

Transakcje bazodanowe

(w oparciu o MySQL/MariaDB)

<https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html>

<https://mariadb.com/kb/en/set-transaction/>

Transakcje bazodanowe

Transakcja bazodanowa - jest to zbiór operacji SQL na danych, który ma za zadanie wykonać się w całości lub wcale.

W szczegółach cechy transakcji mogą nieznacznie różnić się pomiędzy sobą w zależności od systemu zarządzania bazą danych (np. poziomy izolacji transakcji).

W systemach MySQL/MariaDB transakcje w pełni obsługuje silnik InnoDB i przez długi czas był jedynym silnikiem o tej cesze. Obecnie transakcje obsługują również m.in. silniki: MyRocks i TokuDB. Natomiast większość silników nie posiada zaimplementowanej obsługi transakcji.

Transakcję najczęściej oficjalnie rozpoczynamy (odpowiednim poleceniem SQL) i kończymy (poleceniem akceptującym zmiany naniesione przez polecenia SQL lub cofamy w całości).

Transakcje bazodanowe

W MySQL/MariaDB w InnoDB (obecnie silnik domyślny) transakcję rozpoczynamy automatycznie i kończymy automatycznie dla danego pojedynczego polecenia SQL, pod warunkiem, że ustawiona jest zmienna ***SET SESSION autocommit := 1;*** (stan domyślny).

Jeżeli zmienną ustawimy na 0, to będzie oznaczało, że zestaw poleceń SQL w ramach automatycznie rozpoczętej transakcji pierwszym poleceniem SQL będziemy musieli zatwierdzić commit-em (ew. cofnąć rollback-iem).

```
mysql> SET SESSION autocommit := 0;
Query OK, 0 rows affected
mysql> delete from student where ID_student=36;
Query OK, 1 row affected
mysql> insert into student (imie, nazwisko)
values ('1','2');
Query OK, 1 row affected
mysql> commit;
```

Transakcje bazodanowe

Transakcja jest bardzo ważnym elementem oprogramowania komputerowego opierającym się o bazę danych, gdzie dostęp do aplikacji ma wielu użytkowników w jednakowym czasie. W wielu przypadkach niezastosowanie transakcji może sprawić np. niezamierzone umieszczenie błędnych danych w tabelach (np. systemy rezerwacyjne miejsc w autobusach, kinach, koncertach itp. - gdzie pojedynczy proces rezerwacji składa się z kilku zapytań SQL - np. sprawdzenie liczby dostępnych miejsc dla danego fragmentu kursu, umieszczenie rekordu rezerwacji).

Transakcja jest również ważnym elementem dotyczącym zabezpieczenia aplikacji przed niepożądanym skutkiem awarii - np. awarii systemu operacyjnego, nagłego braku zasilania serwera - polecenia SQL wchodzące w skład transakcji nie wykonają się w ogóle zamiast np. fragmentarycznie - np. przelew pieniędzy z jednego konta na drugie.

Transakcje bazodanowe

Transakcja składa się z trzech etapów:

- rozpoczęcia transakcji, np. poleceniami: *start transaction* lub *begin* lub pierwszym poleceniem SQL (patrz: slajd ze zmienną autocommit)
- wykonywanie sekwencji poleceń SQL. W trakcie transakcji istnieje możliwość cofnięcia do wcześniej zapisanego stanu danych (*rollback to nazwa_stanu*) za pomocą polecenia *savepoint nazwa_stanu*
- zakończenia transakcji, polecenia: *commit* (zatwierdzenie nowego stanu danych), *rollback* (cofnięcie transakcji).

Zerwane połączenie z bazą danych lub awaria procesu BD w systemie operacyjnym anuluje wprowadzone zmiany.

Transakcje bazodanowe

Transakcja składa się z trzech etapów (przykład):

```
mysql> begin;
```

Query OK, 0 rows affected

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
32	Jan	Muszka
33	Janina	Nowakowska

3 rows in set

```
mysql> savepoint stan_1;
```

Query OK, 0 rows affected

```
mysql> delete from student where ID_student = 33;
```

Query OK, 1 row affected

Transakcje bazodanowe

Transakcja składa się z trzech etapów (przykład):

```
mysql> rollback to stan_1;  
Query OK, 0 rows affected
```

```
mysql> delete from student where ID_student = 32;  
Query OK, 1 row affected
```

```
mysql> commit;  
Query OK, 0 rows affected
```

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
33	Janina	Nowakowska

2 rows in set

Transakcje bazodanowe

W zależności od ustawień administracyjnych, automatyczne kończenie transakcji może mieć również miejsce po następujących poleceniach: `alter table`, `create index`, `create table`, `drop database`, `drop table`, `lock tables`, `rename table`, `set autocommit = 1`, `truncate`, `unlock tables`.

Zdarza się również w SZBD (w tym różnych ich wersjach), że niektóre powyższe polecenia (np. *alter table*) są blokowane do momentu zakończenia transakcji operującej na spornej tabeli.

Np. w MariaDB od wersji 5.5: <https://mariadb.com/kb/en/metadata-locking/>

Transakcje bazodanowe - zabicie procesu blokującego transakcję

Podczas wykonywania transakcji może zdarzyć się, że nastąpi zablokowanie tabeli na wyłączność z niemożnością jej odblokowania przed upływem czasu ustawionego przez administratora MySQL/MariaDB (`lock_wait_timeout` - https://mariadb.com/kb/en/server-system-variables/#lock_wait_timeout), a tym samym może się zdarzyć, że aplikacja kliencka może przestać odpowiadać (zawiesi się).

Z pomocą przychodzi np. polecenie *kill id_procesu* wykonane w ramach drugiego połączenia z bazą danych.

Przykład na kolejnym slajdzie.

