

Powtórzenie transakcji na przykładzie serwera MySQL

Na bazie loty.

Co to jest transakcja w bazach danych?

Transakcja w bazach danych to zbiór operacji, które są wykonywane jako całość. Może obejmować jedno lub więcej zapytań, takich jak **INSERT**, **UPDATE**, **DELETE** i inne, które mają na celu modyfikację danych. Transakcja zapewnia, że operacje są realizowane w sposób atomowy, tzn. albo wszystkie operacje zakończą się sukcesem, albo żadna z nich nie będzie miała wpływu na dane. To oznacza, że dane w bazie są w spójnym stanie niezależnie od tego, co się stanie w trakcie przetwarzania transakcji.

Transakcje są szczególnie przydatne w środowiskach wieloużytkownikowych, gdzie mogą wystąpić równoczesne operacje na tej samej bazie danych. Transakcje pomagają uniknąć problemów związanych z niejednoznacznościami lub częściowymi zmianami danych, które mogą prowadzić do błędów i nieścisłości.

Cechy transakcji

Każda transakcja w systemie bazodanowym jest oceniana na podstawie czterech podstawowych właściwości, znanych jako **ACID**.

ACID – Właściwości transakcji

ACID to akronim czterech podstawowych właściwości transakcji, które zapewniają spójność danych w bazie danych:

1. A - Atomicity (Atomowość)

Atomowość oznacza, że transakcja jest traktowana jako **jednostka atomowa**. Oznacza to, że operacje w ramach transakcji są wykonywane w całości lub wcale. Jeśli któraś z operacji nie powiedzie się, cała transakcja jest wycofywana (rollback) i żadne zmiany w bazie danych nie są wprowadzane.

Przykład:

- Jeśli transakcja obejmuje operację **INSERT** i **UPDATE**, ale jedna z tych operacji się nie powiedzie, cała transakcja zostanie cofnięta, a żadna z operacji nie zostanie zapisana w bazie.

2. C - Consistency (Spójność)

Spójność zapewnia, że transakcja zmienia stan bazy danych z jednego spójnego stanu do drugiego. Po zakończeniu transakcji dane w bazie muszą być zgodne z wszelkimi zdefiniowanymi regułami i restrykcjami (np. kluczami obcymi, ograniczeniami, itp.).

Przykład:

- Jeśli transakcja ma na celu dodanie nowego rekordu, po jej zakończeniu rekord powinien spełniać wszystkie zasady walidacji bazy danych, takie jak unikalność klucza, prawidłowy typ danych, itp.

3. I - Isolation (Izolacja)

Izolacja oznacza, że operacje wykonywane w ramach jednej transakcji są izolowane od innych transakcji. Nawet jeśli inne transakcje są wykonywane równocześnie, transakcja A nie będzie miała wpływu na transakcję B i vice versa. Wartość izolacji zależy od ustawionego poziomu izolacji, który określa, jak transakcje mogą oddziaływać na siebie.

Istnieją różne poziomy izolacji:

- **Read Uncommitted** – pozwala na odczytanie danych, które zostały zmodyfikowane, ale jeszcze nie zostały zatwierdzone.
- **Read Committed** – pozwala na odczytanie tylko zatwierdzonych danych.
- **Repeatable Read** – zapewnia, że dane odczytane przez transakcję nie zmieniają się w trakcie jej trwania.
- **Serializable** – najwyższy poziom izolacji, gdzie transakcje są wykonywane w pełnej izolacji, blokując dostęp do danych przez inne transakcje.

Przykład:

- Transakcja A może zaktualizować cenę lotu, ale nie wpłynie to na dane w innych transakcjach, które pracują na tej samej tabeli, do momentu zatwierdzenia transakcji.

4. D - Durability (Trwałość)

Trwałość oznacza, że po zatwierdzeniu transakcji, jej zmiany są trwałe i nie mogą zostać utracone, nawet w przypadku awarii systemu. Po wykonaniu **COMMIT**, wszystkie zmiany wprowadzone przez transakcję są zapisane na stałe w bazie danych.

