Transakcje

Spis treści

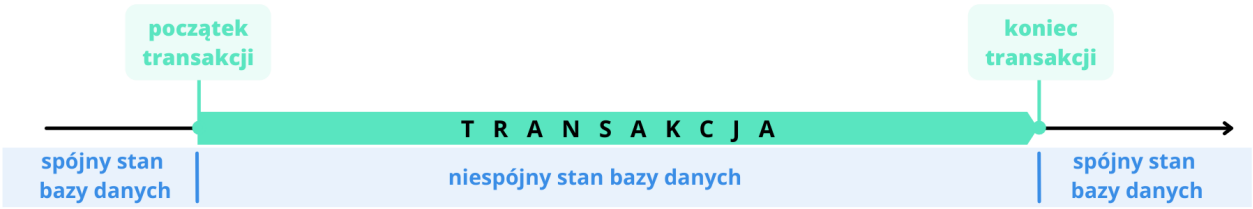
Transakcje

Czym jest transakcja bazodanowa

Zacznijmy od wyjaśnienia z Wikipedii:

> A database transaction symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a database.

Czyli transakcja jest jednostką pracy, która jest wykonywana na bazie danych. Transakcja reprezentuje dokonanie zmiany w bazie danych, czyli przejście z jednego spójnego stanu bazy danych do innego spójnego stanu bazy danych. Gdybyśmy mieli przedstawić to na schemacie, to wyglądałoby to tak:



Obraz 1. Transakcja bazodanowa

Transakcja traktowana jest jako pojedyncza operacja logiczna, która zmienia stan bazy danych ze spójnego na inny, też spójny. Innym używanym określeniem jest *operacja atomowa*, czyli niepodzielna. W obrębie jednej transakcji może być wykonanych wiele operacji. Każda transakcja musi składać się z

rozpoczęcia, wykonania i zamknięcia (trochę jak wstęp, rozwinięcie i zakończenie szkolnej rozprawki 😁). Transakcja może służyć do odczytu danych, ale także do ich zmiany.

Istnienie transakcji w bazach danych ma dwa główne cele:

- W systemach informatycznych z bazy danych korzysta jednocześnie wielu użytkowników. Może się tak zdarzyć, że będą oni korzystali jednocześnie z tych samych informacji. Transakcja ma zapewnić swojego rodzaju izolację pomiędzy zmianami wprowadzanymi przez obu użytkowników. Każdy użytkownik aplikacji (np. klient sklepu) będzie wtedy pracował na swojej dedykowanej transakcji i dopiero gdy zakończy ją z sukcesem (transakcja *A*), zmiany przez niego wprowadzone będą widoczne przez drugiego użytkownika (w transakcji *B*). Wrócimy jeszcze do tej izolacji,
- Zapewnienie spójności danych w przypadku wystąpienia awarii. Jeżeli w trakcie trwania transakcji zostanie odłączony prąd z komputera, na którym jest zainstalowana baza danych to żadne dane nie ulegają zmianie, bo transakcja nie zakończyła się sukcesem.

Transakcje muszą spełniać zasady `ACID`, które dokładniej omówię później, ale trzeba wiedzieć, że dzięki tym zasadom, możliwy jest jednoczesny dostęp do bazy danych wielu transakcji, tj. współbieżność. Za zarządzanie transakcjami odpowiada *DBMS*.

DBMS, *Database Management System* — czyli oprogramowanie, które umożliwia użytkownikom definiowanie, tworzenie, utrzymywanie i kontrolowanie dostępu do bazy danych. System zarządzania baz danych pozwala nam tworzyć konkretne bazy danych oraz na nich operować.

Schemat działania transakcji składa się z trzech podstawowych operacji: `begin`, `commit`, `rollback`.

Obraz 2. Operacje transakcjyjne

- **begin** — komenda rozpoczynająca transakcje,
- **commit** — komenda kończąca transakcję i zatwierdzająca wprowadzone zmiany,
- **rollback** — komenda kończąca transakcję i wycofująca wprowadzone zmiany.

Czyli możemy to zrozumieć w ten sposób, że transakcja rozpoczyna się operacją `begin`. Następnie wykonywane są modyfikacje na bazie danych. Modyfikacje te nie będą trwale widoczne dopóki nie wykonamy `commit`. Jeszcze inaczej można to zrozumieć w taki sposób, że dopóki nie wykonamy `commit` to nasze zmiany są w "trybie roboczym", możemy je wtedy albo zatwierdzić (`commit`), albo wycofać (`rollback`).

