Zaawansowane Technologie Bazodanowe

Wykład p.t.

OLTP

Przetwarzanie Transakcyjne

Pojęcie transakcji w bazach danych

Transakcje

Przetwarzanie danych w bazach danych może mieć charakter *transakcyjny*. Oznacza to, że baza przeprowadzana jest z jednego stanu stabilnego do innego, nowego stanu stabilnego. Takie przetwarzanie określa się jako **On-Line Transactional Processing** (OLTP).

Transakcja – *logicznie niepodzielny ciąg operacji na bazie danych*.

Transakcja to grupa kolejnych instrukcji SQL (i wewnętrznego języka programowania) realizowanych w formie sekwencji przez pojedynczego użytkownika lub aplikację. Transakcja jest jednostką logiczną i może składać się z jednej lub wielu instrukcji SQL. Transakcja kończy się *sukcesem* albo *porażką*. Porażka powoduje anulowanie transakcji (i wszystkich jej efektów cząstkowych).

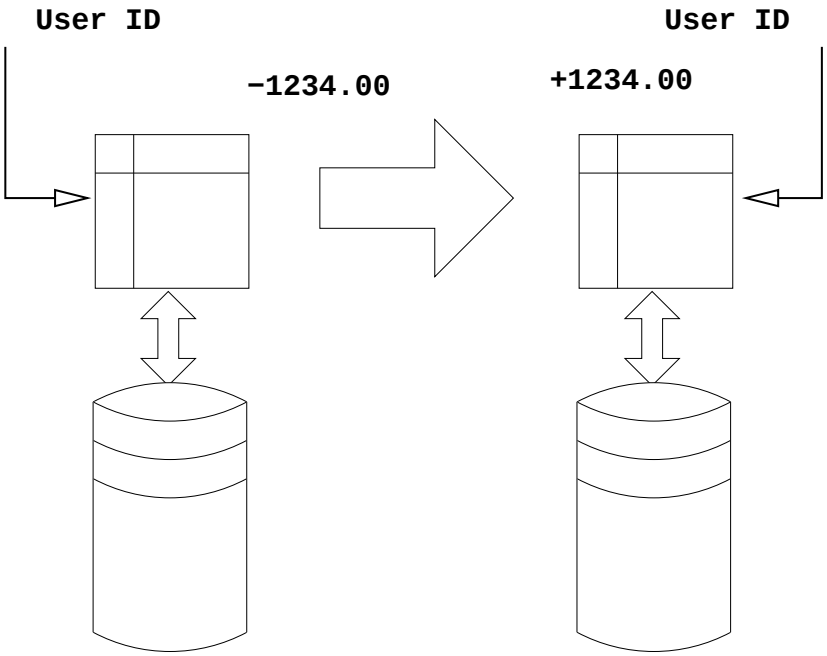
- Transakcja zaczyna się instrukcją `BEGIN [TRANSACTION|WORK]`,
- Transakcję zatwierdza się instrukcją `COMMIT [WORK]`,
- Transakcję anuluje się instrukcją `ROLLBACK`.

Transakcja może zakończyć się porażką w przypadku:

- awarii systemu,
- przerwania połączenia,
- niemożności kontynuowania operacji (np. brak miejsca na dysku),
- niestabilności (niespójności) stanu końcowego (niespełnione więzy integralności).

Przykład transakcji

Transakcja – operacja przelewu



Rysunek 1: Przykład transakcji: przelew pomiędzy kontami

Przykładowa kolejność transakcji:

```
UPDATE Tab_Accounts
   SET Account = Account - 1234.00
    WHERE ID = User_ID_1;
UPDATE Tab_Accounts
   SET Account = Account + 1234.00
    WHERE ID = User_ID_2;
```

Zasady realizacji transakcji w bazach danych

Transakcje – zasady

Zasady realizacji transakcji:

- SZBD automatycznie rozpoczyna transakcję po jej uruchomieniu, gdy nie jest aktywna inna transakcja, blokująca jej realizację,
- Jeżeli transakcja kończy się sukcesem, to zostaje **zatwierdzona** (ang. *committed*).
- Jeżeli transakcja nie może być zrealizowana, to zostaje **odrzucona** (ang. *aborted*).
- Transakcja odrzucona zostaje **wycofana** (ang. *rolled back*; czynność wycofywania określa się jako *rollback*) do stanu sprzed jej rozpoczęcia.
- Transakcja zatwierdzona nie może być wycofana.
- Transakcja może prowadzić do utraty danych (np. `UPDATE`, `DELETE`).
- Transakcje mogą być tylko do odczytu (nie powodują potrzeby blokowania danych).
- W trakcie transakcji można opóźniać kontrolę ograniczeń (`SET CONSTRAINTS MODE`).
- Transakcje mogą określać poziom izolacji blokady nałożonej na dane (`SET TRANSACTION`); standard SQL-92 (ISO) określa cztery poziomy izolacji: `READ UNCOMMITTED`, `READ COMMITTED`, `READ REPETABLE`, `SERIALIZABLE`.

