|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Интеллектуальные системы информационной безопасности»

### Политики безопасности баз данных

### Практическая работа № 2

### Работа со сторонними базами данными. Построение и оптимизация

# ОТЧЁТ

# Выполнил студент группы БСБО-07-20 Любовский С.В.

# Выполнение задания 1.

1. Создайте новую базу данных в PostgreSQL включающие две таблицы: "accounts" и "transactions". Таблица "accounts" должна содержать следующие поля: id (уникальный идентификатор), name (имя), balance (баланс). Таблица "transactions" должна содержать следующие поля: id (уникальный идентификатор), account\_id (ссылка на id в таблице "accounts"), amount (сумма).

-- Создадим таблицы аккаунтов и транзацкий

CREATE TABLE accounts (

    id serial PRIMARY KEY,

    name varchar(255) NOT NULL,

    balance money NOT NULL

);

CREATE TABLE transactions (

    id serial PRIMARY KEY,

    account\_id integer NOT NULL REFERENCES accounts(id),

    amount money NOT NULL

);

1. Проведите проверку что PostgreSQL не допускается аномалия **грязного чтения**, объясните почему.

-- Создадим временную роль для тестирования

CREATE ROLE test\_role;

GRANT ALL ON ALL TABLES IN SCHEMA "public" TO test\_role;

GRANT ALL ON ALL SEQUENCES IN SCHEMA "public" TO test\_role;

SET ROLE test\_role;

-- Создадим незавершенную транзакцию

SELECT setval('accounts\_id\_seq', 1, false);

BEGIN;

INSERT INTO accounts (name, balance) VALUES ('test', 1000);

INSERT INTO transactions (account\_id, amount) VALUES (1, 100);

-- Проверим, что транзакция в процессе (результат должен быть не пустым)

SELECT txid\_current\_if\_assigned();

-- В новом окне подключимся к БД и выполним запрос из под пользователя postgres

BEGIN;

SELECT \* FROM accounts;

SELECT \* FROM transactions;

COMMIT;

Данные запросы не видят данные, созданные в другой транзакции, которая не была закончена. Это происходит, потому что PostgreSQL, как и многие другие СУБД, использует механизмы изоляции транзакций для предотвращения аномалий, включая грязное чтение.

1. Проверьте, что на уровне изоляции Read Committed не предотвращается аномалия фантомного чтения.

Проверим, что на уровне изоляции Read Committed не предотвращается аномалия фантомного чтения. Для этого создадим незавершенную транзакцию на уровне Read Committed, которая будет выводить данные таблицы «accounts».

-- Создадим транзакцию для выборки данных из таблицы accounts с уровнем изоляции READ COMMITTED

BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT \* FROM accounts;

-- В другой сессии изменяем данные в таблице accounts

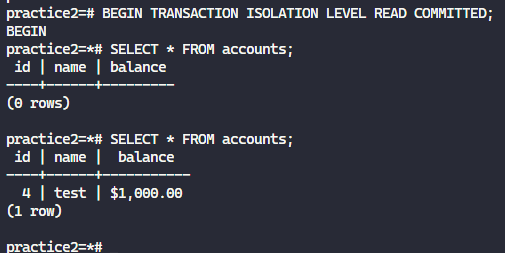
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;

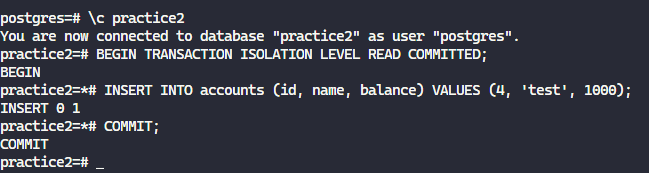
INSERT INTO accounts (id, name, balance) VALUES (4, 'test', 1000);

COMMIT;

-- В первой сессии видим изменения

SELECT \* FROM accounts;

Сессия 1:

Сессия 2:

1. Начните транзакцию с уровнем изоляции Repeatable Read (и пока не выполняйте в ней никаких команд). В другом сеансе удалите строку и зафиксируйте изменения. Видна ли строка в открытой транзакции? Что изменится, если в начале транзакции выполнить запрос, но не обращаться в нем ни к одной таблице?