Tutaj mała uwaga - to w jaki sposób widoczne są modyfikacje pomiędzy poszczególnymi transakcjami jest zależne od poziomu izolacji transakcji. Dlatego też istnieje pewien wyjątek od tego co zostało napisane wyżej, wrócimy do tego zaraz.

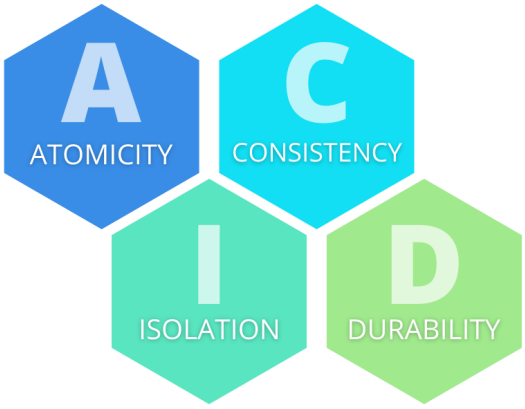
Jeżeli wykonany `commit` - zmiany zostają trwale zapisane w bazie danych. Zamiast `commit`, możemy wykonać `rollback`. Wtedy wszystkie zmiany wprowadzone w obrębie tej transakcji zostaną wycofane i żadna inna transakcja nie dowie się o tych modyfikacjach.

Istnieje tryb automatycznego commitowania transakcji nazywany *autocommit*. Jeżeli ten tryb jest włączony, wtedy każde zapytanie do bazy jest traktowane jako oddzielna transakcja, której wykonanie kończy się komendą *commit*. Co więcej, w takim przypadku nie ma możliwości jej wycofania komendą *rollback*.

Zauważ też, że nie wspominaliśmy o transakcjach na etapie pierwszego kontaktu z JDBC. Tam wszystkie transakcje były ustawione domyślnie w trybie *autocommit*. Gdy korzystamy z JDBC to domyślnym trybem jest *autocommit*.

Właściwości transakcji ACID

ACID to akronim od słów `Atomicity`, `Consistency`, `Isolation` i `Durability`, czyli niepodzielność, spójność, izolacja i trwałość. W odniesieniu do transakcyjności, spełnienie wymienionych właściwości zapewnia poprawne działanie bazy danych nawet w przypadku awarii.



Obraz 3. ACID

Omówmy teraz znaczenie każdego ze słów w odniesieniu do baz danych.

Atomicity (niepodzielność)

Transakcje często składają się z wielu zapytań SQL. Niepodzielność gwarantuje, że każda transakcja jest traktowana jak pojedyncza jednostka - albo wykonała się całkowicie, albo nie powiodła się również całkowicie. Jeżeli którekolwiek z SQL składających się na transakcję się nie powiedzie, cała transakcja kończy się niepowodzeniem - baza danych pozostaje niezmieniona. Atomowość taka musi być gwarantowana w każdej sytuacji, łącznie z błędami zasilania i awariami. Oczywiście ma to takie przełożenie, że jeżeli transakcja *A* rozpoczęła swoją pracę, to transakcja *B* zaobserwuje wprowadzane zmiany, dopiero gdy się one zakończą.

Przykładem atomowej transakcji jest przelew z konta bankowego *A* na konto *B*. Przelew składa się z dwóch operacji, wypłaty pieniędzy z konta *A* i zapisania ich na koncie *B*. Wykonanie tych operacji w sposób niepodzielny zapewnia, że baza danych pozostanie w spójnym stanie, to znaczy pieniądze nie są ani pobrane, ani doładowane, jeśli któraś z tych dwóch operacji skończy się niepowodzeniem.

Consistency (spójność)

Spójność zapewnia, że każda transakcja musi zmienić dane, których dotyczy, tylko w dozwolony sposób. Transakcja może przenosić bazę danych z jednego spójnego stanu do innego, który też jest spójny. Zachowane muszą być przy tym wszystkie określone wcześniej reguły, np. *constraints*. Spójność można również rozumieć w ten sposób, że po pomyślnym zapisie, aktualizacji lub usunięciu rekordu z bazy danych, każde każdy odczyt otrzyma najnowszą wartość wpisu.