Zasady realizacji transakcji – ACID

Zasady ACID przetwarzania transakcyjnego

Współczesne (wielodostępne) SZBD umożliwiają jednoczesne przetwarzanie wielu transakcji (np. systemy rezerwacji miejsc, systemy bankowe, etc.). Poprawność i kompletność realizacji transakcji gwarantuje *moduł zarządzania transakcjami*. Obowiązują cztery poniższe zasady ACID:

- **Atomicity** (atomiczność, niepodzielność) wykonywana jest albo cała transakcja (od początku do końca, albo żaden jej element nie może zostać zeralizowany (nie jest dopuszczalna realizacja częściowa; "wszystko albo nic").
- **Consistency** (spójność) – baza danych musi zachować spójność, dane po wykonaniu transakcji muszą być zgodne z nałożonymi ograniczeniami; transakcja przeprowadza bazę ze stanu stabilnego do nowego stanu stabilnego.
- **Isolation** (izolacja, niezależność) – transakcje są wykonywane niezależnie od siebie. Jeżeli dwie lub więcej transakcji jest przetwarzanych jednocześnie, nie mogą/nie powinny one wzajemnie na siebie oddziaływać; w wyniku jednoczesnego ich przetwarzania nie może się zdarzyć nic, co nie zdarzyłoby się, gdyby były one przetwarzane po kolei.
- **Durability** (trwałość) – po zakończeniu transakcji jej wynik nie może zostać utracony (np. z powodu awarii systemu).

Blokady: Realizacja zasad izolacji (zasadnicza idea) polega na blokowaniu dostępu do pewnych elementów bazy danych podczas realizacji transakcji (np. tabel).
Granulacja blokad: SZBD wprowadzają różne poziomy blokowania; rodzaj i rozmiary blokowanych elementów mogą być różne (np. blokady na poziomie rekordów, bloków na dysku, plików, relacji).

Zasady realizacji transakcji – ACID

Zasady ACID – objaśnienie szczegółowe

- niepodzielność (*ang. atomicity*) - każda transakcja jest traktowana jednostkowo, jeśli chociaż jedno z poleceń objęte pojedynczą transakcją zakończy się niepowodzeniem to wynikiem takiej transakcji będzie operacja odrzucenia (wycofania) wszystkich dotychczasowych zmian przeprowadzonych w ramach tej transakcji - i odwrotnie - aby transakcja zakończyła się sukcesem to wszystkie operacje wchodzące w skład transakcji muszą się zakończyć powodzeniem. Należy w tym przypadku uwzględnić różnorodność przyczyn, jakie mogą spowodować niemożność wykonania danych operacji na rekordach: błędy w składni kwerend (zgłaszanie wyjątków), awarie baz danych, systemu operacyjnego czy sprzętu; w komercyjnych, wysokowydajnych serwerach bazodanowych każda z tych ewentualności musi zostać odpowiednio rozpatrzona i obsłużona na wypadek wystąpienia usterki,

- spójność (*ang. consistency*) - podczas przeprowadzania transakcji nie może dochodzić do sytuacji naruszania zdefiniowanych przez administratora więzów integralności oraz pod żadnym pozorem nie mogą zostać wpisane błędne dane (zły format, nieodpowiednie typy danych, brak wartości domyślnych). Jeżeli taka sytuacja będzie miała miejsce w momencie realizacji któregokolwiek z kroków danej transakcji to taka transakcja musi zostać unieważniona (należy zapewnić tzw. spójny stan bazy danych). Należy zdawać sobie jednak sprawę z tego, że nawet najbardziej restrykcyjnie zdefiniowane więzy integralności w schemacie bazy danych nie ustrzegą przed popełnianymi nierzadko błędami ludzkimi (np. błędy wynikające z nieprawidłowego zaimplementowania algorytmu obliczającego pewne istotne wartości, które w kolejnym etapie biorą bezpośredni udział w realizacji danej transakcji),

- niezależność (*ang. isolation*) - zapewnienie wzajemnego odseparowania wielu transakcji od siebie w sytuacji gdy są one wykonywane równocześnie i skutki zakończenia którejkolwiek z nich mogą bezpośrednio wpływać na stan reszty równolegle wykonywanych transakcji. Wbrew pozorom jest to właściwość, która przysparza cały szereg potencjalnych problemów, które muszą zostać rozwiązane przez programistów implementujących dany system bazodanowy. Zagadnieniom niezależności transakcji (z zakresu współbieżności programowania) poświęcony jest podrozdział 4.2 niniejszej pracy,
- trwałość (*ang. durability*) - pewność, że wszystkie dane, które uległy zmianie podczas zakończonej pomyślnie transakcji nie zostaną w żaden sposób utracone. Praktyczna realizacja tej własności w bazach danych to systemy automatycznej archiwizacji i replikacji (backup) oraz zapisu logów transakcyjnych, pozwalających na natychmiastowe odtworzenie - w razie awarii systemu - ostatniego, spójnego stanu bazy danych.