Przykład:

- Po zatwierdzeniu transakcji, nawet jeśli system ulegnie awarii, dane nie zostaną utracone i po ponownym uruchomieniu systemu będą dostępne.

Transakcje w SQL, w tym w MySQL, pozwalają na grupowanie kilku operacji tak, aby były one wykonywane jako całość. Jeśli jedna z operacji w transakcji się nie powiedzie, możemy wycofać wszystkie operacje. Język **TCL** (Transaction Control Language) w SQL zawiera trzy główne instrukcje: **BEGIN/START TRANSACTION**, **COMMIT** i **ROLLBACK**.

Podstawowe pojęcia transakcji:

- **BEGIN/START TRANSACTION**: Rozpoczyna nową transakcję, grupując operacje.
- **COMMIT**: Zatwierdza wszystkie zmiany wprowadzone przez transakcję. Po **COMMIT** dane zostają trwale zapisane.

- **ROLLBACK:** Wycofuje wszystkie zmiany w transakcji do punktu początkowego lub do określonego punktu zapisu (ang. *savepoint*), jeśli został on ustawiony. Zmiany wycofane za pomocą **ROLLBACK** nie są zapisane w bazie danych.

Przykład transakcji w MySQL:

Założmy, że mamy bazę danych dla linii lotniczych z tabelami **Klienci**, **Loty**, **Rezerwacje** oraz **Płatności**.

Przykład pełnej transakcji:

Chcemy utworzyć rezerwację, dodać płatność i zatwierdzić zmiany tylko wtedy, gdy obie operacje zakończą się sukcesem.

START TRANSACTION;

-- Dodajemy nową rezerwację dla klienta

```
INSERT INTO Rezerwacje (id_klienta, id_lotu, data_rezerwacji)
VALUES (1, 2, NOW());
```

-- Pobieramy ID nowo dodanej rezerwacji

```
SET @last_rezerwacja_id = LAST_INSERT_ID();
```

-- Dodajemy płatność za nowo utworzoną rezerwację

```
INSERT INTO Płatności (id_rezerwacji, kwota, data_platnosci, metoda_platnosci)
VALUES (@last_rezerwacja_id, 300.00, NOW(), 'karta kredytowa');
```

-- Zatwierdzamy transakcję, jeśli wszystko się powiodło

COMMIT;

Jeśli którykolwiek z powyższych kroków zawiedzie, możemy użyć **ROLLBACK**:

ROLLBACK;

Dzięki temu ani rezerwacja, ani płatność nie zostaną zapisane w bazie danych, jeśli coś pójdzie nie tak.

Anomalie transakcyjne

Anomalie transakcyjne pojawiają się, gdy wiele transakcji działa na tej samej bazie danych bez odpowiedniej kontroli współbieżności. Często są wynikiem braku odpowiednich poziomów izolacji. Poniżej opisano główne rodzaje anomalii oraz jak transakcje TCL pomagają im zapobiegać.

a) Dirty Read (brudny odczyt)

Jest to sytuacja, w której transakcja odczytuje dane zmodyfikowane przez inną transakcję, która jeszcze nie została zatwierdzona. Jeśli tamta transakcja wycofa zmiany, dane stają się nieprawidłowe.

Przykład Dirty Read:

1. Transakcja A modyfikuje cenę lotu, ale jeszcze nie wykonuje **COMMIT**.
2. Transakcja B odczytuje tę zmodyfikowaną cenę.
3. Transakcja A wycofuje zmianę (**ROLLBACK**), a więc cena wraca do pierwotnej wartości, jednak Transakcja B już uzyskała nieprawidłowe dane.

Rozwiązanie: Ustawienie poziomu izolacji transakcji na wyższy, np. **READ COMMITTED** lub **REPEATABLE READ**, zapobiegnie odczytywaniu niezatwierdzonych danych.

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

b) Non-repeatable Read (niepowtarzalny odczyt)

Zachodzi, gdy w ramach jednej transakcji dane są odczytywane wielokrotnie, a inne transakcje między odczytami modyfikują te dane. To powoduje, że odczyty są niespójne.

