Czym jest transakcja: ACID rozłożone na cztery obietnice

> Transakcja to obietnica silnika, że grupa operacji zajdzie w całości albo wcale — a ACID to cztery osobne warunki tej obietnicy, z których każdy da się złamać na swój własny, konkretny sposób.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
ACID	Atomicity, Consistency, Isolation, Durability: cztery obietnice transakcji w bazie danych
DML	Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE)
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
JDBC	Java Database Connectivity: standardowy sposób, w jaki program w Javie łączy się z bazą danych
ODBC	Open Database Connectivity: uniwersalny sposób łączenia się z różnymi bazami danych tym samym interfejsem
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Po co to istnieje

Rozważamy przelew między dwoma kontami w tej samej bazie: zdejmujemy 500 złotych z konta A i dopisujemy 500 złotych do konta B. To dwie osobne operacje `UPDATE` na dwóch osobnych wierszach tabeli `konta`. Między pierwszą a drugą operacją mija ułamek sekundy, ale w tym ułamku sekundy może się wydarzyć wszystko — awaria zasilania serwera bazodanowego, restart usługi, przerwane połączenie sieciowe klienta. Jeśli pierwszy `UPDATE` się wykonał i zapisał, a drugi nie zdążył, pieniądze znikają z konta A i nigdzie się nie pojawiają — nie u klienta, nie w banku, tylko w niebycie, bo baza nie ma żadnego mechanizmu, który zapamiętałby „te dwie operacje miały iść

razem", jeśli o to nie poprosimy wprost.

> **Pojęcie: transakcja.** Jednostka pracy złożona z jednej albo wielu operacji na bazie danych, która dla całej reszty systemu wygląda jak jedna, niepodzielna czynność — albo wszystkie operacje w niej się zapiszą, albo żadna. Transakcja nie jest sposobem przyspieszenia czegokolwiek ani mechanizmem bezpieczeństwa w sensie uprawnień — jest odpowiedzią na dokładnie ten problem z przelewem: grupa zmian, która ma sens wyłącznie razem, dostaje granicę, wewnątrz której baza gwarantuje, że nic niedokończonego nie zostanie na trwałe zapisane.

Gwarancje, jakie transakcja daje, streszcza się skrótem ACID — cztery litery, cztery osobne obietnice, z których każda chroni przed inną kategorią katastrofy. Rozkładamy je pojedynczo, bo w praktyce psują się niezależnie od siebie — transakcja może stracić atomowość, nie tracąc izolacji, i odwrotnie.

Atomowość (Atomicity). Wszystkie operacje transakcji zapisują się razem albo żadna z nich się nie zapisuje — nie ma stanu pośredniego widocznego na trwałe. Bez atomowości przelew z akapitu wyżej jest dokładnie tym niebezpiecznym scenariuszem: `UPDATE` na koncie A przechodzi, silnik pada zanim zdąży wykonać `UPDATE` na koncie B, i po ponownym uruchomieniu baza ma 500 złotych mniej na koncie A bez żadnego śladu, dokąd te pieniądze poszły. Atomowość gwarantuje, że w takiej sytuacji silnik przy starcie wycofa niedokończoną transakcję do stanu sprzed pierwszego `UPDATE` — konto A wraca do pierwotnej kwoty, tak jakby przelew nigdy się nie zaczął.

Spójność (Consistency). Transakcja przenosi bazę z jednego stanu spełniającego wszystkie reguły integralności do innego stanu, który też je spełnia — ograniczenia `CHECK`, `FOREIGN KEY`, `NOT NULL` i `UNIQUE` muszą być prawdziwe zarówno przed transakcją, jak i po jej zatwierdzeniu. Bez spójności `UPDATE`, który ustawiłby saldo konta A na wartość ujemną wbrew ograniczeniu `CHECK (saldo >= 0)`, po prostu by przeszedł — baza przestałaby być wiarygodnym źródłem prawdy o tym, ile pieniędzy klient faktycznie ma.

