ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: INFORMATYKA z. 38 Nr kol. 1451

Aleksandra WERNER
Politechnika Śląska, Instytut Informatyki

ZNACZNIKOWANIE CZASU WAŻNOŚCI DANYCH TEMPORALNEJ BAZY DANYCH

Streszczenie. W artykule przedstawiono takie sytuacje, w których sekwencja operacji, dokonywanych w pewnym przedziale czasu na bazie danych w ramach pojedynczej transakcji, może wpływać na własności tej transakcji. Zaprezentowano również możliwe metody postępowania w przypadku wystąpienia konieczności weryfikacji krotek oraz przedstawiono wady i zalety poszczególnych rozwiązań.

TIMESTAMPING OF VALID-TIME OF DATA STORED IN TEMPORAL DATABASE

Summary. Several situations, where a sequence of operations over time within a single transaction can violate its properties are presented in this paper. Possible methods of behavior in tuples revisiting necessity and – additionally – merits and drawbacks of describing approaches, are also contained in this article.

1. Definicje podstawowych pojęć

1.1. Temporalne bazy danych

W związku z tym, że analiza danych zmieniających się w czasie (tzw. danych temporalnych) dostarcza więcej informacji potrzebnych do podejmowania taktycznych decyzji lub decydujących o strategii przedsiębiorstw, w ostatniej dekadzie odnotowano znaczny wzrost zainteresowania możliwością przedstawienia w systemach baz danych wiedzy umiejscowionej w czasie.

System Zarządzania Temporalnymi Bazami Danych (ang. TDBMS) ma zwykle strukturę warstwową i jest rozszerzeniem konwencjonalnego systemu DBMS o dostarczenie wsparcia

do modyfikacji i zapytań danych temporalnych. W temporalnej bazie danych zapisywane są fakty (rozumiane jako pewne ustalone stany rzeczywistości), które niosą ze sobą informacje czasowe. Ze względu na sposób reprezentacji wspieranego przez bazę danych czasu możemy wyodrębnić:

- fakty zapisywane z czasem wiarygodności (ang. valid time), określającym, kiedy fakt był/jest prawdziwy w modelowanej rzeczywistości,
- fakty zapisywane z czasem transakcyjnym (ang. transaction time), określającym, kiedy fakt został zapamiętany w bazie (był prawdziwy w bazie). Czas transakcyjny rozpoczyna się w momencie wprowadzania danych do bazy, a kończy, gdy dane zostają skorygowane.

W artykule rozpatruje się aspekt czasu transakcyjnego. Baza temporalna ze wsparciem czasu transakcyjnego zapamiętuje dane przez skojarzenie z krotką okresu czasu, w którym dane były prawdziwe w bazie. Ten okres czasu może zawierać wartość specjalną UC (w dosłownym tłumaczeniu: *do nadal*), oznaczającą czas bieżący.

Nasze rozważania oprzemy na temporalnej relacji ZAROBKI, stworzonej na bazie modelu stanu[1] w języku ATSQL:

```
CREATE TABLE ZAROBKI (nrp INTEGER, id_wydziału CHAR(4), pensja INTEGER)
                      AS EVENT STATE;
```

Użyta w powyższym zapisie fraza AS EVENT STATE oznacza, że do relacji zostaną automatycznie dodane dwie kolumny: T-start i T-stop, oznaczające odpowiednio: początek i koniec okresu czasu, w którym wartości atrybutów danej krotki były aktualne w bazie.

Zmiany zawartości tabeli ZAROBKI, będące skutkiem wykonania przykładowych operacji, ilustruje rys. 1.

```
    Data bieżąca: '1998-05-07'
① INSERT INTO ZAROBKI VALUES (23, 'RAU1', 1100);
    Data bieżąca: '1998-05-11'
② INSERT INTO ZAROBKI VALUES (31, 'RMT', 1500);
    Data bieżąca: '1998-05-17'
③ COMMIT;
    Data bieżąca: '1998-06-11'
④ INSERT INTO ZAROBKI VALUES (34, 'RMT', 1200);
⑤ DELETE FROM ZAROBKI WHERE nrp = 34;
    Data bieżąca: '1998-07-31'
⑥ UPDATE ZAROBKI SET pensja = 1500 WHERE nrp=23;
```

