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

Институт Информационных технологий (ИТ)

Кафедра Инструментального и прикладного программного обеспечения

(ИиППО)

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №9**

**по дисциплине**

«Технологии обработки транзакций клиент-серверных приложений»

Выполнил студент группы ИКБО-20-21 Фомичев Р.А.

Принял Маличенко С.В.

Москва 2024**ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

1. Напишите тест, проверяющий, что обработка очереди, показанная в демонстрации, работает корректно при выполнении в несколько потоков. Убедитесь, что тест не проходит, если убрать предложение FOR UPDATE SKIP LOCKED.
2. Добавьте в реализацию проверку «зависших» сообщений. Если такая ситуация будет обнаружена, зависшее сообщение должно быть снова принято в работу.
3. Вставьте в таблицу сообщений большое количество строк и проверьте, что:
   1. было обработано каждое сообщение;
   2. каждое сообщение было обработано ровно один раз.
4. Уберите из реализации секундную задержку (имитацию работы), чтобы тест выполнялся быстрее и с достаточным уровнем конкурентности между процессами.

**РЕФЕРАТ**

Отчёт 21 страница, 21 рисунок, 5 источников, 1 приложение.

POSTGRESQL, АСИНХРОННАЯ ОБРАБОТКА, БЛОКИРОВКИ, SQL ОЧЕРЕДЬ

Объектом разработки является база данных.

Цель работы – реализация асинхронных механизмов обработки транзакций для повышения эффективности системы, исследовании проблем в области транзакционной асинхронной обработки.

В процессе работы производилось изучение механизмов асинхронной обработки транзакций и исследование проблем в области транзакционной асинхронной обработки.

Результатом являются сведения по реализации асинхронных механизмов обработки транзакций.

**ВВЕДЕНИЕ**

[**ВВЕДЕНИЕ** 6](#_Toc162260460)

[1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ** 7](#_Toc162260461)

[**1.1 Выполнение задания 1** 7](#_Toc162260462)

[**1.2 Выполнение задания 2** 8](#_Toc162260463)

[**1.3 Выполнение задания 3** 10](#_Toc162260464)

[**1.4 Выполнение задания 4** 11](#_Toc162260465)

[ЗАКЛЮЧЕНИЕ 13](#_Toc162260466)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 1](#_Toc162260467)

[ПРИЛОЖЕНИЕ А 2](#_Toc162260468)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

БД – база данных;

СУБД – система управления базами данных.

**ВВЕДЕНИЕ**

В современном информационном обществе, где огромные объёмы данных являются ключевым ресурсом для бизнеса, науки и повседневной жизни, вопрос обеспечения безопасности и надёжности хранения данных становится более критическим, чем когда-либо. Одним из основных аспектов обеспечения безопасности и надёжности данных в СУБД является блокировки отношений и других объектов.

Целью данной практической работы является реализация асинхронных механизмов обработки транзакций для повышения эффективности системы, исследовании проблем в области транзакционной асинхронной обработки.

1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТ****Ы**
   1. **Выполнение задания 1**

В базе данных были созданы таблицы msg\_queue и msg\_log. Затем были созданы процедуры и функции обработки очереди. Таблица msg\_queue была заполнена записями. Заполнение таблицы представлено на рисунке 1.



Рисунок 1 – Заполнение таблицы msg\_queue

Была запущена обработка очереди в двух сессиях с включённым секундомером. Результат обработки очереди в первой сессии представлен на рисунке 2.

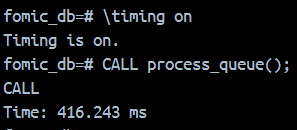


Рисунок 2 – Обработка очереди в первой сессии

Результат обработки очереди во второй сессии представлен на рисунке 3.



Рисунок 3 – Обработка очереди во второй сессии

Все сообщения были обработаны единожды. Вывод результата обработки очереди представлен на рисунке 4.

