|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | |  | |
|  | «Пермский государственный национальный  исследовательский университет» | | | | | | | |
|  | | | |  | | |  | |
|  | | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №7  ПО ДИСЦИПЛИНЕ БД И СУБД  по теме:  «Транзакции» | | | | |  |
|  | | | |  | | |  | |
|  | |  | | |  | Работу  Выполнил  Студент гр.ПМИ-4-20,  3 курс  Пуховкин В.И. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_»\_\_\_\_\_\_\_2023г | | |
|  | | | | Пермь 2023 | | |  | |

1. **Задание**

Для спроектированной ранее базы данных:

1. Продемонстрировать выполняемость всех основных свойств транзакций (ACID).

2. Привести примеры для всех 4 проблем параллельного доступа.

3. Решить проблемы из пункта 2 включая необходимые режимы изоляции данных.

1. **Теория и реализация**

**Транзакции:**Упорядоченное множество операций, переводящих базу данных из одного согласованного состояния в другое

**Atomicity — Атомарность**

Атомарность гарантирует, что каждая транзакция будет выполнена полностью или не будет выполнена совсем. Не допускаются промежуточные состояния.

**Consistency — Согласованность**

Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты.

**Isolation — Изолированность**

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

**Durability — Надёжность**

Если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя. Обесточилась система, произошел сбой в оборудовании? На выполненную транзакцию это не повлияет.

**Уровни изоляции:**

**Read Uncommitted** разрешает доступ к результатам выполнения еще не зафиксированных транзакций и никак не ограничивает выполнения транзакций, тем самым допуская появление любых аномалий;

**Read Committed** — результаты других транзакций становятся доступными после их фиксации, т. е. запрещается аномалия грязного чтения;

**Repeatable Read** — повторное выполнение операций поиска и выборки данных

дает такие же результаты, как первое, т. е. запрещаются аномалии грязного и нечеткого чтения;

**Serializable** требует, чтобы выполнение транзакций было эквивалентно некоторому последовательному выполнению.

**Проблемы параллельного доступа:**

**Грязное чтение -**

Грязное чтение возникает, когда вторая транзакция

выбирает строку, которая обновляется другой транзакцией. Вторая

транзакция читает данные, которые еще не были зафиксированы и могут

быть изменены транзакцией, обновляющей строку. (невозможно в postgres)

**Неповторяемое чтение -**

Неповторяемое чтение возникает, когда вторая транзакция обращается к

одной и той же строке несколько раз и каждый раз считывает разные

данные. Неповторяемое чтение похоже на грязное чтение

в том смысле, что другая транзакция изменяет данные, которые считывает

вторая транзакция. Однако при несогласованном анализе данные, считанные

второй транзакцией, были зафиксированы транзакцией, внесшей изменение.

Кроме того, несогласованный анализ предполагает многократное чтение

(два и более) одной и той же строки, и каждый раз информация изменяется

другой транзакцией; таким образом, термин неповторяющийся читается.

**Фантомные чтения -**

Фантомные чтения происходят, когда действие вставки или удаления

выполняется для строки, принадлежащей диапазону строк, считываемых

транзакцией. Первое чтение транзакцией диапазона строк показывает

строку, которая больше не существует во втором или последующем чтении в

результате удаления другой транзакцией. Точно так же второе или

последующее чтение транзакции показывает строку, которая не

существовала в исходном чтении в результате вставки другой транзакцией.

**Реализация:**

Создаем таблицу users для демонстрации проблем параллельного доступа

CREATE TABLE users

(

id int GENERATED BY DEFAULT AS IDENTITY not null PRIMARY KEY,

fio varchar not null,

bank int not null

);

1. **Тестирование**

**1)Проблема неповторяющегося чтения**

Чтобы решить проблему неповторяющегося чтения в данном примере, мы перешли на следующий уровень изоляции.