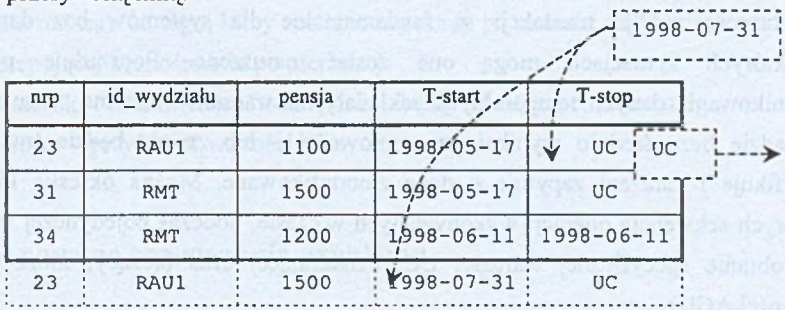
[1] Z definicji: *stan* jest to instancja danych, które istnieją lub trwają przez pewien okres czasu (czyli baza widziana jest jako sekwencja stanów relacji). Każdy zapisany stan jest znacznikowany okresem czasu, w którym stan istnieje.

Wykonanie poleceń oznaczonych symbolami: ①, ③ oraz ④ powoduje wstawienie do tabeli ZAROBKI krotek z danymi w poszczególnych instrukcjach INSERT wartościami atrybutów: nrp, id_wydziału, pensja. Atrybuty znaczników czasu: T-start i T-stop inicjalizowane są czasami odpowiednio: pierwsza krotka czasem: 1998-05-07, druga krotka: 1998-05-11, trzecia: 1998-06-11 oraz wartością UC, reprezentującą czas „do nadal" (nie wyspecyfikowany jawnie punkt czasu, do którego dane zawarte w krotce będą obowiązywać).

Polecenie ③ zatwierdza wykonane w relacji zmiany, a co za tym idzie atrybut T-start w obu krotkach zostaje zmieniony na wartość: 1998-05-17.

Operacja oznaczona symbolem ⑤ nie powoduje utraty informacji o pracowniku nr 34 (zgodnie z założeniem temporalnych baz danych), a jedynie ogranicza okres, w którym wartości atrybutów z danej krotki były obowiązujące w bazie, przez modyfikację wartości atrybutu T-stop z: UC na: 1998-06-11 (czas, w którym wydano polecenie skasowania rekordu).

Aktualizacja danych (polecenie ⑥) powoduje jednoczesne: dodanie do tabeli nowej krotki ze zmodyfikowaną wartością atrybutu pensja, a zarazem zmodyfikowanie, istniejącej w kolumnieT-stop krotki przechowującej poprzednie dane pracownika, wartości UC. Efekt wykonania oraz zakres działania operacji UPDATE ilustrują strzałki oraz prostokąty obramowane przerywaną linią.



Rys. 1. Ilustracja mechanizmu aktualizowania danych w tabeli *ZAROBKI*
Fig. 1. Mechanism of data modification on *ZAROBKI* table

1.2. Transakcja

Transakcja jest elementarną jednostką przetwarzania podobną do procedury. Z punktu widzenia zewnętrznego posiada ona następujące własności (tzw. własności ACID):

- atomowość (ang. Atomicity)

  Transakcja traktowana jest jako jedna jednostka operacyjna; jeżeli transakcja nie kończy wszystkich przewidzianych przez nią akcji, wówczas wszystkie akcje, które zostały przez tę transakcję wykonane, zostają cofnięte,

