Погружение в СУБД. Сезон 2017

Совместный доступ к данным

Дмитрий Барашев

Computer Science Center

Санкт-Петербург 2017

Сверстано в Папирии

papeeria.com

онлайн редактор для LaTeX и Markdown• совместное редактирование в реальном времени • интеграция с Git репозиториями • графики

подсветка синтаксиса • автодополнение • проверка орфографии • предпросмотр математических формул • галерея шаблонов

Транзакции

Транзакции

- ACID свойства транзакционной системы

Транзакции

- Уровни изолированности транзакций

Транзакции

- Что делать, если в СУБД транзакции не поддерживаются

Летайте марсолётами
Марсофлота!

Марсофлот

схема

```
-- Данные о рейсе
CREATE TABLE Journey(id SERIAL PRIMARY KEY, date DATE,
    capacity INT, booked INT,
    CHECK(capacity>0), CHECK(capacity>=booked));
-- Пользовательская запись и остаток денег на счету
CREATE TABLE UserAct(id SERIAL PRIMARY KEY,
    credit INT, CHECK(credit >= 0));

-- Резервирование и кто платит
CREATE TABLE Booking(id SERIAL PRIMARY KEY, refcode TEXT,
    payer INT REFERENCES UserAct);

-- Позиция в билете
CREATE TABLE BookingPos(booking_id INT REFERENCES Booking,
    name TEXT, payed INT, UNIQUE(booking_id, name));

-- Связь между резервированием и рейсами
CREATE TABLE JourneyBooking(
    journey_id INT REFERENCES Journey,
    booking_id INT REFERENCES Booking);
```

Марсофлот

покупка билетов

```
-- Параметры: _user_id = идентификатор пользователя системы
-- Создаем резервирование
INSERT INTO Booking(refcode, payer)
    VALUES ('F23ML9', _user_id) RETURNING id;
-- Заполняем позиции
PERFORM AddBookingPos(_booking_id, 'Дедка', 100);
PERFORM AddBookingPos(_booking_id, 'Внучка', 50);
PERFORM AddBookingPos(_booking_id, 'Жучка', 10);
-- Записываем факт резервирования
PERFORM CheckCapacityAndDoBook(_journey_id, _booking_id);

-- СЧАСТЛИВОГО ПОЛЁТА!
```

Что может пойти не так?

Марсофлот

что может пойти не так?

- Пользователь может отменить резервирование

Марсофлот

что может пойти не так?

- Пользователь может отменить резервирование
- Может отключиться электричество

Марсофлот

что может пойти не так?

- Пользователь может отменить резервирование
- Может отключиться электричество
- Все места могут расхватать

Марсофлот

что может пойти не так?

- Пользователь может отменить резервирование
- Может отключиться электричество
- Все места могут расхватать
- У пользователя может не хватить средств

Пожелания к системе

Если сложная операция прервалась из-за

- явного указания пользователя
- аварийного случая
- нарушения ограничений этой операцией
- действий других операций других пользователей

то операция должна быть отменена целиком.
Частичное применение её действий не допускается.

ACID транзакции

СУБД, поддерживающие ACID транзакции гарантируют

ACID транзакции

СУБД, поддерживающие ACID транзакции гарантируют

- A – атомарность, atomicity

ACID транзакции

СУБД, поддерживающие ACID транзакции гарантируют

- A – атомарность, atomicity
- C – согласованность, consistency

ACID транзакции

СУБД, поддерживающие ACID транзакции гарантируют

- A – атомарность, atomicity
- C – согласованность, consistency
- I – изолированность, isolation

ACID транзакции

СУБД, поддерживающие ACID транзакции гарантируют

- A – атомарность, atomicity
- C – согласованность, consistency
- I – изолированность, isolation
- D – долговечность, durability

Управление транзакцией

```
BEGIN;
INSERT INTO Booking(refcode, payer)
    VALUES ('F23ML9', _user_id) RETURNING id;
PERFORM AddBookingPos(_booking_id, 'Дедка', 100);
PERFORM AddBookingPos(_booking_id, 'Внучка', 50);
PERFORM AddBookingPos(_booking_id, 'Жучка', 10);
PERFORM CheckCapacityAndDoBook(_journey_id, _booking_id);
-- ROLLBACK; <= это же откатит все изменения!

COMMIT;
-- СЧАСТЛИВОГО ПОЛЁТА!
```

