Bazy danych

10. Transakcje

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2020

Transakcje

- Pojedynczy użytkownik — ochrona szczególnie wrażliwych fragmentów. *Transakcja wykonuje się albo w całości, albo wcale.* Jeżeli w trakcie wykonywania transakcji wystąpi jakiś błąd, całą sekwencję operacji można odwołać, przywracając bazę do stanu sprzed rozpoczęcia tej sekwencji.

- System wielodostępny

  1. Jak wyżej.

  2. Różne procesy klienckie odwołujące się do tych samych tabel nie mogą się ze sobą kłócić.

Spójność danych w bazach rozproszonych oznacza, że każdy serwer przechowujący bazę lub jej fragment **musi** zwrócić taką samą odpowiedź na dane zapytanie, gdyby zadać je w tym samym momencie.

Wymóg zachowania spójności najmocniej odróżnia relacyjne (transakcyjne) bazy danych od baz danych NoSQL.

Zasady ACID (T. Hearder, A. Reuter, 1983)

A Atomicity — atomowość.
Transakcja jest niepodzielna, albo wszystko, albo nic.

C Consistency — spójność.
Transakcja nie może naruszać integralności danych: spójności danych pomiędzy różnymi serwerami przechowującymi tę samą bazę lub jej fragmenty, a także więzów narzuconych na dane w tabelach.

I Isolation — izolacja.
Jedna transakcja nie może widzieć wyników działania jakiejś innej, niezatwierdzonej transakcji. Można powiedzieć, że transkacja musi odbywać się tak, jakby żadna inna transakcja nie miała miejsca w tym samym czasie.

D Durability — trwałość.
Zmiany wprowadzone w transakcji muszą być trwałe, niezależnie od możliwych późniejszych błędów sprzętu lub oprogramowania.

Jak to robimy w SQL?

```
START TRANSACTION;
zapytanie_1;
zapytanie_2;
...
zapytanie_N;
COMMIT;
```

Zmiany zostają zatwierdzone

```
START TRANSACTION;
zapytanie_1;
zapytanie_2;
...
zapytanie_N;
ROLLBACK;
```

Zmiany zostają odwołane

Wielodostępność — co może pójść źle?

1. Niespójność odczytów — jedna transakcja może odczytać dane zmieniane przez drugą transakcję, chociaż transakcja ta nie zatwierdziła jeszcze zmian.

2. Niepowtarzalność odczytów — transakcja odczytuje dane, nieco później odczytuje je ponownie, a odczytane dane są inne, mimo iż transakcja odczytująca nie została jeszcze zatwierdzona.

3. Odczyty fantomowe — jedna tabela dodaje wiersz, druga transakcja aktualizuje wiersze. Nowy wiersz powinien być zaktualizowany, a nie jest.

Poziomy izolacji ANSI

Poziom izolacji	Niespójność odczytów	Niepowtarzalność odczytów	Odczyty fantomowe
Read uncommitted	OK	OK	OK
Read committed	NIE	OK	OK
Repeatable read	NIE	NIE	OK
Serializable	NIE	NIE	NIE

Serializowalność oznacza, że wynik sekwencji przeplatających się działań, wykonywanych przez zatwierdzane transakcje, musi ściśle odpowiadać sytuacji, w której wszystkie transakcje wykonywane są kolejno, jedna po zakończeniu drugiej.

Jak realizujemy izolację?

W celu zapewnienia izolacji w systemie wielodostępnym, transakcje blokują tabele (fragmenty tabel), które są im potrzebne. Najczęściej stosowane mechanizmy to:

2PL *Strict two-phase locking*: Każda transakcja zakłada blokadę na każdy rekord, który chce odczytać, przed dokonaniem odczytu. Blokady do odczytu mogą być współdzielone z innymi transakcjami. Każda transakcja zakłada też wyłączną blokadę na każdy fragment danych, który chce zapisać. Wszystkie blokady są utrzymywane aż do zakończenia transakcji. Jest to algorytm "pesymistyczny".

