Bazy danych
przed 13 wykładem

Andrzej Giniewicz

14.06.2024

Kolejna porcja materiałów dotyczy transakcji w języku SQL.

1 Transakcje

Transakcje są zdefiniowane w standardzie SQL jako sposób zgrupowania więcej niż jednego zapytania tak, aby tworzyło nierozerwalną całość z punktu widzenia odzyskiwania danych. Oznacza to, że cała transakcja albo się uda, albo cała transakcja się nie uda. Nie ma możliwości, aby połowa transakcji się udała a połowa nie. W szczególności nie ma możliwości, że w tabeli wydatków dopiszemy już kwotę a przy usuwaniu tej samej kwoty do tabeli środków dostępnych dla firmy, pojawi się błąd, przez co dane utracą spójność — niby pieniądze wydane, ale na koncie dalej są. To, czy transakcje są dostępne, czy też nie, zależy od silnika. Domyślny silnik dostępny w MariaDB, czyli InnoDB, obsługuje transakcje. O innych silnikach wspierających transakcje możemy przeczytać w dokumentacji[1].

Rodzaj transakcji wspierany przez MySQL to tak zwane transakcje ACID, gdzie pierwsze litery skrótu pochodzą od czterech cech dobrze zaimplementowanych transakcji:

Atomicity, czyli niepodzielność. Oznacza to, że transakcje muszą być albo kompletne, aby całość transakcji jest wycofywana. Oznacza to, że silnik musi mieć możliwość cofnięcia modyfikacji do tabeli, przynajmniej tak długo, jak trwa transakcja. Obciąża to dodatkowo zapytanie, ponieważ baza musi przechowywać dodatkowe informacje w czasie trwania całej transakcji;

Consistency, czyli spójność. Oznacza to, że nie można doprowadzić do powstania niespójności, jak w przypadku przykładu opisującego wydatki i budżet — każda niespójność to poważne niebezpieczeństwo dla wszystkich decyzji podjętych na podstawie niespójnych danych;

[1] Patrz `https://mariadb.com/kb/en/mariadb-transactions-and-isolation-levels-for-sql-server-users/#transactions-storage-engines-and-the-binary-log`.

Isolation, czyli izolacja. Oznacza to, że aby transakcja działała poprawnie, dane modyfikowane przez całą transakcją nie mogą być używane przez inną transakcję, do czasu, aż ta się nie zakończy. Bez tej gwarancji, istniałaby możliwość zmiany części tabel, inna transakcja mogłaby z nich skorzystać, następnie mogłoby być konieczne wycofanie zmian — podczas gdy inna transakcja korzystałaby wciąż z połowicznie zmienionych wartości;

Durability, czyli trwałość. Oznacza to, że jeśli transakcja się zakończy, wszystkie zmiany w niej wykonane będą trwałe, nawet jeśli urządzenie utraci źródło zasilania lub zostanie ponownie uruchomione z jakiegoś powodu. Po ponownym uruchomieniu musimy mieć gwarancję, że wszystkie zakończone transakcje są zapisane. Wbrew pozorom nie jest to zawsze prawdą — wiele baz danych zapisuje dane do pamięci RAM i trafiają one na dysk twardy raz na jakiś czas. W takim przypadku zapisana w pamięci RAM transakcja może zniknąć, jeśli utracimy źródło zasilania.

Ogólny schemat transakcji w MariaDB to

```
START TRANSACTION;
-- kilka komend: SELECT, UPDATE, etc;
COMMIT; -- lub ROLLBACK;
```

Przy czym `START TRANSACTION` zaczyna transakcję, `COMMIT` kończy ją sukcesem, a `ROLLBACK` kończy, wycofując wszystkie zmiany.

Co ważne, nieintuicyjna może się wydawać logika automatyzacji używa w MySQL oraz MariaDB. Domyślnie wszystkie komendy są automatycznie zamykane w jednolinijkowych transakcjach. Oznacza to, że gdy do tej pory pisaliśmy

```
SELECT * FROM tabela;
SELECT * FROM inna;
```

W rzeczywistości wykonywaliśmy

```
START TRANSACTION;
SELECT * FROM tabela;
COMMIT;
START TRANSACTION;
SELECT * FROM inna;
COMMIT;
```

Nie zawsze jest to wydajne i opłacalne, dlatego to zachowanie można wyłączyć, pisząc

```
SET autocommit = 0;
```

W ten sposób, automatycznie tworzona jest transakcja, ale nie jest kończona. Oznacza to, że

```
SET autocommit = 0;
SELECT * FROM tabela;
SELECT * FROM inna;
```

będzie równoważne z

```
SET autocommit = 0;
START TRANSACTION;
SELECT * FROM tabela;
SELECT * FROM inna;
```

przy czym nie pojawi się tam `COMMIT`. Oznacza to, że jeśli wyłączymy `autocommit`, musimy napisać

```
SET autocommit = 0;
SELECT * FROM tabela;
SELECT * FROM inna;
COMMIT;
```

Ręczne użycie `START TRANSACTION` wyłącza `autocommit`, ale tylko do najbliższego wystąpienia `COMMIT` lub `ROLLBACK` — potem ponownie jest włączane.

O ile automatyczne robienie `COMMIT` jest w MySQL i MariaDB, nie ma automatycznego robienia `ROLLBACK` w momencie pojawienia się błędu. Oznacza to, że jeśli wykonamy:

```
START TRANSACTION;
-- UPDATE działający;
-- UPDATE z błędem;

-- UPDATE działający;
-- UPDATE poprawiony już działa;
COMMIT;
```

pierwszy działający `UPDATE` wykona się dwa razy — tylko zapytanie zakończone błędem będzie cofnięte. Oznacza to, że powinniśmy napisać

```
START TRANSACTION;
-- UPDATE działający;
-- UPDATE z błędem;
ROLLBACK;
START TRANSACTION;
-- UPDATE działający;
-- UPDATE poprawiony już działa;
COMMIT;
```

lub ewentualnie, co zadziała, ale nie jest rekomendowane

```
START TRANSACTION;
-- UPDATE działający;
-- UPDATE z błędem;

-- UPDATE poprawiony już działa;
COMMIT;
```

Jeśli wykonujemy kod z poziomu aplikacji w Pythonie, R lub innym języku, to po naszej stronie (lub po stronie biblioteki, z której korzystamy), jest wywołanie `ROLLBACK` w momencie wystąpienia wyjątku.

Wykonanie `ROLLBACK` może zostać zautomatyzowane za pomocą procedur składowanych. Choć nie korzystaliśmy z nich do tej pory, zainteresowanych zachęcam do zapoznania się z dokumentacją na stronie `https://mariadb.com/kb/en/stored-procedure-overview/`. Interesujący nas fragment, to

```
DELIMITER //

CREATE PROCEDURE safecall()
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

    START TRANSACTION;
    -- zapytania;
    COMMIT;
END//

DELIMITER ;

CALL safecall;
```

W kodzie powyżej tworzymy procedurę `safecall` i wywołujemy ją. Słowo kluczowe `DELIMITER` zmienia symbol końca zapytania ze średnika na `//` oraz ponownie na średnik po końcu procedury, aby całość procedury, włącznie ze średnikami, była jednym zapytaniem i wykonana została na raz. Jeśli ręczne zrobienie `ROLLBACK` w kodzie lub w aplikacji w Pythonie lub R nie jest możliwe, tego typu procedura składowana pozwala obejść problem i zautomatyzować cofanie transakcji, które zakończą się wyjątkiem.