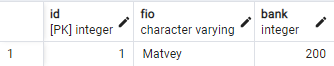
**Пример с ошибкой:**

insert into users (fio, bank) values ('Matvey', 200);

--В первой консоли

BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

select \* from users;



--Во второй консоли

/\*

BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

update users set bank=bank-10;

select \* from users;

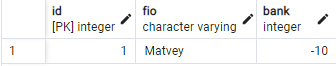


COMMIT;

\*/

update users set bank=bank-200;

select \* from users;



--Ошибка, меняем на другой уровень изоляции BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

COMMIT;

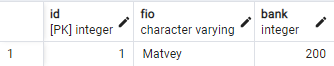
**Решение ошибки:**

insert into users (fio, bank) values ('Matvey', 200);

--В первой консоли

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

select \* from users;



--Во второй консоли

/\*

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

update users set bank=bank-10;

select \* from users;



COMMIT;

\*/

update users set bank=bank-200;

select \* from users;



--Теперь все верно, возникает ошибка, которая не дает первой консоли загнать банк в минус

COMMIT;

**2)Проблема фантомного чтения**

Чтобы решить проблему фантомного чтения в данном примере, мы перешли на следующий уровень изоляции.

**Пример с ошибкой:**

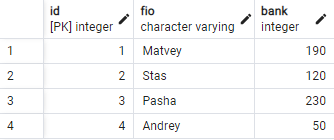
insert into users (fio, bank) values ('Stas', 120);

insert into users (fio, bank) values ('Pasha', 230);

insert into users (fio, bank) values ('Andrey', 50);

BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

select \* from users;



--Во второй консоли

/\*

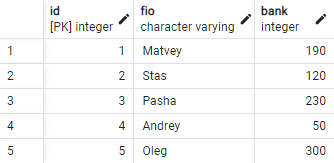
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

insert into users (fio, bank) values ('Oleg', 300);

COMMIT;

\*/

select \* from users;



--Ошибка, первая консоль получает уже другой набор данных, меняем на BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

COMMIT;

**Решение ошибки:**

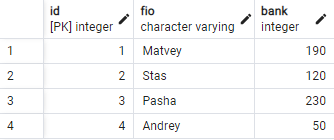
insert into users (fio, bank) values ('Stas', 120);

insert into users (fio, bank) values ('Pasha', 230);

insert into users (fio, bank) values ('Andrey', 50);

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

select \* from users;



--Во второй консоли

/\*

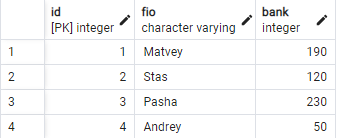
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

insert into users (fio, bank) values ('Oleg', 300);

COMMIT;

\*/

select \* from users;



--Теперь всё верно, первая консоль видит те же данные, что и в начале своей транзакции.

COMMIT;

**3)Пример с SERIALIZABLE**

SERIALIZABLE исключает любые аномалии

**Пример с ошибкой:**

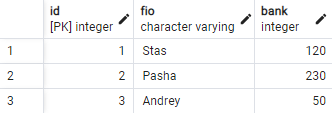
insert into users (fio, bank) values ('Stas', 120);

insert into users (fio, bank) values ('Pasha', 230);

insert into users (fio, bank) values ('Andrey', 50);

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

select \* from users;



update users set bank = bank + 15 where fio = 'Andrey';

select sum(bank) from users;

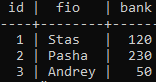


--Во второй консоли

/\*

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

select \* from users;



update users set bank = bank + 10 where fio = 'Stas';

select sum(bank) from users;



COMMIT;

\*/

select \* from users;

--Ошибка, суммы отличаются, меняем на BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE READ;

COMMIT;

**Решение ошибки:**

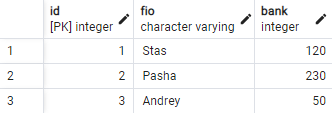
insert into users (fio, bank) values ('Stas', 120);

insert into users (fio, bank) values ('Pasha', 230);

insert into users (fio, bank) values ('Andrey', 50);

BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

select \* from users;



update users set bank = bank + 15 where fio = 'Andrey';

select sum(bank) from users;

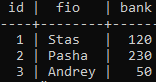


--Во второй консоли

/\*

BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

select \* from users;



update users set bank = bank + 10 where fio = 'Stas';

select sum(bank) from users;



COMMIT;

\*/

select \* from users;



--Теперь всё верно, данный уровень изоляции не дает первой транзакции завершиться

COMMIT;