OCC *Optimistic Concurrency Control*: Wiele transakcji mogą odczytywać i modyfikować fragment danych bez zakładania blokad. Transakcje zapamiętują

historię dokonywanych odczytów i zapisów. Przed zatwierdzeniem transakcja sprawdza historię w celu wykrycia ewentualnych konfliktów z innymi transakcjami. Jeśli jakieś konflikty zostaną wykryte, jedna z transakcji wywołujących konflikt zostaje odwołana.
OCC zakłada, że większość transakcji może się zakończyć pomyślnie bez popadania w konflikt z innymi transakcjami (konflikt polega na żądaniu dostępu do tych samych rekordów bazy — OCC domyślnie zakłada niewielkie współzawodnictwo o dostęp do tych samych danych). Wobec tego na bazę nie są nakładane blokady, zabezpieczające przez konfliktami, ale spowalniające działanie.

Aby zminimalizować blokowanie (i opóźnienie innych transakcji), stosuje się mechanizm OCC. Działa on dobrze gdy konflikty są rzadkie, ale może jednak wygenerować duży koszt, jeżeli konflikty *nie* są rzadkie; w tych wypadkach mechanizm 2PL jest efektywnie szybszy.

Etapy algorytmu OCC

OCC przebiega w następujących etapach:

Begin Rozpocznij transakcję zapisując *timestamp* jej rozpoczęcia.

Modify Odczytaj dane z bazy i tymczasowo zapisz zmiany (**WAL** - patrz niżej).

Validate Sprawdź, czy inne transakcje nie modyfikowały danych, z których korzystała (czytała lub zapisywała) bieżąca transakcja. Należy sprawdzić tarnsakcje zakończone po rozpoczęciu transakcji bieżącej, niekiedy także *inne* transakcje, które jeszcze się nie zakończyły.

Commit/Rollback Jeśli nie wystąpił konflikt, zapisz dane (`COMMIT`). Jeśli konflikt wystąpił, odwołaj transakcję (`ROLLBACK`).

Uwaga! Sprawdzenie, czy wystąpił konflikt i decyzja o zatwierdzeniu/odrzuceniu transakcji *samo* musi mieć charakter operacji atomowej, w przeciwnym razie może wystąpić błąd typu *time of check to time of use (TOCTTOU)*.

Uwaga

- Instrukcje DDL (Data Description Language), czyli instrukcje tworzące i usuwające bazy oraz tworzące, usuwające i modyfukujące tabele nie są “transakcyjne” — nie można ich wycofać.

- W MySQL tabele, które chcemy zabezpieczać transakcjami, muszą być typu InnoDB.

Write-ahead logging — WAL

Wszystkie zmiany w bazie, zanim zostaną ostatecznie zatwierdzone, są tymczasowo zapisywane w specjalnym pliku, w RDBMS zwanym *dziennikiem systemowym*. Zapisy w dzienniku systemowym są indeksowane za pomocą *timestamp*, z reguły z dokładnością do milisekund. Dziennik systemowy służy do

- sprawdzania, czy nie wystąpił konflikt pomiędzy transakcjami,
- odtwarzania stanu systemu w przypadku awarii.

Dopiero jeśli dla jakiejś transakcji zostanie wydane polecenie `COMMIT`, zmiany wprowadzane przez tę transakcję są przepisywane z dziennika systemowego do tabel.

Oprócz dziennika systemowego niekiedy (w systemach rozproszonych praktycznie zawsze) istnieje też *undo log* — plik zawierający informację o tym, jakie zmiany być może w przyszłości będzie należało wycofać.

Two-phase COMMIT — 2PC

W przypadku rozproszonych baz danych lub baz przechowywanych w systemie skalowania poziomego, należy szczególnie zadbać o spójność (*consistency*) danych przechowywanych na różnych serwerach. Najczęściej robi się to za pomocą protokołu **Two-phase COMMIT (2PC)**.

