|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |  |
|  | | | |  | | |  | | |
|  | ФГАОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | | |  |
|  | | | |  | | |  | | |
|  | | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №7  ПО ДИСЦИПЛИНЕ БД И СУБД  по теме:  «Транзакции» | | | | |  | |
|  | | | |  | | |  | | |
|  | | Работу выполнил  Студент гр. МФТИ-1,  2 курс  Бешляга Г.К.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_ 2023 г. | | |  | Проверил  ст. преподаватель  кафедры МОВС  Постаногов И.С.  \_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г. | | |  |
|  | | | | Пермь 2023 | | |  | | |

# 1. Постановка задачи

Спроектировать базу данных и продемонстрировать проблемы параллельного доступа к данным. Скрипты при их параллельном запуске должны демонстрировать проблемы уровней изоляции PostgreSQL, а также решение этих проблем путём повышения уровня изоляции до минимально необходимого.

Кроме того, продемонстрировать использование точек сохранения.

# 2. Определение транзакции

Транзакция -  группа последовательных операций с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще, и тогда она не должна произвести никакого эффекта.

Требования к транзакционным системам принято описывать в терминах свойств ACID — по первым буквам английских названий: атомарность (atomicity), согласованность (consistency), изоляция (isolation), долговечность (durability). Все эти свойства требуют дополнительных пояснений.

Атомарность означает, что любая транзакция должна быть либо выполнена полностью, либо, если по каким-либо причинам завершение транзакции невозможно, ее частичное выполнение не должно оставлять никаких следов ни в базе данных, ни в результатах работы приложений. Это требование накладывает ограничения как на СУБД (которая должна устранять последствия неполного выполнения транзакций), так и на приложения, которые не должны выводить какие-либо результаты, зависящие от не полностью выполненных транзакций.

Требование согласованности означает, что транзакция должна переводить базу данных из одного согласованного состояния в другое согласованное состояние.

Требование изоляции означает, что СУБД должна обеспечить выполнение без помех со стороны других транзакций — ограничение, входящее в определение транзакции. Нарушение изоляции транзакций может приводить к появлению некорректных результатов и состояний базы данных. Подобные ситуации называются аномалиями конкурентного выполнения. Для предотвращения всех возможных аномалий необходима полная изоляция транзакций, однако требование изоляции может вступать в противоречие с требованием высокой пропускной способности, поэтому довольно часто используются ослабленные условия изоляции.

Свойство долговечности предъявляет очень сильные и трудно реализуемые требования к СУБД. Оно означает, что никакие изменения, выполненные завершенными транзакциями, не могут быть потеряны, что бы ни происходило с сервером базы данных или вычислительной системой, на которой этот сервер работает. Таким образом, это требование, по существу, определяет отказоустойчивость базы данных. Конечно, это требование вовсе не означает, что данные, записанные транзакцией, не могут быть изменены другими транзакциями. Возможно, следующая транзакция изменит их через миллисекунды, однако она в своей работе будет учитывать результаты работы завершенных транзакций.

Из свойства атомарности следует, что любая транзакция может завершиться одним из двух способов.

* Нормальное завершение транзакции называется фиксацией (commit). Операция фиксации выполняется приложением для того, чтобы сообщить СУБД, что все операции транзакции выполнены.
* Невозможность полного выполнения приводит к необходимости обрыва транзакции (abort).

# 3. Алгоритм выполнения

Для начала, создадим в БД таблицу people со следующей структурой:

|  |  |
| --- | --- |
| Имя столбца | Тип данных |
| person\_id | int |
| name | text |
| balance | decimal(10,2) |

Таблица 1.1 – структура таблицы people.

Которая для демонстрации первых трех аномалий будет иметь следующее наполнение:

|  |  |  |
| --- | --- | --- |
| Person\_id | name | balance |
| 1 | mikhail | 0 |
| 2 | pavel | 100 |

Если несколько пользователей одновременно работают с базой данных, то СУБД должна не только гарантировать правильное выполнение каждой отдельной транзакции и восстанавливать согласованное состояние базы данных после сбоев, но также обеспечивать правильную параллельную работу всех пользователей с одними и теми же данными. Это означает, что СУБД должна решать проблемы, которые возникают при одновременной обработке транзакций, чтобы избежать конфликтов и сохранить целостность данных.