Создадим две транзакции – одна будет пустой и иметь уровень изоляции repeatable read, вторая удалит строчку

--- Добавим запись в таблицу accounts

INSERT INTO accounts (id, name, balance) VALUES (100, 'test', 1000);

--- Откроем транзакцию с уровнем REPETABLE READ

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--- В другой сессии удалим строку и таблицы accounts

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

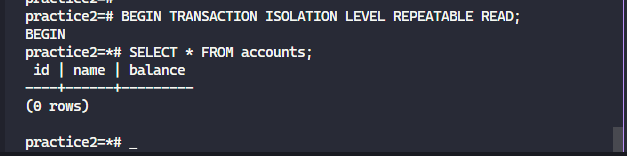
DELETE FROM accounts WHERE id = 100;

COMMIT;

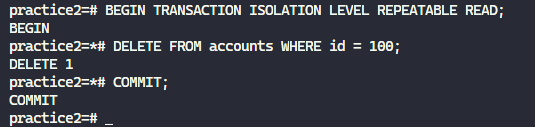
--- В первой сессии выведем таблицу accounts

SELECT \* FROM accounts;

Сессия 1:



Сессия 2:



После обращения к таблице из первой транзакции мы видим, что данных в таблице нет.

Теперь добавим в первую транзакцию запрос, не затрагивающий ни одну другую таблицу.

--- Снова добавим запись в таблицу accounts

INSERT INTO accounts (id, name, balance) VALUES (100, 'test', 1000);

--- Откроем транзакцию с уровнем REPETABLE READ и сделаем SELECT который не затрагивает ни одну из таблиц

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SELECT 1;

--- В другой сессии удалим строку и таблицы accounts

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

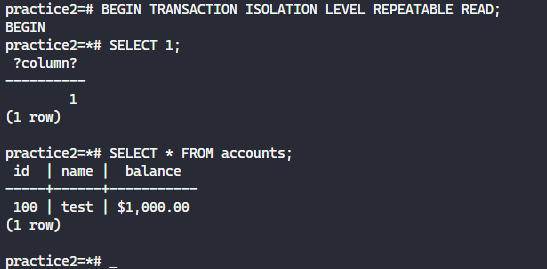
DELETE FROM accounts WHERE id = 100;

COMMIT;

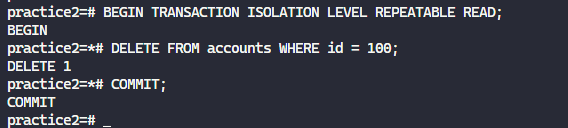
--- В первой сессии выведем таблицу accounts

SELECT \* FROM accounts;

Сессия 1:



Сессия 2:



В этот раз данные остались видны в первой транзакции.

1. Напишите функцию, которая позволяет выполнить перевод средств с одного счета на другой, используя транзакции. Функция должна использовать уровень изоляции транзакции "Serializable". Протестируйте функцию с использованием нескольких параллельных сеансов, чтобы убедиться, что переводы не могут быть выполнены дважды.

--- Создадим функции для перевода денег с одного счета на другой

CREATE OR REPLACE FUNCTION transfer\_money(

    p\_from\_acc int,

    p\_to\_acc int,

    p\_amount money

)

RETURNS void AS $$

BEGIN

    UPDATE

        accounts

    SET

        balance = balance - p\_amount

    WHERE

        id = p\_from\_acc;

    UPDATE

        accounts

    SET

        balance = balance + p\_amount

    WHERE

        id = p\_to\_acc;

    INSERT INTO

        transactions (account\_id, amount)

    VALUES

        (p\_from\_acc, -1 \* p\_amount),

        (p\_to\_acc, p\_amount);

END;

$$ LANGUAGE plpgsql;

--- Создадим два аккаунта

INSERT INTO

    accounts (id, name, balance)

VALUES

    (10, 'Alice', 100),

    (11, 'Bob', 50);

--- Переведем 10 долларов с аккаунта Алисы на аккаунт Боба двумя паралельными транзакиями с уровнем изоляции SERIALIZABLE.

