Bazy Danych

w wykładzie wykorzystano:

1. dr inż. Piotr Macioł, wykłady
2. dr Maciej Bobrowski, *Wykład z programowania baz danych*, http://gryf.mif.pg.gda.pl/~mate/wyklady/bazy/

Transakcje

Krzysztof Regulski
WIMiIP, KISiM,
regulski@agh.edu.pl
B5, pok. 408

Transakcje i blokady

- co to są transakcje?
- reguły ACID
- transakcje z pojedynczym użytkownikiem
- ograniczenia transakcji
- transakcje z wieloma użytkownikami
- poziomy izolacji ANSI
- tryby związany i niezwiązany
- blokowanie
- zakleszczanie i jawne blokady

Co to są transakcje?

- *Przykład zastosowania:* musimy wykonać kilka komend SQL zmieniających dane w tablicach. Jak to zrobić aby komendy wykonał się poprawnie do końca albo wcale?
- *Transakcja jest* logiczną jednostką działań, której nie można podzielić.
- *Logiczna jednostka działań* to zbiór logicznych zmian w bazie danych, które należy wykonać wszystkie albo nie wykonywać żadnej.
- *Moduł zarządzania transakcjami* (*transaction management*) zapewnia pozostawanie bazy w spójnym (poprawnym) stanie pomimo wystąpienia błędu gdziekolwiek w systemie (np. przerwa w zasilaniu) lub w samej transakcji.

Co to są transakcje?

- Zmiany są kontrolowane przez trzy kluczowe frazy:
  - Fraza **`BEGIN WORK`** rozpoczyna transakcję;
  - **`COMMIT WORK`** informuje, że wszystkie elementy transakcji są kompletne i powinny zostać zatwierdzone na stałe oraz stać się dostępne dla wszystkich transakcji;
  - **`ROLLBACK WORK`** mówi, że należy porzucić transakcję, a wszystkie zmiany danych dokonane przez transakcję SQL mają być anulowane. Baza danych, z punktu widzenia użytkowników, powinna się znajdować w takim stanie, jakby nie były wykonywane żadne zmiany.

Co to są transakcje?

- Standard ANSI/SQL nie definiuje frazy SQL BEGIN WORK, transakcje w tym standardzie powinny wykonywać się automatycznie. Fraza ta jednak jest wymagana prawie we wszystkich implementacjach baz danych.
- Słowo WORK we frazie COMMIT WORK, ROLLBACK WORK można pominąć.
- SQL obejmuje polecenia rozpoczęcia i zakończenia transakcji, a także blokowania danych dla współbieżnych operacji (**`START TRANSACTION, COMMIT, ROLLBACK`**)

Co to są transakcje?

- Dowolna transakcja w bazie danych powinna być odizolowana od wszystkich pozostałych transakcji, które są wykonywane w tym samym czasie.
- W idealnej sytuacji każda transakcja zachowuje się tak jakby posiadała wyłączny dostęp do bazy danych.
- Niestety realia związane z osiągnięciem dobrej wydajności wymagają kompromisów o których powiemy później.

KLIENT 1	KLIENT 2	Wolne miejsce
Czy są wolne miejsca?		1
	Czy są wolne miejsca?	1
Oferuje miejsce		1
	Oferuje miejsce	1
Pytanie o kartę kredytową lub konto		1
	Pytanie o kartę kredytową lub konto	1
Podaje numer karty	Podaje numer konta	1
Autoryzacja		1
	Autoryzacja	1
Przypisanie miejsca		1
	Przypisanie miejsca	1
Obciążenie karty		1
	Obciążenie karty	1
Zmniejszenie liczby wolnych miejsc		0
	Zmniejszenie liczby wolnych miejsc	-1

Reguły ACID

- **ACID** to mnemonik służący do opisania transakcji, jaka powinna być:
  - Niepodzielna (**A**tomic)
  - Spójna (**C**onsistent)
  - Odizolowana (**I**solated)
  - Trwała (**D**urable)

Reguły ACID

- **Niepodzielność (*Atomicity*)** – transakcja składa się z sekwencji instrukcji. Transakcja musi być wykonana w całości, jeśli nastąpi awaria, częściowe zmiany powinny zostać wycofane.

  Transakcja, mimo że jest zbiorem odwołań, musi być wykonywana jako pojedyncza jednostka. Transakcja musi wykonywać się w jednym momencie i nie może dzielić się na podzbiory.

- **Spójność (*Consistency*)** – baza danych musi być zgodna z rzeczywistością, co określamy jako spójność. Transakcje modyfikujące należy traktować jako przejście bazy danych z jednego spójnego stanu w drugi, także spójny. Reguły spójności określane są przez więzy integralności.

