywanych w artykule pozwala uniknąć niepotrzebnych problemów podczas tworzenia i użytkowania systemów informatycznych współpracujących z bazami danych, gdyż duża część tych utrudnień jest wynikiem fizycznych ograniczeń zrealizowanych systemów baz danych.

W sekcji 2 artykułu omówione zostały ograniczenia występujące w systemach zarządzania bazami danych (SZBD), zarówno w systemach ze zcentralizowaną bazą, jak również w systemach z bazami rozproszonymi. Druga sekcja przedstawia metody wykorzystywane w celu wyeliminowania konfliktów w dostępach do współdzielonych zasobów podczas wy-

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

konywania transakcji. Sekcja 4 zawiera informacje dotyczące wzajemnego wpływania na siebie transakcji w zależności od stosowanego poziomu izolacji w SZBD. Sekcja 5 uzupełnia wcześniej opisane mechanizmy wykorzystywane do zarządzania transakcjami o dodatkowe, stosowane tylko w systemach z rozproszonymi bazami danych. Ostatnia sekcja zawiera podsumowanie omawianych w artykule zagadnień.

2. Ograniczenia w systemach zarządzania bazami danych

W celu zapewnienia komunikacji i wymiany danych pomiędzy aplikacjami typowego systemu informatycznego można korzystać z wielu mniej lub bardziej złożonych metod. Do najprostszych i najmniej wydajnych metod zaliczyć można wymianę plików z danymi pomiędzy aplikacjami. W bardziej złożonych i skomplikowanych systemach wykorzystuje się możliwość wykonywania transakcji rozproszonych, obejmujących jednocześnie wiele baz danych, używając do tego celu zarówno mechanizmów obsługi transakcji rozproszonych udostępnianych przez system zarządzania bazami danych, lub poprzez zastosowanie warstw pośrednich zarządzających transakcjami wykonywanymi na wielu bazach znajdujących się zwykle w różnych lokalizacjach.

Oczywiste jest, że im bardziej złożony jest system, tym trudniejsza jest jego realizacja i utrzymanie. Należy jednak pamiętać, że nawet w prostych systemach wykorzystujących bazę centralną występują opisane w dalszej części artykułu ograniczenia, wynikające z obowiązujących standardów czy też założeń producentów systemów zarządzania bazami danych.

Tworzenie wydajnych aplikacji współpracujących z bazami danych wymaga znajomości mechanizmów kontroli współbieżności, do których zaliczane są blokady, semafory oraz poziomy izolacji dla transakcji. Ich zastosowanie pozwala uzyskać większą kontrolę nad sposobem wykonywania transakcji, a także nad kolejnością ich wykonywania, pozwala także zarządzać wzajemnym wpływem transakcji konkurujących ze sobą o zasoby systemowe. Ponadto zastosowanie tych mechanizmów pozwala tworzyć wydajne aplikacje pomimo wspomnianych wcześniej ograniczeń SZBD, występujących w szczególności w systemach OLTP wykonujących jednocześnie bardzo duże ilości konkurencyjnych transakcji, które potrzebują dostępu do zasobów znajdujących się w pamięci lub na dyskach SZDB.

Jak wszyscy wiemy, współczesne relacyjne SZBD wspierają wykonywanie konkurujących wzajemnie ze sobą transakcji, umożliwiając aplikacjom nawiązywanie wielu jednoczesnych połączeń do tej samej bazy danych. Zrównoleglenie wykonywania transakcji wiąże się z trudnościami związanymi z jednoczesnym dostępem wielu użytkowników do tych samych zasobów systemu. Opracowanych zostało wiele algorytmów wykorzystywanych do realizacji współbieżności w SZBD, a także w standardzie SQL określono sposób, w jaki transakcje takie powinny być wykonywane. W zależności od sposobu działania SZBD wystąpić mogą pewne ograniczenia, powodujące, że system nie będzie działał tak, jak byśmy tego od niego żądali.