Transakcje bazodanowe - zabicie procesu blokującego transakcję

Przykład:

1. Przejęcie przez transakcję tabeli na wyłączność (o tym na kolejnych slajdach):

```
mysql> begin;
```

Query OK, 0 rows affected

```
mysql> select * from produkt for update;
```

2. Próba zmiany nazwy tabeli *produkt* z poziomu interfejsu programu klienckiego. Jeżeli program kliencki przestanie odpowiadać, można połączyć się ponownie z poziomu nowej instancji programu klienckiego z SZBD i wykonać polecenie:

```
mysql> show processlist;
```

a następnie:

```
mysql> kill 4192;
```

Podstawy baz danych / Artur Niewiarowski

gdzie liczba po *kill* to numer procesu z tabeli *processlist* w kolumnie *Id*.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

W ramach transakcji bazodanowych można zastosować parametry ograniczające dostęp do danych. Są one jednym z mechanizmów współdzielenia zasobów bazy danych (obok tzw. wielowersyjności, backupu i dziennika logów).

W zależności od SZBD może być ich zdefiniowanych więcej oraz pod innymi nazwami lub z dodatkowymi cechami.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Wielowersyjność (ang. Multiversion Concurrency Control, MVCC) - to technika zarządzania współbieżnością w bazach danych, która pozwala na jednoczesne wykonywanie operacji odczytu i zapisu przez wiele transakcji, bez konieczności blokowania dostępu do danych.

W systemach baz danych stosujących MVCC, każda transakcja odczytuje dane z określonego momentu czasu (wersji danych) - zazwyczaj z momentu rozpoczęcia transakcji. W wyniku tego, transakcje odczytujące dane nie muszą czekać na zakończenie innych transakcji zapisujących, ponieważ odczytują one dane z wcześniejszej wersji. Podobnie, transakcje zapisujące nie blokują transakcji odczytujących, gdyż tworzą nową wersję danych, która nie jest jeszcze widoczna dla innych transakcji.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

MVCC poprawia wydajność systemu baz danych w środowiskach, gdzie wiele transakcji czytających i zapisujących jest wykonywanych równocześnie. Pozwala to na lepszą skalowalność oraz minimalizuje problemy związane z blokadami i zakleszczeniami (ang. deadlocks). Należy jednak pamiętać, że MVCC nie rozwiązuje wszystkich problemów związanych z zarządzaniem współbieżnością i może wprowadzać dodatkowe wyzwania, takie jak konieczność zarządzania wieloma wersjami danych.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Dziennik logów (ang. transaction log, log file) - jest to ważny element w systemach baz danych, który służy do przechowywania informacji o wszystkich operacjach i transakcjach wykonywanych na danej bazie danych. Głównym celem dziennika logów jest zapewnienie niezawodności, trwałości i odzyskiwania danych w przypadku awarii systemu.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Kiedy transakcja jest wykonywana na bazie danych, system rejestruje wszystkie zmiany w **dzienniku logów** przed ich zastosowaniem do bazy danych. Dziennik logów zapisuje informacje takie jak typ operacji (np. wstawienie, aktualizacja, usunięcie), identyfikator transakcji, wartości przed i po zmianie oraz inne metadane.

W przypadku awarii systemu, takiej jak nagłe wyłączenie zasilania czy awaria sprzętu, dziennik logów umożliwia odzyskanie utraconych danych poprzez tzw. "redo" (ponowne zastosowanie) zatwierdzonych transakcji oraz "undo" (cofnięcie) niezatwierdzonych transakcji. Dzięki temu, baza danych może zostać przywrócona do spójnego i aktualnego stanu.

Dziennik logów jest również wykorzystywany w innych procesach, takich jak replikacja danych pomiędzy serwerami czy tworzenie kopii zapasowych bazy danych.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

W ramach transakcji w MySQL/MariaDB możemy stosować:

- *klauzulę for update* - tj. *exclusive lock*, wywoływana jest na końcu polecenia *select*. Blokuje rekordy tabeli wywołane poleceniem *select* (po wartości PK w *where*) do momentu zatwierdzenia lub cofnięcia transakcji. Inna transakcja nie może założyć blokady, ani nie może modyfikować danych zawartych w zablokowanych rekordach tabeli,
- *klauzulę lock in share mode* - tj. *shared lock*, wywoływana jest również na końcu polecenia *select*. Działa podobnie jak blokada omawiana powyżej, natomiast umożliwia innym transakcjom założenie blokady na tą samą tabelę oraz odczyt danych,
- dodatkowo poziomy (stopnie) izolacji - umożliwiają określenie domyślnej izolacji transakcji od innych transakcji oraz wolnych poleceń SQL.

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Dodatkowo poza transakcją można zablokować wybrane tabele na wyłączność sesji poleceniem *lock tables*, a następnie odblokować poleceniem *unlock tables*.

Blokady są następujące:

- read
- read local
- write
- low_priority write
- write concurrent

Więcej szczegółów: <https://mariadb.com/kb/en/lock-tables>

Parametry ograniczające dostęp do danych

Przykład użycia blokady „read”

```
root local - Console
File Edit View Window
mariadb> lock tables uzytkownicy read;
Query OK, 0 rows affected (0.00 sec)
mariadb>
```

```
root local - Console -- Processing
File Edit View Window
mariadb> select * from uzytkownicy limit 1;
+-----+-----+-----+-----+-----+
| ID_uzytkownicy | nazwisko | imie | login      | haslo |
+-----+-----+-----+-----+-----+
| 8              | nowakowski | jan | jnowakowski | NULL  |
+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)

mariadb> delete from uzytkownicy;
```

Blokada możliwości modyfikacji danych w tabeli z poziomu innych sesji, ale nie ich odczytu

