22 Transakcje, blokady i zakleszczenia

ZAGADNIENIA

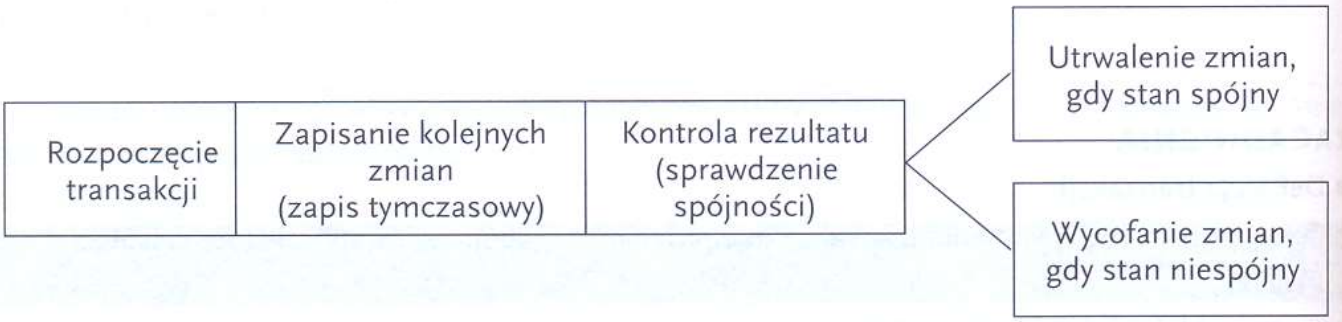
- **Definicja transakcji**
- **Cechy poprawnie wykonanej transakcji: niepodzielność, spójność, współbieżność, izolacja, trwałość**
- **Co to są blokady i zakleszczenia?**

Akcja lub seria akcji przeprowadzanych przez pojedynczego użytkownika lub aplikację w celu uzyskania dostępu do SZBD związanego z odczytem lub modyfikacją zawartości bazy danych nazywana jest transakcją.

Wymiana danych pomiędzy bazą danych a użytkownikami lub programami może być narażona na utratę spójności. Stan bazy danych określa się jako spójny, gdy jej zawartość odpowiada rzeczywistości, a w przechowywanych danych nie pojawiają się błędy. Aby baza danych nie straciła spójności, wprowadzanie i modyfikacja danych powinny zachodzić w sposób gwarantujący poprawność przechowywanych informacji. Skończony proces wymiany danych pomiędzy bazą danych a użytkownikiem lub programem nazywamy **transakcją**. Poprawnie wykonana transakcja powinna mieć pięć podstawowych cech. Powinna być niepodzielna, spójna, współbieżna, izolowana i trwała.

- **Niepodzielna**
  Transakcja może składać się z kilku akcji. Gdy zdarzy się, że któraś z akcji narazi bazę danych na utratę spójności i tym samym nie będzie mogła zostać wykonana, całość transakcji (wszystkie pozostałe akcje) również nie powinna być wykonana.
- **Spójna**
  Transakcje nie mogą pozostawić bazy danych w stanie niespójnym, tzn. że baza danych, która jest w stanie spójnym przed wykonaniem transakcji, powinna być w tym samym stanie po wykonaniu transakcji.
- **Współbieżna**
  To cecha transakcji, która pozwala na pracę z bazą danych wielu użytkownikom w tym samym czasie i ma zagwarantować, że operacje wykonywane przez tych użytkowników nie wprowadzą bazy danych w stan niespójny. W praktyce oznacza to, że transakcje podlegają wcześniej określonym mechanizmom zsynchronizowanego blokowania i izolowania, które blokują dostęp do danych jednemu użytkownikowi, w momencie gdy te same dane modyfikuje drugi użytkownik.
- **Izolowana**
  Baza danych musi blokować (izolować) dane w momencie trwania jednej transakcji, tak aby wykonując modyfikacje wraz z innymi transakcjami, baza danych nie utraciła spójności.