Poniższe przykłady przedstawiają sytuacje, kiedy takie zdarzenia mogą wystąpić, przy czym aplikacje A i B wykonują swoje operacje w ramach pojedynczych transakcji.

- Utracone aktualizacje (*lost updates*). Dwie aplikacje A i B jednocześnie odczytują ten sam wiersz z bazy danych, a następnie wykorzystują wcześniej odczytaną wartość w obliczeniach nowej wartości dla elementu tego wiersza. Jeśli aplikacja A zmodyfikuje wiersz nową wartością, a następnie aplikacja B zmodyfikuje ten sam wiersz swoją wyliczoną wartością, to w takim przypadku modyfikacja wykonana przez aplikację A zostaje utracona.
- Dostęp do danych niezatwierdzonych (*access to uncommited data*). Aplikacja A modyfikuje dane w bazie, a aplikacja B odczytuje te dane zanim zostaną one zatwierdzone. Jeżeli zmodyfikowane przez transakcję A dane nie zostaną zatwierdzone, ale wycofane, wówczas obliczenia wykonywane w aplikacji B wykonywane są na niezatwierdzonych i najprawdopodobniej niepoprawnych danych.
- Niepowtarzalne odczyty (*non-repetable reads*). Aplikacja A odczytuje wiersz danych z bazy, a następnie przechodzi do wykonywania kolejnych instrukcji SQL. Podczas wykonywania tych instrukcji aplikacja B modyfikuje lub usuwa ten wiersz a następnie zatwierdza zmiany. Jeśli później aplikacja A próbuje ponownie odczytać, zmodyfikować lub usunąć oryginalny wiersz, otrzymuje zmodyfikowane dane lub informację, że wiersz taki już nie istnieje (pomimo to, że aplikacja A przez cały czas wykonuje tę samą transakcję!).
- Odczyt danych nieistniejących (*phantom read*). Odczyt taki pojawia się wówczas, gdy aplikacja A odczytuje dane z tabeli, a w tym samym czasie aplikacja B dodaje nowe wiersze do tej tabeli, również spełniające warunki instrukcji w zapytaniu aplikacji A. W wyniku tego aplikacja A podczas ponownego wykonania tej samej instrukcji SQL zwraca inne wyniki uwzględniające dodatkowe dane (phantoms), mimo że wiersze spełniające warunki zapytanie nie zostały zmodyfikowane.

Zapobieganie wymienionym zachowaniom SZBD można osiągnąć poprzez odpowiednie zastosowanie w aplikacjach mechanizmu blokad oraz wykorzystanie poziomów izolacji dla transakcji.

3. Eliminowanie konfliktów dostępu do zasobów – blokady

Blokada (*lock*) jest mechanizmem służącym do zapobiegania występowaniu konfliktów w dostępie do zasobów w środowisku współdzielonym przez wielu użytkowników. Transakcja, przed uzyskaniem dostępu do określonego zasobu, musi wcześniej zgłosić żądanie w celu uzyskania blokady na ten zasób. Po zakończeniu zadania transakcja ta musi zwolnić zajęte zasoby, aby były one dostępne dla innych transakcji. W praktyce blokady realizuje się poprzez wykorzystywanie mechanizmów kolejkowania oraz stosowanie semaforów.

W zależności od sposobu dostępu do zasobów blokady mogą być wyłączne i dzielone. W przypadku uzyskania przez transakcję blokady wyłącznej na określone zasoby, zasoby te przestają być dostępne dla wszystkich wykonywanych równolegle transakcji. Blokada wyłączna może być uzyskana tylko przez jedną transakcję, podczas gdy blokady dzielone mogą być zakładane jednocześnie przez kilka transakcji, których liczba zależy od specyfiki aplikacji.

Blokady mogą być zakładane automatycznie przez SZBD lub bezpośrednio przez użytkownika. Blokady bezpośrednio zakładane są podczas wykonywania modyfikacji danych, przy pobieraniu danych (*SELECT FOR UPDATE*), w trakcie wykonywania czynności administracyjnych (*LOCK TABLE*), a także podczas przetwarzania blokowego danych.