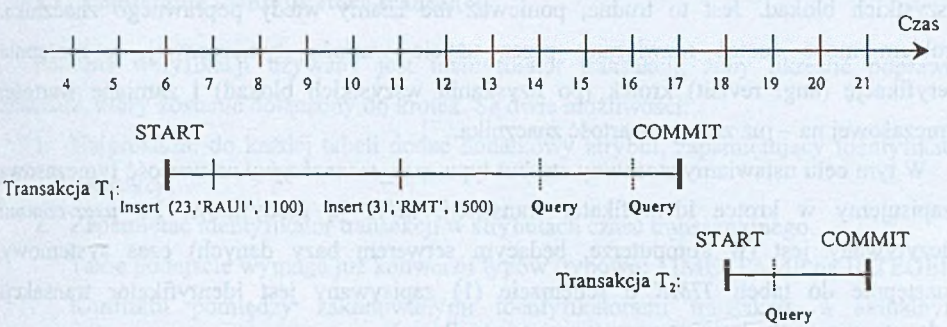
- spójność (ang. Consistency)
  Zarówno przed wykonaniem transakcji, jak i po niej baza danych znajduje się w stanie zgodnym (tzn. w stanie stałej poprawności danych, znajdujących się w bazie). Podczas działania transakcji stan bazy danych może nie być spójny, ale tylko w ramach tej transakcji,
- izolacja (ang. Isolation)
  Transakcja nie uwzględnia działania innych transakcji, czyli widzi bazę danych jako spójną,
- trwałość (ang. Durability)
  Po zakończeniu transakcji jej skutki są trwałe i nie mogą być odwrócone poprzez jakiekolwiek przypadkowe zdarzenie.

Jedną z konsekwencji ACID jest tzw. szeregowalność, która oznacza, że końcowy skutek danego zestawu transakcji nie może zależeć od kolejności, w jakiej te transakcje są wykonywane.

2. Specyfikacja problemu

Własności ACID transakcji są fundamentalne dla systemów baz danych, jednak w niektórych sytuacjach mogą one zostać naruszone. Poprzednie podejścia do znacznikowania danych temporalnych zakładały, że transakcja nie ma czasu trwania – co w zasadzie bezpośrednio wynikało z atomowości – lub że nie będzie transakcji, która modyfikuje i zarazem zapytuje o dane zmodyfikowane. Można określić kilka sytuacji, w których sekwencja operacji wykonywanych w czasie, podczas pojedynczej transakcji, lub złe dobranie specyficznej wartości UC, oznaczającej czas bieżący, może wpływać na własności ACID.

Żeby dostać spójny transakcyjnie obraz modyfikacji bazy danych, wartość UC musi być czasem potwierdzenia transakcji. Zakładając, że mamy transakcję T o czasie trwania od 1998-05-06 do 1998-05-17, podczas której dokonywana jest zmiana danych zapisanych w bazie, zamiast początkowej wartości UC powinna zostać wpisana wartość: 1998-05-17. To – w momencie gdy dane są modyfikowane i wykonywane są na nich zapytania (ang. query) w ramach tej samej transakcji – może powodować problem przedstawiony na rys.2: zapytanie może dotyczyć czasu, który jeszcze nie jest znany (por. rys. 1).



Rys. 2. Możliwy przebieg transakcji w czasie
Fig. 2. Possible transaction temporal proceeding

Należy zapewnić zachodzenie następujących reguł:

1. Fakt, że aktualna transakcja trwa pewien odcinek czasu, nie może być wykrywalny przez jakiekolwiek zapytanie.
2. Zapytanie wykonane dwukrotnie w czasie tej samej transakcji, przy założeniu że nie wystąpiły w tym czasie żadne modyfikacje, musi zwrócić identyczny rezultat.
3. Zapytanie wykonane w ramach pewnej transakcji T_1 nie może dać w odpowiedzi rezultatu różnego od tego, który zostanie uzyskany po zadaniu zapytania w transakcji T_2, następującej bezpośrednio po pierwszej.
4. Te znaczniki czasu, które ostatecznie będą identyczne, nie mogą – nawet przejściowo – przybierać wartości różnych.

3. Proponowane rozwiązanie problemu

Żeby osiągnąć spójny obraz poprzednich stanów bazy, konieczne jest użycie tego samego znacznika czasu dla wszystkich modyfikacji dokonanych podczas transakcji. W przeciwnym przypadku możliwe byłoby wycofanie transakcji do punktu czasu pomiędzy chwilami występowania dwóch modyfikacji, a to z kolei zaprzeczałoby atomowości transakcji. Użyty znacznik musi być wprowadzony po czasie, w którym wszystkie blokady zostaną nabyte (osiągnięte), gdyż inaczej znaczniki nie będą poprawnie odzwierciedlały szeregowego porządku transakcji. Taką wiadomość o uzyskaniu wszystkich blokad uzyskujemy – w konwencjonalnym DBMS – dopiero po osiągnięciu ostatniego polecenia w transakcji (czyli po *user-commit*), jednak w celu umożliwienia transakcji ujrzenia jej własnych modyfikacji, może być konieczne skojarzenie znaczników z krotkami przed osiągnięciem

wszystkich blokad. Jest to trudne, ponieważ nie znamy wtedy poprawnego znacznika. Problem może zostać rozwiązany przez użycie wartości tymczasowej, a następnie weryfikację (ang. revisit) krotek (po uzyskaniu wszystkich blokad) i zamianę wartości tymczasowej na – już znaną – wartość znacznika.