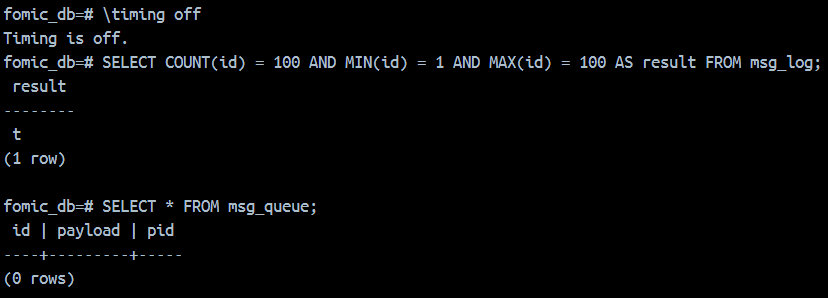


Рисунок 4 – Результат обработки очереди

Из кода обработки очереди была удалена команда FOR UPDATE SKIP LOCKED. Таблица msg\_queue была очищена и заполнена заново. Результат заполнения таблицы msg\_queue представлен на рисунке 5.



Рисунок 5 – Заполнение таблицы msg\_queue

Была запущена обработка очереди в двух сессиях. Результат обработки в первой сессии представлен на рисунке 6.



Рисунок 6 – Обработка очереди в первой сессии

Результат обработки очереди во второй сессии представлен на рисунке 7.



Рисунок 7 – Обработка очереди во второй сессии

Сообщения были обработаны повторно. Вывод результата обработки очереди представлен на рисунке 8.

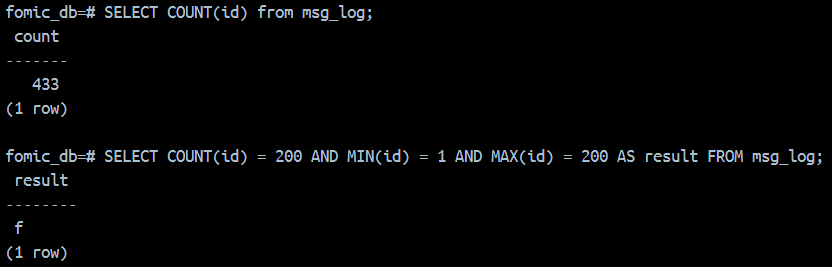


Рисунок 8 – Результат обработки очереди

* 1. **Выполнение задания 2**

Таблицы были приведены в начальное состояние. Была добавлена возможность обрабатывать сообщения, обработчик которых не действует в настоящее время. Результат заполнения таблицы msg\_queue представлен на рисунке 9.



Рисунок 9 – Заполнение таблицы msg\_queue

Была заблокирована работа обработчика очереди для появления зависшего сообщения. Результат блокировки представлен на рисунке 10.

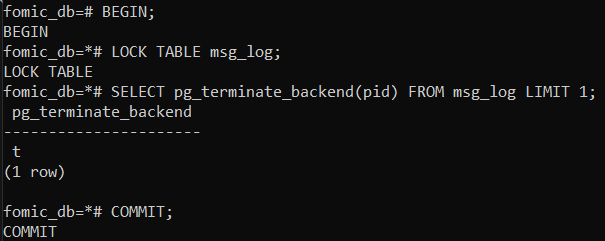


Рисунок 10 – Прерывание обработки очереди во второй сессии

Обработка была прервана с ошибкой. Сообщение об ошибке представлено на рисунке 11.

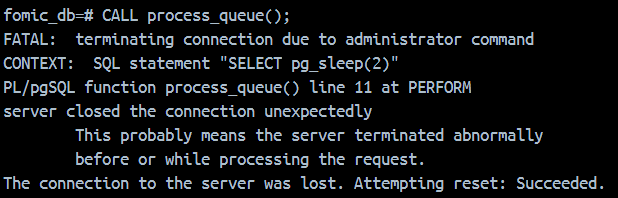


Рисунок 11 – Результат прерывания очереди

После остановки обработки появились зависшие сообщения. Результат проверки зависших сообщений представлен на рисунке 12.

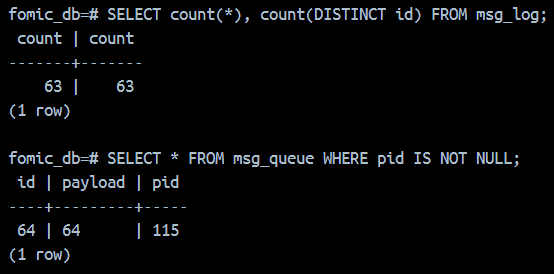


Рисунок 12 – Проверка наличия зависших сообщений

Процесс обработки был запущен заново, после чего зависшие сообщения были обработаны. Повторная обработка зависших сообщений представлена на рисунке 13.