Rysunek 1 przedstawia sposób, w jaki zakładana jest automatyczna blokada podczas wstawiania danych do tabeli przez SZBD Oracle. Zastosowanie automatycznej blokady[1)] powoduje, że dane wstawiane w pierwszej sesji użytkownika A widoczne są w sesji drugiej B dopiero po zatwierdzeniu transakcji w sesji A. Identyfikator sesji użytkownika A to 146, natomiast użytkownika B – 141. Początkowo tabela *lock_tbl* zawiera 5 rekordów, a w sesjach obydwu użytkowników nie ma żadnych blokad (informacje o blokadach widoczne są w widoku *v$lock*). Po dodaniu danych przez użytkownika A dane te widoczne są tylko w sesji A, oraz automatycznie założona zostaje blokada. Po zatwierdzeniu zmian (*commit*) dane zostają zapisane i są dostępne ze wszystkich sesji, natomiast założona wcześniej blokada zostaje zwolniona po zatwierdzeniu danych.

Samo zastosowanie blokad nie gwarantuje zachowania poprawności logiki aplikacji. W przykładzie na rysunku 2 widzimy dwie konkurencyjne transakcje A i B modyfikujące ten sam wiersz. Transakcja w sesji użytkownika B oczekuje na zwolnienie blokady wiersza, (blokada założona automatycznie po jego modyfikacji w sesji A) i dopiero po zwolnieniu blokady operacja modyfikacji w B może zostać zakończona. Przykład ten pokazuje, że pomimo zastosowania blokady dane modyfikowane przez wykonywaną współbieżnie transakcję w sesji B nadpisują dane wcześniej zatwierdzone przez transakcję A. W przykładzie tym początkowo w tabeli *lock_tbl* znajduje się 5 wierszy i taki stan jest widoczny w sesjach A i B. W chwili, gdy transakcja A próbuje zmodyfikować dane, okazuje się, że zostały one wcześniej zablokowane w wyniku wykonywanej modyfikacji tych samych danych w sesji A. Transakcja w sesji B zostaje wstrzymana do czasu zwolnienia blokady założonej na żądane przez nią zasoby. Podczas oczekiwania na zwolnienie zasobów użytkownik B nie może wykonywać żadnych operacji, i sesja użytkownika pozostaje w „zawieszenie" do momentu zatwierdzenia lub wycofania zmian przez sesję blokującą. Po zatwierdzeniu danych w sesji A widoczne są zmiany wykonane w tej sesji, natomiast w sesji B zwracana jest informacja o zmodyfikowaniu jednego wiersza. Zatwierdzenie modyfikacji wykonanej w sesji B powoduje, że zmodyfikowane równolegle dane w transakcji A zostają nadpisane nową wartością użytkownika B.

[1)] Dane przedstawione zostały na podstawie SZBD Oracle 10.2.0.3 (TX – blokada).

```
(A) SQL> select username, sid from v$session
         where audsid = userenv('SESSIONID');

USERNAME                              SID
------------------------------ ----------
A                                     152

(A) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
```

```
(B) SQL> select username, sid from v$session
         where audsid = userenv('SESSIONID');

USERNAME                              SID
------------------------------ ----------
B                                     148

(B) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
```

```
(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
no rows selected

(A) SQL> insert into lock_tbl values (6,'InterBase');
1 row created.

(A) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
      6 InterBase
```

```
(B) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
```

```
(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);

       SID TY        ID1      LMODE    REQUEST
---------- -- ---------- ---------- ----------
       152 TM      67405          3          0
       152 TX     327691          6          0

(A) SQL> commit;
Commit complete.

(A) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
      6 InterBase
```

```
(B) SQL> select * from lock_tbl;

   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server
      6 InterBase
```

```
(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
no rows selected
```