  Jeżeli w komendach SQL są ograniczenia to powinny być sprawdzone na końcu transakcji – słowo kluczowe DEFERRABLE

  ```
  DEFERRABLE INITIALLY DEFERRED or DEFERRABLE INITIALLY IMMEDIATE
  ```

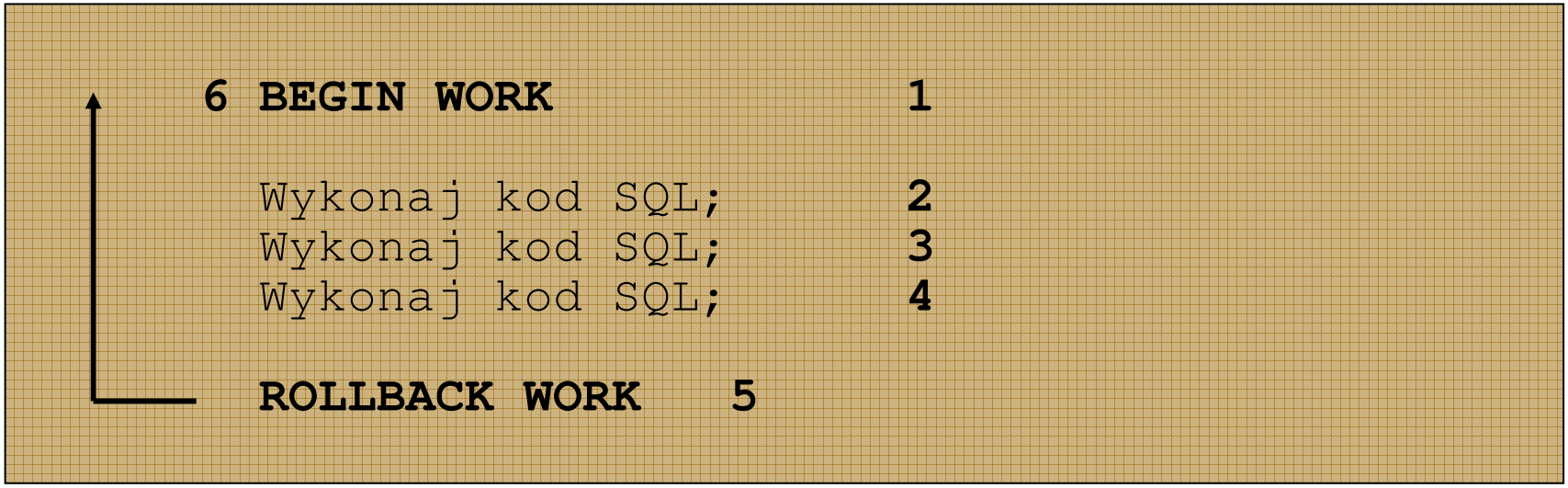
Reguły ACID

- **Izolacja (*Isolation*)** – równoczesne wykonywanie transakcji na tych samych danych przez różnych użytkowników może prowadzić do zachwiania spójności, dlatego poszczególne transakcje muszą zostać odizolowane od siebie jest to zadanie modułu zarządzania współbieżnością (*concurrency control manager)*. Izolacja sprawia niestety kłopoty z wydajnością bazy danych
- **Trwałość (*Durability*)** – po pomyślnym wykonaniu transakcji zmiany powinny zostać utrwalone tak, aby zostały zachowane nawet w przypadku awarii sprzętu lub oprogramowania. Wykonywane jest to za pomocą dziennika transakcji.