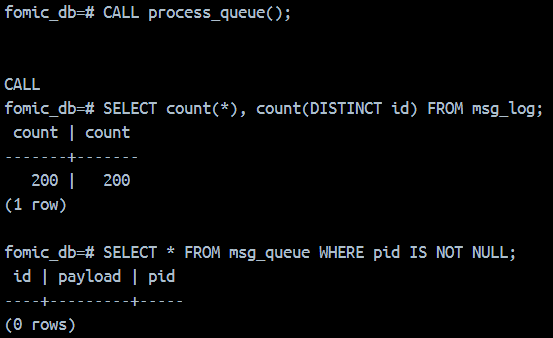


Рисунок 13 – Повторная обработка очереди

* 1. **Выполнение задания 3**

Таблица была приведена в начальное состояние. Затем была заполнена 3000 записями. Результат заполнения таблицы msg\_queue представлен на рисунке 14.

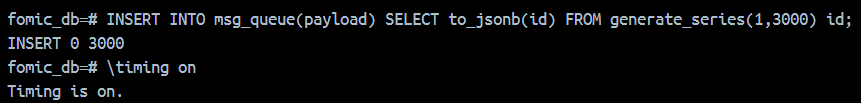


Рисунок 14 – Заполнение таблицы msg\_queue

Был начат процесс обработки с включённым секундомером в двух сессиях. Результат обработки очереди в первой сессии представлен на рисунке 15.

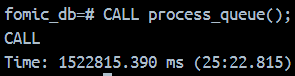


Рисунок 15 – Обработка очереди в первой сессии

Результат обработки очереди во второй сессии представлен на рисунке 16.



Рисунок 16 – Обработка очереди во второй сессии

Сообщения были обработаны единожды. Результат проверки обработки очереди представлен на рисунке 17.

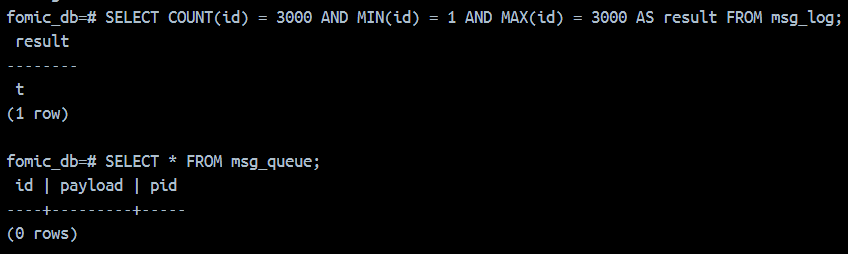


Рисунок 17 – Результат проверки обработки очереди

* 1. **Выполнение задания 4**

Таблица была приведена в начальное состояние. Была убрана задержка, имитирующая работу. Таблица msg\_queue была заполнена записями. Результат заполнения представлен на рисунке 18.



Рисунок 18 – Заполнение таблицы msg\_queue

Результат обработки очереди в первой сессии представлен на рисунке 19.

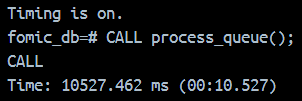


Рисунок 20 – Обработка очереди в первой сессии

Результат обработки очереди во второй сессии представлен на рисунке 20.



Рисунок 20 – Обработки очереди во второй сессии

Все сообщения были обработаны единожды. Результат проверки обработки очереди представлен на рисунке 21.