Isolation (izolacja)

Izolacja określa, w jaki sposób transakcja jest widoczna dla innych użytkowników. Niższy poziom izolacji zwiększa możliwość uzyskiwania dostępu do tych samych danych w tym samym czasie przez wielu użytkowników. Jednocześnie zwiększana jest wtedy ilość **efektów współbieżności transakcji** (do poziomów izolacji i efektów współbieżności zaraz przejdziemy). Wyższy poziom izolacji zmniejsza ilość efektów współbieżności, które mogą napotkać użytkownicy, ale zwiększa szanse, że jedna transakcja zablokuje inną.

Durability (trwałość)

Trwałość gwarantuje, że zacommitowane transakcje nie mogą już zostać wycofane nawet w przypadku awarii systemu.

Wyobraźmy sobie teraz przykład z życia — przelew bankowy:

- Jeżeli z jednego konta znikają pieniądze, to na drugim koncie pieniądze się pojawiają,
- Kwota pobrana z konta źródłowego jest identyczna z kwotą dodaną na koncie docelowym,
- Zmiany wykonane przez proces przelewu stają się widoczne dopiero po pomyślnym wykonaniu transakcji, tzn. nie ma momentu, w którym kwota przelewu jest na obu kontach, bądź kwoty przelewu brakuje na obu kontach jednocześnie,
- Pomyślne wykonany przelew nie może zostać cofnięty - możemy natomiast wykonać kolejny.

Negatywne efekty współbieżności transakcji

Wracając do izolacji, bazy danych pozwalają na ustawienie różnych poziomów izolacji transakcji (o tym szczegółowo później). Wybranie konkretnego poziomu może skutkować pewnymi efektami współbieżności transakcji. Negatywne efekty współbieżności transakcji występują przy niskim poziomie izolacji transakcji. Czyli w zależności od tego jaki ustawimy poziom izolacji transakcji w bazie danych, możemy się wtedy natknąć na inne negatywne efekty podjętej decyzji. W pierwszej kolejności skupimy się na przykładowych efektach współbieżności transakcji, następnie przejdziemy do poziomów izolacji transakcji i tego który poziom należy ustawić żeby uniknąć danego efektu współbieżności. Możemy wyróżnić kilka **efektów współbieżności transakcji**:

Dirty Read

Dirty Read występuje, gdy jedna transakcja może odczytywać dane z wiersza, który został zmodyfikowany przez inną działającą transakcję i nie został jeszcze zacommitowany.

W poniższym przykładzie, `Transakcja 2` zmienia zawartość wiersza o id 10, natomiast zmiany te nie są

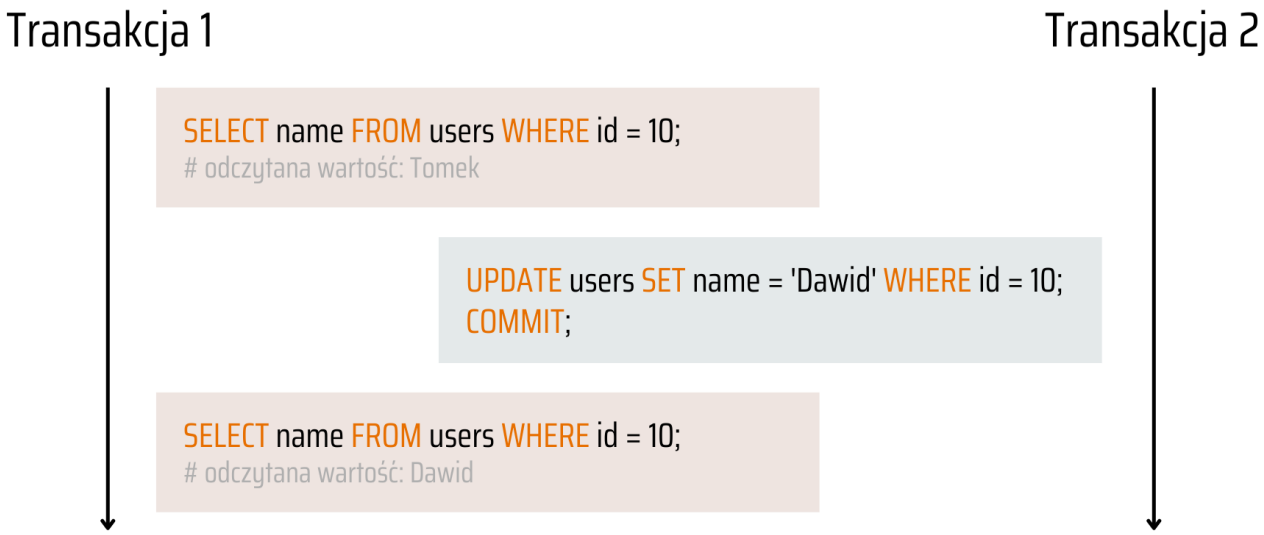
commitowane. `Transakcja 1` odczytuje dane, które nie zostały zacommitowane. Jeżeli w tym momencie `Transakcja 2` zrobi `rollback` wprowadzanych zmian (które były już odczytane przez `Transakcję 1`), wtedy `Transakcja 1` będzie miała zaczytane nieprawidłowe dane. Spójrz na poniższy schemat:

Obraz 4. Dirty Read

Non-repeatable Read

Non-repeatable Read (*niepowtarzalny odczyt*) ma miejsce, gdy w trakcie transakcji dany wiersz jest pobierany dwukrotnie i odczytane wartości będą się różnić pomiędzy odczytami.

W poniższym przykładzie, `Transakcja 1` odczytuje pewne dane. Następnie `Transakcja 2` aktualizuje te dane i transakcja zostaje pomyślnie zacommitowana. Oznacza to, że zmiany wprowadzone w ramach `Transakcji 2` powinny już być widoczne przez inne transakcje. Pojawia się teraz taka kwestia, czy jeżeli `Transakcja 1` wykona odczyt ponownie to czy powinna zobaczyć zacommitowane zmiany w obrębie `Transakcji 2` czy też nie. Będzie to wynikało z ustawionego poziomu izolacji transakcji. Jeżeli wystąpi efekt **Non-repeatable Read**, będzie to oznaczało, że `Transakcja 1` odczyta ten sam wiersz dwukrotnie i otrzyma inny wynik. Spójrz na poniższy schemat:

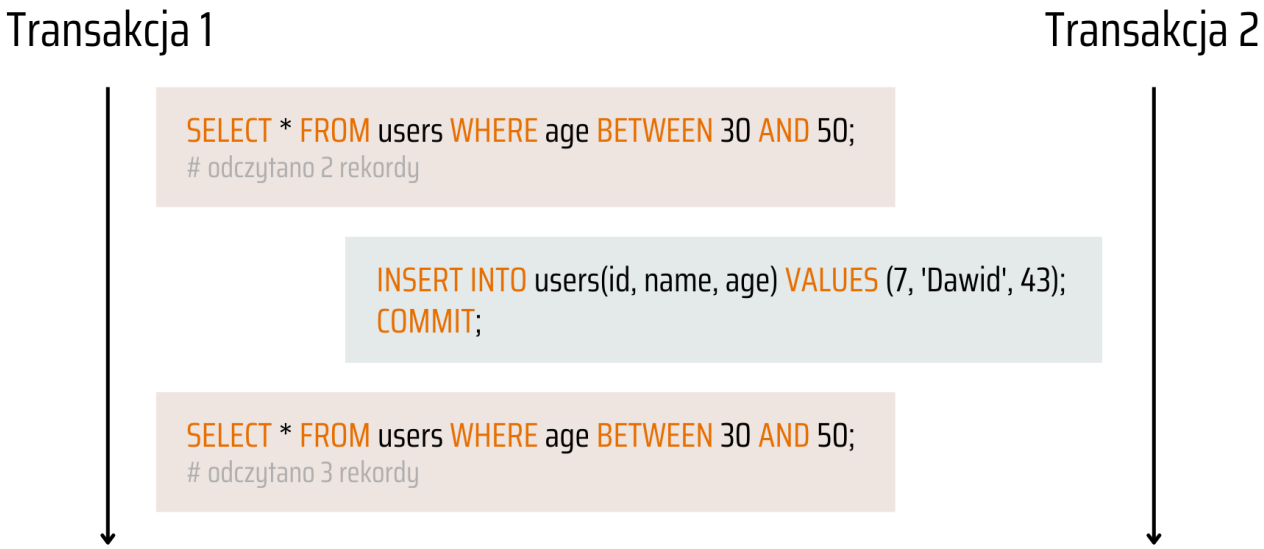


Obraz 5. Non-repeatable Read

Phantom Read

Phantom Read ma miejsce, gdy w trakcie trwania transakcji, do odczytywanych rekordów dodawane są nowe wiersze lub usuwane przez inną transakcję.