W tym celu ustawiamy stosowny atrybut typu *czas_transakcyjny* na wartość tymczasową i zapisujemy w krotce identyfikator transakcji, która ją modyfikuje. Po *user-commit* odczytywany jest (w komputerze, będącym serwerem bazy danych) czas systemowy, a następnie do tabeli *TIME* o schemacie (1) zapisywany jest identyfikator transakcji i odczytany czas (będący już poprawnym znacznikiem).

TIME (id_transakcji INTEGER, czas_potwierdzenia TIMESTAMP) (1)

W następnym kroku krotki modyfikowane przez transakcję są weryfikowane z użyciem znacznika z tabeli *TIME*. Schemat postępowania przedstawiono w tabeli 1.

Nasuwają się pytania:

1. Jak są skojarzone identyfikatory transakcji z krotkami (systemy wielożytkownikowe)?
2. Jaka jest wartość tymczasowa atrybutów czasu transakcyjnego w krotkach modyfikowanych w czasie transakcji?
3. Jak powinna być reprezentowana wartość UC?
4. Czy modyfikowane krotki muszą być weryfikowane przed potwierdzeniem transakcji?

Tabela 1

Odwzorowanie przykładowych zdań w języku ATSQL na język SQL-92

Zdania ATSQL	Odpowiedniki w SQL-92
INSERT INTO *tabela* VALUES (A_1 ... A_n)	INSERT INTO *tabela* VALUES (A_1 ... A_n, *wart_tymczasowa*, UC)
DELETE FROM *tabela* WHERE *warunek*	UPDATE *tabela* SET T-Stop = *wart_ tymczasowa* WHERE *warunek* AND T-Stop = UC
COMMIT	INSERT INTO Time VALUES (*id_trans*, *czas_bieżący*) UPDATE *tabela* SET T-Start = *znajdź_czas*(*id_trans*) WHERE *krotka_wstawiona_przez_transakcję* (*id_trans*) UPDATE *tabela* SET T-Stop = *znajdź_czas*(*id_trans*) WHERE *krotka_skasowana_przez_transakcję* (*id_trans*) DELETE FROM Time WHERE id_transakcji = *id_trans* COMMIT
SELECT * FROM *tabela*	SELECT * FROM *tabela* WHERE T-Stop = UC
TRANSACTION SELECT * FROM *tabela*	SELECT T-Start, T-Stop, A_1, ..., A_n FROM *tabela*

3.1. Pamiętanie identyfikatora transakcji

Podczas weryfikacji używany jest identyfikator transakcji, żeby określić poprawny znacznik, który zostanie dołączony do krotek. Są dwie możliwości:

1. Najprostsza: do każdej tabeli dodać dodatkowy atrybut, zapamiętujący identyfikator transakcji.
2. Zapamiętać identyfikator transakcji w atrybutach czasu transakcyjnego.
   Takie podejście wymaga już konwersji typów (typowo: TIMESTAMP na INTEGER). Konfliktu pomiędzy zakodowanymi identyfikatorami transakcji a aktualnymi znacznikami można uniknąć, ponieważ znaczniki transakcyjne są większe niż czas stworzenia bazy danych. Tak więc identyfikatory transakcji mogą być zależne od najmniejszego znacznika (zwykle: 0001-01-01); pierwszy identyfikator transakcji jest wtedy odwzorowywany na najmniejszą wartość w domenie czasu, drugi identyfikator transakcji jest odwzorowywany jako kolejna najmniejsza wartość w dziedzinie czasu itd.

Wybór pomiędzy tymi dwoma rozwiązaniami jest rozwiązaniem kompromisowym między kosztami: zajętości przestrzeni i zużytego czasu: konwersja może być użyteczna, ale niezbyt wygodna (w SQL-92 trzeba dokonać konwersji pośredniej przy użyciu typu INTERVAL. Oznacza to, że najpierw dokonujemy konwersji identyfikatora transakcji do typu INTERVAL, a następnie dodajemy najmniejszy znacznik czasu – np. 0001-01-01).