Izolacja (Isolation). Transakcje wykonywane równolegle nie widzą nawzajem swoich niedokończonych, jeszcze niezatwierdzonych zmian — dla każdej z nich wygląda to tak, jakby była jedyną transakcją w systemie. Bez izolacji druga sesja, licząca w tym samym momencie saldo klienta do raportu, mogłaby odczytać stan konta A tuż po pierwszym `UPDATE` przelewu, ale przed drugim — zobaczyłaby pieniądze zniknięte z A i jeszcze niedopisane do B, mimo że z punktu widzenia całej transakcji przelew nigdy nie był w tym stanie „na trwałe". Strona Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom rozwija dokładnie to, co się dzieje, gdy izolacja jest niepełna.

Trwałość (Durability). Po tym, jak silnik potwierdzi zatwierdzenie transakcji, jej skutki przetrwają każdą kolejną awarię — restart serwera, zanik zasilania, padnięcie procesu bazy — nawet jeśli te zmiany fizycznie nie zdążyły jeszcze trafić do głównych plików danych na dysku. Bez trwałości `COMMIT` byłby pustą deklaracją: klient dostaje potwierdzenie, że przelew przeszedł, restartuje się serwer minutę później, i po ponownym uruchomieniu przelewu nie ma, bo zmiana istniała tylko w

pamięci. Mechanizmem, który tę obietnicę faktycznie spełnia, jest log transakcyjny opisany na stronie Log transakcyjny: WAL, redo, transaction log.

Spójność jest z tej czwórki najbardziej sporną literą, bo w odróżnieniu od pozostałych trzech nie opisuje mechanizmu, który silnik sam z siebie realizuje — opisuje właściwość, za której zapewnienie w dużej mierze odpowiada autor schematu i autor aplikacji, nie sam silnik. Atomowość, izolacja i trwałość to rzeczy, które baza gwarantuje niezależnie od tego, co wpiszemy w `CREATE TABLE` — silnik sam decyduje, kiedy wycofać transakcję, jak izolować sesje i jak zapisać log. Spójność zależy od tego, czy w ogóle zdefiniowaliśmy ograniczenia, które mają być pilnowane: baza bez żadnego `CHECK` na saldo pozwoli zapisać ujemną kwotę i formalnie nie złamie niczego, bo nie miała czego pilnować. To rozróżnienie bywa źródłem sporu akademickiego o to, czy „C" w ogóle powinno stać obok pozostałych trzech liter, skoro nie jest własnością silnika w tym samym sensie co atomowość czy trwałość — ale w praktyce inżynierskiej ta litera przypomina po prostu, że transakcyjność bez ograniczeń integralności chroni tylko przed awarią, nie przed błędną logiką.

Zestawienie czterech liter obok siebie, razem z tym, co się dzieje przy ich braku, daje jeden obraz:

Litera	Obietnica	Co się psuje bez niej
Atomowość	wszystko albo nic	pieniądze znikają z konta A i nie docierają do konta B po awarii w środku przelewu
Spójność	reguły integralności przed i po transakcji	saldo konta spada poniżej zera mimo ograniczenia `CHECK`
Izolacja	transakcje nie widzą nawzajem swoich niedokończonych zmian	raport liczony w trakcie przelewu widzi pieniądze zniknięte z A, jeszcze niedopisane do B
Trwałość	zatwierdzona zmiana przetrwa awarię	`COMMIT` się powiódł, serwer padł minutę później, przelewu nie ma po restarcie

Analogia

Transakcja bankowa, z której skrót ACID pożyczył swój najczęstszy przykład, tłumaczy się sama sobą lepiej niż jakakolwiek metafora spoza świata pieniędzy — bo przelew między kontami dosłownie jest transakcją w obu znaczeniach tego słowa naraz.