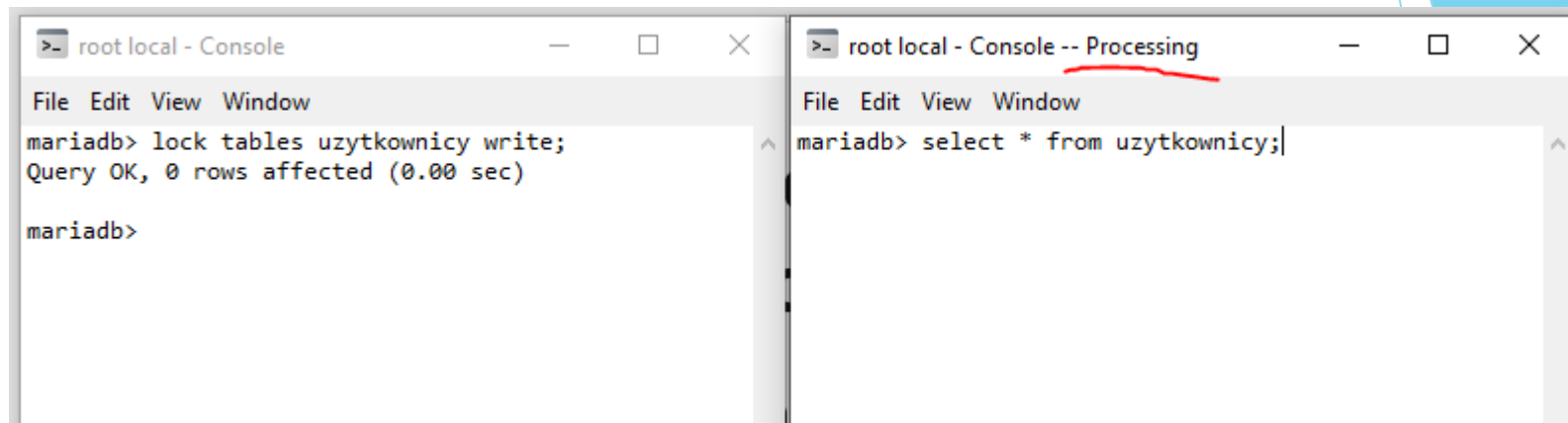
```
mariadb> unlock tables;
Query OK, 0 rows affected (0.00 sec)
mariadb> |
```

```
mariadb> delete from uzytkownicy;
Query OK, 7 rows affected (48.29 sec)

mariadb>
```

Parametry ograniczające dostęp do danych

Przykład użycia blokady „write”



The image shows two terminal windows side-by-side. The left window, titled 'root local - Console', shows a MariaDB session where the command 'lock tables uzytkownicy write;' has been executed successfully, returning 'Query OK, 0 rows affected (0.00 sec)'. The right window, titled 'root local - Console -- Processing', shows a MariaDB session where the command 'select * from uzytkownicy;' is being entered. The title bar of the right window has a red underline under the text '-- Processing', indicating that the query is being processed in the background.

```
root local - Console
File Edit View Window
mariadb> lock tables uzytkownicy write;
Query OK, 0 rows affected (0.00 sec)

mariadb>

root local - Console -- Processing
File Edit View Window
mariadb> select * from uzytkownicy;
```

Blokada możliwości modyfikacji danych oraz ich odczytu z poziomu innych sesji

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

```
mysql> select * from produkt;
```

ID_produk	nazwa_produk
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

```
7 rows in set
```

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

Transakcja T1

```
mysql> begin;
Query OK, 0 rows affected

mysql> select * from produkt where
ID_produk=7 for update;
+-----+-----+
| ID_produk | nazwa_produk |
+-----+-----+
|          7 | xx           |
+-----+-----+
1 row in set
```

Transakcja T2

```
mysql> begin;
Query OK, 0 rows affected

mysql> insert into produkt
(nazwa_produk) values ('yy');
Query OK, 1 row affected
update produkt set nazwa_produk = 'zz'
where ID_produk = 1;
Query OK, 1 row affected
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update produkt set nazwa_produk
= 'ww' where ID_produk = 7;

--[tutaj system czeka na zakończenie
transakcji T1, która zablokowała rekord]
```

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

Transakcja T1

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from produkt for  
update;
```

ID_produkt	nazwa_produktu
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

7 rows in set

Transakcja T2

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> insert into produkt  
(nazwa_produktu) values ('zz');
```

--[tutaj system czeka na zakończenie transakcji T1, która zablokowała tabelę]

--[po zbyt długim oczekiwaniu:]

1205 - Lock wait timeout exceeded;
try restarting transaction

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

Transakcja T1

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from produkt for  
update;
```

ID_produk	nazwa_produk
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

7 rows in set

Transakcja T2

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from produkt for  
update;
```

--[tutaj system czeka na zakończenie
transakcji T1, która zablokowała
tabelę]

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

Transakcja T1

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from produkt for  
update;
```

ID_produk	nazwa_produk
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

7 rows in set

„Poza transakcją”

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> commit;  
Query OK, 0 rows affected
```

```
mysql> update produkt set  
nazwa_produk = 'zz' where  
ID_produk=7;
```

--[tutaj system czeka na zakończenie
transakcji T1, która zablokowała
tabelę]

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *for update* - symulacja użycia

Poza oficjalnym blokiem transakcji

```
mysql> select * from produkt for update;
```

ID_produk	nazwa_produk
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

7 rows in set

Poza oficjalnym blokiem transakcji

```
mysql> select * from produkt for update;
```

ID_produk	nazwa_produk
1	Monitor
2	Chleb
3	Komputer
4	Myszka
5	Książka
6	Stół
7	xx

7 rows in set

```
mysql> update produkt set nazwa_produk = 'zz' where ID_produk=7;
```

Query OK, 1 row affected

Rows matched: 1 Changed: 1 Warnings: 0

Transakcje bazodanowe - parametry ograniczające dostęp do danych

Klauzula *lock in share mode* - symulacja użycia

Transakcja T1

```
mysql> begin;
Query OK, 0 rows affected

mysql> select * from produkt lock in share
mode;
+-----+-----+
| ID_produkt | nazwa_produktu |
+-----+-----+
|          1 | Monitor        |
|          2 | Chleb          |
|          3 | Komputer       |
+-----+-----+
3 rows in set
```

Transakcja T2

```
mysql> begin;
Query OK, 0 rows affected

mysql> select * from produkt lock in
share mode;
+-----+-----+
| ID_produkt | nazwa_produktu |
+-----+-----+
|          1 | Monitor        |
|          2 | Chleb          |
|          3 | Komputer       |
+-----+-----+
3 rows in set

mysql> insert into produkt
(nazwa_produktu) values ('ww');
--[tutaj system czeka na zakończenie
transakcji T1, która zablokowała tabelę]

1205 - Lock wait timeout exceeded; try
restarting transaction
```

Transakcje bazodanowe - ACID

Transakcję niezależnie od systemu charakteryzują zasady ACID (<https://mariadb.com/kb/en/acid-concurrency-control-with-transactions>).