Dziennik transakcji

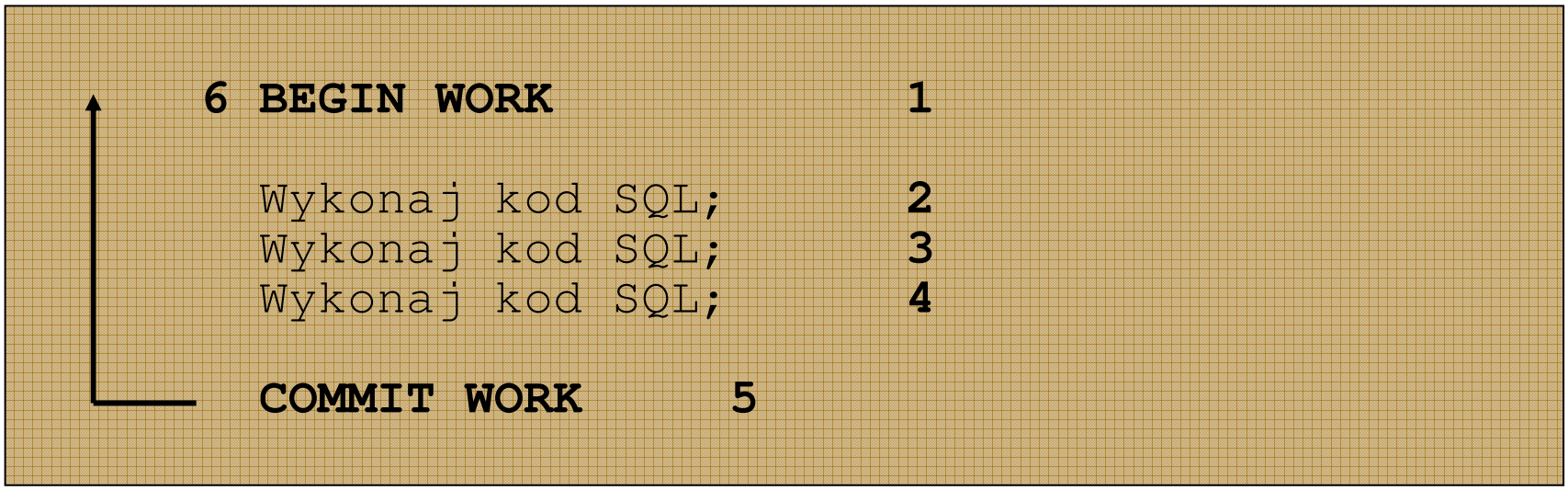
- Dziennik transakcji (log) działa następująco: w czasie wykonywania transakcji zmiany są zapisywane nie tylko w bazie danych ale także do pliku dziennika.
- Po zakończeniu transakcji zapisywany jest znacznik, który informuje, że transakcja została zakończona i dane z dziennika transakcji zostały zatwierdzone do zapisu na stałe do bazy danych.
- Gwarantuje to bezpieczeństwo danych nawet w przypadku awarii bazy. Jeżeli serwer bazy danych przestanie działać w czasie transakcji to po ponownym jego uruchomieniu automatycznie jest sprawdzane, czy zakończone transakcje zostały właściwie odzwierciedlone w bazie danych (poprzez przeglądanie transakcji w dzienniku transakcji a nie w bazie danych).
- Jeśli w bazie danych nie ma transakcji, które trwały w czasie awarii serwera, transakcja jest utrwalana bez udziału użytkownika

Transakcje z pojedynczym użytkownikiem



Gdy stwierdzamy, że nie chcemy zakończenia wykonania transakcji to wykonujemy **ROLLBACK WORK**

Transakcje z pojedynczym użytkownikiem



Gdy stwierdzamy, że chcemy zakończenia wykonania transakcji to wykonujemy **COMMIT WORK**

Transakcje z pojedynczym użytkownikiem

- Ograniczenia transakcji
  - Nie wolno zagnieżdżać transakcji
  - Zaleca się aby transakcje były niewielkie. Należy wykonać wiele działań aby upewnić się, że transakcje wielu użytkowników są rozdzielone. Te elementy, które biorą udział w transakcjach, muszą być zablokowane
  - Transakcja, której wykonanie trwa długo i obejmuje wiele tabel, uniemożliwia innym użytkownikom korzystanie z danych, do czasu zakończenia lub anulowania transakcji

Transakcje z pojedynczym użytkownikiem

- Ograniczenia transakcji c.d.
  - » Należy unikać transakcji w czasie dialogu z użytkownikiem. Należy najpierw pobrać potrzebne dane a dopiero potem uruchamiać transakcję.
  - » Instrukcja **`COMMIT WORK`** trwa zazwyczaj szybko.
  - » Dużo dłużej trwa **`ROLLBACK WORK`**

Transakcje z wieloma użytkownikami

- Co oznacza że transakcje są **izolowane**?
- Samo osiągnięcie izolacji nie jest trudne. Zezwolenie na pojedyncze połączenie z bazą danych, z zaledwie jedną transakcją wykonywaną w danym czasie zapewnia izolację pomiędzy różnymi transakcjami.
- Osiągnięcie bardziej praktycznej izolacji, bez znacznego obniżenia wydajności powoduje uniemożliwienie uzyskania dostępu do bazy danych przez wielu użytkowników.
- Osiągnięcie prawdziwej izolacji bez obniżenia wydajności bazy jest bardzo trudne dlatego standard SQL ANSI definiuje różne poziomy izolacji, które baza danych może implementować.

