**Санкт-Петербургский политехнический университет Петра Великого**

**Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Управление транзакциями**

по дисциплине «Основы построения защищенных баз данных»

Выполнил

студент гр. 4841001/00101 *<подпись>* А.С. Петушков

Проверил

преподаватель *<подпись>* М.А. Полтавцева

Санкт-Петербург  
2022

**Цель работы**

Установить влияние уровня изоляции транзакций на возможные аномалии при многопользовательской работе с данными.

**Задачи работы**

**Часть 1. Уровни изоляции транзакций**

Разработать тестовую таблицу из двух числовых полей код – значение, первичный ключ поле «код». Заполнить таблицу записями со значением 0 (более 1 000 000 записей). Далее перед каждым прогоном инициализируйте таблицу заново.

Тест 1

1. Установите наименьший доступный уровень изоляции транзакций.
2. Создайте процесс – писатель с бесконечным циклом, внутри которого находится транзакция, меняющая значения двух случайно выбранных полей на +1 и -1 (чтобы сумма значений всей таблицы продолжала оставаться 0). После транзакции установите в цикле задержку в 100 миллисекунд.
3. Создайте процесс – читатель, в виде транзакции считающей сумму значений по всей таблице.
4. Задайте просмотр блокировок для каждого процесса.
5. Запустите процесс читатель несколько (не менее 10) раз последовательно при запущенном процессе писателе. Соберите полученные результаты и сделайте выводы.
6. Повторите тест для всех других уровней изоляции.

Тест 2

1. Измените процесс-читатель так, чтобы он проводил подсчет суммы значений в таблице, затем следовала задержка, затем снова проводился подсчет. Запускайте его на небольшом диапазоне записей (например, на 1000).
2. Разработайте процесс-писатель, который будет внутри транзакции добавлять запись в таблицу с кодом больше существующего и значением 111. После вставки перед концом транзакции установите задержку в 100 миллисекунд.
3. Задайте просмотр блокировок для каждого процесса.
4. Запустите процесс-читатель и затем, во время паузы, запустите второй процесс-писатель.
5. Проведите тест для всех уровней изоляции по 5 раз. Соберите полученные результаты и сделайте выводы.
6. Проведите тест для всех уровней изоляции по 3 раза. Соберите полученные результаты и сделайте выводы.

Тест 3

1. Оставьте прежний процесс читатель.
2. Для процесса писателя перед концом транзакции установите задержку и, после нее, откат транзакции.
3. Запустите процесс – писатель и затем, во время паузы запустите второй процесс – читатель.
4. Отрегулируйте время задержек таким образом, чтобы первый подсчет происходил во время задержки писателя, а второй уже после его отката.
5. Задайте просмотр блокировок для каждого процесса.
6. Проведите тест для всех уровней изоляции по 5 раз. Соберите полученные результаты и сделайте выводы.

Тест 4

1. Разработайте процесс – писатель, который внутри транзакции:

* подсчитывает сумму значений в таблице и записывает ее в переменную;
* содержит задержку на 3 (или более) секунды;
* проверяет условие на равенство переменной суммы 0;
* если условие выполняется, вставляет запись с кодом 1200000 и значением 111.

1. Разработайте процесс – писатель, который вставляет запись с кодом 1200000 и значением 111. После вставки перед концом транзакции установите задержку в 100 миллисекунд.
2. Запустите первый процесс – писатель и затем, во время паузы, запустите второй процесс – писатель.
3. Задайте просмотр блокировок для каждого процесса.
4. Проведите тест для всех уровней изоляции по 2 раза. Соберите полученные результаты и сделайте выводы.

**Часть 2**

* 1. Если в вашей СУБД возможны не рассмотренные аномалии по работе с данными (уточните на основе документации), разработайте дополнительные тесты на основе приведенных выше и продемонстрируйте аномалии и их устранение с повышением уровня изоляции транзакций.
  2. Разработайте программу обработки ошибок блокировки (на примере базы данных из работы 1-2). Проведите тесты программы, вызвав ситуации блокировки на основании разработанных ранее пользовательских запросов.
  3. Соберите данные о блокировках, накопленные СУБД в процессе выполнения тестов, включая последний – при возникновении взаимоблокировки. Проведите их анализ, учитывая возможные типы блокировок в СУБД и объекты блокировки.

**Ход работы**

Согласно документации СУБД PostgreSQL, уровень изоляции READ UNCOMMITTED обрабатывается как READ COMMITTED, поэтому в тестах он не будет учтен.