Proszę nie traktować ich jako niepotrzebnej definicji, tylko jako ważne zasady mające bezpośredni wpływ na funkcjonowanie programu komputerowego opierającego się o bazę danych.

Zasady ACID:

- Atomicity - Atomowość
- Consistency - Spójność
- Isolation - Izolacja
- Durability - Trwałość

Transakcje bazodanowe - ACID

Atomicity - Atomowość

Najbardziej charakterystyczna cecha transakcji - transakcja wykonuje się w całości albo wcale.

Czyli:

- Akceptujemy naniesione zmiany poleceniem *commit*
- Lub odrzucamy naniesione zmiany poleceniem *rollback* (+ awarie itp.)

Symboliczny przykład - przelew pieniędzy z konta na konto:

1. `begin;`
2. `update konta set kwota = kwota - 105.69 where ID_konta=45632;`
3. `update konta set kwota = kwota + 105.69 where ID_konta=44123;`
4. `commit;`

Jeżeli nastąpiłaby awaria pomiędzy punktami 2 a 3 i polecenia nie byłyby ujęte w transakcję, to przelewana kwota zostałaby utracona, jak również byłaby trudna lub niemożliwa do odzyskania ze względu na brak pewności o dostarczeniu kwoty na drugie konto (oczywiście przykład jest symboliczny).

Transakcje bazodanowe - ACID

Consistency - Spójność

Po wykonaniu transakcji baza danych nie traci spójności, dotyczy to:

- względów spójności encji - np. po zakończeniu transakcji w kolumnie PK nie będzie dwóch takich samych wartości, lub np. ograniczenie *check* nie zostanie zlekceważone, lub np. kolumna z ograniczeniem *not null* nie będzie zawierała w ramach rekordu wartości *null*

```
CREATE TABLE `klient` (  
  `ID_klient` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `imie` varchar(30) NOT NULL,  
  `nazwisko` varchar(80) NOT NULL,  
  `data_zapisu` datetime DEFAULT NULL,  
  `data_rezygnacji` datetime DEFAULT NULL,  
  PRIMARY KEY (`ID_klient`),  
  CONSTRAINT `CONSTRAINT_1` CHECK (`data_zapisu` < `data_rezygnacji`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

-oraz względów spójności referencyjnej PK-FK - nie zostanie przypisana wartość dla FK, która nie ma odpowiednika w PK

Transakcje bazodanowe - ACID

Durability - Trwałość

Po zatwierdzeniu transakcji poleceniem *commit*, wprowadzone lub zmodyfikowane dane stają się trwale dostępne (tj. "nie znikną").

Transakcje bazodanowe - ACID

Isolation - Izolacja

Stopień widoczności zmian zachodzących w innych transakcjach (rezultatów wykonywania poleceń SQL) z danej transakcji o tym poziomie.

W MySQL/MariaDB mamy cztery poziomy izolacji transakcji: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE.

W MySQL/MariaDB domyślnym poziomem jest REPEATABLE READ, co niesie za sobą pewne konsekwencje nieprzemyślanego braku jego zmiany na inny dla danego przypadku/problemu w wybranym module aplikacji.

Transakcje bazodanowe - ACID

Isolation - Izolacja

Poziom określający widoczność zmian zachodzących w innych transakcjach z poziomu danej transakcji o tym poziomie.

Poziomy izolacji transakcji:

1. READ UNCOMMITTED - zmiany wykonywane przez polecenia wewnątrz innych transakcji są na bieżąco widoczne poza nie w transakcji o tym poziomie, pomimo nie zatwierdzenia poleceniem commit.
2. READ COMMITTED - zmiany wykonane przez polecenia wewnątrz innych transakcji są widoczne w transakcji o tym poziomie dopiero po zatwierdzeniu poleceniem commit.
3. REPEATABLE READ - zmiany wykonane przez polecenia wewnątrz innych transakcji nie są widoczne w tej o tym poziomie, nawet pomimo zatwierdzenia ich poleceniem commit. Tryb domyślnie używany przez silnik InnoDB.
4. SERIALIZABLE - działa tak samo jak REPEATABLE READ i dodatkowo blokuje wiersze odczytywane przez pierwszą transakcję - tak jak klauzula lock in share mode.

Transakcje bazodanowe - ACID

Poziomy izolacji transakcji

Sprawdzanie aktualnego poziomu izolacji (starsze wersje i MariaDB):

```
mysql> select @@tx_isolation;
```

```
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
```

Sprawdzanie aktualnego poziomu izolacji poprzez "show variables"

```
mysql> show variables like 't%_isolation';
```

```
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| transaction_isolation  | REPEATABLE-READ |
| tx_isolation           | REPEATABLE-READ |
+-----+-----+
```

Transakcje bazodanowe - ACID

Poziomy izolacji transakcji

Ustawienie poziomu izolacji na czas trwania sesji:

```
mysql> set session transaction isolation level read uncommitted;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> set session transaction isolation level repeatable read;  
Query OK, 0 rows affected (0.00 sec)
```

Ustawienie poziomu izolacji na czas trwania jednej transakcji (tj. tej po zmianie poziomu tym poleceniem) w ramach sesji:

```
mysql> set transaction isolation level read uncommitted;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> set transaction isolation level repeatable read;  
Query OK, 0 rows affected (0.00 sec)
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Przykład - "uproszczona" rezerwacja miejsca na koncercie - zmiana poziomu izolacji na czas trwania sesji, rozpoczęcie transakcji, założenie blokady *for update*, zmiana zawartości tabeli, zakończenie transakcji

```
CREATE DEFINER=`artur`@`%` PROCEDURE `dodaj_rezerwacje`(IN  
`vID_koncerty` int, IN vimie varchar(60), IN vnazwisko  
varchar(100))  
BEGIN
```

```
set session transaction isolation level read committed;  
start transaction;
```

```
set @m1m=0;  
select maks_liczba_miejsc into @m1m from koncerty where  
ID_koncerty=vID_koncerty;
```

```
set @ile_rezerwacji=0;  
select count(*) into @ile_rezerwacji from rezerwacje where  
ID_koncerty=vID_koncerty for update;
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