Jeden węzeł sieci, koordynator lub Transaction Manager (TM), działa jako *master*. Na pozostałych węzłach działają Resource Managers (RM); TM na ogół jest RM swojego węzła. Wszystkie węzły uczestniczące w transakcji tworzą *kohortę*.

2PC jest rozpoczynane przez koordynatora. Członkowie kohorty albo zgadzają się na zapisanie zmian, albo wysyłają sygnał o konieczności przerwania transakcji. W wielu architekturach dla różnych transakcji różne węzły mogą być koordynatorami.

Faza **commit request** (faza głosowania):

- TM wysyła zapytanie o gotowość `COMMIT` do wszystkich członków kohorty i czeka na ich odpowiedź.
- Każdy RM kohorty wykonuje transakcję aż do punktu, w którym należałoby wydać polecenie `COMMIT` (WAL!) i przygotowuje swój *undo log*.
- Członkowie kohorty, którym udało się wykonać powyższy punkt, wysyłają koordynatorowi informację, że zgadzają się na zatwierdzenie transakcji. Członkowie kohorty, którzy napotkali błąd uniemożliwiający zatwierdzenie transakcji, wysyłają koordynatorowi informację, że transakcji nie można zatwierdzić.

Faza **commit** (zakończenie transakcji): Jeśli *wszyscy* członkowie kohorty potwierdzą gotowość do wykonania transakcji,

- TM wysyła polecenie `COMMIT` do wszystkich członków kohorty,
- wszyscy RM zatwierdzają transakcję na swoich węzłach i zwalniają wszystkie blokady nałożone na dane w związku z daną transakcją,
- wszyscy RM wysyłają potwierdzenie do koordynatora,
- TM zatwierdza transakcję i zwalnia blokady po otrzymaniu *wszystkich* potwierdzeń.

Jeśli nie ma zgody na transakcję, czyli gdy *którykolwiek* z członków kohorty zasygnalizuje w fazie głosowania brak zgody na transakcję **lub** gdy przekroczony zostanie czas oczekiwania (*timeout*) koordynatora,

- koordynator wysyła polecenie `ROLLBACK` do wszystkich członków kohorty,
- każdy z członków kohorty wycofuje transakcję korzystając ze swojego *undo log*, po czym zwalnia wysztkie blokady nałożone na dane w związku z daną transakcją,
- członkowie kohorty wysyłają powiadomienia do koordynatora,
- koordynator kończy (wycofuje) transakcję po otrzymaniu wszystkich powiadomień, po czym zwalnia wszystkie blokady nałożone na dane w związku z daną transakcją.

2PC jest protokołem asynchronicznym. Jeśli w transakcji uczestniczy N węzłów, zatwierdzenie transakcji wymaga przesłania $3N-3$ komunikatów (*messages*) i następuje z opóźnieniem potrzebnym do wysłania trzech komunikatów.

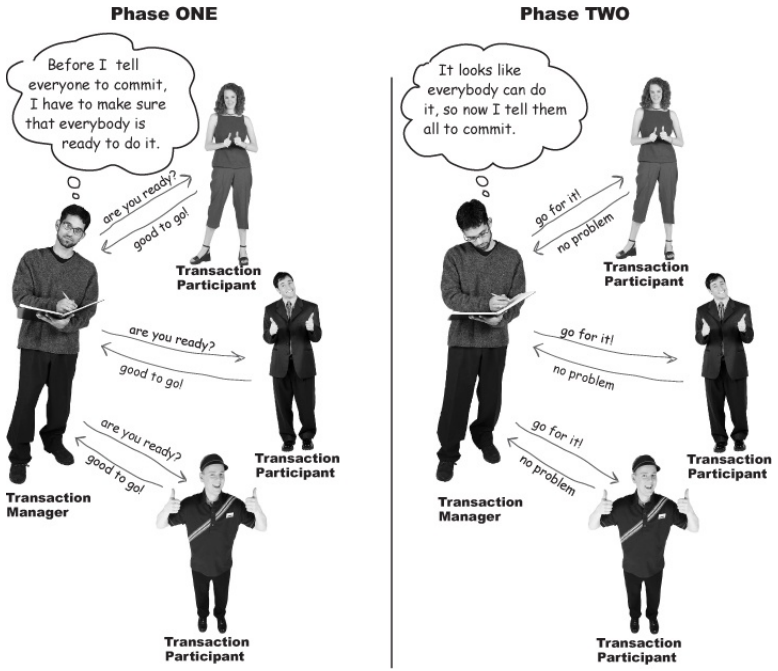
2PC w ogromnej większości przypadków pozwala na poprawne przeprowadzenie transakcji w systemie rozproszonym, ale nie zapewnia, że stanie się tak *w każdym możliwym* przypadku. 2PC gwarantuje natomiast, że żadna transakcja nie pozostanie w stanie nierozstrzygniętym bez powiadomienia o tym fakcie użytkownika.

