Poziomy izolacji i co który naprawdę gwarantuje

> Poziom izolacji to deklaracja, ile niespójności współbieżności godzimy się zobaczyć w zamian za przepustowość — a to, co standard obiecuje na papierze i to, co faktycznie robi konkretny silnik na danym poziomie, to dwie różne rzeczy, które ta strona rozdziela.

◫◫ Skróty użyte na tej stronie (4)

Skrót	Co znaczy
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
MVCC	Multi-Version Concurrency Control: baza trzyma kilka wersji wiersza, żeby czytający nie blokował piszącego
SET	Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

◫◫ ZMIENIA SYSTEM

Ustawienie poziomu izolacji dla całej sesji

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Ustawienie działa dla wszystkich kolejnych transakcji tej sesji, dopóki ktoś go nie zmieni ponownie albo sesja się nie zakończy — przydatne, gdy cała aplikacja albo cały skrypt ma pracować na podniesionym poziomie przez cały czas swojego działania.

Silnik	Ustawienie dla sesji

PostgreSQL	`SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL REPEATABLE READ;`
Oracle	brak odpowiednika dla całej sesji — poziom ustawia się tylko per transakcja
SQL Server	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` (obowiązuje sesję do zmiany)
MySQL / MariaDB	`SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;`

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ ZMIENIA SYSTEM

Ustawienie poziomu izolacji tylko dla następnej transakcji

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Ustawienie działa wyłącznie dla transakcji, która się właśnie zaczęła — po jej `COMMIT` albo `ROLLBACK` kolejna transakkcja wraca do poziomu domyślnego sesji. To bezpieczniejszy wybór niż zmiana poziomu dla całej sesji, jeśli podniesiony poziom izolacji potrzebny jest tylko do jednej konkretnej operacji, na przykład przelewu środków albo aktualizacji stanu magazynowego wrażliwej na skos zapisu.

Przykład:

```
produkcja=# BEGIN;
BEGIN
produkcja=*# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET
produkcja=*# SHOW transaction_isolation;
 transaction_isolation
-----------------------
 serializable
(1 row)

produkcja=*# COMMIT;
COMMIT
produkcja=# SHOW transaction_isolation;
```

```
 transaction_isolation
-----------------------
 read committed
(1 row)
```

Po `COMMIT` poziom izolacji wraca do domyślnego `read committed` tej sesji — podniesienie dotyczyło wyłącznie transakcji, która się właśnie zakończyła.

Oracle nie rozróżnia tych dwóch wariantów w ogóle — `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;` zawsze dotyczy wyłącznie bieżącej, jeszcze niezatwierdzonej transakcji i musi zostać powtórzone po każdym `COMMIT`/`ROLLBACK`, jeśli podniesiony poziom ma obowiązywać dalej.

SET = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

Po co to istnieje

> **Pojęcie: poziom izolacji.** Ustawienie transakcji albo sesji, które określa, na które z anomalii współbieżności opisanych na stronie Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom transakcja pozostaje podatna, a przed którymi jest chroniona. Im wyższy poziom izolacji, tym mniej anomalii transakcja może zaobserwować — ale też tym więcej pracy silnik musi wykonać, żeby tę ochronę zapewnić, czy to przez trzymanie blokad dłużej, czy przez utrzymywanie dodatkowych wersji wierszy opisanych na stronie MVCC: dlaczego czytający nie blokuje piszącego. Poziom izolacji nie jest więc wyborem między „bezpiecznie" a „niebezpiecznie" — jest wyborem punktu na skali między spójnością widoku danych a przepustowością systemu pod obciążeniem współbieżnym.

Standard SQL definiuje cztery poziomy izolacji, uporządkowane od najsłabszego do najsilniejszego, i opisuje każdy z nich wyłącznie przez to, które z trzech oficjalnych anomalii dopuszcza:

Poziom izolacji	Brudny odczyt	Niepowtarzalny odczyt	Fantom
READ UNCOMMITTED	dopuszczalny	dopuszczalny	dopuszczalny
READ COMMITTED	zablokowany	dopuszczalny	dopuszczalny
REPEATABLE READ	zablokowany	zablokowany	dopuszczalny
SERIALIZABLE	zablokowany	zablokowany	zablokowany

`SERIALIZABLE` to poziom, na którym wynik działania wielu transakcji równoległych musi być identyczny z wynikiem, jaki dałoby ich wykonanie jedna po drugiej w jakiejś kolejności — to jedyny poziom ze standardu, który formalnie chroni też przed skosem zapisu opisanym na poprzedniej stronie, choć nie każda implementacja realizuje tę ochronę tym samym mechanizmem.