Transakcje z wieloma użytkownikami

- Standard SQL ANSI definiuje poziomy izolacji w odniesieniu do niepożądanych zjawisk, które mogą się zdarzyć w czasie interakcji transakcji dla wielodostępnych baz danych;
- Niepożądane zjawiska:
  - *Brudny odczyt* – ma miejsce wtedy, gdy pewne instrukcje SQL wewnątrz transakcji odczytują dane, które zostały zmienione przez inną transakcję ale nie potwierdzone jeszcze poleceniem COMMIT
  - *Odczyty nie dające się powtórzyć* – Podobne do brudnego odczytu. Zjawisko zachodzi wtedy, gdy transakcja odczytuje zbiór danych, następnie czyta dane ponownie i okazuje się, że dane nie są identyczne, odczytuje bowiem zmiany zatwierdzone przez COMMIT

Transakcje z wieloma użytkownikami

- Niepożądane zjawiska c.d.:
  - *Odczyty widmo* – problem podobny do odczytów nie dających się powtórzyć, ale mający miejsce wtedy gdy w tabeli znajdzie się nowy wiersz. Gdy inna transakcja aktualizuje tabelę nawy wiersz powinien być dodany a tak się nie staje.
  - *Utracone aktualizacje* - Zachodzi wtedy gdy do bazy zapisywane są dwie różne zmiany i druga aktualizacja powoduje, że pierwsza zostaje utracona.

Poziomy izolacji ANSI

Definicja poziomu izolacji ANSI /ISO	Brudny odczyt	Odczyt nie dający się powtórzyć	Widmo
READ UNCOMMITTED odczyt nie zatwierdzony	***możliwy***	***możliwy***	***możliwy***
READ COMMITTED odczyt zatwierdzony	*niemożliwy*	***możliwy***	***możliwy***
REPETABLE READ odczyt dający się powtórzyć	*niemożliwy*	*niemożliwy*	***możliwy***
SERIALIZABLE odczyt uszeregowany	*niemożliwy*	*niemożliwy*	*niemożliwy*

Przykład

– Zakładamy, że mamy następującą tablicę tab z danymi:

```
mysql> CREATE TABLE tab (f INT) TYPE = InnoDB;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tab values (1),(2),(3),(4),(55);
Query OK, 5 rows affected (0.00 sec)
```

– Na początek sprawdźmy jaki poziom izolacji transakcji obowiązuje w danej chwili (jeśli tego nie zmieniliśmy my lub administrator to domyślnym poziomem izolacji transakcji w MySQL jest REPEATABLE READ).

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

Repeatable Read

– Zobaczmy, czy polecenie INSERT wykonane w obrębie jednej transakcji i potwierdzone następnie poleceniem COMMIT jest widoczne z poziomu drugiej transakcji.

Sesja 1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
+------+
5 rows in set (0.00 sec)
```

Sesja 2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tab VALUES(6);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

– nie ma znaczenia dla transakcji w sesji 2, że polecenie SELECT zostało wykonane po poleceniu COMMIT. W obrębie transakcji nowy rekord jest natychmiast ''widzialny'' (równie dobrze moglibyśmy wykonać SELECT przed wykonaniem polecenia COMMIT).

Sesja 1

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
+------+
5 rows in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

- To właśnie jest idea blokowania typu **Repeatable Read**.
- Wykonane polecenie SELECT zwraca wynik, który charakteryzuje się spójnością, a nowe rekordy dodane do tablicy z poziomu innej transakcji nie są od razu widoczne.
- Aby były widoczne należy bezwzględnie zakończyć transakcję.

Uncommitted Read

- Zobaczmy jak zachowują się transakcje w trybie **Uncommitted Read**. Musimy w tym celu zmienić poziom izolacji transakcji z domyślnego na Uncommitted Read właśnie.
- Aby to uczynić musimy mieć przywilej SUPER.

```
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL
       READ UNCOMMITTED;
Query OK, 0 rows affected (0.00 sec)
```

Sesja 1

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

Sesja 2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tab VALUES (7),(8);
Query OK, 1 row affected (0.06 sec)
```

Sesja 1

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.00 sec)
```

– To właśnie jest tzw. *dirty read* - nowe rekordy nie zostały jeszcze nawet potwierdzone w drugiej transakcji a już są widoczne z poziomu pierwszej transakcji.

Sesja 2

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

- **Sesja 1**

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

- Taki poziom izolacji jest niebezpieczny i właściwie łamie zasady ACID. Używa się takiego trybu pracy transakcji w przypadku, kiedy nie interesuje nas spójność danych, a jedynie dostęp do najświeższych danych z poziomu dowolnej transakcji.

Committed Read

– Ponownie trzeba zmienić poziom izolacji i uruchomić dwie nowe (!) sesje.