Problemy z wielodostępem: czytanie brudnopisu

Dirty Read – brudny odczyt

```
pracownicy=# select * from tab_pobory ;
 id_prac | pobory
---------+---------
 707     | 1234.00

UPDATE tab_pobory
SET pobory=pobory+432
--      WHERE ID_prac='707'
;
```

BEGIN – COMMIT:

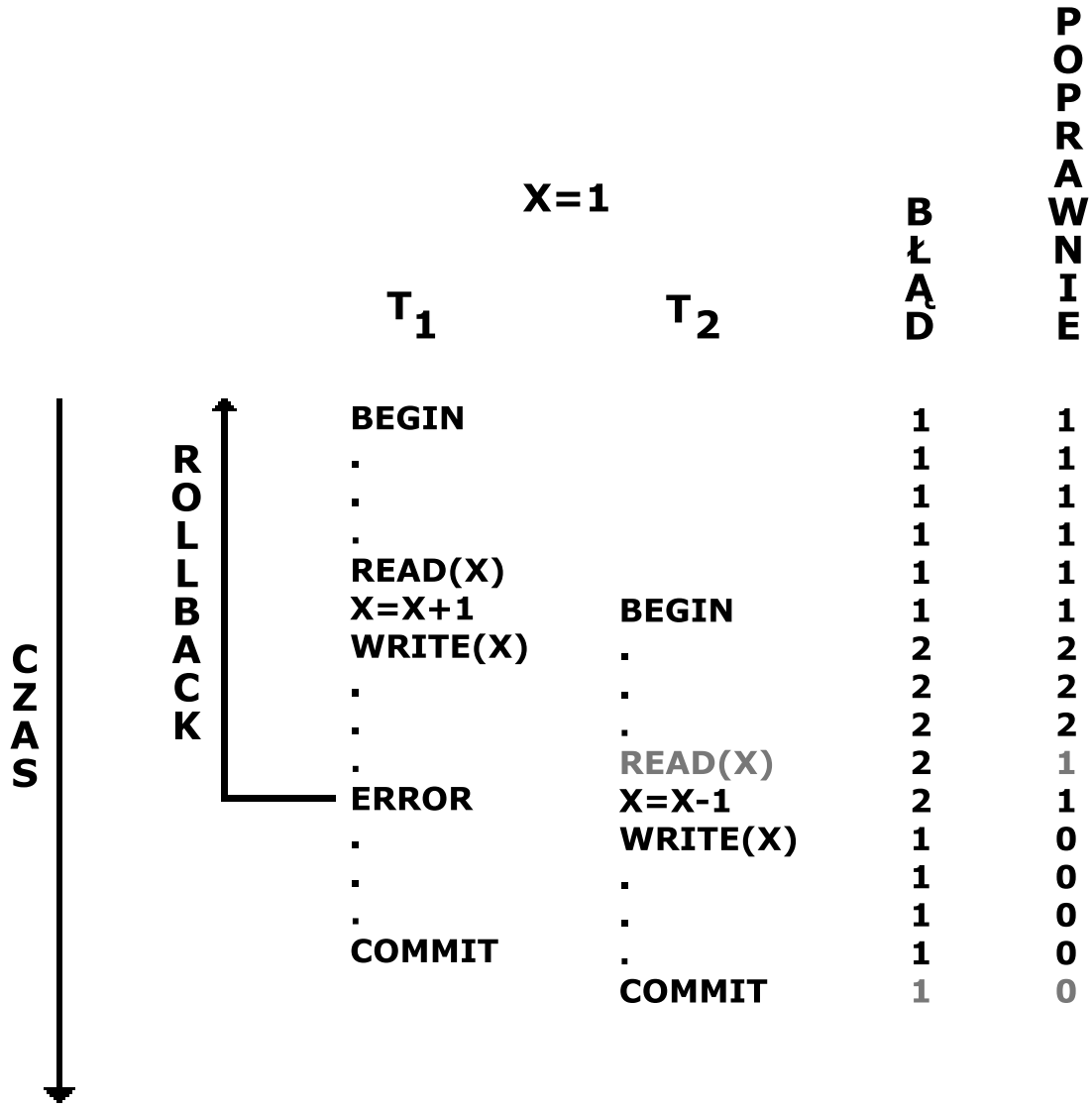
T	T1 SQL	T1 View	T2 View	No DR T2 View
1	`BEGIN WORK`	1234.00	1234.00	1234.00
2	`SET pobory=pobory+432`			
3		1666.00	1666.00	1234.00
4	`COMMIT WORK`			
		1666.00	1666.00	1666.00

BEGIN – ROLLBACK:

T	T1 SQL	T1 View	T2 View	No DR T2 View
1	`BEGIN WORK`	1666.00	1666.00	1666.00
2	`SET pobory=pobory-432`			
3		1234.00	1234.00	1666.00
4	`ROLLBACK WORK`			
5		1666.00	1666.00	1666.00

Przykład transakcji równoczesnych

Transakcje – dirty read



Rysunek 2: Przykład transakcji równoczesnych i problemu niepowtarzalnych odczytów