Przykład Non-repeatable Read:

1. Transakcja A odczytuje cenę lotu.
2. Transakcja B zmienia cenę lotu i zatwierdza zmiany (**COMMIT**).
3. Transakcja A ponownie odczytuje cenę lotu i widzi różne wartości.

Rozwiązanie: Ustawienie poziomu izolacji na **REPEATABLE READ** zapewni, że transakcja A będzie miała spójne odczyty.

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

c) Phantom Read (odczyt widma)

Występuje, gdy jedna transakcja odczytuje zestaw wierszy spełniających określony warunek, a inna transakcja w międzyczasie dodaje lub usuwa wiersze, które spełniają ten warunek. Powoduje to zmiany w zestawie danych przy kolejnym odczycie.

Przykład Phantom Read:

1. Transakcja A odczytuje liczbę wszystkich rezerwacji na dany lot.
2. Transakcja B dodaje nową rezerwację na ten sam lot i zatwierdza zmiany.
3. Transakcja A ponownie odczytuje liczbę rezerwacji i widzi nową liczbę.

Rozwiązanie: Ustawienie poziomu izolacji na **SERIALIZABLE**, który zapobiega tej anomalii, blokując tabelę na czas trwania transakcji.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Przykład transakcji z obsługą poziomów izolacji

Przy użyciu **SERIALIZABLE** unikniemy zjawiska **Phantom Read**:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

-- Sprawdzenie liczby rezerwacji na lot o id_lotu = 2

SELECT COUNT(*) FROM Rezerwacje WHERE id_lotu = 2;

-- Dodanie nowej rezerwacji

INSERT INTO Rezerwacje (id_klienta, id_lotu, data_rezerwacji)

VALUES (1, 2, NOW());

COMMIT;

Dzięki poziomowi **SERIALIZABLE** inne transakcje będą musiały poczekać na zakończenie powyższej transakcji przed wykonaniem operacji na tej tabeli, co zapobiega anomalii typu Phantom Read.

Na bazie loty z poprzedniego ćwiczenia wykonaj następujące przykłady:

Przykład1:

W tej transakcji najpierw dodajemy nowego klienta, a następnie tworzymy rezerwację i płatność. Jeśli którakolwiek z operacji się nie powiedzie, cała transakcja zostanie wycofana.

BEGIN;

-- Dodajemy nowego klienta

```
INSERT INTO Klienci (imie, nazwisko, email, telefon, adres)
VALUES ('Anna', 'Nocul', 'anna.nocul@example.com', '123456789', 'ul. Zielona 5,
Warszawa');
```

```
-- Pobieramy ostatnie id_klienta
SET @last_klient_id = LAST_INSERT_ID();
```

```
-- Tworzymy rezerwację dla nowego klienta
INSERT INTO Rezerwacje (id_klienta, id_lotu, data_rezerwacji)
VALUES (@last_klient_id, 2, NOW());
```

```
-- Pobieramy ostatnie id_rezerwacji
SET @last_rezerwacja_id = LAST_INSERT_ID();
```

```
-- Tworzymy płatność dla nowej rezerwacji
INSERT INTO Płatności (id_rezerwacji, kwota, data_platnosci, metoda_platnosci)
VALUES (@last_rezerwacja_id, 350.00, NOW(), 'karta kredytowa');
```

```
COMMIT;
```

Jeśli jakkolwiek część powyższej transakcji się nie powiedzie, można użyć **ROLLBACK**, aby cofnąć wszystkie zmiany w ramach tej transakcji.

Przykład 2:

Ta transakcja aktualizuje status istniejącej rezerwacji oraz zmienia metodę płatności. O id_rezerwacji=10.

```
BEGIN;
```

```
-- Zaktualizuj status rezerwacji dla id_rezerwacji = 10
UPDATE Rezerwacje
```

```
SET status = 'potwierdzona'
```

```
WHERE id_rezerwacji = 10;
```

```
-- Zaktualizuj metodę płatności dla płatności związanej z id_rezerwacji = 10
```

```
UPDATE Płatności
```

```
SET metoda_platnosci = 'BLIK'
```

```
WHERE id_rezerwacji = 10;
```

```
COMMIT;
```