```
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL
         READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)
```

Sesja 1

```
mysql>BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

Sesja 2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES (7),(8);
Query OK, 1 row affected (0.05 sec)
```

Sesja 1

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
+------+
6 rows in set (0.00 sec)
```

Sesja 2

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Sesja 1

```
mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Committed Read c.d.

- Istotną różnicą jest to, że niepotwierdzone poleceniem COMMIT polecenie INSERT nie wpłynęło na aktualny stan bazy danych widziany z poziomu drugiej transakcji.
- Dopiero po potwierdzeniu transakcji (po jej zakończeniu) widoczne są zmiany w tablicach.
- Jest też różnica pomiędzy tym poziomem izolacji (READ COMMITTED) a omówionym pierwszym domyślnym poziomem izolacji (REPEATABLE READ).
- W trybie READ COMMITTED zmiany w tablicach widoczne są już wówczas, gdy transakcja, w której zostały wykonane potwierdzi je poleceniem COMMIT, nawet wówczas gdy w danej transakcji nie wykonano jeszcze COMMIT.
- W trybie REPEATABLE READ, zmiany są widoczne dopiero wówczas, gdy w obu transakcjach wykonane zostaną polecenia potwierdzenia (COMMIT).

Serializable

```
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL
SERIALIZABLE;
Query OK, 0 rows affected (0.00 sec)
```

– Tryb SERIALIZABLE posuwa się o krok dalej niż tryb REPEATABLE READ. W trybie SERIALIZABLE wszystkie zwyczajne polecenia SELECT są traktowane jakby były wykonywane z klauzulą LOCK IN SHARE MODE.

Sesja 1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM tab
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.00 sec)
```

Sesja 2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.06 sec)

mysql> UPDATE tab SET f=88 WHERE f=8;
```

– Z powodu wykonania polecenia SELECT w sesji 1 polecenie UPDATE wykonywane w sesji 2 czeka aż po poleceniu SELECT (w sesji 1) nie zostanie wykonane polecenie COMMIT (tak, jak przy zwykłym LOCK IN SHARE MODE). Dopiero, kiedy w sesji 1 wykonane zostanie polecenie COMMIT kończące transakcję, wówczas zostanie wykonanie polecenie UPDATE w sesji 2.

Sesja 1

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Sesja 2

```
Query OK, 1 rows affected (4.23 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
|    6 |
|    7 |
|   88 |
+------+
8 rows in set (0.00 sec)
```

Poziomy izolacji ANSI

- Tablice InnoDB wspierają wszystkie cztery poziomy izolacji transakcji. Przy przenoszeniu kodów SQL na inny system baz danych, należy mieć świadomość, że nie wszystkie poziomy izolacji są wspierane przez inne systemy baz danych, a co więcej, w niektórych z nich domyślnym poziomem izolacji jest zupełnie inny poziom niż w MySQL.
- **MySQL** - REPEATABLE READ jest domyślnym poziomem izolacji transakcji i nie powinniśmy tego raczej zmieniać, może się zdarzyć bowiem że będziemy kiedyś oczekiwać długie godziny, zanim nasze polecenia wykonywane w bazie danych odniosą trwały skutek.
- **SQL SERVER** - domyślnie READ COMMITTED, poza tym, nie ma żadnych innych poziomów izolacji.
- **Oracle** - domyślnie READ COMMITTED, poza tym można wybrać też SERIALIZABLE i niestandardowy READ ONLY.
- **DB2** - domyślnie REPEATABLE READ, poza tym można wybrać też UNCOMMITTED READ oraz inne niestandardowe poziomy izolacji.
- **PostgreSQL** - domyślnie REPEATABLE READ, poza tym można też wybrać SERIALIZABLE.

Tryb chained i unchained

- W MySQL można wykonywać zmiany w bazie danych bez polecenia BEGIN WORK czy START TRANSACTION.
- Dzieje się tak, ponieważ MySQL działa domyślnie w trybie autozatwierdzania (AUTOCOMMIT), zwanym czasem trybem chained, lub trybem niejawnych transakcji.
- Każda instrukcja zmieniająca dane działa w tym trybie tak, jakby była kompletną transakcją. Dzięki temu możliwe jest wykonywanie pojedynczych instrukcji nie zakończonych instrukcjami COMMIT ani ROLLBACK.
- polecenie BEGIN WORK powoduje, że system przełączy się automatycznie do trybu, w którym kolejne polecenia będą częścią transakcji zakończonej przez COMMIT lub ROLLBACK.
- W standardzie SQL przewiduje się, że wszystkie instrukcje są częścią transakcji. Każda transakcja zaczyna się automatycznie i trwa do instrukcji COMMIT lub ROLLBACK. Stąd w standardzie nie ma polecenia BEGIN WORK, chociaż jest bardzo popularne.

Blokady

- Wiele baz danych implementuje transakcje, a w szczególności izolację różnych transakcji użytkownika za pomocą blokad, które ograniczają dostęp do danych innym użytkownikom.
- Istnieją dwa typy blokad:
  - *blokada współdzielona* (*ang. shared lock*), która pozwala innym użytkownikom odczytywać dane, ale nie zezwala na ich aktualizację.
  - *blokada wyłączna*, która nie zezwala innym transakcjom nawet na odczyt danych.

Zakleszczenia

Sesja 1	Sesja 2
Aktualizacja wiersza 14	
	Aktualizacja wiersza 15
Aktualizacja wiersza 15	
	Aktualizacja wiersza 14

Obie sesje zablokują się i po krótkiej przerwie pojawi się komunikat