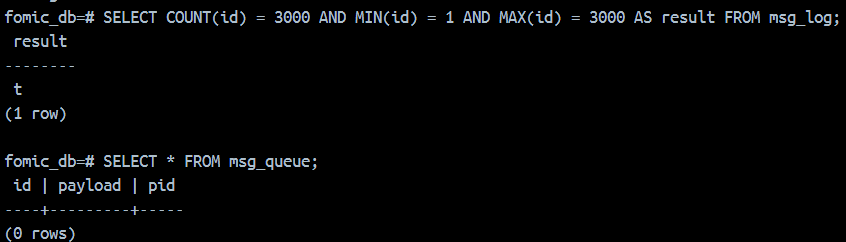


Рисунок 21 – Результат проверки обработки очереди

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были реализованы асинхронные механизмы обработки транзакций для повышения эффективности системы, исследованы проблемы в области транзакционной асинхронной обработки.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. PostgreSQL: Официальная документация [Электронный ресурс] – URL: https://www.postgresql.org/docs/ (дата обращения: 14.02.2024).
2. PostgreSQL Tutorial for Beginners [Электронный ресурс] – URL: https://www.tutorialspoint.com/postgresql/index.htm (дата обращения: 14.02.2024).
3. PostgreSQL: Википедия [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/PostgreSQL (дата обращения: 14.02.2024).
4. Лузанов П.В. и др. Postgres. Первое знакомство. [Электронный ресурс] – URL: https://postgrespro.ru/education/books/introbook (дата обращения: 22.02.2024)
5. Новиков Б. А. Лекции Основы технологий баз данных. [Электронный ресурс] – URL: https://postgrespro.ru/education/university/dbtech (дата обращения: 22.02.2024)

# ПРИЛОЖЕНИЕ А

**ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

Листинг 1 – SQL код для выполнения задания 1

-- Session 1 --

INSERT INTO msg\_queue(payload) SELECT to\_jsonb(id) FROM generate\_series(1,100) id;

\timing on

CALL process\_queue();

-- Session 2 --

CALL process\_queue();

-- Session 1 --

\timing off

SELECT COUNT(id) = 100 AND MIN(id) = 1 AND MAX(id) = 100 AS result FROM msg\_log;

SELECT \* FROM msg\_queue;

-- Session 1 --

INSERT INTO msg\_queue(payload) SELECT to\_jsonb(id) FROM generate\_series(1,200) id;

CALL process\_queue();

-- Session 2 --

CALL process\_queue();

-- Session 1 --

SELECT COUNT(id) from msg\_log;

SELECT COUNT(id) = 200 AND MIN(id) = 1 AND MAX(id) = 200 AS result FROM msg\_log;

Листинг 2 – SQL код для выполнения задания 2 практической работы

-- Session 1 --

INSERT INTO msg\_queue(payload) SELECT to\_jsonb(id) FROM generate\_series(1,200) id;

CALL process\_queue();

-- Session 2 --

BEGIN;

LOCK TABLE msg\_log;

SELECT pg\_terminate\_backend(pid) FROM msg\_log LIMIT 1;

COMMIT;

-- Session 1 --

SELECT count(\*), count(DISTINCT id) FROM msg\_log;

SELECT \* FROM msg\_queue WHERE pid IS NOT NULL;

CALL process\_queue();

SELECT count(\*), count(DISTINCT id) FROM msg\_log;

SELECT \* FROM msg\_queue WHERE pid IS NOT NULL;

Листинг 3 – SQL код для выполнения задания 3 практической работы

-- Session 1 --

INSERT INTO msg\_queue(payload) SELECT to\_jsonb(id) FROM generate\_series(1,3000) id;

\timing on

CALL process\_queue();

-- Session 2 --

CALL process\_queue();

-- Session 1 --

\timing off

SELECT COUNT(id) = 3000 AND MIN(id) = 1 AND MAX(id) = 3000 AS result FROM msg\_log;

SELECT \* FROM msg\_queue;

Листинг 4 – SQL код для выполнения задания 4 практической работы

-- Session 1 --

INSERT INTO msg\_queue(payload) SELECT to\_jsonb(id) FROM generate\_series(1,3000) id;

\timing on

CALL process\_queue();

-- Session 2 --

CALL process\_queue();

-- Session 1 --

\timing off

SELECT COUNT(id) = 3000 AND MIN(id) = 1 AND MAX(id) = 3000 AS result FROM msg\_log;

SELECT \* FROM msg\_queue;

Листинг 5 – SQL код реализации очереди для задания 1

CREATE FUNCTION take\_message(OUT msg msg\_queue) AS $$

BEGIN

SELECT \*

INTO msg

FROM msg\_queue

WHERE pid IS NULL

ORDER BY id LIMIT 1

FOR UPDATE SKIP LOCKED;

UPDATE msg\_queue

SET pid = pg\_backend\_pid()

WHERE id = msg.id;

END;

$$ LANGUAGE plpgsql VOLATILE;

CREATE FUNCTION complete\_message(msg msg\_queue) RETURNS void AS $$

DELETE FROM msg\_queue

WHERE id = msg.id;

$$ LANGUAGE sql VOLATILE;