Największą wadą protokołu two-phase COMMIT jest blokowanie. Członkowie kohorty po wysłaniu zgody na transakcję do koordynatora, czekają blokując dane, aż otrzymają ostateczne `COMMIT` lub `ROLLBACK` od koordynatora. Jeśli koordynator ulegnie w tym czasie awarii lub jeśli utracona zostanie komunikacja pomiędzy koordynatorem a członkami kohorty, niektórzy członkowie kohorty nie będą wiedzieli, w jaki sposób mają zakończyć transakcję. Pozostaną w stanie zawieszenia, utrzymując blokady nałożone na tabele. Jeśli stan zawieszenia trwa zbyt długo, występuje błąd typu *timeout*; działający członkowie kohorty odwołują wówczas transakcję na swoich węzłach i powiadamiają o tym użytkownika. Nie wiadomo natomiast, jaki jest stan koordynatora (TM).

Pewne modyfikacje protokołu 2PC pozwalają na wybranie nowego TM spośród działających RM, jeśli komunikacja z TM zostaje utracona. Pojawia się jednak problem, gdy pierwotny TM "wróci do życia" przed zakończeniem transakcji. Jeśli RM otrzyma *sprzeczne* decyzje od dwóch węzłów twierdzących, że są TM, nie wiadomo, jak RM ma się zachować.

Two-phase commit — podsumowanie

Wszystkie komputery w sieci muszą się zgodzić na pewne działanie. Jeśli któryś się nie zgodzi lub nie odpowie w określonym czasie, operacja zostaje odwołana



Ryzyko związane z transakcjami

1. Długo działające transakcje blokują dostęp innych użytkowników do danych, na których działa transakcja, dopóki nie zostanie ona zatwierdzona lub odwołana.

2. Należy unikać transakcji wtedy, gdy wymagana jest interakcja z użytkownikiem — należy najpierw zebrać wszytskie dane, a dopiero potem rozpoczynać transakcję.

(B)lokowanie tabel

`LOCK TABLES` nazwa_tabeli `[READ | [LOW PRIORITY] WRITE];`

- Tryb `READ` — chcę czytać tabelę i w tym czasie nie zezwalam innym na zapis.
- Tryb `WRITE` — chcę zmieniać zawartość tabeli i w tym czasie nie zezwalam innym ani na zapis, ani na odczyt.

- Tryb `LOW PRIORITY WRITE` — pozwala innym wątkom na założenie blokady `READ`; w tym czasie wątek, który chce nałożyć blokadę `LOW PRIORITY WRITE`, musi czekać, aż tamten wątek zwolni blokadę.

`UNLOCK TABLES;` — zwalnia wszystkie zablokowane przez dany wątek tabele.

Tabel nie należy blokować zbyt długo lub niepotrzebnie.

Uwaga praktyczna: Aplikacja powinna *najpierw* zebrać wszystkie potrzebne dane od użytkownika, *później* inicjować transakcję lub blokować tabele.