W poniższym przykładzie, `Transakcja 1` odczytuje pewne dane w jakimś zakresie. Następnie `Transakcja 2` dodaje nowe dane we wspomnianym zakresie transakcja zostaje pomyślnie zacommitowana. Oznacza to, że zmiany wprowadzone w ramach `Transakcji 2` powinny już być widoczne przez inne transakcje. Pojawia się teraz taka kwestia, czy jeżeli `Transakcja 1` wykona odczyt z tym samym zakresem danych ponownie to czy powinna zobaczyć zacommitowane zmiany w obrębie `Transakcji 2` czy też nie. Będzie to wynikało z ustawionego poziomu izolacji transakcji. Jeżeli wystąpi efekt **Phantom Read**, będzie to oznaczało, że `Transakcja 1` odczytując dane z tego samego zakresu, zobaczy nowe wpisy. Spójrz na poniższy schemat:



Obraz 6. Phantom Read

Poziomy izolacji transakcji

Sposobem na uniknięcie powyższych problemów jest zastosowanie jednego z poziomów izolacji

opracowanych przez ANSI, American National Standards Institute

Poziom izolacji transakcji jest parametrem określającym, jak wykonywana transakcja jest widziana przez inne transakcje. Można to też opisać jako sposób określenia, kiedy zmiany wprowadzone przez jedną transakcje stają się widoczne dla innych transakcji. Poziom izolacji ustawia się jako właściwość bazy danych.

Im niższy poziom izolacji, tym większy jest dostęp to tych samych danych w tym samym czasie, ale razem z dostępnością rośnie ryzyko wystąpienia negatywnych efektów współbieżności. Im poziom izolacji jest wyższy, tym to ryzyko jest niższe, za to rośnie możliwość, że jedna transakcja zablokuje dostęp do danych innej transakcji.

Obraz 7. Poziomy izolacji

Jakie wyróżniamy **poziomy izolacji transakcji**:

- **Read uncommitted** — Transakcja ma dostęp do niezacommitowanych przez inną transakcję danych (co może powodować `dirty read`),
- **Read committed** — Transakcja ma dostęp jedynie do danych zacommitowanych przez inne transakcje,
- **Repeatable read** — Zapytanie odczytujące dane zawsze zwraca te same wartości, nawet jeżeli jakaś współbieżna transakcja zmodyfikowała odczytywane dane,
- **Serializable** — Wszystkie współbieżne transakcje zawsze zwracają te same wartości, co w praktyce wymusza sekwencyjne wykonywanie tych transakcji.

Jakie są konsekwencje wyboru konkretnego poziomu izolacji? Jeden aspekt takiego wyboru został już omówiony wcześniej - mowa tutaj o negatywnych efektach współbieżności transakcji. Z drugiej strony stosując najwyższy poziom izolacji, należy liczyć się z tym, że możemy zablokować dostęp do danych.

Na czym polega ta blokada dostępu? Poniższy opis będzie mocno uproszczony ☺.

Najwyższy poziom izolacji blokuje dostęp do danych. Baza danych zakłada wtedy blokadę na dostęp do danych (tzw. *lock*) i uniemożliwia dostęp do tych samych danych więcej niż jednej transakcji. Jeżeli wiele transakcji chce jednocześnie pracować z tymi samymi danymi, może to doprowadzić do zablokowania kolejnych transakcji w oczekiwaniu na skończenie pracy przez poprzednie transakcje, które założyły blokadę. W uproszczeniu - `Transakcja 2` będzie czekała aż `Transakcja 1` skończy co miała zrobić. Dodajmy do tego skalę 10 000 użytkowników korzystających z systemu jednocześnie i mamy ciekawy problem.

Z tego właśnie powodu, w programowaniu decyzje trzeba podejmować świadomie. Każda decyzja ma swoje konsekwencje i dlatego zanim zostanie podjęta - trzeba te konsekwencje znać.

Różne silniki baz danych mogą inaczej implementować poziomy izolacji. Przykładowo w dokumentacji PostgreSQL znajdziemy taką informację:

> Read Committed is the default isolation level in PostgreSQL.

Poniżej zebraliśmy różne poziomy izolacji zestawione z tym jakim efektom współbieżności mają one zapobiegać.

Tabela 1. Poziom izolacji vs efekty współbieżności

Poziom	Dirty Reads	Non-repeatable Reads	Phantom Reads
Read uncommitted	Może wystąpić	Może wystąpić	Może wystąpić
Read committed	Nie występuje	Może wystąpić	Może wystąpić
Repeatable read	Nie występuje	Nie występuje	Może wystąpić
Serializable	Nie występuje	Nie występuje	Nie występuje