Rys. 1. Blokada automatyczna podczas wstawiania danych do tabeli w bazie Oracle. Wstawianie danych przez użytkownika A powoduje wystąpienie blokady i zmiany widoczne są w sesji użytkownika B dopiero po zatwierdzenie zmian w sesji A i automatycznym zwolnienie blokady

```
(A) SQL> select username, sid from v$session
         where audsid = userenv('SESSIONID');
USERNAME                         SID
------------------------------ ----------
A                                 152

(B) SQL> select username, sid from v$session
         where audsid = userenv('SESSIONID');
USERNAME                         SID
------------------------------ ----------
B                                 148

(A) SQL> select * from lock_tbl;
   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server

(B) SQL> select * from lock_tbl;
   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Postgres-R
      5 MS SQL Server

(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
no rows selected

(A) SQL> update lock_tbl set name = 'Sybase'
         where IDX = 4;
1 row updated.

(B) SQL> update lock_tbl set name = 'MySQL'
         where IDX = 4;


(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
       SID TY        ID1      LMODE    REQUEST
---------- -- ---------- ---------- ----------
       148 TX     393263          0          6
       152 TM      67405          3          0
       148 TM      67405          3          0
       152 TX     393263          6          0

(A) SQL> commit;
Commit complete.

 1 row updated.

(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
       SID TY        ID1      LMODE    REQUEST
---------- -- ---------- ---------- ----------
       148 TM      67405          3          0
       148 TX      65563          6          0

(A) SQL> select * from lock_tbl;
   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 Sybase
      5 MS SQL Server

(B) SQL> commit;
Commit complete.

(sys) SQL> select sid,type,id1,lmode,request from v$lock where sid in (148,152);
no rows selected

(A) SQL> select * from lock_tbl;
   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 MySQL
      5 MS SQL Server

(B) SQL> select * from lock_tbl;
   IDX NAME
------- ----------------
      1 Oracle
      2 Informix
      3 DB2
      4 MySQL
      5 MS SQL Server
```

Rys. 2. Nadpisywanie danych przez równolegle wykonywane transakcje pomimo blokad

4. Wzajemny wpływ transakcji na siebie – poziomy izolacji

Poziomy izolacji transakcji SQL (*transaction isolation levels*) określane są na podstawie umożliwiania (lub nie umożliwiania) przez te transakcje występowania trzech zjawisk:

1) Brudny odczyt (*dirty read*) – dane niezatwierdzone w transakcji A mogą być odczytane w transakcji B.
2) Niepowtarzalny odczyt (*nonrepetable read*) – jeśli dane odczytane w czasie $t1$ przez transakcję A zostaną zmodyfikowane i zatwierdzone przez transakcję B, to ponowny ich odczyt w transakcji A w czasie $t2$ zwraca zmienione dane.
3) Złudny odczyt (*phantom read*) – jeśli do tabeli, z której transakcja A odczytuje są dane w czasie $t1$, zostaną dodane nowe rekordy w transakcji B, może to wpłynąć na wyniki zwracane w transakcji A przez to samo zapytanie w czasie $t2$.

Standard SQL nie narzuca SZDB określonego schematu blokowania, ale opisuje te poziomy w związku z tymi trzema zjawiskami – pozwalając jednocześnie na istnienie wielu różnorodnych mechanizmów blokowania i kontroli zgodności (*locking*/*concurrency mechanisms*).

Możliwość występowania tych zjawisk dla poszczególnych poziomów izolacji przedstawia tabela 1.

Tabela 1
Poziomy izolacji w zależności od sposobu odczytywania danych

Poziom izolacji	*Dirty read*	*Nonrepetable read*	*Phantom read*
READ UNCOMMITTED	Tak	Tak	Tak
READ COMMITTED	–	Tak	Tak
REPEATABLE READ	–	–	Tak
SERIALIZABLE	–	–	–

W większości stosowanych w praktyce systemów zarządzania bazami danych zaimplementowane są cztery poziomy izolacji. W niektórych systemach zdefiniowane są również dodatkowe, opisywane wcześniej niestandardowe poziomy, natomiast w części systemów nie wszystkie poziomy zostały zrealizowane. Tabela 2 przedstawia poziomy izolacji w najczęściej stosowanych systemach zarządzania bazami danych.