Рассмотрим более подробно проблемы, которые возникают при параллельной обработке транзакций:

1. Грязное чтение

Важное уточнение: PostgreSQL не допускает исправляет такие проблемы автоматически, поэтому данную проблему продемонстрировать на практике не предоставляется возможным

Чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится).

Имеются две транзакции, открытые различными скриптами, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| Транзакция 1 | Транзакция 2 |
| BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED  UPDATE people SET balance = 7 WHERE name = 'mikhail'; |  |
|  | BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED  SELECT balance FROM people WHERE name = 'mikhail'; |
| ROLLBACK; |  |

В транзакции 1 изменяется значение поля balance, а затем в транзакции 2 выбирается значение этого поля. После этого происходит откат транзакции 1. В результате значение, полученное второй транзакцией, будет отличаться от значения, хранимого в базе данных.

Результат выборки во второй транзакции:

|  |
| --- |
| balance |
| 7 |

Результат выборки во второй транзакции на уровне изоляции READ COMMITED:

|  |
| --- |
| balance |
| 0 |

1. Неповторяющееся чтение

Ситуация, когда при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными.

Имеются две транзакции, открытые различными скриптами, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| Транзакция 1 | Транзакция 2 |
| BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;  SELECT balance FROM people where name = 'mikhail'; |  |
|  | BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;  UPDATE people SET balance = -1000 WHERE name = 'mikhail';  COMMIT; |
| SELECT balance FROM people where name 'mikhail'; |  |

В транзакции 1 выбирается значение поля balance, затем в транзакции 2 изменяется значение поля balance. При повторной попытке выбора значения из поля balance в транзакции 1 будет получен другой результат. Эта ситуация особенно неприемлема, когда данные считываются с целью их частичного изменения и обратной записи в базу данных.

Результат первой выборки в первой транзакции:

|  |
| --- |
| balance |
| 0 |

Результат второй выборки в первой транзакции:

|  |
| --- |
| balance |
| -1000 |

Для решения данный проблемы на PostgreSQL необходимо использовать уровень изоляции REPEATABLE READ (повторяемость чтения) вместо READ COMMITED.

Результат первой выборки во второй транзакции на уровне изоляции REPEATABLE READ:

|  |
| --- |
| balance |
| 0 |

Результат второй выборки во второй транзакции на уровне изоляции REPEATABLE READ:

|  |
| --- |
| balance |
| 0 |

1. Фантомное чтение

Ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк.

Имеются две транзакции, открытые различными скриптами, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| Транзакция 1 | Транзакция 2 |
| BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED; SELECT balance FROM people WHERE name = ‘mikhail’; |  |
|  | BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED; INSERT INTO people VALUES (3,'mikhail','1000'); COMMIT; |
| SELECT balance FROM people WHERE name = ‘mikhail’; COMMIT; |  |

В транзакции 1 выполняется SQL-оператор, использующий все значения поля balance, поле name которых будет равно ‘mikhail’. Затем в транзакции 2 выполняется вставка новой строки, приводящая к тому, что повторное выполнение SQL-оператора в транзакции 1 выдаст другой результат. Такая ситуация называется чтением фантома (фантомным чтением). От неповторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (фантомных) данных.

Результат первой выборки во второй транзакции:

|  |
| --- |
| balance |
| 0 |

Результат второй выборки во второй транзакции:

|  |
| --- |
| balance |
| 0 |
| 1000 |

Для решения данный проблемы на PostgreSQL необходимо использовать уровень изоляции REPEATABLE READ (повторяемость чтения) вместо READ COMMITED.

Результат первой выборки во второй транзакции на уровне изоляции REPEATABLE READ:

|  |
| --- |
| balance |
| 0 |

Результат второй выборки во второй транзакции на уровне изоляции REPEATABLE READ:

|  |
| --- |
| balance |
| 0 |

1. Аномалия сериализации:

Уровень SERIALIZABLE обеспечивает самую строгую изоляцию транзакций. На этом уровне моделируется последовательное выполнение всех зафиксированных транзакций, как если бы транзакции выполнялись одна за другой, последовательно, а не параллельно. Однако, как и на уровне REPEATABLE READ, на этом уровне приложения должны быть готовы повторять транзакции из-за сбоев сериализации. Фактически этот режим изоляции работает так же, как и REPEATABLE READ, только он дополнительно отслеживает условия, при которых результат параллельно выполняемых сериализуемых транзакций может не согласовываться с результатом этих же транзакций, выполняемых по очереди. Это отслеживание не привносит дополнительных препятствий для выполнения, кроме тех, что присущи режиму REPEATABLE READ, но тем не менее создаёт некоторую добавочную нагрузку, а при выявлении исключительных условий регистрируется аномалия сериализации и происходит сбой сериализации.