Уровни изоляции транзакций

Уровни изоляции транзакций

- При параллельном выполнении возникают проблемы разного рода
- Уровень изоляции определяет, каких проблем точно не случится
- Программист может выбрать уровень изоляции для транзакции

Потерянных обновлений не бывает

```
id  |  booked
1       10
```

Транзакция T_1

```
UPDATE Journey
SET booked = booked + 2
WHERE id=1;
```

Транзакция T_2

```
UPDATE Journey
SET booked = booked + 3
WHERE id=1;
```

```
id  |  booked
1       15
```

Это не потерянное обновление

хотя казалось бы

```
id  |  booked
1       10
```

Транзакция T_1

```
SELECT booked INTO _booked
FROM Journey
WHERE id=1;
_booked = _booked + 3;
UPDATE Journey
SET booked = _booked
WHERE id=1;
```

Транзакция T_2

```
SELECT booked INTO _booked
FROM Journey
WHERE id=1;
_booked = _booked + 2;
UPDATE Journey
SET booked = _booked
WHERE id=1;
```

```
id  |  booked
1       12
```

Проблема: грязное чтение

dirty read, чтение неподтверждёных данных

Транзакция T_1

```
-- Коля Герасимов и Алиса
BEGIN;

SELECT booked INTO _booked
FROM Journey
WHERE id=1;
UPDATE Journey
SET booked = _booked + 2
WHERE id=1;



-- Ой, а как же Вертер!
ROLLBACK;
```

Транзакция T_2

```
-- Артур Дент
BEGIN TRANSACTION
ISOLATION LEVEL
READ UNCOMMITTED;



SELECT booked INTO _booked
FROM Journey
WHERE id=1;
-- Хнык, уже все забукано
ROLLBACK;
```

Проблема: грязное чтение

dirty read, чтение неподтверждёных данных

Транзакция T_1

```
-- Коля Герасимов и Алиса
BEGIN;

SELECT booked INTO _booked
FROM Journey
WHERE id=1;
UPDATE Journey
SET booked = _booked + 2
WHERE id=1;




-- Ой, а как же Вертер!
ROLLBACK;
```

Транзакция T_2

```
-- Скрипт на сайте
BEGIN TRANSACTION
ISOLATION LEVEL
READ UNCOMMITTED;




SELECT id
FROM Journey
WHERE booked >= capacity - 1;
-- Эй, поторопитесь,
-- на рейс id прямо сейчас
-- покупают последние билеты!
```

Явное указание уровня изоляции

```
-- PostgreSQL
BEGIN TRANSACTION ISOLATION LEVEL
    READ UNCOMMITTED;
-- или
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

-- Microsoft SQL Server
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN TRANSACTION;

-- MySQL
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN WORK;
```

READ UNCOMMITTED

- гарантирует отсутствие потерянных обновлений
- допускает грязное чтение

READ COMMITTED

чтение подтверждённых данных

- гарантирует отсутствие грязного чтения
- транзакция читает данные, являющиеся подтверждёнными на момент начала очередного оператора
- уровень по умолчанию во многих СУБД

SELECT FOR UPDATE

- UPDATE блокирует строку от записи другими транзакциями
- SELECT не блокирует и не блокируется
- SELECT ... FOR UPDATE – это SELECT, который блокирует и блокируется как UPDATE

Compare-and-Swap

CAS

- CAS(mem, old, new):
  `if mem == old then mem = new else throw Exception`
- CAS – атомарная операция
- CAS своими руками:

```
UPDATE Journey SET booked=_new_value
WHERE id=1 AND booked=_old_value
```

Проблема: неповторямое чтение

non-repeatable read

Транзакция T_1

```
-- Юбилейный пассажир!
BEGIN;

IF 100500 = TotalBooked()
THEN




  -- ПОЗДРАВЛЯЕМ!
  PERFORM SendEmail(
      GetWinnerName(),
      TotalBooked());
END IF;
```

Транзакция T_2

```
-- Артур Дент отказывается
-- от полета
BEGIN;
-- какие-то действия

UPDATE Journey
SET booked = booked-1
WHERE id=1;
-- еще действия
COMMIT;
```

Внезапно

From: info@marsoflot.earth
To: kolya.gerasimov1984@govorunmail.com

Дорогой Коля Герасимов, поздравляем тебя, ты стал нашим 100499 пассажиром!