Problemy z wielodostępem – niepowtarzalne odczyty

Non-Repeatable Read – odczyt nie dający się powtórzyć

```
pracownicy=# select * from tab_pobory ;
 id_prac | pobory
---------+---------
 707     | 1234.00
```

```
UPDATE tab_pobory
SET pobory=pobory+432
--      WHERE ID_prac='707'
;
```

BEGIN – COMMIT:

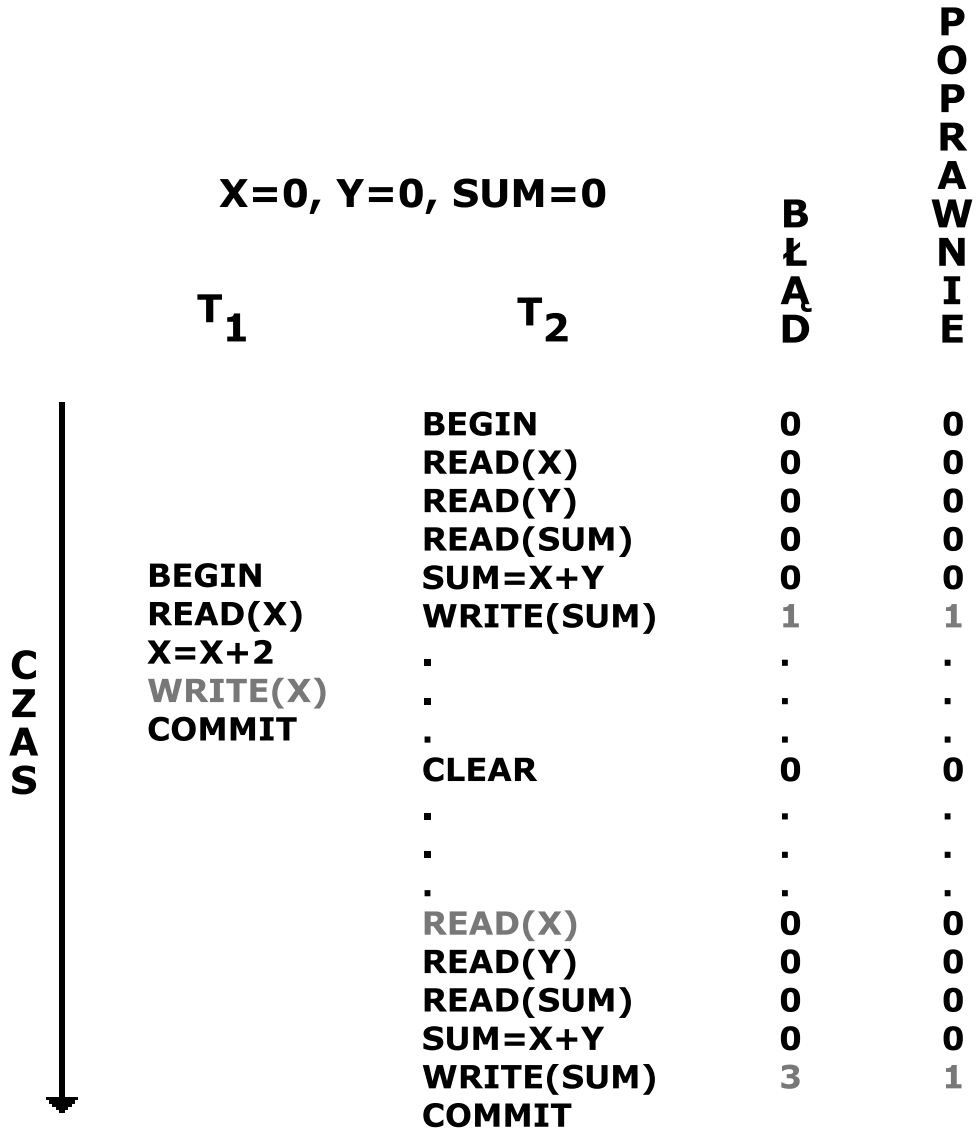
T	T1 SQL	T1 View	T2 View	No NRR T2 View
1	`BEGIN WORK`		(`BEGIN WORK`)	(`BEGIN WORK`)
2		1234.00	1234.00	1234.00
3	`SET pobory=pobory+432`			
4		1666.00	1234.00	1234.00
5	`COMMIT WORK`			
		1666.00	1666.00	1234.00
6			`COMMIT WORK`	`COMMIT WORK`
7			`BEGIN WORK`	`BEGIN WORK`
8	`SELECT pobory`	1666.00	1666.00	1666.00

BEGIN – ROLLBACK: zmiany nie są widoczne (brak czytania z brudnopisu).

Transakcja T2 widzi zmiany wykonane i zatwierdzone przez T1, pomimo, że sama się nie zakończyła. W trakcie realizacji T2 widzialne zmiany mogą następować wielokrotnie (wiele transakcji wykonuje COMMIT).