Имеются две транзакции, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| Транзакция 1 | Транзакция 2 |
| **BEGIN** **TRANSACTION** **ISOLATION** **LEVEL** **serializable**;  **update** people **set** balance = '1000' **where** **name** = 'pavel';  **select** pg\_sleep(5); |  |
|  | **BEGIN** **TRANSACTION** **ISOLATION** **level** **serializable**;  **update** people **set** balance = '-1000' **where** **name** = 'mikhail';  **select** \* **from** people;  **COMMIT**; |
| **SELECT \* FROM people; COMMIT;** |  |

Транзакция 1 выводит изменяет баланс pavel,

Транзакция 2 тоже изменяет баланс, но уже у Mikhail. Затем обе транзакции пытаются зафиксироваться. В случае уровня изоляции REPEATABLE READ эти транзакции зафиксируются, но полученный результат не соответствовал бы полученному порядку. В случае уровня изоляции SERIALIZABLE будет зафиксирована лишь одна транзакция, а вторая закончится откатом с сообщением:

ОШИБКА: не удалось сериализовать доступ из-за зависимостей чтения/записи между транзакциями

Это объясняется тем, что при выполнении транзакции 1 перед транзакцией 2 транзакция 1 показала бы одно состояние таблицы, а при выполнении в обратном порядке было бы другое состояние.

1. Демонстрация точек сохранения

Точка сохранения — это специальная отметка внутри транзакции, которая позволяет откатить все команды, выполненные после неё, и восстановить таким образом состояние на момент установки этой точки.

Для демонстрации создадим следующую транзакцию:

begin transaction;

savepoint moneyback;

update people set balance = balance + 1000 where name = 'mikhail';

select \* from people;

rollback to moneyback;

commit;

select \* from people;

Здесь изменяется, а затем считывается баланс для Mikhail

Таблица people до создания точки сохранения:

|  |  |  |
| --- | --- | --- |
| id | name | people |
| 1 | mikhail | 0 |
| 2 | pavel | 100 |

Таблица popalis\_dps после создания точки сохранения и выполнении операции обновления:

|  |  |  |
| --- | --- | --- |
| id | name | people |
| 1 | Mikhail | 1000 |
| 2 | pavel | 100 |

Таблица people после отката к точке сохранения.

|  |  |  |
| --- | --- | --- |
| id | name | Promili\_spirta |
| 1 | mikhail | 0 |
| 2 | pavel | 100 |

3. Особенности реализации транзакций и изоляций на SQL

Синтаксис блока транзакции:

BEGIN TRANSACTION ISOLATION LEVEL <уровень изоляции>

-- Тело транзакции

[COMMIT | ROLLBACK]

BEGIN TRANSACTION ISOLATION LEVEL начинает блок транзакции, то есть обозначает, что все операторы после команды BEGIN и до явной команды COMMIT или ROLLBACK будут выполняться в одной транзакции. Уровень изоляции определяет, какие изменения, произведенные в одной транзакции, будут видны другим транзакциям до или после их фиксации (commit).

1. READ UNCOMMITTED – на этом уровне каждая транзакция видит незафиксированные изменения другой транзакции
2. READ COMMITTED – на этом уровне транзакция может видеть только те изменения, произведенные в одной транзакции
3. REPEATABLE READ – на этом уровне транзакция может видеть только те данные, которые были зафиксированы на момент начала транзакции, и не будет видеть изменения, произведенные другими транзакциями после ее начала.
4. SERIALIZABLE – на этом уровне транзакции будут выполняться последовательно, как если бы они были выполнены одна за другой, гарантируя сохранение данных в целостности и отсутствие конфликтов между транзакциями.