CREATE PROCEDURE process\_queue() AS $$

DECLARE

msg msg\_queue;

BEGIN

LOOP

SELECT \* INTO msg FROM take\_message();

EXIT WHEN msg.id IS NULL;

COMMIT;

-- обработка

PERFORM pg\_sleep(1);

INSERT INTO msg\_log(id, pid) VALUES (msg.id, pg\_backend\_pid());

PERFORM complete\_message(msg);

COMMIT;

END LOOP;

END;

$$ LANGUAGE plpgsql;

Листинг 6 – SQL код реализации очереди для задания 2

CREATE FUNCTION take\_message(OUT msg msg\_queue) AS $$

BEGIN

SELECT \*

INTO msg

FROM msg\_queue

WHERE pid IS NULL

ORDER BY id LIMIT 1;

UPDATE msg\_queue

SET pid = pg\_backend\_pid()

WHERE id = msg.id;

END;

$$ LANGUAGE plpgsql VOLATILE;

CREATE FUNCTION complete\_message(msg msg\_queue) RETURNS void AS $$

DELETE FROM msg\_queue

WHERE id = msg.id;

$$ LANGUAGE sql VOLATILE;

CREATE PROCEDURE process\_queue() AS $$

DECLARE

msg msg\_queue;

BEGIN

LOOP

SELECT \* INTO msg FROM take\_message();

EXIT WHEN msg.id IS NULL;

COMMIT;

-- обработка

PERFORM pg\_sleep(1);

INSERT INTO msg\_log(id, pid) VALUES (msg.id, pg\_backend\_pid());

PERFORM complete\_message(msg);

COMMIT;

END LOOP;

END;

$$ LANGUAGE plpgsql;

Листинг 7 – SQL код реализации очереди для задания 3

CREATE OR REPLACE FUNCTION take\_message(OUT msg msg\_queue) AS $$

BEGIN

SELECT \*

INTO msg

FROM msg\_queue

WHERE pid IS NULL OR pid NOT IN (SELECT pid FROM pg\_stat\_activity)

ORDER BY id LIMIT 1

FOR UPDATE SKIP LOCKED;

UPDATE msg\_queue

SET pid = pg\_backend\_pid()

WHERE id = msg.id;

END;

$$ LANGUAGE plpgsql VOLATILE;

CREATE FUNCTION complete\_message(msg msg\_queue) RETURNS void AS $$

DELETE FROM msg\_queue

WHERE id = msg.id;

$$ LANGUAGE sql VOLATILE;

CREATE PROCEDURE process\_queue() AS $$

DECLARE

msg msg\_queue;

BEGIN

LOOP

SELECT \* INTO msg FROM take\_message();

EXIT WHEN msg.id IS NULL;

COMMIT;

-- обработка

PERFORM pg\_sleep(1);

INSERT INTO msg\_log(id, pid) VALUES (msg.id, pg\_backend\_pid());

PERFORM complete\_message(msg);

COMMIT;

END LOOP;

END;

$$ LANGUAGE plpgsql;

Листинг 8 – SQL код реализации очереди для задания 4

CREATE OR REPLACE FUNCTION take\_message(OUT msg msg\_queue) AS $$

BEGIN

SELECT \*

INTO msg

FROM msg\_queue

WHERE pid IS NULL OR pid NOT IN (SELECT pid FROM pg\_stat\_activity)

ORDER BY id LIMIT 1

FOR UPDATE SKIP LOCKED;

UPDATE msg\_queue

SET pid = pg\_backend\_pid()

WHERE id = msg.id;

END;

$$ LANGUAGE plpgsql VOLATILE;

CREATE FUNCTION complete\_message(msg msg\_queue) RETURNS void AS $$

DELETE FROM msg\_queue

WHERE id = msg.id;

$$ LANGUAGE sql VOLATILE;

CREATE PROCEDURE process\_queue() AS $$

DECLARE

msg msg\_queue;

BEGIN

LOOP

SELECT \* INTO msg FROM take\_message();

EXIT WHEN msg.id IS NULL;

COMMIT;

-- обработка

INSERT INTO msg\_log(id, pid) VALUES (msg.id, pg\_backend\_pid());

PERFORM complete\_message(msg);

COMMIT;

END LOOP;

END;

$$ LANGUAGE plpgsql;