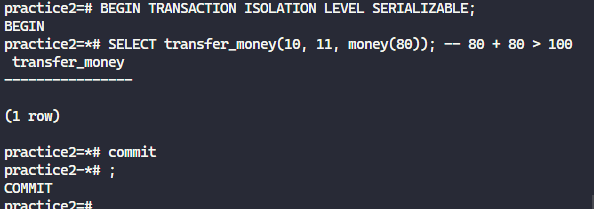
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT transfer\_money(10, 11, money(80)); -- 80 + 80 > 100

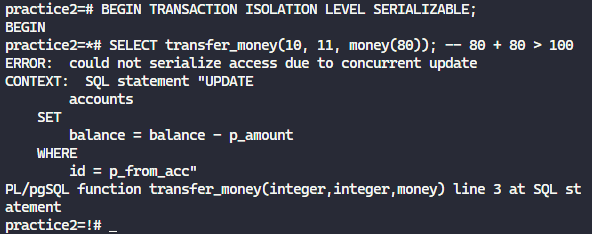
--- Попытаемся применить изменения в обеих транзакциях

COMMIT;

Сессия 1:



Сессия 2:



Мы видим, что при выполнении функции в первой транзакции вторая блокируется. После применения первой транзакции вторая упадет с ошибкой и не применится.

1. Начните транзакцию Repeatable Read и выполните какой-нибудь запрос. В другом сеансе создайте таблицу. Видно ли в первой транзакции описание таблицы в системном каталоге? Можно ли в ней прочитать строки таблицы?

Создадим транзакцию с уровнем изоляции Repeatable Read, и выполним какой-нибудь запрос. Также, в другом сеансе, создадим новую таблицу. В итоге, в первой транзакции описание таблицы в системной каталоге видно. Прочитать строки в это таблицы возможность отсутствует.

1. Убедитесь, что команда DROP TABLE транзакционна.

--- Дропнем таблицу в транзакции

BEGIN;

DROP TABLE accounts CASCADE;

--- Убедимся, что таблица была удалена

SELECT table\_name FROM information\_schema.tables

WHERE table\_schema NOT IN ('information\_schema', 'pg\_catalog')

AND table\_schema IN('public', 'myschema');

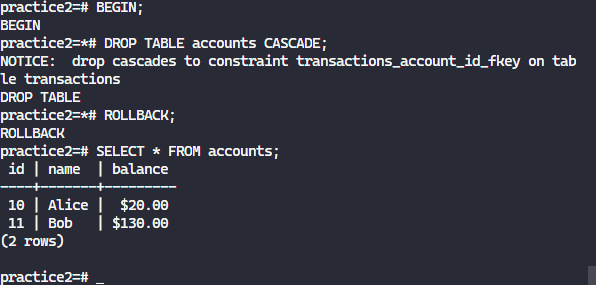
--- Откатим транзакцию

ROLLBACK;

--- Убедимся, что таблица не была удалена

SELECT \* FROM accounts;

Сессия:



Операция DROP (CASCADE) является транзакционной.

# Выполнение задания 2.

1. Установите расширение [pageinspect](https://www.postgresql.org/docs/15/pageinspect.html).
2. Создать базу данных с именем versions\_db. Создать таблицу users со следующими полями:
   1. id: уникальный идентификатор пользователя (integer, primary key, auto-increment).
   2. username: имя пользователя (varchar(255)).
   3. email: электронный адрес пользователя (varchar(255)).
   4. version: версия строки (integer).

--- Создадим базу данных versions\_db

CREATE DATABASE versions\_db;

--- Создадим таблицу users

CREATE TABLE users (

    id serial PRIMARY KEY,

    username varchar(255) NOT NULL,

    email varchar(255) NOT NULL,

    version integer NOT NULL DEFAULT 1

)

1. Создать триггер, который будет автоматически увеличивать версию строки при любом обновлении.

