Домашнее задание № 10

Кокорин Илья, M3439

2 декабря 2019 г.

Весь код писался и тестировался в PostgreSQL 12.1 on x86_64-apple-darwin18.7.0, compiled by Apple clang version 11.0.0 (clang-1100.0.33.12), 64-bit

1 Для каждой хранимой процедуры из предыдущего домашнего задания выберите минимальный допустимый уровенить изоляции транзакций (с обоснованием)

1.1 FreeSeats

Операцию реализуем с использованием одного селекта из таблиц Flights, Seats и Reservations. Минимально допустимый уровень изоляции - Read Commited.

1. Нам его хватает, так как нам не мешает наличие иномалий «неповторяемое чтение» и «фантомная запись», так как мы делаем только одно чтение, а эти аномалии проявляются только при повторном чтении. Наличие аномалии «косая запись» нам тоже не мешает, так как мы ничего не пишем в этом запросе.

2. Уровня Read Uncommited нам не хватает по следующей причне. Рассмотрим следующий вариант исполнения: пользователь A захотел купить билет на рейс, на котром осталось одно свободное место. Это место в параллельной транзакции пытается купить пользователь B. Транзакция пользователя B уже сделала запись в Reservation, на упала, так как у пользователя B недостаточно средств на счете. В итоге транзакция пользователя A увидит это незафиксированное изменение и скажет пользователю, что на рейсе не осталось свободных мест. В итоге это место не будет куплено ни пользователем A, ни пользователем B, и мы потеряли клиента. Аналогичный результат мы получим, если мест будет больше одного, но и упавших транзакций будет больше одной.

Максимально допустимый уровень - Serializable.

1.2 Reserve

Схема работы: проверить существование места и возможность исполнения операции с ним (в том числе проверяем отсутствие записи с данными flight_id и seat_no в таблице Reservations или её существование, но там должна быть невыкупленная истёкшая бронь), если все условия истинны, добавить новую запись в таблицу Reservations. Если же запись в таблице Reservations уже сущесвует, но при этом эта бронь не выкуплена и у неё истёк срок, перезаписываем её данные с помощью UPDATE.

Минимально допустимый уровень - Serializable. Мы хотим избавиться от аномалии «фантомная запись», так как мы хотим гарантировать, что между проверкой отсутствия записи в таблице Reservation и вставкой новой брони не будет вставлена в таблицу другая бронь на то же место и тот же рейс.

Максимально допустимый уровень - Serializable.

1.3 ExtendReservation

Схема работы: проверить существование записи в таблице Reservations с данными flight_id, user_id и seat_no, возможность продление брони для него (не закрытие продажи билетов на рейс, невыкупленность этой брони, не истечение брони и т.д.). Если такая бронь есть, обновить её timestamp.

Минимально допустимый уровень изоляции - Read Commited.

1. Нам его хватает, так как нам не мешает наличие иномалий «неповторяемое чтение» и «фантомная запись», так как мы делаем только одно чтение, а эти аномалии проявляются только при повторном чтении. Наличие аномалии «косая запись» нам тоже не мешает, так как мы пишем только в одну строку, однозначно определяемую аргументами flight_id иseat_no.

2. Уровня Read Uncommited нам не хватает, так как это пишущая транзакция.

Максимально допустимый уровень - Serializable.

1.4 BuyFree

Схема работы: проверить существование места и возможность исполнения операции с ним (в том числе проверяем отсутствие записи с данными flight_id и seat_no в таблице Reservations или её существование, но там должна быть невыкупленная истёкшая бронь), если все условия истинны, добавить новую запись в таблицу Reservations. Если же запись в таблице Reservations уже сущесвует, но при этом эта бронь не выкуплена и у неё истёк срок, перезаписываем её данные с помощью UPDATE.

Минимально допустимый уровень - Serializable. Мы хотим избавиться от аномалии «фантомная запись», так как мы хотим гарантировать, что между проверкой отсутствия записи в таблице Reservation и вставкой новой брони не будет вставлена в таблицу другая бронь на то же место и тот же рейс.

Максимально допустимый уровень - Serializable.