Przykład transakcji równoczesnych

Transakcje – non-repeatable-read



Rysunek 3: Przykład transakcji równoczesnych i problemu niepowtarzalnych odczytów

Problemy z wielodostępem: odczyty widmo

Phantom – odczyty widmo

```
pracownicy=# select * from tab_pobory ;
 id_prac | pobory
---------+---------
 707     | 1234.00
```

```
UPDATE tab_pobory
SET pobory=pobory+432
--      WHERE ID_prac='707'
;
```

```
INSERT INTO tab_pobory
VALUES ('710,1234.00);
```

BEGIN – COMMIT:

T	T1 SQL	T1 View	T2 SQL	T2 View	No Phantom
1	BEGIN WORK		BEGIN WORK		
2		1234.00		1234.00	1234.00
3	pobory+432				
4		1666.00		1234.00	1234.00
5			INSERT (1234.00)		
6		1666.00		1234.00	1234.00
7	COMMIT WORK				
8		1666.00		1234.00	1666.00
9			COMMIT WORK		
10	SELECT pobory	1666.00	SELECT pobory	1666.00	1666.00
11		1234.00		1234.00	1666.00

Problemy z wielodostępem: utracona aktualizacja

Lost Update – utracona aktualizacja

```
pracownicy=# select * from tab_pobory ;
 id_prac | pobory
---------+---------
 707     | 1234.00
```

```
UPDATE tab_pobory
SET pobory=pobory+432
--      WHERE ID_prac='707'
;
```

```
UPDATE tab_pobory
SET pobory=pobory+234
--      WHERE ID_prac='707'
;
```

BEGIN – COMMIT:

T	T1 SQL	T1 View	T2 SQL	T2 View
1	`BEGIN WORK`		`BEGIN WORK`	
2		1234.00		1234.00
3	`SELECT pobory`	1234.00	`SELECT pobory`	1234.00
4	`pobory+432`	1666.00		1234.00
5	`COMMIT WORK`	1666.00		1234.00
6		1666.00		(1234.00/1666.00)
7		1666.00	`pobory+234`	
8			`COMMIT WORK`	
9		1486.00		1468.00

Problemy z wielodostępem: utracona aktualizacja

Lost update – rozwiązanie

Proste rozwiązanie problemu polega na zblokowaniu wyszukiwania rekordów do modyfikacji i samej modyfikacji w jednej instrukcji. Obowiązuje to jednak wszystkich użytkowników/aplikacje.

```
pracownicy=# BEGIN WORK;
BEGIN
pracownicy=# SELECT * FROM tab_pobory;
 id_prac | pobory
---------+---------
 707     | 1234.00
(1 row)
pracownicy=# UPDATE tab_pobory
pracownicy-# SET pobory=pobory+432.00
pracownicy-# WHERE pobory=1234;
UPDATE 1
pracownicy=# COMMIT;
COMMIT
pracownicy=# select * from tab_pobory ;
 id_prac | pobory
---------+---------
 707     | 1666.00
(1 row)

pracownicy=# UPDATE tab_pobory
pracownicy-# SET pobory=pobory+432.00
pracownicy-# WHERE pobory=1234;
UPDATE 0
```

Jak widać, ponowna próba modyfikacji rekordów nie powiodła się.

Poziomy izolacji

Isolation Levels

Standard SQL definiuje cztery poziomy izolacji:

- **READ UNCOMMITTED** – odczyt niezatwierdzony,
- **READ COMMITTED** – odczyt zatwierdzony,
- **REPEATABLE READ** – odczyt dający się powtórzyć,
- **SERIALIZABLE** – odczyt uszeregowany.

Przypomnijmy najważniejsze problemy współdziałania transakcji na wspólnym obszarze pamięci:

- *Dirty read:* pierwsza transakcja modyfikuje rekord(y), a druga czyta zmodyfikowane dane **zanim zmiana została zachowana** przez `COMMIT`; jeśli pierwsza transakcja zostanie wycofana, to druga z nich przeczytała dane, które nie istnieją.
- *Non-Repeatable read:* pierwsza transakcja czyta dane, a druga usuwa je lub modyfikuje i zatwierdza przez `COMMIT`; teraz pierwsza (niezakończona) transakcja **czytając ponownie te dane (tym samym zapytaniem) otrzyma inne wartości**,
- *Phantom:* pierwsza transakcja odczytuje/modyfikuje dane spełniające pewien predykat; druga transakcja równolegle wstawia/modyfikuje rekordy, tak, że nowe rekordy spełniają dany predykat – **następne wykonanie tego samego zapytania przez pierwszą transakcję da inny wynik** (rekordy wstawiane przez drugą transakcję nie zostały obsłużone przez pierwszą).