- **Trwała**
  Ta cecha transakcji oznacza, że po jej wykonaniu wprowadzone przez transakcje zmiany powinny zostać utrwalone na wypadek awarii oprogramowania lub sprzętu. Zakończenie transakcji nie musi oznaczać wprowadzenia zmian. Transakcja może zakończyć się wycofaniem zmian, gdy któryś z ich elementów będzie błędny. Przebieg transakcji może ilustrować następujący schemat:



Rys. 22.1. Przebieg transakcji

Koncepcja transakcji w języku SQL określa transakcję jako grupę komend połączonych i wykonywanych jako jedna operacja.

Transakcje są atomowe – są wykonywane całościowo jako jedna operacja albo wcale (w przypadku błędu). Aby lepiej zrozumieć istotę transakcji, można posłużyć się przykładem transferu pieniędzy z jednego konta bankowego na drugie. Potwierdzenie wykonania transferu wiąże się z sytuacją, w której pieniądze zostaną usunięte z jednego konta i pojawią się na drugim, docelowym koncie. Jeśli transakcja miałaby zakończyć się błędem, pieniądze powinny pozostać w całości na koncie pierwszym, z którego transfer miał być wykonany. Niedopuszczalne są zatem sytuacje, w których w wyniku błędu lub problemów sprzętowych pieniądze znalazłyby się na koncie docelowym i nie zostałyby usunięte z konta źródłowego. Inna niedopuszczalna sytuacja miałaby miejsce, gdyby pieniądze zostały usunięte i niezaksięgowane na koncie docelowym (zginęłyby podczas transferu). Kolejnym przypadkiem, którego nie można zaakceptować, byłby transfer tylko pewnej części kwoty. Transakcje działają podobnie jak w przykładzie przelewów bankowych. Muszą być wykonane i potwierdzone w całości jako jedna operacja. W przeciwnym razie w całości muszą zostać wycofane.

Transakcje rozpoczynamy słowem kluczowym **BEGIN** (rozpocznij), natomiast kończymy poleceniem **COMMIT** (potwierdź).

PostgreSQL każde polecenie (nawet te nierozpoczęte słowem **BEGIN**) traktuje jak transakcje. Jeśli polecenie zakończy się sukcesem, implikowane jest słowo **COMMIT**, natomiast jeśli w poleceniu wystąpi błąd, wówczas automatycznie implikowane jest słowo **ROLLBACK**. Transakcja rozpoczęta słowem **BEGIN** wprowadza zmiany na wirtualnej – tymczasowej kopii danych. Zmiany wprowadzane są do bazy danych i widoczne dla pozostałych użytkowników w chwili ich zatwierdzenia słowem **COMMIT**.

PRZYKŁAD 22.1

Aby prześledzić konstrukcję transakcji, posłużymy się tabelą **pracownicy_bhp**.

```
 id_pracownika |   imie   | nazwisko | wiek
---------------+----------+----------+------
             1 | Jan      | Nowak    |   24
             2 | Wojciech | Kowalski |   30
             3 | Marian   | Nowak    |   33
             4 | Marcin   | Tracz    |   26
             5 | Ewa      | Werner   |   23
```

Rys. 22.2. Wyświetlenie zawartości tabeli

Po słowie kluczowym **BEGIN**, które rozpoczyna transakcję, występuje średnik. Polecenie **UPDATE** powoduje, że wiek wszystkich pracowników o **id_pracownika** większym od dwa zmieniany jest na 18.

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=18 WHERE id_pracownika > 2;
UPDATE 3
mojabaza=# SELECT * FROM pracownicy_bhp;
 id_pracownika |   imie   | nazwisko | wiek
---------------+----------+----------+------
             1 | Jan      | Nowak    |   24
             2 | Wojciech | Kowalski |   30
             3 | Marian   | Nowak    |   18
             4 | Marcin   | Tracz    |   18
             5 | Ewa      | Werner   |   18
(5 wierszy)
```

Rys. 22.3. Rozpoczęcie transakcji

Możemy teraz wycofać zmianę, stosując słowo kluczowe **ROLLBACK**.