**Тест 1**

Тест №1 предназначен для выявления аномалии неповторяющегося чтения. В случае отсутствия данной аномалии процесс-читатель всегда получит нулевой результат суммы значений по столбцу.

В таблице 1 представлены результаты выполнения теста для всех уровней изоляции транзакций.

Таблица 1 – Результаты

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Уровень изоляции транзакций | Номер испытания | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Read Committed | -2 | 2 | 6 | 12 | -15 | 12 | -3 | 8 | 6 | -10 |
| Repeatable Read | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Serializable | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Тест выявил, что только уровни Repeatable Read и Serializable полностью исключают аномалию неповторяющегося чтения.

Код теста приведен в приложении 1.

**Тест 2**

Тест №2 предназначен для выявления аномалии фантомного чтения. В случае отсутствия данной аномалии процесс-читатель всегда получит нулевой результат суммы значений по столбцу.

В таблице 2 представлены результаты выполнения теста для всех уровней изоляции транзакций (указан только результат повторного вычисления суммы, т.к. первый всегда нулевой по условию).

Таблица 2 – Результаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уровень изоляции транзакций | Номер испытания | | | | |
| 1 | 2 | 3 | 4 | 5 |
| Read Committed | 111 | 111 | 111 | 111 | 111 |
| Repeatable Read | 0 | 0 | 0 | 0 | 0 |
| Serializable | 0 | 0 | 0 | 0 | 0 |

Тест выявил, что только уровни Repeatable Read и Serializable полностью исключают аномалию фантомного чтения, т.к. результат повторного подсчета для этих уровней не изменился.

Код теста приведен в приложении 2.

**Тест 3**

Тест №3 предназначен для выявления аномалии грязного чтения. В случае отсутствия данной аномалии процесс-читатель всегда получит нулевой результат суммы значений по столбцу.

В таблице 3 представлены результаты выполнения теста для всех уровней изоляции транзакций (указан только результат повторного вычисления суммы, т.к. первый всегда нулевой по условию).

Таблица 3 – Результаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уровень изоляции транзакций | Номер испытания | | | | |
| 1 | 2 | 3 | 4 | 5 |
| Read Committed | 0 | 0 | 0 | 0 | 0 |
| Repeatable Read | 0 | 0 | 0 | 0 | 0 |
| Serializable | 0 | 0 | 0 | 0 | 0 |

Тест выявил, что все уровни изоляции полностью исключают аномалию грязного чтения, т.к. результат повторного подсчета не изменился.

Код теста приведен в приложении 3.

**Тест 4**

Тест №3 предназначен для выявления аномалии потерянного обновления.

На рисунке 1 представлено сообщение об ошибке, полученное на каждом уровне изоляции.



Рисунок 1 – Ошибка

Таким образом, сообщение об ошибке показывает, что СУБД не позволяет нарушить целостность данных. Аномалия потерянного обновления для всех уровней отсутствует.

**Вывод**

В результате выполнения данной лабораторной работы были изучены уровни изоляции транзакций и возможные аномалии при многопользовательской работе с данными. Практическое исследование проведено на примере СУБД PostgreSQL. В данной СУБД уровень изоляции транзакций READ UNCOMMITTED не поддерживается (реализован как READ COMMITTED), поэтому не все аномалии возможно воспроизвести.

Приложение 1 – Код скрипта для теста 1

CREATE OR REPLACE PROCEDURE w1(

tmp1 int,

tmp2 int

)

AS $w1$

BEGIN

UPDATE "\_table" SET "val" = "val" + 1 WHERE "id" = tmp1;

UPDATE "\_table" SET "val" = "val" - 1 WHERE "id" = tmp2;

COMMIT;

END $w1$

LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE writer1()

AS $writer1$

DECLARE

tmp1 int;

tmp2 int;

BEGIN

WHILE TRUE

LOOP

tmp1 := random() \* 1000000;

tmp2 := random() \* 1000000;

WHILE tmp1 = tmp2

LOOP

tmp2 := random() \* 1000000;

END LOOP;

CALL w1(tmp1, tmp2);

PERFORM pg\_sleep(0.1);

END LOOP;

END $writer1$

LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE reader1()

AS $reader1$

DECLARE

i int;

j int;

s int;

tmp int;

BEGIN

i := 0;

WHILE i < 10

LOOP

s := 0;

j := 1;