Wyobrażamy sobie okienko kasowe w banku, gdzie klient prosi o przelanie gotówki z jednej koperty do drugiej. Kasjer, który potraktowałby to jako dwie oddzielne, niepowiązane czynności — najpierw wyjmij pieniądze z koperty A, dopiero później, może za chwilę, może po przerwie na kawę, włóż je do koperty B — naraża klienta dokładnie na scenariusz braku atomowości: coś przerywa kasjera między jedną czynnością a drugą, i pieniądze przez jakiś czas nie istnieją w żadnej kopercie. Bank, który traktuje przelew poważnie, każe kasjerowi wykonać obie czynności jako jedną nierozerwalną procedurę — a jeśli coś mu w trakcie przeszkodzi, cofnąć się do stanu sprzed rozpoczęcia, nie

zostawiać transakcji w połowie.

Izolacja w tej analogii to zasada, że drugi klient, stojący przy tym samym okienku i pytający o stan koperty A w trakcie, gdy kasjer jest w środku przelewu, nie dostaje odpowiedzi opartej na częściowo wykonanej operacji — dostaje albo stan sprzed przelewu, albo stan po jego pełnym zakończeniu, nigdy stan pośredni. Trwałość to pokwitowanie, które klient dostaje po zakończonym przelewie — dokument, który obowiązuje nawet jeśli bank spłonie minutę później, bo zapis operacji trafił już do księgi rachunkowej, a nie tylko do pamięci kasjera.

Jak to działa

Rozwiń: jak to działa pod spodem

Autocommit. Domyślnym trybem pracy każdego z czterech silników z tej książki jest tryb, w którym każde pojedyncze polecenie SQL jest samo w sobie osobną, kompletną transakcją — wysyłamy `UPDATE`, silnik od razu opakowuje je w niewidoczne `BEGIN` i `COMMIT`, i jeśli polecenie się powiedzie, zmiana jest natychmiast trwała, bez żadnego dodatkowego polecenia z naszej strony. Ten tryb nazywa się autocommit i tłumaczy, dlaczego przez większość codziennej pracy w kliencie SQL nikt nie pisze `BEGIN` ani `COMMIT` — pojedyncze `UPDATE konta SET saldo = saldo - 500 WHERE id = 1;` zatwierdza się samo, w momencie wykonania. Problem pojawia się dopiero wtedy, gdy operacja logicznie wymaga więcej niż jednego polecenia — jak w przelewie z dwoma `UPDATE`ami — bo autocommit z definicji nie wie, że te dwa polecenia mają się trzymać razem; trzeba wtedy jawnie otworzyć transakcję poleceniami opisanymi na stronie BEGIN, COMMIT, ROLLBACK, SAVEPOINT.

Sprawdzenie, czy sesja pracuje w trybie autocommit, wygląda inaczej w każdym silniku:

Silnik	Domyślny tryb	Jak sprawdzić bieżące ustawienie
PostgreSQL	autocommit włączony	`SHOW autocommit;`
Oracle	autocommit włączony w większości klientów (SQL*Plus ma własne, opisane niżej)	`SHOW AUTOCOMMIT` w SQL*Plus
SQL Server	autocommit włączony, chyba że sesja jest w trybie `IMPLICIT_TRANSACTIONS`	`DBCC USEROPTIONS;` — wiersz `implicit_transactions`
MySQL / MariaDB	autocommit włączony	`SELECT @@autocommit;`