3.2. Znajdowanie wartości tymczasowej atrybutów czasu transakcyjnego

Wartość tymczasowa musi spełniać dwa żądania:

- musi kwalifikować krotkę do bieżącego stanu transakcyjnego (kiedy w klauzuli WHEN pojawi się odniesienie do atrybutów czasu transakcyjnego),
- musi być racjonalną (tzn. zrozumiałą dla zadającego zapytanie użytkownika) wartością (kiedy atrybuty transakcyjne będą użyte w klauzuli SELECT).

Te wymogi stawiają przed nami następujące możliwości wyboru:

- użycie czasu startu transakcji,
- użycie czasu, w którym wartość tymczasowa po raz pierwszy okazała się być potrzebna (tzn. użycie czasu pierwszej modyfikacji),
- użycie różnych wartości – tzn. czasu bieżącego – w czasie trwania transakcji (to rozwiązanie może jednak prowadzić do niepowtarzalnych odczytów w ramach jednej transakcji, spowodowanych dwukrotnym zadaniem identycznego zapytania).

3.3. Reprezentacja wartości UC

Reprezentacja UC jest zupełnie odrębnym zagadnieniem. Wszystkie krotki mają w atrybucie *T-stop*[1] tę wartość tak długo, dopóki nie zostaną skasowane (w sensie logicznym) z bazy. Pole to nie może przyjmować żadnych wartości z przedziału: [*czas_powstania_bazy_danych* , *czas_bieżący*], stąd możliwe są do przyjęcia następujące wartości:

- dowolnie wybrany czas przed powstaniem bazy danych[2],
- największa wartość z zakresu (zwykle: 9999-31-12),
- wartość NULL (zwykle żąda mniej przestrzeni w bazie niż inne znaczniki, ale zarazem może uniemożliwić użycie indeksu).

3.4. Strategie weryfikacji krotek

Ponieważ weryfikacja (rozumiana jako modyfikacja tymczasowej wartości znacznika czasu zapisanego w krotce) wpływa na wydajność systemu, istnieje kilka różnych strategii kontroli i rewizji krotek:

- Eager,
  Dla każdej transakcji poprawny znacznik jest zapisywany natychmiast w czasie user-commit. Realizacji tego rozwiązania można dokonać za pomocą tzw. after-trigerów i jest ono dobre dla częstego zapytywania i modyfikacji znaczników, ale koszt znacząco się zwiększa w sytuacjach, kiedy poprawne znaczniki są rzadko potrzebne.
- Low-system-usage,
  Weryfikacja krotek odbywa się wtedy, gdy nie ma zbyt dużego obciążenia systemu. Tak jest np. w postrelacyjnym systemie Postgres, ale wymaga to śledzenia asynchronicznego procesu opartego na obciążeniu systemu.
- Piggy-backing,
  Najpierw odbywa się sprawdzanie, czy są jakiekolwiek krotki do weryfikacji, a dopiero potem, w razie potrzeby, dokonywana jest weryfikacja.
- Explicitly scheduled revisiting,
  Weryfikacja krotek następuje w ściśle określonym czasie (np. codziennie o 2 w nocy). Podejście jest dobre jedynie przy zapytaniach kompatybilnych z SQL-92 (tzw. Top

[1] Czas, oznaczający koniec okresu, w którym wartości atrybutów danej krotki były aktualne w bazie.

[2] Uwaga: w konsekwencji czas zakończenia transakcji może być mniejszy niż jej czas rozpoczęcia (co może być sprzeczne z nałożonymi ograniczeniami).

Upward Compatible); inne pytania mogą nie być wykonywane poprawnie, jeżeli weryfikacja nie wystąpiła.

- Lazy,

  Weryfikuje się tylko te krotki, które mają niepoprawne znaczniki, a prawidłowa wartość znacznika jest niezbędna do uzyskania poprawnego wyniku zadanego zapytania. Takie rozwiązanie (ze względu na dobry koszt), równolegle z podejściem Low-system-usage, znalazło swoje zastosowanie w postrelacyjnym systemie Postgres.