--- Создадим триггер, который будет автоматически увеливать версию строки при любом обновлении

CREATE OR REPLACE FUNCTION update\_version() RETURNS TRIGGER AS

$$

BEGIN

    NEW.version = OLD.version + 1;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

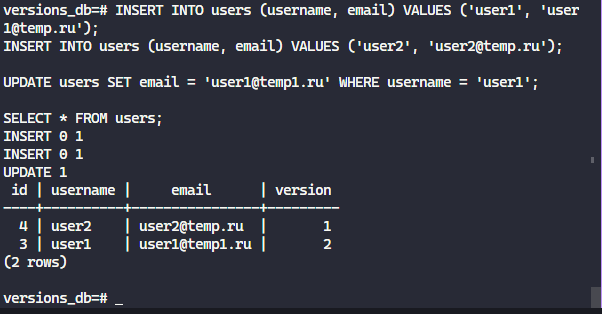
CREATE TRIGGER update\_version\_trigger

    BEFORE UPDATE ON users

    FOR EACH ROW

    EXECUTE FUNCTION update\_version();

1. Вставить в таблицу users строку с различными данными, а затем обновите.



1. При помощи следующего запроса:

**SELECT '(0,'||lp||')' AS ctid,**

**t\_xmin as xmin,**

**t\_xmax as xmax,**

**CASE WHEN (t\_infomask & 256) > 0  THEN 't' END AS xmin\_c,**

**CASE WHEN (t\_infomask & 512) > 0  THEN 't' END AS xmin\_a,**

**CASE WHEN (t\_infomask & 1024) > 0 THEN 't' END AS xmax\_c,**

**CASE WHEN (t\_infomask & 2048) > 0 THEN 't' END AS xmax\_a**

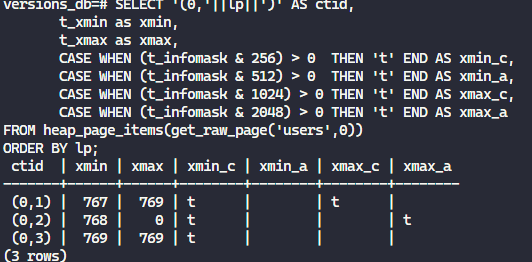
**FROM heap\_page\_items(get\_raw\_page('**users**',0))**

**ORDER BY lp;**

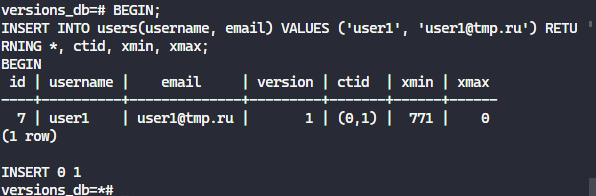
Где,

* **ctid** является ссылкой на следующую, более новую, версию той же строки. У самой новой, актуальной, версии строки ctid ссылается на саму эту версию
* **xmin** и **xmax** определяют видимость данной версии строки в терминах начального и конечного номеров транзакций.
* **xmin\_c, xmin\_a, xmax\_c, xmax\_a** содержит ряд битов, определяющих свойства данной версии

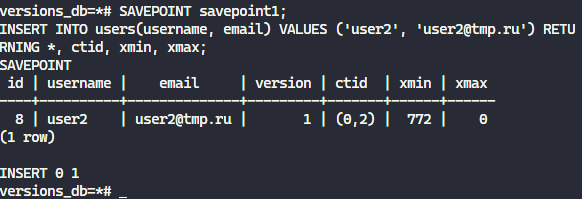
Выведите информацию о версиях строк, узнав сколько версий строк щас находится в таблице и сравнить их с атрибутом (version)



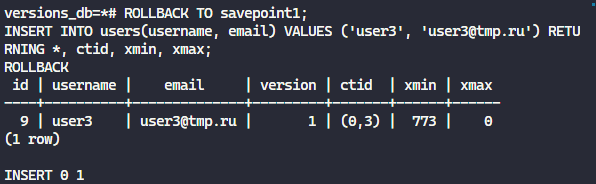
1. Опустошим таблицу при помощи **TRUNCATE;**
2. Начините транзакцию и вставьте новую строку и узнайте номер текущей транзакции (это можно сделать при помощи след команды: **INSERT INTO users(...) VALUES (...) RETURNING \*, ctid, xmin, xmax;**