Jak to działa

Rozwiń: jak to działa pod spodem

Prawda o domyślnych poziomach. Tabela wyżej to litera standardu — to, co silniki robią naprawdę z domyślnym ustawieniem, różni się znacząco między nimi:

Silnik	Domyślny poziom izolacji
PostgreSQL	READ COMMITTED
Oracle	READ COMMITTED
SQL Server	READ COMMITTED
MySQL / MariaDB (InnoDB)	REPEATABLE READ

MySQL/InnoDB jest tu jedynym wyjątkiem — startuje domyślnie o jeden stopień wyżej niż pozostała trójka, co oznacza, że kod migrowany z PostgreSQL czy SQL Server na MySQL bez świadomej zmiany ustawienia automatycznie zyskuje ochronę przed niepowtarzalnym odczytem, której mógł nie mieć wcześniej — a kod migrowany w drugą stronę tę ochronę traci, jeśli nikt jej jawnie nie ustawi.

READ UNCOMMITTED w PostgreSQL działa jak READ COMMITTED. PostgreSQL formalnie przyjmuje polecenie `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;` bez błędu, ale pod spodem traktuje je dokładnie tak samo jak `READ COMMITTED` — silnik nigdy nie pozwala jednej transakcji zobaczyć niezatwierdzonych zmian drugiej, niezależnie od tego, o jaki najniższy poziom prosimy. Powodem jest architektura MVCC opisana na stronie MVCC: dlaczego czytający nie blokuje piszącego: każdy odczyt w PostgreSQL i tak działa na spójnej migawce danych, więc `READ UNCOMMITTED` w znaczeniu „pokaż mi też niezatwierdzone zmiany innych" nie ma tam żadnej realizacji technicznej. Ten sam mechanizm sprawia, że w PostgreSQL brudny odczyt w ogóle nie jest osiągalny, niezależnie od zadeklarowanego poziomu.

Przykład:

```
produkcja=# BEGIN;
BEGIN
```

```
produkcja=*# SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET
produkcja=*# SHOW transaction_isolation;
 transaction_isolation
-----------------------
 read uncommitted
(1 row)

produkcja=*# SELECT wartosc FROM zamowienia WHERE id = 104;
-- widzi wyłącznie zatwierdzone dane, mimo zadeklarowanego READ UNCOMMITTED
```

REPEATABLE READ w InnoDB blokuje też fantomy. MySQL/InnoDB idzie dalej niż wymaga standard: na poziomie `REPEATABLE READ` — swoim ustawieniu domyślnym — chroni nie tylko przed niepowtarzalnym odczytem, ale też przed fantomem, dzięki mechanizmowi blokad przerw między kluczami (gap locks), który blokuje nie tylko istniejące wiersze, ale też przestrzeń, w której mogłyby powstać nowe wiersze pasujące do warunku `WHERE`. To sprawia, że tabela zgodności ze standardu, poprawna dla PostgreSQL i większości innych silników, jest myląca dla InnoDB — tam `REPEATABLE READ` w praktyce zachowuje się bliżej `SERIALIZABLE`, choć formalnie nim nie jest, bo nie chroni przed skosem zapisu w każdym scenariuszu.

SNAPSHOT w SQL Server jako osobny tryb. SQL Server, poza czterema standardowymi poziomami, oferuje piąty, niestandardowy tryb `SNAPSHOT`, który trzeba osobno włączyć na poziomie bazy danych przed użyciem:

```
ALTER DATABASE produkcja SET ALLOW_SNAPSHOT_ISOLATION ON;
```

`SNAPSHOT` daje każdej transakcji spójny obraz danych z momentu jej rozpoczęcia, budowany na wersjonowaniu wierszy w `tempdb` — mechanizm opisany dokładniej na stronie MVCC: dlaczego czytający nie blokuje piszącego — i formalnie chroni przed tymi samymi trzema anomaliami co `SERIALIZABLE`, ale bez blokowania czytających przez piszących, kosztem możliwego błędu `3960` przy konflikcie zapisu wykrytym dopiero przy `COMMIT`. Pokrewny, ale osobny tryb `READ_COMMITTED_SNAPSHOT` zmienia zachowanie domyślnego `READ COMMITTED` tak, żeby też korzystał z wersjonowania zamiast blokad odczytu, nie zmieniając przy tym nazwy poziomu izolacji zgłaszanej aplikacji.

Częste błędy

Podnoszenie poziomu izolacji tam, gdzie problemem jest jeden konkretny wiersz, nie cała transakcja. `SERIALIZABLE` chroni przed każdą anomalią, ale kosztem częstszych błędów konfliktu wymagających ponowienia transakcji po stronie aplikacji i mniejszej przepustowości pod

obciążeniem. Jeśli faktyczny problem to utracona aktualizacja jednego konkretnego wiersza — jak w scenariuszu magazynowym ze strony Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom — zwykle tańszym i równie skutecznym rozwiązaniem jest blokada jawna `SELECT ... FOR UPDATE` opisana na stronie Blokady: współdzielone, wyłączne, eskalacja albo atomowy `UPDATE produkty SET ilosc = ilosc - 3`, zamiast podnoszenia izolacji całej transakcji do najwyższego poziomu.