Przykład:

```
mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

Oracle jest tu wyjątkiem wartym osobnego zdania: klient SQL*Plus domyślnie ma autocommit wyłączony — polecenia DML czekają na jawny `COMMIT`, a wyjście z SQL*Plus bez niego wykonuje niejawny `COMMIT` przy poprawnym zamknięciu sesji albo `ROLLBACK` przy zerwaniu połączenia. Sterowniki aplikacyjne do Oracle (JDBC, ODBC) zwykle włączają autocommit z powrotem, więc zachowanie zależy od tego, czym się łączymy z bazą, nie tylko od samego silnika.

Dlaczego akurat tak. Autocommit jest domyślny nie dlatego, że transakcje wieloznaczeniowe są rzadkością — jest domyślny, bo odwrotne założenie byłoby niebezpieczne dla każdego, kto o transakcjach nie myśli. Gdyby domyślnym trybem było czekanie na jawny `COMMIT`, każde pojedyncze polecenie `UPDATE` wykonane przez kogoś, kto nie zna tego mechanizmu, zostawiałoby otwartą transakcję trzymającą blokady na zmienionych wierszach — aż do końca sesji, restartu klienta albo momentu, gdy ktoś przypadkiem zamknie okno. Autocommit sprawia, że najprostszy, najczęstszy przypadek użycia — jedno polecenie, jeden efekt — działa bezpiecznie bez żadnej wiedzy o transakcjach, a cenę płaci tylko ten, kto świadomie potrzebuje więcej niż jednego polecenia naraz i musi o to jawnie poprosić przez `BEGIN`.

Częste błędy

Zakładanie, że wiele poleceń w jednym oknie klienta to jedna transakcja. Wklejenie dziesięciu poleceń `UPDATE` do okna zapytań i wciśnięcie „wykonaj" nie tworzy między nimi żadnej więzi, jeśli sesja pracuje w trybie autocommit — każde polecenie zatwierdza się osobno, w momencie swojego wykonania. Awaria w połowie listy zostawia pierwsze pięć poleceń zatwierdzonych na trwałe i ostatnie pięć niewykonanych wcale — nie ma powrotu do stanu sprzed pierwszego polecenia, bo z punktu widzenia silnika nie było jednej transakcji, tylko dziesięć osobnych.

Traktowanie spójności jako czegoś, co silnik pilnuje automatycznie. Baza bez zdefiniowanych ograniczeń `CHECK`, `FOREIGN KEY` czy `NOT NULL` nie ma czego bronić — `ACID` gwarantuje, że reguły integralności, które istnieją, przetrwają transakcję nienaruszone, ale nie tworzy tych reguł za nas. Schemat zaprojektowany bez ograniczeń jest w pełni atomowy, izolowany i trwały, a mimo to potrafi zgromadzić dane, które nie mają żadnego logicznego sensu.

Mylenie trybu SQL*Plus w Oracle z zachowaniem samego silnika. Skrypt migracyjny napisany i przetestowany w SQL*Plus, gdzie każdy `COMMIT` jest jawny i widoczny, po przeniesieniu do aplikacji łączącej się przez sterownik z włączonym autocommitem zaczyna zatwierdzać każde polecenie osobno — różnica w zachowaniu bierze się z klienta, nie z bazy, i łatwo ją przeoczyć, jeśli ktoś testował skrypt tylko w jednym środowisku.

Mylenie izolacji z atomowością. Ktoś, kto zaobserwował, że dwie równoległe transakcje „nie widzą się nawzajem", bywa skłonny sądzić, że dowolna z nich w razie awarii automatycznie wycofa się razem z drugą — to dwie osobne gwarancje. Izolacja mówi tylko, że transakcje nie ujawniają sobie nawzajem niedokończonej pracy w trakcie działania; nie mówi nic o tym, co się stanie z każdą z nich osobno przy awarii. Transakcja A może zostać w pełni zatwierdzona, a transakcja B równoległa do niej — wycofana z zupełnie innego powodu, i żadna z tych dwóch rzeczy nie ma wpływu na drugą.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — przelew bez transakcji