Poziomy izolacji

Isolation Levels – objaśnienie

- READ UNCOMMITTED - jest najniższym poziomem izolacji transakcji, nie zabezpiecza przed żadnym z wymienionych wcześniej problemów (nawet przed problemem odczytu niepewnych danych!) co rzecz jasna automatycznie eliminuje możliwość zastosowania tego poziomu w profesjonalnych implementacjach systemów, bowiem musimy mieć absolutną pewność, że nikt oprócz nas nie korzysta w danej chwili z bazy danych. Co jednak robić w przypadku gdy nie ma możliwości podwyższenia poziomu niezależności (np. archiwalne instancje systemów)? Można zastosować tzw. techniki optymistyczne, które polegają na każdorazowym porównaniu owej wartości - w momencie gdy dochodzi do zmiany wartości rekordu w transakcji - z wartością poprzednią (poprzez klauzulę `WHERE`), co oczywiście wymaga istnienia bufora przechowującego poprzedni stan wartości. Należy zdawać sobie jednak sprawę z obniżenia wydajności całego systemu wobec zastosowania optymistycznych transakcji (istotne wartości opóźnień w przypadku dużej ilości przetwarzanych transakcji),
- READ COMMITTED - w wielu systemach relacyjnych baz danych jest to domyślny poziom niezależności transakcji, w pełni zabezpiecza przed problemem „dirty reads”, nieskuteczny jednak w eliminowaniu fantomów i problemu niepowtarzalności danych. Pozwala transakcjom na zakładanie lub zwalnianie blokad na poziomie rekordu, nad którym aktualnie dokonują modyfikacji. Żadna transakcja nie otrzyma dostępu do danego rekordu (sekcji krytycznej) dopóki transakcja, która blokadę taką założyła nie zwolni współdzielonych zasobów.

- REPEATABLE READ - jeden z najwyższych poziomów izolacji, nie zabezpiecza jedynie przed fantomami, w wielu zastosowaniach zupełnie wystarczający. Pozwala transakcjom na zakładanie i zwalnianie blokad na wszystkie bieżąco przetwarzane wiersze co sprawia, że nawet wielokrotne odwołanie się do tego samego zestawu danych w obrębie tej samej transakcji spowoduje wygenerowanie identycznych rezultatów.
- SERIALIZABLE - najwyższy poziom niezależności, oznacza całkowitą izolację transakcji nawzajem od siebie. W efekcie otrzymujemy rodzaj kolejki, w której znajdują się zlecone do wykonania transakcje, które w sposób sekwencyjny zostają następnie wykonane w określonym porządku, co może w sposób znaczący obniżyć wydajność serwera bazy danych. Oczywiście nic nie stoi na przeszkodzie aby można było wykonać równolegle kilka transakcji, jeśli tylko nie korzystają one podczas wykonania z tych samych zasobów, jednak utrata szybkości działania systemu w porównaniu z innymi poziomami izolacji jest zawsze zauważalna, należy z rozwagą stosować ten tryb pracy.

Definiowanie poziomu izolacji

Poziomy izolacji

POZIOM IZOLACJI	Dirty read	Non-Repeatable	Phantom
READ UNCOMMITTED	TAK	TAK	TAK
READ COMMITTED	NIE	TAK	TAK
REPETABLE READ	NIE	NIE	TAK
SERIALIZABLE	NIE	NIE	NIE

Do definiowania poziomu izolacji transakcji służy instrukcja:

```
SET TRANSACTION { ISOLATION LEVEL
          {READ UNCOMMITTED | READ COMMITTED
           REPETABLE READ | SERIALIZABLE }
          | { READ ONLY | READ WRITE }
          | { DIAGNOSTICS SIZE ilosc warunkow }};
```

W PostgreSQL:

```
SET TRANSACTION
    [ ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE } ]
    [ READ WRITE | READ ONLY ]

SET SESSION CHARACTERISTICS AS TRANSACTION
    [ ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE } ]
    [ READ WRITE | READ ONLY ]
```

`READ COMMITTED` – zezwolenie na odczyt danych **zatwierdzonych przed rozpoczęciem instrukcji** wewnątrz transakcji; możliwe odczyty nie dające się powtórzyć (i fantomy). Jest to poziom domyślny (ang. default). `SERIALIZABLE` – zezwolenie na odczyt danych zatwierdzonych przed wykonaniem pierwszej instrukcji DML (SELECT/INSERT/DELETE/UPDATE/FETCH/COPY_TO) w obrębie transakcji.

Blokady i ograniczenia

Blokady

Izolacja transakcji jest realizowana za pomocą blokowania dostępu do danych:

- blokada współdzielona (ang. *shared mode*) pozwala innym czytać dane (bez aktualizacji),
- blokada wyłączna (ang. *exclusive mode*) nie pozwala na dostęp do danych (nawet odczyt).

Blokady mogą być jawne lub niejawne. Niejawne blokady są stosowne automatycznie w trakcie wykonywania instrukcji. Jawne blokady definiuje się instrukcją `LOCK TABLE` oraz `SELECT ... FOR UPDATE`. Blokady mogą być na poziomie:

- wybranych wierszy: `SELECT 1 FROM <table> WHERE <cond> FOR UPDATE`,
- tabeli: `LOCK TABLE <table>`.

Ograniczenia

Ograniczenia (ang. *constraints*) mogą być sprawdzane:

- bezpośrednio po wykonaniu instrukcji lub,
- po zakończeniu transakcji (tzw. opóźnianie ograniczeń).