```
ERROR: Dead lock detected
```

Sesja, w której pojawił się komunikat o zakleszczeniu jest anulowana, natomiast druga jest kontynuowana.

Jak nie dopuszczać do zakleszczeń?

1) Transakcja powinna dotyczyć pojedynczej aktualizacji
2) Gdy zachodzi potrzeba aktualizacji kilku wierszy w jednej transakcji należy we wszystkich takich transakcjach przetwarzać wiersze w takiej samej kolejności
3) Zastosować jawną blokadę

Zapobieganie zakleszczeniom - timeout

Gdy transakcja czeka bezczynnie na zwolnienie blokady dłużej niż ustalony okres oczekiwania ***timeout***, transakcja zostaje wycofana przez system.

Zjawisko "zagłodzenia" transakcji

- Nawet jeśli nie będziemy dopuszczać do zakleszczenia, mogą wystąpić "pechowe" transakcje, które będą wielokrotnie wybierane do wycofania i w konsekwencji mogą nigdy nie doczekać się realizacji.
- Przypisz transakcji *priorytet*, związany z momentem pierwszej próby jej realizacji (nazywany też *znacznikiem czasowym*). Wycofuj transakcję w cyklu, która ma najniższy priorytet. Z czasem transakcja uzyskuje coraz wyższy priorytet i w końcu "wygra" z innymi, później rozpoczętymi transakcjami.

Jawne blokady

Blokowanie wierszy – fraza FOR UPDATE

```
SELECT * FROM item
WHERE cena>25
FOR UPDATE
```

Blokowanie tabel

```
LOCK [ TABLE ] nazwa;
```

Kontrola współbieżności:

- **Moduł zarządzania współbieżnością** (*concurrency control manager*) steruje interakcjami pomiędzy współbieżnymi transakcjami w celu zapewnienia poprawności i spójności bazy danych.
- Na moment zapisu danych do bazy dane muszą być niedostępne w tym czasie dla innych użytkowników. **Blokady** (*locks*) mogą dotyczyć pliku (relacji), rekordu, pola. Rezultatem blokowania dostępu do zasobów jest kolejkowanie dostępu.
- Blokady:
  - Odczytu – daje dostęp tylko do odczytu i zapobiega ich modyfikacji przez inne transakcję (gdy stosujemy zapytanie do relacji)
  - Zapisu – daje dostęp do danych zarówno do odczytu i zapisu, jednocześnie uniemożliwia dostęp innym transakcjom do tego elementu danych
  - Totalne – blokowanie zapisu do relacji
  - Optymistyczne – blokowanie kolejnych rekordów w miarę modyfikowania
- **Zakleszczenie** (*dead lock*) – różne transakcje otrzymały dostęp do jednej relacji dzięki blokadzie optymistycznej, ale blokują sobie wzajemnie dostęp do poszczególnych rekordów, w efekcie żadna nie może być kontynuowana.

Przykłady:

```
START TRANSACTION;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

```
BEGIN TRANSACTION;
UPDATE ListaProduktow
SET Cecha = 4
WHERE IdProduktu = 2944;
ROLLBACK;
```

```
LOCK TABLES trans READ, customer WRITE;
UPDATE customer
SET total_value=sum_from_previous_statement
WHERE customer_id=some_id;
UNLOCK TABLES;
```

Składnia polecenia **START TRANSACTION**

- Domyślnie, MySQL pracuje z włączoną opcją AUTOCOMMIT (zresztą nie tylko MySQL). Aby więc możliwe było wykonywanie transakcji należy tą opcję wyłączyć.

```
mysql> SET AUTOCOMMIT = 0;
```

- Należy jednak pamiętać, że tak wyłączona opcja oznacza, że każda operacja na bazie danych nie zostanie trwale zapisana, dopóty, dopóki nie wykonamy polecenia `COMMIT`.
- W praktyce więc transakcja rozpoczyna się nie wykonaniem polecenia `BEGIN` ale poleceniem `SET AUTOCOMMIT = 0;` i podobnie, transakcja nie powinna się kończyć poleceniem `COMMIT` ale dodatkowo należy jeszcze włączyć z powrotem opcję `AUTOCOMMIT: SET AUTOCOMMIT = 1;.`
- Dopiero wtedy będziemy mieli wykonaną nasza transakcję i będziemy mogli wykonywać normalne zmiany w bazie danych bez używania transakcji.