REPEATABLE READ

что гарантируется

- гарантирует отсутствие грязного чтения
- гарантирует что транзакция в любой момент прочитает одни и те же данные

REPEATABLE READ

что НЕ гарантируется

- что данные на самом деле останутся неизменными
- что не появится новых данных, попадающих под условия выборки (фантомное чтение)

REPEATABLE READ

дополнительные гарантии

- в PostgreSQL транзакция читает данные, являющиеся подтверждёнными на момент начала транзакции
- в других СУБД это может быть не так

Расписание транзакций

- *Расписание* - последовательность выполнения действий транзакций
- В действительности в расписании действия разных транзакций перемешаны

Последовательное расписание

- Расписание, в котором действия разных транзакций не перемешаны

Ожидаемые гарантии

- У меня есть n транзакций и $n!$ последовательных расписаний
- Любое из них даст мне согласованный результат

Ожидаемые гарантии

- У меня есть n транзакций и $n!$ последовательных расписаний
- Любое из них даст мне согласованный результат
- Планировщик делает непоследовательное расписание? Ok.
- Ну пусть его результат совпадет с каким-то из $n!$ последовательных. Мне всё равно с каким.

Сериализуемое расписание

- Расписание, эквивалентное какому-то последовательному

СУБД старается

REPEATABLE READ

Транзакция T_1

```
BEGIN ISOLATION LEVEL
REPEATABLE READ;
SELECT credit FROM UserAct
WHERE id=1;



UPDATE UserAct
SET credit = credit + 100
WHERE id=1;
COMMIT;
```

Транзакция T_2

```
--
--
--
BEGIN ISOLATION LEVEL
REPEATABLE READ;
UPDATE UserAct
SET credit = credit*2
WHERE id=1;
COMMIT
```

Наблюдаемый порядок выполнения

apparent execution order

Транзакция T_1

```
BEGIN ISOLATION LEVEL
REPEATABLE READ;
SELECT credit FROM UserAct
WHERE id=1;
UPDATE UserAct
SET credit = credit + 100
WHERE id=1;
COMMIT;
```

Транзакция T_2

```
--
--
--
--
--
--
--
BEGIN ISOLATION LEVEL
REPEATABLE READ;
INSERT INTO Something
SELECT credit*2 FROM UserAct
WHERE id=1;
COMMIT
```

$$T_1 \rightarrow T_2$$

Наблюдаемый порядок выполнения

apparent execution order

Транзакция T_1

```
BEGIN ISOLATION LEVEL
REPEATABLE READ;
SELECT credit FROM UserAct
WHERE id=1;


UPDATE UserAct
SET credit = credit + 100
WHERE id=1;
COMMIT;
```

Транзакция T_2

```
--
--
--
--
BEGIN ISOLATION LEVEL
REPEATABLE READ;
INSERT INTO Something
SELECT credit*2 FROM UserAct
WHERE id=1;


COMMIT;
```

$$T_2 \to T_1$$

Три транзакции

первая

```
-- Транзакция T1 добавляет запись в BookingPos
-- если у пользователя достаточно средств
BEGIN;
SELECT credit INTO _credit
FROM UserAct WHERE id=1;

IF _credit > 100 THEN
  INSERT INTO BookingPos(booking_id, name, payed)
      VALUES(1, 'Коля Герасимов', 100);
END IF;
COMMIT;
```

Три транзакции

вторая

```
-- Транзакция T2 изменяет кредит пользователя
BEGIN;
UPDATE UserAct SET credit = credit - 100
WHERE id=1;
COMMIT;
```

Три транзакции

третья

```
-- Транзакция Т3 начисляет бонусные мили
BEGIN;
UPDATE UserAct SET miles = credit*0.01 + 0.1 * (
  SELECT SUM(payed)
  FROM BookingPos BP JOIN Booking B
      ON BP.booking_id = B.id
  WHERE B.payer = 1
);
COMMIT;
```

Три транзакции

последовательность запуска

- Транзакции T_1 и T_2 запускаются одновременно
- Транзакция T_3 запускается после подтверждения транзакции T_2

Три транзакции

действительное расписание

1. T_1 начинается и делает всё кроме COMMIT
2. T_2 начинается и подтверждается
3. T_3 начинается и делает UPDATE

Уровень изоляции SERIALIZABLE

- Гарантирует, что результат будет эквивалентен какому-то последовательному выполнению
- Добавляет некоторые накладные расходы