WHILE j < 1000001

LOOP

SELECT "val" FROM "\_table" WHERE "id" = j INTO tmp;

s := s + tmp;

j := j + 1;

END LOOP;

COMMIT;

RAISE NOTICE '%', s;

i := i + 1;

END LOOP;

END $reader1$

LANGUAGE plpgsql;

/\* -- \*/

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level read committed;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level repeatable read;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level serializable;

CALL writer1();

CALL reader1();

UPDATE "\_table" SET "val" = 0;

Приложение 2 – Код скрипта для теста 2

CREATE OR REPLACE PROCEDURE writer2()

AS $writer2$

BEGIN

INSERT INTO "\_table" VALUES (1000999, 111);

perform pg\_sleep(0.1);

COMMIT;

END $writer2$

LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE reader2()

AS $reader2$

DECLARE

i int;

j int;

\_sum int;

tmp int;

BEGIN

\_sum := 0;

j := 1;

tmp := 0;

WHILE j < 1001001

LOOP

SELECT SUM("val") FROM "\_table" WHERE "id" >= j AND "id" < (j + 1000) INTO tmp;

IF tmp IS DISTINCT FROM NULL THEN

\_sum := \_sum + tmp;

END IF;

j := j + 1000;

END LOOP;

RAISE NOTICE '%', \_sum;

PERFORM pg\_sleep(10);

\_sum := 0;

j := 1;

tmp := 0;

WHILE j < 1001001

LOOP

SELECT SUM("val") FROM "\_table" WHERE "id" >= j AND "id" < (j + 1000) INTO tmp;

IF tmp IS DISTINCT FROM NULL THEN

\_sum := \_sum + tmp;

END IF;

j := j + 1000;

END LOOP;

RAISE NOTICE '%', \_sum;

END $reader2$

LANGUAGE plpgsql;

/\* -- \*/

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level read committed;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level repeatable read;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level serializable;

CALL writer2();

CALL reader2();

UPDATE "\_table" SET "val" = 0;

DELETE FROM "\_table" WHERE "id" = 1000999;

Приложение 3 – Код скрипта для теста 3

CREATE OR REPLACE PROCEDURE writer3()

AS $writer2$

BEGIN

INSERT INTO "\_table" VALUES (1000999, 111);

RAISE NOTICE 'pause';

perform pg\_sleep(7);

ROLLBACK;

END $writer2$

LANGUAGE plpgsql;

/\* -- \*/

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level read committed;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level repeatable read;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level serializable;

CALL writer3();

CALL reader2();

UPDATE "\_table" SET "val" = 0;

DELETE FROM "\_table" WHERE "id" = 1000999;

Приложение 4 – Код скрипта для теста 4

CREATE OR REPLACE PROCEDURE writer4\_1()

AS $writer4\_1$

DECLARE

\_sum int;

j int;

tmp int;

BEGIN

\_sum := 0;

j := 1;

tmp := 0;

SELECT SUM("val") FROM "\_table" INTO \_sum;

RAISE NOTICE '%', \_sum;

PERFORM pg\_sleep(5);

IF \_sum IS NOT DISTINCT FROM 0 THEN

INSERT INTO "\_table" VALUES (1200000, 111);

RAISE NOTICE 'INSERT 1';

END IF;

END $writer4\_1$

LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE writer4\_2()

AS $writer4\_2$

BEGIN

INSERT INTO "\_table" VALUES (1200000, 111);

RAISE NOTICE 'INSERT 2';

PERFORM pg\_sleep(0.1);

COMMIT;

END $writer4\_2$

LANGUAGE plpgsql;

/\* -- \*/

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level read committed;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level repeatable read;

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION level serializable;

CALL writer4\_1();

CALL writer4\_2();

UPDATE "\_table" SET "val" = 0;

delete from "\_table" where "id" = 1200000;

Приложение 5 – Код скрипта для инициализации

CREATE TABLE "\_table"(

"id" int NOT NULL,

"val" int NOT NULL,

CONSTRAINT "\_table\_pk" PRIMARY KEY ("id")

);

CREATE OR REPLACE PROCEDURE init()

AS $init$

DECLARE

j int;

BEGIN

j := 1;

WHILE j < 1000001

LOOP

INSERT INTO "\_table" ("id", "val") VALUES (j, 0);

j := j + 1;

END LOOP;

END;

$init$ LANGUAGE plpgsql

/\* -- \*/

CALL init();