```
if (@ile_rezerwacji<@mlm) then
insert into rezerwacje (ID_koncerty, imie, nazwisko) values
(vID_koncerty, vimie, vnazwisko);
commit;
else
rollback;
SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=30001, MESSAGE_TEXT='Brak
miejsc!';
end if;

END
```

* słowo BEGIN w tym przykładzie otwiera ciało procedury, tj. nie rozpoczyna transakcji - robi to: start transaction

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

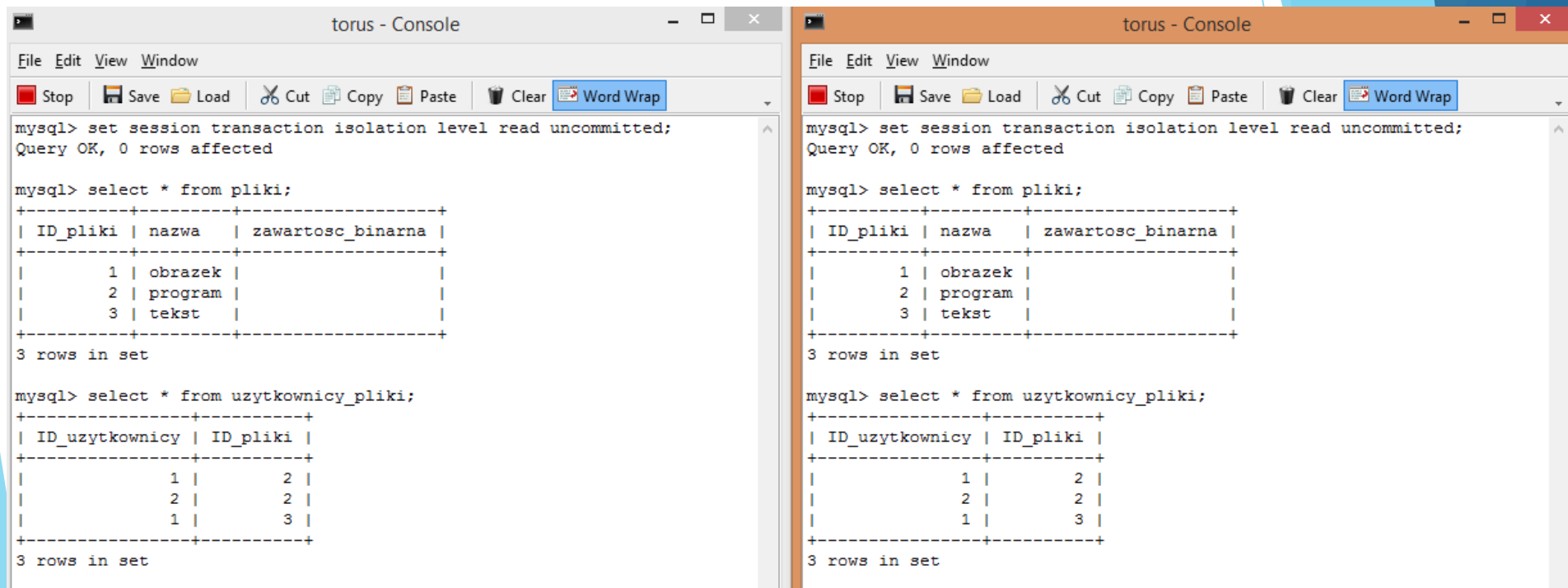
Wywołanie procedury rezerwacji miejsca

```
mysql> call dodaj_rezerwacje(3546, 'Jan', 'Kowalski');  
Query OK
```

```
mysql> call dodaj_rezerwacje(3546, 'Michał', 'Nowakowski');  
Query OK
```

```
mysql> call dodaj_rezerwacje(3546, 'Anna', 'Nowakowska');  
30001 - Brak miejsc!  
mysql>
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji



The image shows two side-by-side terminal windows, both titled "torus - Console". Each window contains the following MySQL commands and results:

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected

mysql> select * from pliki;
+-----+-----+-----+
| ID_pliki | nazwa | zawartosc_binarna |
+-----+-----+-----+
| 1 | obrazek | 
| 2 | program | 
| 3 | tekst | 
+-----+-----+-----+
3 rows in set

mysql> select * from uzytkownicy_pliki;
+-----+-----+
| ID_uzytkownicy | ID_pliki |
+-----+-----+
| 1 | 2 |
| 2 | 2 |
| 1 | 3 |
+-----+-----+
3 rows in set
```

Powyższy rysunek przedstawia prosty sposób na przetestowanie poziomów izolacji transakcji - wystarczy uruchomić obok siebie dwa okna konsolowe, ustawić odpowiedni poziom, uruchomić transakcje i wykonywać polecenia SQL. Według powyższego sposobu były testowane poziomy na kolejnych slajdach.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Read uncommitted

- Niezatwierdzony odczyt (*dirty reads*) - transakcje mają dostęp do modyfikacji wprowadzonych przez inne transakcje, nawet jeszcze nie zatwierdzone. Odczytywane są niezatwierdzone dane.
- Niepowtarzalny odczyt (*nonrepeatable (fuzzy) reads*) - każdy kolejny odczyt danych przez jedną transakcję może zwrócić inne wyniki. Dane zostały w międzyczasie zmodyfikowane przez równoległą, zatwierdzoną transakcję.
- Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Read uncommitted - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level
read uncommitted;
Query OK, 0 rows affected
```

```
mysql> select * from pliki;
```

ID_pliki	nazwa	zawartosc_binarna
1	obrazek	
2	program	
3	tekst	

3 rows in set

```
mysql> select * from uzytkownicy_pliki;
```

ID_uzytkownicy	ID_pliki
1	2
2	2
1	3

3 rows in set

```
mysql> set session transaction isolation level
read uncommitted;
Query OK, 0 rows affected
```

```
mysql> select * from pliki;
```