Powstało wiele interfejsów programistycznych do obsługi transakcji, umożliwiających modyfikację poziomu izolacji dla sesji użytkownika. Jednym z nich jest sterownik JDBC, który poprzez interfejs Connection umożliwia zarówno odczytywanie poziomu izolacji (metoda getTransactionIsolation) oraz jego zmianę (metoda setTransactionIsolation). Rysunek 3 przedstawia przykładowy fragment programu wykonującego modyfikację danych z ustawieniem poziomu izolacji dla wykonywanej transakcji na SERIALIZABLE.

Poziom izolacji można odczytać bezpośrednio ze słownika bazy danych (np. *SELECT ISOLATION FROM SYSCAT.PACKAGES* w DB2, czy też przeglądnięcie widoków *v$transaction* oraz *v$session* pod kątem bieżącej sesji w przypadku SZBD Oracle). Zmiana pozio-

mu izolacji może być wykonywana bezpośrednio w bazie danych i obejmować całą bazę (np. *SET CURRENT ISOLATION = UNCOMITED READ* w DB2, *SET TRANSACTION ISOLATION LEVEL SERIALIZABLE* w Oracle) lub tylko bieżącą sesję (np. *ALTER SESSION SET ISOLATION LEVEL = SERIALIZABLE* w Oracle).

```
Connection con = DriverManager.getConnection(connectionURL, userName,password);
con.setAutoCommit(false);
con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
Statement st = con.createStatement();
statementStr = "update table_name set col1 = " + newValue + " where pk = 1";
out(statementStr);
st.executeUpdate(statementStr);
con.commit();
```

Rys. 3. Przykład programu wykorzystującego JDBC do ustawiania poziomu izolacji dla transakcji modyfikacji danych

Tabela 2
Poziomy izolacji w zależności od sposobu odczytywania danych

RDBMS	Domyślny poziom	Dozwolone poziomy
Oracle	COMMITTED READ	READ COMMITTED SERIALIZABLE niestandardowy READ ONLY
SQL Server	COMMITTED READ	UNCOMMITTED READ COMMITTED READ REPEATABLE READ SERIALIZABLE niestandardowy SNAPSHOT
MySQL (InnoDB)	REPEATABLE READ	UNCOMMITTED READ COMMITTED READ REPEATABLE READ SERIALIZABLE
Informix	COMMITTED READ	UNCOMMITTED READ COMMITTED READ REPEATABLE READ SERIALIZABLE

W przypadku niewłaściwie zaprojektowanej aplikacji można doprowadzić do sytuacji, w której użytkownik przez długi czas będzie oczekiwał na zakończenie swojej transakcji i zwrócenie wyników, a otrzymane wyniki i tak nie będą poprawne. Taka sytuacja może wystąpić w przypadku, gdy poziom izolacji ustawiony jest na COMMITED READ. Jeżeli w sesji A generowany jest raport na podstawie danych z tabeli i jednocześnie sesja B zmo-

dyfikuje te dane w tabeli, które jeszcze nie zostały pobrane przez sesję A, to wówczas sesja tworząca raport zostanie zablokowana do momentu zatwierdzenia transakcji modyfikującej dane. Jeśli dodatkowo transakcja B zmodyfikuje również dane odczytane wcześniej przez transakcję tworzącą raport A, to pomimo wydłużonego oczekiwania na wynik użytkownik może otrzymać nieprawidłowy raport uwzględniający tylko część zmian. W celu zapobiegania opisanej powyżej sytuacji, systemy SZDB udostępniają mechanizmy umożliwiające odczytywanie danych aktualnych w chwili uruchomienia transakcji. Dane odczytywane są wówczas z „obrazu" (*snapshot*) przedstawiającego stan danych w momencie uruchomienia transakcji, a więc nie uwzględniają zmian wykonywanych przez równoległe transakcje – nawet tych, które zostały zatwierdzone.