Влияние на производительность

- Падение производительности может произойти из-за ожидания блокировок или из-за более высокого процента откатов транзакций
- В реализации PostgreSQL (predicate locks) потери невелики по сравнению с другими реализациями
- Результат сравнения с другими уровнями зависит от сценария использования
  - в некоторых сценариях SERIALIZABLE повышает производительность
  - в некоторых ухудшает на 2-5%
  - возможно где-то падение будет более значительным

Рекомендации разработчиков

- Явно отмечать read-only транзакции

  ```
  BEGIN ISOLATION LEVEL SERIALIZABLE READ ONLY;
  ...
  ```

- Делать транзакции короткими
- Делать в одной транзакции только то, что требуется для поддержания целостности
- Не использовать явные или неявные блокировки на уровне SERIALIZABLE
- Помнить, что верить результатам даже read-only транзакции можно только если она подтвердилась

Уровни изоляции транзакций

Транзакции DIY

```
-- генератор номеров транзакций
CREATE sequence TxnId;

-- Коллекция
CREATE TABLE Collection(id INT PRIMARY KEY,
    name TEXT,
    committed_version INT,
    uncommitted_version INT);

-- Элемент коллекции
CREATE TABLE Item(id INT PRIMARY KEY,
    collection_id INT,
    name TEXT,
    version INT);
```

Процедура начала транзакции

```
SELECT committed_version FROM Collection
    WHERE id=1 AND
          uncommitted_version IS NULL;
SELECT nextval('TxnId'); -- =1

UPDATE Collection SET uncommitted_version=1
    WHERE id=1 AND
          committed_version=0 AND
          uncommitted_version IS NULL;
```

Во время транзакции

WRITE транзакция

```
INSERT INTO Item(
  id, collection_id,
  name, version)
VALUES (1002, 1, 'Item2', 1);
```

Конкурентные транзакции

```
-- Читает из той же коллекции
SELECT committed_version
FROM Collection WHERE id=1;

SELECT * FROM Item
WHERE collection_id=1
AND version=0;

-- Обновляет элемент
-- другой коллекции
SELECT nextval('TxnId'); -- =2
UPDATE Item
SET name='', version=2
WHERE id=2001 AND version=0;
```

Процедура подтверждения транзакции

```
-- Проверяем наличие изменений
SELECT FROM Item
WHERE collection_id=1 AND version>1;


UPDATE Collection
SET committed_version=1,
    uncommitted_version=NULL
WHERE id=1 AND uncommitted_version=1;
```

Что нужно

- Атомарное изменение строки
- Генератор монотонно возрастающих номеров
- Версионирование строк

Что нужно запомнить

о транзакциях вообще

- Аббревиатуру ACID
- Система с ACID транзакциями гарантирует
  - атомарность, согласованность, изолированность и долговечность

Что нужно запомнить

о транзакциях вообще

- Уровень изоляции можно выбирать
  - A и D не обсуждаются, I и C можно варьировать

Что нужно запомнить

о транзакциях вообще

- Выше уровень изоляции $\Rightarrow$ более надежные гарантии и больше откатившихся транзакций

Что нужно запомнить

об уровнях изоляции

- READ COMMITTED + SELECT FOR UPDATE
  - транзакция читает только подтверждённые данные
  - достаточный уровень для многих приложений
  - хорошо работает при активной конкуренции

Что нужно запомнить

об уровнях изоляции

- REPEATABLE READ/SNAPSHOT ISOLATION даёт транзакции согласованный снимок
  - не блокирует и не откатывает читающие транзакции
  - хорошо работает при низкой конкуренции

Что нужно запомнить

об уровнях изоляции

- SERIALIZABLE даёт гарантию сериализуемости
  - гарантии даются только транзакциям, использующим SERIALIZABLE
  - может откатывать читающие транзакции
  - производительность зависит от сценариев использования

Что нужно запомнить

об уровнях изоляции

- Используешь REPEATABLE READ/SERIALIZABLE?
  - будь готов повторять транзакции

Что нужно запомнить

об уровнях изоляции

- Используешь REPEATABLE READ/SERIALIZABLE?
  - будь готов повторять транзакции
  - впрочем, забудь. Просто всегда будь готов.

Что нужно запомнить

о системах без ACID транзакций

- Не используй их без нужды

Что нужно запомнить

о системах без ACID транзакций

- Если есть атомарное обновление строки или compare-and-swap то можно сделать application-level транзакции