ID_pliki	nazwa	zawartosc_binarna
1	obrazek	
2	program	
3	tekst	

3 rows in set

```
mysql> select * from uzytkownicy_pliki;
```

ID_uzytkownicy	ID_pliki
1	2
2	2
1	3

3 rows in set

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

- Proszę mieć na uwadze to, że obie transakcje nie muszą być tego samego poziomu, wystarczy jedna - ale dla czytelności przykładów przyjąłem taki sposób.
- Odstępy pomiędzy poleceniami SQL w przykładach są istotne i oznaczają kolejność wykonywania poleceń pomiędzy transakcjami, które wykonują się współbieżnie

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Read uncommitted - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level
read uncommitted;
Query OK, 0 rows affected
```

```
mysql> begin;
Query OK, 0 rows affected
```

```
mysql> delete from uzytkownicy_pliki where
ID_uzytkownicy = 2 and ID_pliki = 2;
Query OK, 1 row affected
```

```
mysql> select * from uzytkownicy_pliki;
+-----+-----+
| ID_uzytkownicy | ID_pliki |
+-----+-----+
|                1 |        2 |
|                1 |        3 |
+-----+-----+
2 rows in set
```

```
mysql> rollback;
Query OK, 0 rows affected
```

```
mysql>
```

```
mysql> set session transaction isolation level
read uncommitted;
Query OK, 0 rows affected
```

```
mysql> begin;
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy_pliki;
+-----+-----+
| ID_uzytkownicy | ID_pliki |
+-----+-----+
|                1 |        2 |
|                1 |        3 |
+-----+-----+
2 rows in set
```

```
mysql> insert into uzytkownicy_pliki values
(2,2);
--tutaj SZBD czeka!
1062 - Duplicate entry '2-2' for key 'PRIMARY'
mysql>
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Read uncommitted - przykład fragmentu aplikacji

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

Poziom ten sprawia, że transakcja po prawej stronie posiada uaktualnione dane, po usunięciu wierszy przez transakcję po lewej stronie, pomimo niezatwierdzenia lewej transakcji.

Jednocześnie widzimy zabezpieczenie w postaci czekania w transakcji prawej na decyzję zaakceptowania lub cofnięcia transakcji lewej. W transakcji prawej chcemy dodać kombinację wartości klucza głównego, która to kombinacja została usunięta w transakcji lewej. Jeżeli po wykonaniu inserta zaakceptujemy transakcję lewą to dane zostaną umieszczone bez błędu, w przeciwnym wypadku (czyli cofnięciu transakcji) dostaniemy komunikat o zdublowanych danych klucza głównego i insert w transakcji prawej zwróci błąd.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Read committed

~~-Niezatwierdzony odczyt (*dirty reads*) - transakcje mają dostęp do modyfikacji wprowadzonych przez inne transakcje, nawet jeszcze nie zatwierdzone. Odczytywane są niezatwierdzone dane.~~

-Niepowtarzalny odczyt (*nonrepeatable (fuzzy) reads*) - każdy kolejny odczyt danych przez jedną transakcję może zwrócić inne wyniki. Dane zostały w międzyczasie zmodyfikowane przez równoległą, zatwierdzoną transakcję.

-Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.

Transakcje bazodanowe - ACID - poziomy...

Read committed - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected
```

```
mysql> begin;
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> insert into uzytkownicy values (null, 'alex', '');
Query OK, 1 row affected
```

```
mysql> commit;
Query OK, 0 rows affected
```

```
mysql>
```

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected
```

```
mysql> begin;
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
5	alex	

5 rows in set

```
mysql> commit;
Query OK, 0 rows affected
```

Transakcje bazodanowe - ACID - poziomy...

Read committed - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level read committed;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c

```
mysql> set session transaction isolation level read committed;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c

Poziom ten różni się od poprzedniego tym, że dane zmodyfikowane/dodane w transakcji lewej będą widoczne w transakcji prawej dopiero po zakończeniu transakcji lewej poleceniem **commit**.

Bez zatwierdzenia zmian w transakcji lewej - transakcja po prawej stronie nie widzi naniesionych zmian.

Transakcje bazodanowe - ACID - poziomy...

Read committed - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level read committed;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> insert into uzytkownicy values (null, 'alex','');  
Query OK, 1 row affected
```

```
mysql> rollback;  
Query OK, 0 rows affected
```

```
mysql> set session transaction isolation level read committed;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> insert into uzytkownicy values (null, 'alex','');
```

--tutaj SZBD czeka!!!!

```
Query OK, 1 row affected
```

```
mysql> commit;  
Query OK, 0 rows affected
```

... i analogiczny przykład z czekaniem na decyzję dotyczącą zatwierdzenia lub cofnięcia zmian.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read - domyślny poziom w InnoDB

- ~~-Niezatwierdzony odczyt (*dirty reads*) - transakcje mają dostęp do modyfikacji wprowadzonych przez inne transakcje, nawet jeszcze nie zatwierdzone. Odczytywane są niezatwierdzone dane.~~
- ~~-Niepowtarzalny odczyt (*nonrepeatable (fuzzy) reads*) - każdy kolejny odczyt danych przez jedną transakcję może zwrócić inne wyniki. Dane zostały w międzyczasie zmodyfikowane przez równoległą, zatwierdzoną transakcję.~~
- Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read

- ~~-Niezatwierdzony odczyt (*dirty reads*) - transakcje mają dostęp do modyfikacji wprowadzonych przez inne transakcje, nawet jeszcze nie zatwierdzone. Odczytywane są niezatwierdzone dane.~~
- ~~-Niepowtarzalny odczyt (*nonrepeatable (fuzzy) reads*) - każdy kolejny odczyt danych przez jedną transakcję może zwrócić inne wyniki. Dane zostały w międzyczasie zmodyfikowane przez równoległą, zatwierdzoną transakcję.~~
- ~~-Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.~~

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read

-Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.

W MySQL/MariaDB ta cecha tego poziomu nie występuje. Jest to związane z pojęciem migawki (snapshot) i tym, że pierwszy odczyt danych z danej tabeli jest w ramach migawki zapamiętany do końca trwania transakcji. Innymi słowy, jeżeli dana transakcja np. wyczyści tabelę T i zakończy działanie commit-em, to równoległa transakcja o tym poziomie nadal będzie wyświetlała w ramach polecenia select na tabeli T dane, które już dawno nie istnieją. Jednakże operacje na usuniętych danych nie będą odnosiły żadnego efektu (przykład na kolejnych slajdach).

Table 2.4. Problems Permitted by Isolation Levels

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantom Rows
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	No
SERIALIZABLE	No	No	No

Transakcje bazodanowe - ACID - poziomy...

Repeatable read - przykład fragmentu aplikacji - cz. 1/2