Dzięki wykorzystaniu mechanizmu *snapshot* (np. *temdb* w SQL Server 2005, *undo tablespace* w Oracle) można wykonywać odczyty *REPEATABLE READ* bez konieczności blokowania innych użytkowników. W przypadku, gdy transakcja odwołuje się do wiersza zmodyfikowanego przez inną transakcję, w wyniku uwzględnione są dane z chwili, gdy transakcja była uruchamiana.

5. Dodatkowe ograniczenia w systemach rozproszonych

Ograniczenia występujące w przypadku pojedynczych instancji SZBD, mają oczywiście również podczas wykonywania transakcji rozproszonych, które obejmują przynajmniej dwie bazy danych najczęściej znajdujące się w różnych lokalizacjach. Blokowanie zasobów oraz dłuższe niż w przypadku bazy centralnej czasy odpowiedzi rozproszonych baz w połączeniu z opisanymi ograniczeniami SZBD bardzo komplikują wykonywanie transakcji rozproszonych. Często transakcję te mogą blokować się wzajemnie w różnych bazach biorących udział w transakcji, powodując wydłużenie czasu ich odpowiedzi, a nawet uniemożliwiając ich zakończenie.

Do realizacji transakcji rozproszonych opracowano różne algorytmy i protokoły, z których do najpopularniejszych należą protokoły wielofazowego zatwierdzania 2-fazowy (*Two-Phase Commit Protocol*) i 3-fazowy (*Three-Phase Commit Protocol*). Protokół 3-fazowego zatwierdzania wymaga przesyłania większej ilości komunikatów niż protokół 2-fazowego zatwierdzania i jak pokazuje [9] protokół 2-fazowego zatwierdzania jest wydajniejszy.

W implementowanych systemach wykorzystuje się również metodę 2-Phase Locking, która zapewnia serializację (wykonywanie transakcji w taki sposób, jak gdyby były one wykonywane kolejno).

Protokoły działające w oparciu o metodę 2-Phase Locking wymagają jednoczesnego stosowania blokad na wszystkich wierszach tabel lub tabelach objętych przez wykonywaną transakcję. Blokady te wymagają odpowiedniego zarządzania w momencie ich zwalniania, zarówno po poprawnym zakończeniu transakcji, jak też w przypadku jej wycofania lub niepoprawnego zakończenia. W przypadku nieprawidłowego zakończenia transakcji (np. ze względu na błąd w jednym z węzłów biorących udział w transakcji) często zasoby takie pozostają zablokowane przez dłuższy czas. Blokady takie mogą być zwolnione dopiero po

upływie określonego czasu (*timeout*) lub też po wykonaniu odpowiednich czynności przez osobę zarządzającą systemem. Jednak takie oczekiwania na zwolnienie blokad powoduje znaczny spadek wydajności całego systemu.

6. Podsumowanie

Stosowanie blokad w aplikacjach klienckich jest stosunkowe proste i ich realizacja nie wymaga dodatkowych zasobów po stronie aplikacji. Dzięki odpowiedniemu zastosowaniu blokad oraz poziomów izolacji transakcji można tworzyć aplikacje zabezpieczające przed opisywanymi ograniczeniami SZDB. Należy jednak pamiętać o tym, że jeśli użyjemy bardziej restrykcyjnego poziomu izolacji oraz zwiększymy liczbę blokad, to wówczas zamiast współbieżnego wykonywania operacji wielu użytkowników, uzyskujemy kolejkowanie zadań i długie oczekiwania na ich zakończenie.