- Never,

  To podejście zestawia znaczniki w osobnej tabeli, np. o nazwie *TIME*. Jeżeli zapytanie potrzebuje prawidłowego znacznika, znajduje go w tabeli *TIME*. To wymaga łączenia tabeli *TIME* z tabelą temporalną, a to może – dla dużych tabel – znacząco zwiększyć koszty. Rozwiązanie jest użyteczne tylko wtedy, gdy zapytania rzadko dotyczą atrybutów czasu transakcyjnego.

Dla omówionych wyżej rozwiązań weryfikacja może być wykonana z różną granularnością:

- w zależności od pewnego ustalonego czasu,

  Punktem wyjścia staje się zapytanie: znajdowany jest największy czas w zapytaniu i weryfikowane są te krotki, które były wstawione do bazy przed wyspecyfikowanym momentem czasu.

  Takie podejście daje dobre rezultaty, jeżeli zapytanie porównuje czas transakcyjny krotek z pewną stałą czasową (tak jak w zapytaniu (1)) nie znajduje jednak zastosowania w przypadku, gdy w klauzuli *WHERE* zapytania znajduje się porównanie wartości atrybutów czasu transakcyjnego różnych krotek (zapytanie (2)).

```
TRANSACTION SELECT * FROM tabela WHERE                    (1)
            BEGIN (TRANSACTION(tabela)) >= '1990-01-01';

NONSEQUENCED TRANSACTION                                  (2)
SELECT * FROM tabela T1, T2 WHERE
        BEGIN (TRANSACTION (T1)) > END (TRANSACTION (T2));
```

- na poziomie krotek,

  Każda krotka, którą pobiera zapytanie, jest sprawdzana pod kątem konieczności zastosowania weryfikacji. Jeżeli zachodzi taka potrzeba, wykonywana jest weryfikacja.

  Niestety, podobnie jak poprzednie również i to podejście nie jest zbyt ogólne: wydaje się być przeznaczone jedynie dla zapytań typu (2).

- na poziomie tabeli,

  Weryfikowane są wszystkie potrzebne do realizacji zapytania tabele, przy czym przed wykonaniem zapytania dokonuje się również znacznikowania czasem bieżącej transakcji.

  Podejście wydaje się być ogólne, jednak pod warunkiem, że nie zachodzi sytuacja, gdzie tabela ulega częstym modyfikacjom, a weryfikacja odbywa się rzadko (jak to może mieć miejsce w rozwiązaniu Explicitly scheduled revisiting).
- na poziomie bazy,

  Rozwiązanie to różni się od poprzedniego tylko tym, że weryfikowane są wszystkie tabele należące do bazy.

  Również jest to podejście ogólne, tylko – w porównaniu z granularnością na poziomie tabeli – czas odpowiedzi na zapytanie jest znacznie większy.

4. Podsumowanie

W związku z tym, że w czasie wykonywania przez pojedynczą transakcję jej kolejnych operacji, baza temporalna może przejściowo znajdować się w stanie niespójnym, należy wybrać najoptymalniejszą odpowiedź na ewentualne wystąpienie w transakcji zapytania o jej (nieznany jeszcze) czas transakcyjny.

Istnieje kilka możliwych rozwiązań problemu:

- zabronienie zadawania zapytań, które żądają dostępu do znaczników (uwaga: jest to równoznaczne ze znacznym ograniczeniem języka zapytań),
- zgłaszanie błędu semantycznego,
- zwrócenie wartości tymczasowej (rozumianej jako jedna wartość w ramach jednej transakcji) równocześnie z ostrzeżeniem użytkownika, że wyświetlane czasy transakcyjne mogą ulec zmianie po potwierdzeniu transakcji, a następnie weryfikacja krotek.

Wydaje się, że najbardziej poprawna pod kątem semantyki transakcji jest, wymieniona jako ostatnia, taktyka ostrzegania i weryfikacji.