```
mysql> set session transaction isolation level repeatable read;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
8	alex	

5 rows in set

```
mysql> delete from uzytkownicy where ID_uzytkownicy = 8;  
Query OK, 1 row affected
```

```
mysql> insert into uzytkownicy values (null, 'monika', '');  
Query OK, 1 row affected
```

```
mysql> set session transaction isolation level repeatable read;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
8	alex	

5 rows in set

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
8	alex	

5 rows in set

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
8	alex	

5 rows in set

Transakcje bazodanowe - ACID - poziomy...

Repeatable read - przykład fragmentu aplikacji - cz. 2/2

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
9	monika	

5 rows in set

```
mysql> commit;
```

Query OK, 0 rows affected

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
9	monika	

5 rows in set

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
8	alex	

5 rows in set

```
mysql> insert into uzytkownicy values (null, 'monika', '');
```

-- tutaj SZBD czeka!

1062 - Duplicate entry 'monika' for key
'uniq_uzytkownicy_login'

```
mysql> commit;
```

Query OK, 0 rows affected

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	michal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
9	monika	

5 rows in set

Ten poziom izolacji transakcji sprawia, że dane zmodyfikowane i zatwierdzone w transakcji lewej nie będą widoczne w transakcji prawej do momentu jej zakończenia - czyli na poziomie modułu programu (tj. w ramach kolejnej transakcji)

Transakcje bazodanowe - ACID - poziomy...

Repeatable read - przykład fragmentu aplikacji

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
9	monika	

5 rows in set

```
mysql> insert into uzytkownicy values (null, 'franek', '');  
Query OK, 1 row affected
```

```
mysql> delete from uzytkownicy where login='monika';  
Query OK, 1 row affected
```

```
mysql> commit;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
14	franek	

5 rows in set

Przykład powyżej pokazuje, że w tym poziomie izolacji dane w transakcji prawej są aktualne w ramach pierwszego selekt-a (pierwszy dostęp do danych w tabeli). Po kolejnych modyfikacjach w transakcji lewej, dane w transakcji prawej pozostaną niezmienione - snapshot.

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read - kolejny przykład z migawką (snapshot) - cz. 1/3

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
32	Jan	Muszka
33	Janina	Nowakowska
34	Stefania	Nowakowski

```
4 rows in set
```

```
mysql> delete from student where  
ID_student = 34;  
Query OK, 1 row affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
32	Jan	Muszka
33	Janina	Nowakowska
34	Stefania	Nowakowski

```
4 rows in set
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read - kolejny przykład z migawką (snapshot) - cz. 2/3

```
mysql> delete from student where  
ID_student = 34;  
Query OK, 1 row affected
```

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
32	Jan	Muszka
33	Janina	Nowakowska

```
3 rows in set
```

```
mysql> commit;
```

```
mysql> select * from student;
```

ID_student	imie	nazwisko
31	Jan	Kowalski
32	Jan	Muszka
33	Janina	Nowakowska
34	Stefania	Nowakowski

```
4 rows in set
```

```
mysql> update student set nazwisko =  
'Nowak' where ID_student = 34;
```

```
Query OK, 0 rows affected
```

```
Rows matched: 0 Changed: 0 Warnings: 0
```

Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Repeatable read - kolejny przykład z migawką (snapshot) - cz. 3/3

```
mysql> commit;
```

Nic nie zmieniono, ponieważ
rekord już fizycznie nie istnieje →

Nadal mamy "stare" dane
w ramach migawki, czyli
update nie operował na
danych w migawce →

```
mysql> update student set nazwisko =  
'Nowak' where ID_student = 34;  
Query OK, 0 rows affected  
Rows matched: 0 Changed: 0 Warnings: 0
```

```
mysql> select * from student;  
+-----+-----+-----+  
| ID_student | imie   | nazwisko |  
+-----+-----+-----+  
|          31 | Jan    | Kowalski |  
|          32 | Jan    | Muszka   |  
|          33 | Janina | Nowakowska |  
|          34 | Stefania | Nowakowski |  
+-----+-----+-----+  
4 rows in set
```

```
mysql>
```


Transakcje bazodanowe - ACID - poziomy izolacji transakcji

Serializable

- ~~-Niezatwierdzony odczyt (*dirty reads*) - transakcje mają dostęp do modyfikacji wprowadzonych przez inne transakcje, nawet jeszcze nie zatwierdzone. Odczytywane są niezatwierdzone dane.~~
- ~~-Niepowtarzalny odczyt (*nonrepeatable (fuzzy) reads*) - każdy kolejny odczyt danych przez jedną transakcję może zwrócić inne wyniki. Dane zostały w międzyczasie zmodyfikowane przez równoległą, zatwierdzoną transakcję.~~
- ~~-Fantomy (*phantom reads*) - każdy kolejny odczyt danych przez jedną transakcję, może zwrócić inne wyniki. Dane zostały w międzyczasie dodane przez równoległą, zatwierdzoną transakcję.~~

Transakcje bazodanowe - ACID - poziomy...

Serializable - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level serializable;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> commit;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

-- tutaj SZBD czeka!

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64
21	monika	
22	franek	

```
mysql> set session transaction isolation level serializable;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micchal	1b95538f3035f4cba2189716fd96173c
4	edyta	207023ccb44feb4d7dadca005ce29a64

4 rows in set

```
mysql> insert into uzytkownicy values (null, 'monika', '');
```

--tutaj SZBD czeka!