```
mojabaza=# ROLLBACK;
ROLLBACK
mojabaza=# SELECT * FROM pracownicy_bhp;
 id_pracownika |   imie   | nazwisko | wiek
---------------+----------+----------+------
             1 | Jan      | Nowak    |   24
             2 | Wojciech | Kowalski |   30
             3 | Marian   | Nowak    |   33
             4 | Marcin   | Tracz    |   26
             5 | Ewa      | Werner   |   23
(5 wierszy)
```

Rys. 22.4. Wycofanie transakcji

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET imie='Marcin' WHERE id_pracownika > 1;
UPDATE 4
mojabaza=# SAVEPOINT punkt_pierwszy;
SAVEPOINT
mojabaza=# UPDATE pracownicy_bhp SET nazwisko='Zagloba' WHERE id_pracownika > 1;
UPDATE 4
mojabaza=# SELECT * FROM pracownicy_bhp;
 id_pracownika |  imie  | nazwisko | wiek
---------------+--------+----------+------
             1 | Jan    | Nowak    |   24
             2 | Marcin | Zagloba  |   30
             3 | Marcin | Zagloba  |   33
             4 | Marcin | Zagloba  |   26
             5 | Marcin | Zagloba  |   23
(5 wierszy)
```

Rys. 22.5. Użycie punktów zapisu w transakcji

Na przykładzie ilustrowanym za pomocą rysunku 22.4 zastosowaliśmy wycofanie transakcji. Możemy się o tym przekonać, stosując kwerendę **SELECT * FROM pracownicy_bhp;**. Wiek pracowników wrócił do poprzedniej wartości. Jeśli zmianę chcemy zatwierdzić, używamy polecenia **COMMIT**, które – podobnie jak **ROLLBACK** – kończy transakcję.

Podczas wykonywania transakcji możemy określić również punkty zapisu, tzw. **SAVEPOINTS**.

Po słowie kluczowym **SAVEPOINT** używamy identyfikatora. Dzięki zdefiniowaniu identyfikatorów punktów zapisu możemy cofnąć zmiany przeprowadzane przez transakcję do określonego za pomocą identyfikatora punktu (rys. 22.5).

Aby cofnąć zmiany do zdefiniowanego wcześniej punktu **SAVEPOINT**, używamy polecenia:

```
mojabaza=# ROLLBACK TO SAVEPOINT punkt_pierwszy;
ROLLBACK
mojabaza=# SELECT * FROM pracownicy_bhp;
 id_pracownika |  imie  | nazwisko | wiek
---------------+--------+----------+------
             1 | Jan    | Nowak    |   24
             2 | Marcin | Kowalski |   30
             3 | Marcin | Nowak    |   33
             4 | Marcin | Tracz    |   26
             5 | Marcin | Werner   |   23
(5 wierszy)
```

Rys. 22.6. Cofanie transakcji do poprzednio zdefiniowanych punktów zapisu

Kontrola transakcji

Podczas wykonywania transakcji możliwy jest podział ich na mniejsze części za pomocą słowa kluczowego **SAVEPOINT**. Aby usunąć **SAVEPOINT**, używamy polecenia:

```
RELEASE SAVEPOINT nazwa_punktu_przywracania;
```

Całość transakcji można nadal wycofać, używając słowa kluczowego **ROLLBACK**.

Blokady i zakleszczenia

Zakleszczenia występują w bazie danych w chwili, gdy co najmniej dwie transakcje nałożą blokadę na dane, które mają być użyte przez inne transakcje. Wówczas transakcje czekają, aż każda z nich zostanie wykonana, i dochodzi do zakleszczenia. Sytuacja ta spowodowana jest blokadami nakładanymi na dane w chwili ich modyfikacji.

PRZYKŁAD 22.2

Aby przeanalizować powyższy przypadek, posłużymy się dwoma oknami sesji numerowanymi **okno1**, **okno2**.