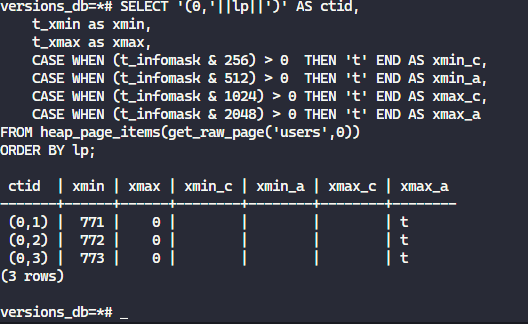
1. Поставьте точку сохранения и добавьте новую строку использовав команду из пункта 7.



1. Откатимся к точке сохранения и добавим новую строчку аналогично 7 и 8 пункту.



1. Выведем сведения о версиях строк.



# Выполнение задания 3.

1. Создать таблицу t с полями id(integer) и name (char(2000)) с параметром filfactor = 75%.

--- Создадим таблицу «t» с полями: id, name (с параметром filfactor = 75%).

CREATE TABLE t (

    id INT,

    name VARCHAR(2000)

) WITH (FILFACTOR = 75);

1. Создать индекс над полем t(name)

--- Создадим индекс над полем t(name).

CREATE INDEX t\_name\_idx ON t (name);

1. Установить расширение *pageinspect.*
2. Создать представление, которое будет включать в себя информацию о версиях строк при помощи след запроса:

**CREATE VIEW t\_v AS**

**SELECT '(0,'||lp||')' AS ctid,**

**CASE lp\_flags**

**WHEN 0 THEN 'unused'**

**WHEN 1 THEN 'normal'**

**WHEN 2 THEN 'redirect to '||lp\_off**

**WHEN 3 THEN 'dead'**

**END AS state,**

**t\_xmin || CASE**

**WHEN (t\_infomask & 256) > 0 THEN ' (c)'**

**WHEN (t\_infomask & 512) > 0 THEN ' (a)'**

**ELSE ''**

**END AS xmin,**

**t\_xmax || CASE**

**WHEN (t\_infomask & 1024) > 0 THEN ' (c)'**

**WHEN (t\_infomask & 2048) > 0 THEN ' (a)'**

**ELSE ''**

**END AS xmax,**

**CASE WHEN (t\_infomask2 & 16384) > 0 THEN 't' END AS hhu,**

**CASE WHEN (t\_infomask2 & 32768) > 0 THEN 't' END AS hot,**

**t\_ctid**

**FROM heap\_page\_items(get\_raw\_page('t',0))**

**ORDER BY lp;**

* флаг Heap Hot Updated показывает, что надо идти по цепочке ctid,
* флаг Heap Only Tuple показывает, что на данную версию строки нет ссылок из индексов.

1. Спроецировать ситуацию в таблице t, при которой произойдет внутристраничная очистка без участия HOT-обновлений.

--- Спроецируем ситуацию в таблице «t», при которой произойдет внутри страничная очистка без участия HOT-обновлений

INSERT INTO t (id, name) VALUES (1, '11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111);

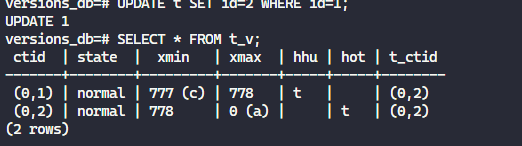
1. После воспроизвести ситуацию, но уже с HOT-обновлением

--- После воспроизведем ситуацию, но уже с HOT-обновлением.

UPDATE t SET id=2 WHERE id=1;

--- Посмотрим, что получилось.

SELECT \* FROM t\_v;



**Звездочка.**

Использовались факторы заполнения меньше стандартного значения выгодно только при большом количестве вставок в таблицу, однако при этом размер индекса будет расти значительно быстрее чем количество данных. При использовании фактора заполнения больше стандартного скорость работы индекса увеличивается, его размер уменьшается, но вставки и обновления будут занимать больше времени.