```
Query OK, 1 row affected
```

```
mysql> insert into uzytkownicy values (null, 'franek', '');
```

```
Query OK, 1 row affected
```

```
mysql> commit;  
Query OK, 0 rows affected
```

Transakcje bazodanowe - ACID - poziomy...

Serializable - przykład fragmentu aplikacji

```
mysql> set session transaction isolation level serializable;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micHAL	1b95538f3035f4cba2189716fd96173c

```
mysql> set session transaction isolation level serializable;  
Query OK, 0 rows affected
```

```
mysql> begin;  
Query OK, 0 rows affected
```

```
mysql> select * from uzytkownicy;
```

ID_uzytkownicy	login	haslo
1	ala	1b95538f3035f4cba2189716fd96173c
2	jan	1b95538f3035f4cba2189716fd96173c
3	micHAL	1b95538f3035f4cba2189716fd96173c

W tym poziomie izolacji dane w tabelach są blokowane blokadą *lock in share mode*, w tym przypadku najpierw lewą, a później prawą.

Środowiska programistyczne a transakcje bazodanowe

Wiele środowisk programistycznych posiada zaimplementowane metody w ramach obiektów (czy też wywołań proceduralnych) umożliwiających połączenie z bazą danych i obsługę transakcji.

Przykład w języku PHP:

```
mysqli_autocommit($link, FALSE);  
mysqli_commit($link);  
mysqli_rollback($link);
```

Procedury powyżej analogiczne do poznanych już poleceń poniżej:

```
mysqli_query($link, "SET autocommit=0;");  
mysqli_query($link, "commit;");  
mysqli_query($link, "rollback;");
```

Środowiska programistyczne a transakcje bazodanowe

Wiele środowisk programistycznych posiada zaimplementowane metody w ramach obiektów (czy też wywołań proceduralnych) umożliwiające połączenie z bazą danych i obsługę transakcji.

Przykład w języku C# .NET:

```
MySqlConnection myTrans;  
myTrans = myConnection.BeginTransaction();
```

```
try  
{  
    //...  
    myTrans.Commit();  
}  
catch(Exception e)  
{  
    myTrans.Rollback();  
}
```

Środowiska programistyczne a transakcje bazodanowe

Proszę mieć na uwadze to, że omówione cechy transakcji w tym blokady tabel/rekordów powodujące np. chwilowe 'zawieszenie' odpowiedzi SZBD na zapytania SQL wpływają bezpośrednio na zachowanie (w tym wydajność) aplikacji komputerowych łączących się z bazą danych.

Środowiska pro bazodanowe

Proszę mieć na uwadze blokady tabel/rekordów odpowiedzi SZBD na

```
<?php
$link = mysqli_connect("localhost", "artur", "", "artur") or
die (mysqli_connect_error());

mysqli_query($link, "begin;");
mysqli_query($link, "select * from student for update;");
mysqli_query($link, "commit;");
?>
```

artur student.mck - Cons... x artur student.mck - Cons... x artur student.mck - Cons... x artur student.mck - Cons... x

File Edit View Window

Stop Save Load Cut Copy Paste Clear Word Wrap

mysql> begin;
Query OK, 0 rows affected

mysql> select * from student for update;
+-----+
| ID_student | imie | nazwisko |
+-----+
| 31 | Jan | Kowalski |
| 33 | Janina | Nowakowski |
+-----+
2 rows in set

mysql> select sleep(30); commit;
+-----+
| sleep(30) |
+-----+
| 0 |
+-----+
1 row in set

Query OK, 0 rows affected

mysql>

Podstawy baz danych / Artur Niewiarowski

Wszelkie blokady warto dobrze przemyśleć - tj. czy trzeba, które rekordy i jaki rodzaj blokady, itp. itd.

student.mck.pk.edu.pl/~aniewiarov x

← → ↻ 🏠 🔒 student.mck.p 67% 🔍 Szukaj ⬇️ 📖 ⏏️ ☰

⚙️ Często odwiedzane 🌐 Pierwsze kroki

🔍 Inspektor 📄 Konsola 🐛 Debugger {} Edytor stylów 🔄 Wydajność ⬆️ ⬆️ Sieć ⏏️

🗑️ Filtrowanie adresów || 🔍 🔒 Trwałe dzienniki 🚫 Wyłącz pamięć podręczną Bez ograniczeń 📏 HAR 📏

Wszystkie HTML CSS JS XHR Czcionki Obrazy Media WebSocket Inne

	Dc	Plik	Prz	Prz	Nagłówki	Ciasteczka	Parametry	Odpowiedź	Pomiary czasu
2	🔍	/~aniew...	doc	h 239	Łączenie:	0 ms			
					Konfigurowanie TLS:	0 ms			
					Wysyłanie:	0 ms			
					Oczekiwanie:				28,72 s
					Odbieranie:				0 ms

🕒 2 zapytania Przesłano: 20 ?

Transakcje bazodanowe

Przykładowe fragmenty funkcjonalności aplikacji do rozważenia

-rejestracja użytkownika - umieszczenie danych w kilku tabelach, np. takich jak: użytkownicy, pliki użytkowników, kilku użytkowników w jednym czasie chce zarejestrować ten sam login

Transakcje bazodanowe

Przykładowe fragmenty funkcjonalności aplikacji do rozważenia

-koszyk zakupów - nie można zamówić produktu, jeżeli nie ma go na stanie magazynu. W trakcie wyboru produktu był jeszcze na stanie i jednocześnie w koszyku innego użytkownika jako ostatni egzemplarz.

Transakcje bazodanowe

Przykładowe fragmenty funkcjonalności aplikacji do rozważenia

- rezerwacja przedmiotów wybieralnych przez studentów z poziomu systemu. Limity na przedmiot o danej godzinie.

Transakcje bazodanowe

Przykładowe fragmenty funkcjonalności aplikacji do rozważenia

-rezerwacja biletów lotniczych, wybór biletu rezerwuje go na czas trwania sesji.
Oprogramować moment wyboru biletu z uwzględnieniem wielu użytkowników dokonujących tej samej rezerwacji w danej chwili.

Transakcje bazodanowe

Przykładowe fragmenty funkcjonalności aplikacji do rozważenia

-przelew bankowy - wykonywany przez dwóch współwłaścicieli konta w jednakowym czasie, z różnych komputerów, z uwzględnieniem debetu.