OKNO1

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=wiek+10 WHERE id_pracownika > 1;
UPDATE 4
mojabaza=#
```

Rys. 22.7. Rozpoczęcie pierwszej transakcji

OKNO2

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=10 WHERE id_pracownika>1;
```

Rys. 22.8. Rozpoczęcie drugiej transakcji

Jak prezentuje powyższy przykład, transakcja w drugim oknie została wstrzymana (napis **UPDATE 4** nie został wyświetlony), co oznacza, że zadziałał mechanizm blokujący dostęp do krotek, na których aktualnie wykonywana jest już inna transakcja. Aby odblokować drugie okno, musimy zakończyć transakcję w pierwszym oknie poleceniem **COMMIT** lub **ROLLBACK**. W naszym przykładzie w obu oknach (w kolejności **okno1**, **okno2**) zakończymy transakcję wycofaniem zmian. Po wpisaniu w pierwszym oknie polecenia **ROLLBACK** transakcja w oknie numer 2 zostanie odblokowana.

```
mojabaza=# UPDATE pracownicy_bhp SET wiek=10 WHERE id_pracownika>1;
UPDATE 4
```

Rys. 22.9. Wykonanie polecenia transakcji w drugim oknie po zakończeniu pierwszej transakcji

Powyższy przykład pozwolił na zilustrowanie mechanizmu blokowania. Nadeszła pora na prześledzenie, kiedy dochodzi do zakleszczenia.

Do zakleszczenia dochodzi, gdy pierwsza transakcja nakłada blokadę na modyfikowane krotki, w tym czasie druga transakcja chce wykonać operacje i czeka na zakończenie transakcji pierwszej, pierwsza transakcja również nie może wykonać zmian, ponieważ w drugiej kolejności krotki blokowane są przez transakcję drugą. W efekcie obie transakcje czekają na swoje zakończenie.

PRZYKŁAD 22.3

Przykład zakleszczenia:

1. TRANSAKCJA1 (zablokowanie krotek o **id_pracownika = 1**)

```
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=55 WHERE id_pracownika = 1;
UPDATE 1
mojabaza=#
```

Rys. 22.10. Rozpoczęcie pierwszej transakcji i nałożenie blokady na dane

2. TRANSAKCJA2 (zablokowanie krotki o **id_pracownika = 2**)

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=60 WHERE id_pracownika=2;
UPDATE 1
mojabaza=#
```

Rys. 22.11. Rozpoczęcie drugiej transakcji i nałożenie blokady na dane

3. TRANSAKCJA1 (próba zmiany krotki o **id_pracownika = 2** – oczekiwanie na zdjęcie blokady przez TRANSAKCJĘ2)

```
mojabaza=# BEGIN;
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=55 WHERE id_pracownika = 1;
UPDATE 1
mojabaza=# UPDATE pracownicy_bhp SET wiek=55 WHERE id_pracownika = 2;
```

Rys. 22.12. Próba modyfikacji danych blokowanych przez transakcję drugą – blokada

4. TRANSAKCJA2 (próba zmiany krotki o **id_pracownika = 1** – oczekiwanie na zdjęcie blokady nałożonej przez TRANSAKCJĘ1, czyli zakleszczenie).

```
BEGIN
mojabaza=# UPDATE pracownicy_bhp SET wiek=60 WHERE id_pracownika=2;
UPDATE 1
mojabaza=# UPDATE pracownicy_bhp SET wiek=60 WHERE id_pracownika=1;
BŁĄD:  wykryto zakleszczenie
SZCZEGÓŁY:  Proces 4820 oczekuje na ShareLock na transakcja 837; zablokowany przez 4456.
Proces 4456 oczekuje na ShareLock na transakcja 838; zablokowany przez 4820.
PODPOWIEDŹ:  Przejrzyj dziennik serwera by znaleźć szczegóły zapytania.
mojabaza=#
```

Rys. 22.13. Próba modyfikacji danych blokowanych przez transakcję pierwszą przez okno drugiej transakcji – zakleszczenie

Jak to ilustruje powyższy przykład, PostgreSQL wykrył zakleszczenie, a transakcja z tego powodu została wycofana.