1.5 BuyReserved

Схема работы: проверить существование записи в таблице Reservations с данными flight_id, user_id и seat_no, возможность выкупа брони для него (не закрытие продажи билетов на рейс, невыкупленность этой брони, не истечение брони и т.д.). Если такая бронь есть, обновить её timestamp и поставить is_bought = TRUE.

Минимально допустимый уровень изоляции - Read Commited.

1. Нам его хватает, так как нам не мешает наличие иномалий «неповторяемое чтение» и «фантомная запись», так как мы делаем только одно чтение, а эти аномалии проявляются только при повторном чтении. Наличие аномалии «косая запись» нам тоже не мешает, так как мы пишем только в одну строку, однозначно определяемую аргументами flight_id иseat_no.

2. Уровня Read Uncommited нам не хватает, так как это пишущая транзакция.

Максимально допустимый уровень - Serializable.

1.6 FlightStatistics

Схема работы: сделать один SELECT, группируя по flight_id и используя функции типа sum и count для подсчёта статистики.

Минимально допустимый уровень изоляции - Read Uncommited. Нам его хватает, так как нам не мешает наличие иномалий «неповторяемое чтение» и «фантомная запись», так как мы делаем только одно чтение, а эти аномалии проявляются только при повторном чтении. Наличие аномалии «косая запись» нам тоже не мешает, так как мы ничего не пишем в этом запросе. Наличие грязного чтения нам так же не мешает, так как мы собираем обобщённую статистику. Также, этот запрос, скорее всего, будет долгим, поэтому мы не хотим брать блокировки, чтобы не мешать остальным запросам исполняться в это время.

Максимально допустимый уровень - Serializable.

1.7 FlightStat

Схема работы: как в прошлый раз, но при выборке из таблицы Flights фильтровать по flight_id.

Минимально допустимый уровень изоляции - Read Uncommited, максимально допустимый - Serializable. Рассужения аналогичны предыдущему пункту.

1.8 CompressSeats

Сложная работа с курсорами, минимальный и максимальный уровни изоляции - Serializable.

2 Реализуйте сценарий работы

Напишем псевдокод

```
async def user_booking(flight_id):
        # получаем логин и пароль из текущей пользовательской сессии
        user_id, user_password = get_user_credentials(session)

        sql = 'SELECT check_credentials({0}, {1};'.format(user_id, user_password)

        if get_single_bool_from_sql_function(sql):
                sql = 'SELECT free_seats({0});'.format(flight_id)
                # функция get_seats_list_from_sql выполняет SQL-запрос
                # и получает список свободных мест из отношения-результата
                free_seats = get_seats_list_from_sql(sql)
                # rander_seats преобразовывает список мест в формат,
                # удобный для обображения пользователю (например, JSON)
                rendered_seats = render_seats(free_seats)
                # send_rendered_data_to_user посылает отрендеренный список мест на клиент
                # для отображения пользователю
                await send_rendered_data_to_user(rendered_seats)

                # get_number_from_user и get_action_type_from_user получают с клиента
                # ввод пользователя: какое место и что он с ним хочет сделать
                # (купить или забронировать)
                seat_number = await get_number_from_user()
                action_type = await get_action_type_from_user()
                if action_type == 'RESERVE':
                        sql = 'SELECT reserve({0}, {1}, {2}, {3})'\
                                .format(user_id, user_password, flight_id, seat_number)
                else:
                        sql = 'SELECT buy_free({0}, {1}, {2}, {3})'\
                                .format(user_id, user_password, flight_id, seat_number)

                try:
                        # execute_sql_and_get_boolean_result исполняет SQL-код и возвращает результат:
                        #единственный boolean
                        if execute_sql_and_get_boolean_result(sql):
                                await send_rendered_data_to_user(
                                        'Операция завершилась успешно')
                        else:
                                await send_rendered_data_to_user(
                                        'Не удалось завершить операцию')
                except TransactionException:
                        await send_rendered_data_to_user(
                                'Не удалось завершить операцию')
```

3 Реализуйте отмену всех броней на заданном рейсе

```
CREATE OR REPLACE PROCEDURE remove_all_bookings(flight_id_arg INT) AS
$$
BEGIN
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
    DELETE
    FROM Reservation
    WHERE flight_id = flight_id_arg
      AND NOT is_bought;
END;
$$
    LANGUAGE plpgsql;
```