Wyobrażamy sobie tabelę `konta (id INTEGER PRIMARY KEY, wlasciciel TEXT, saldo NUMERIC CHECK (saldo >= 0))` z dwoma wierszami: `(1, 'Konto A', 1000.00)` i `(2, 'Konto B', 500.00)`. Sesja w trybie autocommit wykonuje po kolei `UPDATE konta SET saldo = saldo - 500 WHERE id = 1;` i zaraz potem `UPDATE konta SET saldo = saldo + 500 WHERE id = 2;`, ale połączenie z bazą zrywa się dokładnie między tymi dwoma poleceniami. Jakie jest saldo obu kont zaraz po zerwaniu połączenia?

Odpowiedź

Konto A ma saldo 500.00, konto B wciąż ma 500.00 — pierwszy `UPDATE` zdążył się zatwierdzić samodzielnie, bo sesja pracowała w trybie autocommit i każde polecenie było własną transakcją. Drugi `UPDATE` nigdy się nie wykonał, bo połączenie padło wcześniej. W sumie zniknęło 500 złotych, które nie istnieją w żadnym z kont — dokładnie problem, przed którym chroni opakowanie obu poleceń w jedną jawną transakcję.

Ćwiczenie 2 — która litera zawiodła

Aplikacja rezerwacji sal wykonuje w jednej jawnej transakcji: `UPDATE sale SET zarezerwowana = TRUE WHERE id = 3;`, a zaraz potem wysyła e-mail z potwierdzeniem do klienta. Serwer pocztowy jest

niedostępny, więc kod aplikacji łapie wyjątek i… kontynuuje, zostawiając transakcję zatwierdzoną `COMMIT`em, mimo że e-mail nigdy nie poszedł. Klient dzwoni pół godziny później, twierdząc, że nie dostał potwierdzenia, a sala w systemie jest zajęta. Która litera ACID w ogóle nie ma tu zastosowania i dlaczego problem nie jest błędem bazy danych?

Odpowiedź

Żadna z czterech liter nie zawiodła — baza zrobiła dokładnie to, o co ją poproszono: zapisała `UPDATE` atomowo, spójnie, w izolacji od innych sesji i trwale. Wysyłka e-maila nie jest częścią transakcji bazodanowej i ACID nie obejmuje niczego poza operacjami na danych w silniku. To błąd projektu aplikacji — decyzja o tym, czy `COMMIT` sali ma czekać na potwierdzenie wysyłki maila, czy jest z nią rozdzielony, leży całkowicie po stronie logiki aplikacyjnej, nie bazy.

Pytania kontrolne

1. Dlaczego atomowość i trwałość to dwie różne gwarancje, skoro obie dotyczą tego, co się dzieje z zapisanymi danymi po awarii?

Odpowiedź

Atomowość dotyczy awarii w trakcie niedokończonej transakcji — gwarantuje, że taka transakcja zostanie wycofana w całości, tak jakby się nie zaczęła. Trwałość dotyczy awarii po tym, jak transakcja już się zatwierdziła (`COMMIT` się powiódł) — gwarantuje, że ta zatwierdzona zmiana przetrwa awarię, zamiast zniknąć razem z zawartością pamięci operacyjnej. Pierwsza chroni przed połowicznym zapisem, druga przed utratą zapisu, który formalnie już się udał.

2. Sesja MySQL ma `@@autocommit` równe 1 i wykonuje pojedyncze polecenie `DELETE FROM zamowienia WHERE status = 'anulowane';` bez wcześniejszego `BEGIN`. Czy da się to cofnąć poleceniem `ROLLBACK` wykonanym zaraz po?

Odpowiedź

Nie. Przy autocommit włączonym to pojedyncze polecenie jest już zatwierdzoną transakcją w momencie swojego wykonania — `ROLLBACK` wysłany później nie ma czego wycofać, bo żadna otwarta transakcja już nie istnieje. Żeby operację dało się cofnąć, `BEGIN` musiałby poprzedzić `DELETE`, zanim `ROLLBACK` miałby jakikolwiek sens.

Powiązane: BEGIN, COMMIT, ROLLBACK, SAVEPOINT · Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom · Log transakcyjny: WAL, redo, transaction log · Ograniczenia: NOT

NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT

Źródła: ISO/IEC 9075-2:2016 § 4.31 (transactions) · Jim Gray, Andreas Reuter, *Transaction Processing: Concepts and Techniques* (1993), rozdział 1 · dokumentacja PostgreSQL, rozdział „Transactions" · dokumentacja Oracle Database Concepts, sekcja „Transactions"

Revision #47
Created 2026-09-24 22:42:23 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:26 UTC by Tomasz Konieczny