W celu szacunkowego określenia kosztów wykonania weryfikacji stworzono w systemie Oracle 7.3.3 bitemporalną[1] relację z dwoma numerycznymi i czterema czasowymi atrybutami, reprezentującymi czas wiarygodności (v_begin, v_end) i transakcyjny (t_start, t_stop). Każdy z nich stowarzyszono z atrybutem id_trans, specyfikującym zmieniającą dany

[1] Wspierającą zarówno czas transakcyjny jak i czas wiarygodności.

atrybut transakcję. Użyto dziewięciu indeksów w tabeli: jeden oparty na atrybucie numerycznym i po jednym na każdym znaczniku (korzystano z nich podczas modyfikacji) oraz na id_trans (korzystano z niego podczas weryfikacji)[1]. Ilość czasu CPU i liczbę wykonanych operacji wejścia/wyjścia mierzono przez obserwację zawartości tablic dynamicznych słownika bazy danych: v$filestat, v$sysstat oraz v$datafile.

Porównano dwa skrajne algorytmy: Eager i Lazy. Z przeprowadzonych obserwacji wynika, że wielkością krytyczną była liczba modyfikacji w transakcji (liczba transakcji wykonanych pomiędzy weryfikacjami nie wpływała na wynik). Jeżeli liczba modyfikacji oscylowała w granicach: [1, 20], to zarówno czas wydatkowany przez CPU, jak i liczba operacji wejścia/wyjścia były mniejsze dla strategii Lazy. Dla większej niż 20 liczby modyfikacji w transakcji podejście Lazy stawało się od 50% do 100% bardziej kosztowne niż Eager.

Opierając się na oczywistym założeniu, że większa część wykonywanych w transakcji zapytań do bazy nie odwołuje się do jej znaczników czasowych oraz biorąc pod uwagę czasochłonność poszczególnych rozwiązań, możemy przyjąć, że najdogodniejszym sposobem weryfikacji krotek jest taktyka Lazy.

LITERATURA

1. Date C. J.: Wprowadzenie do baz danych. WNT, Warszawa 1981.
2. Ullman J. D.: Systemy baz danych. WNT, Warszawa 1988.
3. Subieta K.: Ingres, System Zarządzania Relacyjną Bazą Danych. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994.
4. Snodgrass R., Ahn I.: Temporal Databases. IEEE Computer. Vol. 19. Nr 9. 1986.
5. Jensen Ch., Clifford J., Soo M.: A Consensus Glossary of Temporal Database Concepts. Aalborg University, 1993. Adres internetowy: tdbglossary@cs.arizona.edu.
6. Snodgrass R., Jajodija S., Clifford J.: Temporal Databases: Theory, Design and Implementation. Benjamin/Cummings Publishing Company, 1993.
7. TIME International Workshop Series. Florida, 1994 – 1998. Adres internetowy: http://www.cs.uregina.ca/~temporal/
8. Böhlen M., Snodgrass R., Jensen C.: Adding Valid Time to SQL/Temporal. ANSI Expert's Contribution, 1996. Adres strony internetowej: http://www.timeconsult.com.

[1] Usunięcie jakiegokolwiek indeksu negatywnie wpływało na wydajność.

9. Steiner A.: A Generalization Approach to Temporal Data Models and their Implementations. Informatik Dept. Zurich 1997. Adres strony internetowej: http://www.timeconsult.com.
10. TimeCenter Technical Reports, 1997-1999. Adres strony internetowej: http://www.cs.auc.dk/research/DBS/tdb/TimeCenter.

Recenzent: Dr hab. inż. Stanisław Wołek
Prof. Pol. Rzeszowskiej

Wpłynęło do Redakcji 21 czerwca 1999 r.

Abstract

The topic of this article is the overview of correct and efficient timestamping of temporal data. The article is organized as follows. The first part is connected with fundamental to database systems definitions of Temporal Database and transaction. This is the basic platform to examine the problem of modifying and querying temporal data within the same transaction, which is specified in the second part. Finally, several strategies for timestamping and revisiting tuples are offered in section 3.

The main purpose of the paper is to identify the difficult problem of possible violating transaction ACID properties by sequence of operations over time within a single transaction (fig. 2). To get a transaction-consistent view of database modifications, the transaction-time period of the tuple should be: [*transaction_start_time*, *commit_time_of_the_transaction*]. Correct implementing of most queries that make explicit reference to this unknown transaction time is describing in section 3 (table 1). It is shown, that expensive tuples revisiting step may be postponed (section 4) and that the best strategy is to revisit only these tuples, which incorrect timestamps are referred by a query.