Ślepe kopiowanie poziomu izolacji między silnikami. Kod przenoszony z PostgreSQL, gdzie `REPEATABLE READ` chroni przed niepowtarzalnym odczytem, ale nie przed fantomem, na MySQL/InnoDB, gdzie ten sam poziom domyślnie chroni też przed fantomem dzięki gap lockom, zachowuje się inaczej niż testy sugerowały — nie dlatego, że coś jest zepsute, tylko dlatego, że nazwa poziomu jest ta sama, a jego rzeczywista implementacja nie.

Zakładanie, że `READ UNCOMMITTED` w PostgreSQL da przyspieszenie znane z innych silników. Deklaracja tego poziomu w PostgreSQL nie zmienia niczego w wydajności ani w widoczności danych — silnik i tak zawsze pracuje jak na `READ COMMITTED`. Ktoś, kto obniża poziom izolacji w nadziei na szybsze odczyty tak, jak zrobiłby to w innym silniku, nie dostaje żadnej korzyści, tylko kod, który wygląda na świadomie mniej bezpieczny, nie będąc w istocie ani trochę szybszym.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — który poziom naprawdę pomaga

Aplikacja księgowa liczy dzienny raport sprzedaży, wykonując w jednej transakcji trzy zapytania `SELECT SUM(wartosc) FROM zamowienia WHERE data = CURRENT_DATE;` w odstępie kilku sekund, i porównuje wyniki między sobą jako kontrolę spójności. Na poziomie `READ COMMITTED` te trzy odczyty czasem się różnią, mimo że nikt nie modyfikuje istniejących wierszy — tylko dokładane są nowe zamówienia z bieżącego dnia. Który standardowy poziom izolacji rozwiąże ten konkretny problem i dlaczego akurat on, a nie `REPEATABLE READ`?

Odpowiedź

`SERIALIZABLE`, a przynajmniej pełna ochrona przed fantomem — nowe zamówienia dokładane w trakcie trwania transakkcji A to fantom, nie niepowtarzalny odczyt, bo żaden istniejący wiersz

się nie zmienia, tylko przybywają nowe wiersze spełniające ten sam warunek `WHERE`. `REPEATABLE READ` ze standardu formalnie chroni tylko przed niepowtarzalnym odczytem, nie przed fantomem — na silniku, który trzyma się litery standardu (jak PostgreSQL), sam `REPEATABLE READ` by tu nie wystarczył. Warto zauważyć, że na MySQL/InnoDB, gdzie `REPEATABLE READ` dodatkowo blokuje fantomy przez gap locki, ten sam poziom by problem rozwiązał — dokładnie ilustracja tego, że nazwa poziomu nie gwarantuje identycznego zachowania między silnikami.

Pytania kontrolne

1. Dlaczego tabela zgodności poziomów izolacji ze standardu SQL nie jest wystarczającą dokumentacją zachowania konkretnego silnika?

Odpowiedź

Bo standard określa wyłącznie, które anomalie poziom izolacji *musi* blokować jako minimum — nie zabrania implementacji blokować więcej, niż standard wymaga. MySQL/InnoDB na `REPEATABLE READ` blokuje też fantomy, których standard na tym poziomie nie wymaga, a PostgreSQL na `READ UNCOMMITTED` i tak blokuje brudne odczyty, mimo że standard by na to pozwolił. Rzeczywiste zachowanie trzeba sprawdzić w dokumentacji konkretnego silnika, nie wywodzić wyłącznie z nazwy poziomu.

2. Transakcja w SQL Server działa w trybie `SNAPSHOT` i przy próbie `COMMIT` dostaje błąd 3960. Co się właśnie stało i czym różni się to od zwykłego oczekiwania na blokadę?

Odpowiedź

Transakcja w trybie `SNAPSHOT` pracowała na spójnym obrazie danych z momentu swojego rozpoczęcia, ale w międzyczasie inna transakcja zmieniła i zatwierdziła te same wiersze — silnik wykrywa ten konflikt dopiero przy próbie `COMMIT`, zamiast każąc transakkcji czekać na blokadę w trakcie działania. Zamiast oczekiwania SQL Server zwraca błąd, a aplikacja musi sama ponowić całą transakcję od początku, korzystając już ze świeżego stanu danych — cena za to, że tryb `SNAPSHOT` nie blokuje czytających przez piszących w trakcie działania.

Powiązane: Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom · Blokady: współdzielone, wyłączne, eskalacja · MVCC: dlaczego czytający nie blokuje piszącego · BEGIN, COMMIT, ROLLBACK, SAVEPOINT

Źródła: ISO/IEC 9075-2:2016 § 4.32 · dokumentacja PostgreSQL, rozdział „Transaction Isolation" · dokumentacja MySQL 8.0, rozdział „InnoDB Locking and Transaction Model" · dokumentacja Microsoft SQL Server, „SET TRANSACTION ISOLATION LEVEL" i „ALTER DATABASE SET Options"

Revision #46
Created 2026-09-24 22:42:28 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:30 UTC by Tomasz Konieczny