W PostgreSQL do wymuszanie opóźnienia w FK: `DEFERRABLE`.

```
CREATE TABLE tab_pobory
(ID_prac char(5) NOT NULL,
 Pobory numeric(8,2),
 CONSTRAINT tab_pobory_fk FOREIGN KEY(ID_prac)
 REFERENCES prac(ID_prac) ON UPDATE CASCADE DEFERRABLE
);
```

Zakleszczenia

Zakleszczenia

Zakleszczenie może mieć miejsce jeżeli dwie transakcje rywalizują o dostęp do tych samych niepodzielnych (blokowany dostęp) zasobów.

Przykład schematu zakleszczenia:

T1	T2
`BEGIN`	`BEGIN`

`UPDATE pobory`
`SET pobory=1234`
`WHERE ID_prac='707';`	

	`UPDATE pobory`
	`SET pobory=1234`
	`WHERE ID_prac='710';`

`UPDATE pobory`
`SET pobory=1234`
`WHERE ID_prac='710';`	

	`UPDATE pobory`
	`SET pobory=1234`
	`WHERE ID_prac='707';`

PostgreSQL automatycznie wykrywa zakleszczenia i je eliminuje. Zmiany wykonane w eliminowanej sesji są tracone.

PostgreSQL – zarządzanie transakcjami

Przegląd instrukcji

`BEGIN WORK|TRANSACTION` – rozpoczęcie transakcji.

`COMMIT WORK|TRANSACTION` – zatwierdzenie transakcji.

`SELECT 1 FROM <table> WHERE <condition> FOR UPDATE` – blokowanie wierszy tablicy wewnątrz transakcji.

`LOCK TABLE <table>` – blokowanie dostępu do tablicy.

`LOCK TABLE <table> IN <lockmode> MODE` – blokowanie dostępu do tablicy.

Parametr `<lockmode>` może przybierać siedem nastepujących wartości:

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE
| SHARE UPDATE EXCLUSIVE
| SHARE | SHARE ROW EXCLUSIVE
| EXCLUSIVE | ACCESS EXCLUSIVE
```

Opis: `http://www.postgresql.org/docs/8.3/static/explicit-locking.html`

Charakterystyki PostgreSQL i zalecenia

Charakterystyki PostgreSQL

- PostgreSQL domyślnie pracuje w trybie *chained* – oznacza to, że instrukcje są autozatwierdzane natychmiast po wykonaniu (tzw. niejawne transakcje – każda instrukcja jest osobną transakcją),
- `BEGIN WORK` – przełącza w tryb transakcyjny, aż do wystąpienia `COMMIT` lub `ROLLBACK`,
- Zakończenie transakcji `ROLLBACK` pozwala na *bezpieczne* eksperymentowanie,
- PostgreSQL nie dopuszcza trybu czytania z brudnopisu (ang. *dirty read*),
- w PostgreSQL można używać skrótów: `BEGIN`, `COMMIT`, `ROLLBACK`,
- w PostgreSQL nie wolno zagnieżdżać transakcji (brak *save points*),
- PostgreSQL stosuje 7 (8) typów blokad.
- PostgreSQL automatycznie wykrywa zakleszczenia i je eliminuje.

Zalecenia dotyczące stosowania transakcji

- transakcje powinny być małe! – kończymy, gdy tylko jest to możliwe,
- nie utrzymywać transakcji podczas dialogu z użytkownikiem,
- aplikacje powinny przetwarzać dane w tej samej kolejności.

SAVEPOINTS

Granularyzacja transakcji – punkty kontrolne

W zaawansowanych systemach bazodanowych możliwa jest wieloetapowa kontrola transakcji z wykorzystaniem tzw. *savepoints*.

Rozważmy przykład przelewu pomiędzy kontami:

```
BEGIN WORK;
UPDATE accounts SET balance = balance-100.00
 WHERE name = 'Alice';
UPDATE branches SET balance = balance-100.00
 WHERE name = (SELECT branch_name FROM accounts WHERE name='Al
UPDATE accounts SET balance = balance+100.00
 WHERE name = 'Bob';
UPDATE branches SET balance = balance+100.00
 WHERE name = (SELECT branch_name FROM accounts WHERE name='Bo
...
COMMIT;
```

Przy zastosowaniu punktów kontrolnych *savepoint* możliwy jest podział transakcji na etapy:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Multiversion Concurrency Control, MVCC

MVCC in PostgreSQL

W systemie PostgreSQL zastosowano mechanizm o nazwie *Multiversion Concurrency Control, MVCC*. Jego idea polega na zapewnieniu każdej transakcji stabilnego obrazu danych z pewnej chwili, niezależnie od stanu aktualnego (ang. *snapshots*).

PostgreSQL od wersji 8.1 wspiera wszystkie poziomy niezależności transakcji (ale realizuje tylko dwa!) oraz system blokad na poziomie tabel oraz wierszy. Zastosowany mechanizm MVCC ma w swoich założeniach zmniejszyć - do niezbędnego minimum - proces użycia blokad.

Zasada działania MVCC jest stosunkowo prosta, model zakłada możliwość istnienia kilku wersji tego samego rekordu w tabeli co z kolei wynika z fizycznego sposobu reprezentacji wierszy w systemie. W PostgreSQL każdy rekord w tabeli posiada dodatkowe, ukryte atrybuty (kolumny), które to decydują o aktualnym stanie danego wiersza. Z punktu widzenia problemu przetwarzania transakcyjnego istotne są cztery kolumny (wartości):

- X_{min} - numer transakcji, w której dany wiersz został stworzony (poleceniem `INSERT`),
- C_{min} - numer polecenia SQL w bieżącej transakcji, który utworzył rekord,
- X_{max} - numer transakcji, w której dany wiersz został usunięty (poleceniem `DELETE`), jeśli wiersz nie jest usunięty to pole przyjmuje wartość 0,
- C_{max} - numer polecenia SQL w bieżącej transakcji, który usunął wiersz.

Użytkownik, który w danym momencie przegląda tabelę ma dostęp tylko do następujących wierszy:

- $x_{min} \leq$ bieżący numer transakcji,
- $x_{max} = 0$,
- $x_{max} \geq$ bieżący numer transakcji.

Niestety, miarę upływu czasu taka tabela z „nieaktualnymi” wierszami zaczyna się szybko rozrastać, co w konsekwencji - szczególnie przy intensywnym użyciu poleceń języka DML - doprowadzi do znacznego spadku wydajności bazy danych. Dlatego też w PostgreSQL istnieje wykraczające poza standardy SQL narzędzie/polecenie - `VACUUM` które dokonuje przeglądu wierszy w tabeli i całkowitego usunięcia rekordów, dla których wartość x_{max} jest mniejsza od najstarszej transakcji w systemie.

Blokady w PostgreSQL

W PostgreSQL możliwe jest także jawne zakładanie blokad na tabele oraz wiersze (dostępnych jest szereg różnych trybów pracy, ich pełna lista znajduje się poniżej). Każdy z tych trybów blokowania danych to blokowanie na poziomie tabeli[1] (nawet jeśli w nazwie pojawia się słowo kluczowe `ROW`). Możemy skorzystać z następujących trybów:

- ACCESS SHARE (AS) - jest to blokada zakładana automatycznie np. przy wykonywaniu polecenia typu `SELECT`, umożliwia odczyt danych w każdej chwili, uniemożliwia dokonywanie modyfikacji,
- ROW SHARE (RS) - zarezerwowany dla poleceń `SELECT ... FOR UPDATE` i `SELECT ... FOR SHARE`, dodatkowo blokuje tabele powiązane ze sobą więzami integralności,
- ROW EXCLUSIVE (RX) - domyślny tryb blokad dla poleceń modyfikujących dane w tabelach - `INSERT`, `UPDATE` oraz `DELETE`, blokuje także zależne tabele,
- SHARE UPDATE EXCLUSIVE (SUX) - stosowany jedynie w momencie wywołania polecenia `VACUUM`, zabezpiecza przed niezsynchronizowanymi zmianami w schemacie bazy danych,
- SHARE (S) - zabezpiecza przed niekontrolowanymi zmianami danych w tabeli (przydatny np. w momencie zakładania indeksu na kolumnę),
- SHARE ROW EXCLUSIVE (SRX) - blokada wiersza tabeli przed wszelkimi modyfikacjami, możliwy odczyt danych,
- EXCLUSIVE (X) - odczyt danych z tabeli jest możliwy wyłącznie w obrębie trwającej transakcji,
- ACCESS EXCLUSIVE (AX) - całkowita blokada tabeli na czas trwania danej transakcji, jako jedyny tryb oferuje możliwość blokady odczytu danych z tabeli poleceniem typu `SELECT`.

[1] Pomijając wyrażenia typu `SELECT ... FOR UPDATE`, gdzie blokowane są bezpośrednio wiersze

W PostgreSQL istnieje możliwość założenia kilku blokad na daną tabelę jednocześnie, są jednak pewne ograniczenia (wzajemnie wykluczające się tryby), pełne zestawienie znajduje się w poniższej tabeli:

Tablica 1: Blokady w PostgreSQL.

	AS	**RS**	**RX**	**SUX**	**S**	**SRX**	**X**	**AX**
AS	+	+	+	+	+	+	+	-
RS	+	+	+	+	+	+	-	-
RX	+	+	+	+	-	-	-	-
SUX	+	+	+	+	-	-	-	-
S	+	+	-	-	+	-	-	-
SRX	+	+	-	-	-	-	-	-
X	+	-	-	-	-	-	-	-
AX	-	-	-	-	-	-	-	-

PostgreSQL

Literatura

1. Thomas Connolly, Carolyn Begg: Systemy baz danych. Praktyczne metody projektowania, implementacji i zarządzania. RM, Warszawa 2004 (tom 2, rodz. 19).

2. Richard Stones, Neil Matthew: Bazy danych i PostgreSQL. Helion, Gliwice 2001.

3. John C. Worsley, Joshua D. Drake: PostgreSQL. Praktyczny przewodnik. Helion/O'Reilly, Gliwice, 2002.

4. `http://www.postgresql.org/docs/8.3/interactive/index.html`