– Inną możliwością jest rozpoczęcie transakcji poleceniem
`START TRANSACTION`

– W takim wypadku tryb AUTOCOMMIT zostaje zawieszony automatycznie na czas wykonywania ciągu operacji i uruchomiony ponownie w momencie wykonania polecenia COMMIT lub ROLLBACK

Przykład:

```
mysql> START TRANSACTION;
mysql> SELECT @A:=SUM(pensja) FROM tab1 WHERE type=1;
mysql> UPDATE tab2 SET suma=@A WHERE type=1;
mysql> COMMIT;
```

– Poleceń BEGIN i BEGIN WORK można używać zamiast polecenia START TRANSACTION w celu rozpoczęcia transakcji. Polecenie START TRANSACTION zostało dodane w wersji 4.0.11 MySQLa.

SAVEPOINT

```
mysql> SAVEPOINT identyfikator
mysql> ROLLBACK TO SAVEPOINT identyfikator
```

- Wyrażenie SAVEPOINT ustawia pewne miejsce w transakcji o nazwie identyfikator. Jeśli jakaś transakcja ma już oznaczone w taki sam sposób (za pomocą tego samego identyfikatora) miejsce, wówczas to miejsce jest zamazywane przez nowe miejsce.
- Wyrażenie ROLLBACK TO SAVEPOINT cofa transakcję do punktu oznaczonego przez identyfikator. Identyfikatory, które zostały ustawione po identyfikatorze, do którego odwołaliśmy się w poleceniu ROLLBACK TO SAVEPOINT są usuwane.
- Jesli wyrażenie ROLLBACK TO SAVEPOINT zwraca błąd

  `ERROR 1181: Got error 153 during ROLLBACK`
  to oznacza to, że nie istnieje miejsce oznaczone przez identyfikator, do którego się odnosiliśmy.
- Jeśli użyjemy zwykłego COMMIT lub ROLLBACK, wówczas wszystkie identyfikatory miejsc zostaną usunięte.

Spójne SELECT'y

– Spójrzmy na proces transakcji z poziomu dwóch różnych sesji.

Sesja 1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tab (f) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tab;
+---+
| f |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

Sesja 2

```
mysql> SELECT * FROM tab;
Empty set (0.00 sec)
```

– A zatem wykonując to samo polecenie SELECT z poziomu różnych sesji (jednej, w czasie której wykonujemy transakcję i drugiej, w czasie której polecenia są wykonywane "na zewnątrz" transakcji) dostaniemy dwa różne rezultaty.

– Dopiero po wykonaniu COMMIT w sesji pierwszej, wynik będzie taki sam z poziomu obu sesji.

Sesja 1

```
mysql> COMMIT; Query OK, 0 rows affected (0.00 sec)
```

Sesja 2

```
mysql> SELECT * FROM tab;
+---+
| f |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

– Taką właściwość nazywa się **spójnym czytaniem** lub **spójnym SELECT**. Każdy wykonany SELECT zwraca dane aktualne do ostatnio ZAKOŃCZONEJ transakcji.

SELECT'y FOR UPDATE

- Może się zdarzyć, że będziemy chcieli przeczytać rekord, w celu zmiany wartości niektórych z jego pól, mając jednocześnie pewność, że nikt inny nie będzie chciał w tym samym czasie wykonać tego samego.
- Na przykład dwóch użytkowników w czasie dwóch różnych sesji czytają ten sam rekord, w celu wstawienia następnego rekordu, w którym pewna wartości w pewnym polu będzie zwiększoną inkrementalnie wartością z pola przeczytanego właśnie rekordu, albo wartością maksymalną w tym polu (bieżącą wartością maksymalną).

Sesja 1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MAX(f) FROM tab;
+---------+
| MAX(f)  |
+---------+
|       3 |
+---------+
1 row in set (0.00 sec)

mysql> INSERT INTO tab(f) VALUES (4);
Query OK, 1 row affected (0.00 sec)
```

Sesja 2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MAX(f) FROM tab;
+--------+
| MAX(f) |
+--------+
|      3 |
+--------+
1 row in set (0.00 sec)

mysql> INSERT INTO tab(f) VALUES (4);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Sesja 1

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    4 |
+------+
5 rows in set (0.00 sec)
```

– W wyniku takich działań powstały dwa rekordy z wartością 4, podczas gdy chcieliśmy mieć jeden rekord z wartością 4 i jeden z wartością 5.

- Aby zabezpieczyć się przed taką sytuacją musimy ograniczyć dostęp do rekordów tablicy.
- Można to zrobić za pomocą zamknięcia dostępu do tablicy do czasu, aż transakcja nie zostanie zakończona.
- Służy do tego klauzula FOR UPDATE dodawana do polecenia SELECT. Jest to więc specjalny SELECT wykonywany z myślą o tym, aby chwilę później wykonać UPDATE.
- W przykładzie najpierw usuwamy błędne rekordy:

```
mysql> DELETE FROM tab WHERE f=4;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MAX(f) FROM tab FOR UPDATE;
+--------+
| MAX(f) |
+--------+
|      3 |
+--------+
1 row in set (0.00 sec)

mysql> INSERT INTO tab(f) VALUES (4);
Query OK, 1 row affected (0.00 sec)
```

Sesja 2

- `mysql> SELECT MAX(f) FROM tab FOR UPDATE;`
  Nie ma żadnych wyników. MySQL czeka, aż aktywna transakcja się zakończy i dopiero wówczas zwróci dane, które będą aktualne po zakończeniu transakcji w sesji 1.

Sesja 1

- `mysql> COMMIT;`
  `Query OK, 0 rows affected (0.00 sec)`
  Dopiero w w tym momencie wyniki są zwracane do sesji 2. Należy jeszcze dodać, że jeśli blokowanie trwało zbyt długo, wówczas MySQL zwróci informację, że został przekroczony czas oczekiwania.

Sesja 2

```
– mysql> SELECT MAX(f) FROM tab FOR UPDATE;
  +---------+
  | MAX(f)  |
  +---------+
  |       4 |
  +---------+
  1 row in set (4.20 sec)

  mysql> INSERT INTO tab(f) VALUES(5);
  Query OK, 1 row affected (0.00 sec)

  mysql> COMMIT;
  Query OK, 0 rows affected (0.01 sec)

  mysql> SELECT * FROM tab;
  +------+
  | f    |
  +------+
  |    1 |
  |    2 |
  |    3 |
  |    4 |
  |    5 |
  +------+
  5 rows in set (0.00 sec)
```

SELECT'y w trybie wspólnym

- Kolejnym typem ograniczenia dostępu do danych jest tzw. `LOCK IN SHARE MODE`.
- Taki sposób ograniczania danych zapewnia dostęp do najświeższych danych (wprowadzanych w czasie transakcji) z zewnątrz transakcji.
- Takie udostępnianie danych blokuje wszystkie zmiany danych (polecenia UPDATE i DELETE) i jeśli ostatnie zmiany nie były jeszcze potwierdzone poleceniem COMMIT, powoduje oczekiwanie na wynik zapytania dopóty, dopóki nie nastąpi potwierdzenie transakcji w sesji, która rozpoczęła tą transakcję.

Sesja 1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MAX(f) FROM tab LOCK IN
SHARE MODE;
+---------+
| MAX(f)  |
+---------+
|       5 |
+---------+
1 row in set (0.00 sec)
```

– W tym czasie użytkownik w innej sesji próbuje wykonać UPDATE

Sesja 2

– `mysql> UPDATE tab SET f = 55 WHERE f=5;`

Jednak polecenie oczekuje dopóty dopóki nie nastąpi zakończenie transakcji w sesji 1.

Sesja 1

– `mysql> COMMIT;`
`Query OK, 0 rows affected (0.00 sec)`

Sesja 2

```
mysql> UPDATE tab SET f = 55 WHERE f=5;
Query OK, 0 rows affected (6.95 sec)
Rows matched: 0 Changed: 0 Warnings: 0

mysql> UPDATE tab SET f = 55 WHERE f=5;
Query OK, 1 row affected (43.30 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM tab;
+------+
| f    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|   55 |
+------+
5 rows in set (0.00 sec)
```

Nie wszystko można cofnąć

- Niektóre polecenia SQL nie mogą być cofnięte, pomimo tego, że wykonywane będą w transakcji. Należą do nich, generalnie, wszystkie polecenia języka DDL, czyli takie, za pomocą których tworzymy lub usuwamy bazy danych, albo tworzymy lub usuwamy tablice w obrębie bazy.
- Należy tak zaprojektować transakcje, aby nie wykonywać w jej obrębie takich poleceń. Jeśli w obrębie transakcji użyjemy polecenia, za pomocą którego utworzymy tablicę, a następnie użyjemy polecenia, które nie zostanie poprawnie wykonane z powodu jakiegoś błędu, wówczas trzeba się liczyć z tym, że po wykonaniu polecenia ROLLBACK nie wszystkie efekty wykonania różnych poleceń zostaną cofnięte.

Wyrażenia, które wywołują automatycznie COMMIT

- Niektóre polecenia automatycznie kończą transakcję pomimo tego, że nie wykonamy explicite polecenia `COMMIT`.
- `ALTER TABLE, BEGIN, CREATE INDEX, DROP DATABASE, DROP INDEX, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE`
- Polecenie `UNLOCK TABLES` kończy transakcję ze skutkiem COMMIT nawet jeśli jakieś tablice są w danym momencie zablokowane.