Jak powszechnie wiadomo, systemy rozproszone są dużo bardziej skomplikowane od systemów scentralizowanych i opracowano dla nich wiele algorytmów i protokołów do obsługi transakcji rozproszonych. Niektóre zaimplementowane systemy, jak np. Bea Tuxedo czy IBM WebSphere, umożliwiają wykonywanie transakcji rozproszonych wykorzystując w tym celu protokół 2-fazowego zatwierdzania. Jednocześnie trwają prace nad utworzeniem algorytmu umożliwiającego zwiększenie wydajności wykonywania transakcji rozproszonych oraz skalowalności całego systemu, szczególnie w przypadku wykonywania bardzo dużych ilości transakcji znacznie zróżnicowanych pod względem sposobu ich wykonywania przez SZBD. Wiele opracowywanych ostatnio metod opiera się o zastosowanie dodatkowych modułów pośredniczących (*middleware*). Taki system pośredniczący ma za zadanie zarządzanie sposobem wykonywania transakcji odpowiednio je kolejkując i przekazując do wykonania do SZBD, zapewniając przy tym cztery cechy transakcji zawarte w modelu „ACID" (atomowość, spójność, integralność i trwałość). Obecnie wykorzystywane rozwiązania oparte o *middleware* są zwykle prostsze w realizacji niż te korzystające bezpośrednio z mechanizmów SZBD, jednak należy pamiętać o tym, że ich stosowanie często wiąże się z utratą wydajności całego systemu.

Literatura

[1] Wrembel R., Bębel B., *Oracle. Projektowanie rozproszonych baz danych*. Helion, 2003.
[2] Conolly T., Begg C., *Database Systems – A practical Approach to Design, Implementation, and Management*. Adison-Wesley, 2002.
[3] Kyte T., *Expert Oracle Database Architecture*. Apress, 2005.
[4] Whitehorn M., Marklyn B., *Inside Relational Databases*. 2nd ed., Springer, 2001.
[5] Eaton C., Cialini E., *High Availability Guide for DB2(R) (Paperback)*. IBM Press, 2004.
[6] Brown E.L., *SQL Server 2005. Wyciśnij wszystko*. Helion, 2005.
[7] Kalen D., *Inside Microsoft SQL Server 2000*. Microsoft Press, 2001.
[8] Worsley J.C., Drake J.D., *PostgreSQL. Praktyczny przewodnik*. Helion, 2002.
[9] Sikorska M., *Praktyczne porównanie mechanizmów 2PC i 3PC dla rozproszonych baz danych*.
[10] Bernstein P.A., Hadzilacos V., Goodman N., *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[11] Thomasian A., *Database Concurrency Control: Methods, Performance, and Analysis*. Springer, 1996.
[12] ANSI X3.135-1992 *SQL92 standard. Information Technology -Database Language SQL.* ANSI.
[13] Armendariz J.E., Gonzalez de Mendivil J.R., Munoz-Escoi F.D., *A lock based algorithm for concurrency control and recovery in a middleware replication software architecture.* IEEE Computer Society, 2005.
[14] Bernstein P.A., Meichun Hsu, Mann B., *Implementing recoverable requests using queues.* ACM Press, 1990.
[15] Kifer M., Bernstein A., Lewis P.M., *Database Systems. An Application-Oriented Approach. Complete Version*. Pearson Addison Wesley, 2006.
[16] Carey M.J., Livny M., *Conflict detection tradeoffs for replicated data*. ACM Press, 1991.
[17] Papadimitriou C., *The theory of database concurrency control*. Computer Science Press, 1986.
[18] Kindler E., *Serializability, Concurrency Control, and Replication Control*. Springer, Berlin, 2000.
[19] Stonebraker M., *Concurrency Control and Consistency of Multiple Copies of Data in Distributed Ingres*. IEEE Press, 1979.
[20] Silberschatz A., Korth H.F., Sudarshan S., *Database System Concepts.* 5th ed., McGraw-Hill, 2006.
[21] Thomas R.H., *A majority consensus approach to concurrency control for multiple copy databases*. ACM Press, 1979.
[22] Gray J., Reuter A., *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 1992.
[23] Kornacker M., Mohan C., Hellerstein J.M., *Concurrency and recovery in generalized search trees*. ACM Press, 1997.
[24] Skeen D., *A Formal Model of Crash Recovery in a Distributed System.* IEEE Transactions on Software Engineering, 1983.
[25] Moss E., *Nested Transactions: An Approach to Reliable Distributed Computing.* The MIT Press, 1985.
[26] Mohan C., Narang I., *Efficient Locking and Caching of Data in the Multisystem Shard Disks Transaction Environment.* Springer-Verlag, London, 1992.