|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 Сидоров С.Д.

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 страниц, 22 рисунка, 5 источников, 1 приложения.

POSTGRESQL, АСИНХРОННАЯ ОБРАБОТКА, ОЧЕРЕДЬ, БЛОКИРОВКИ, SQL

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](#_Toc2266)

[1. ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ 7](#_Toc15883)

[1.1 Выполнение задания 1 7](#_Toc12379)

[1.2 Выполнение задания 2 8](#_Toc14576)

[1.3 Выполнение задания 3 10](#_Toc273)

[1.4 Выполнение задания 4 11](#_Toc16832)

[ЗАКЛЮЧЕНИЕ 13](#_Toc14449)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc24146)

[ПРИЛОЖЕНИЕ А 15](#_Toc29255)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

БД – база данных;

СУБД – система управления базами данных.

**ВВЕДЕНИЕ**

В современном мире, где данные играют важнейшую роль в различных сферах, от бизнеса до повседневной жизни, вопрос обеспечения высокой скорости обработки большого объёма данных является одним из ключевых. Для решения данной задачи используется асинхронная обработка данных с распределением по разным вычислительным машинам с использованием механизма очередей.

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**

Перед началом выполнения задания была создана база данных exec\_async содержащая таблицы msg\_queue и msg\_log. Для выполнения первого задания были созданы процедуры и функции для обработки очереди, представленные в листинге 5 приложения А, а также была заполнена msg\_queue сотней сообщений для обработки, что представлено на рисунке 1.



Рисунок 1 - Заполнение таблицы сообщений

После была запущена обработка очереди в терминале 1 с включённым секундомером и в терминале 2, что представлено на рисунках 2 - 3.

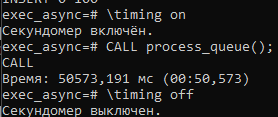


Рисунок 2 - Обработка очереди в первом терминале



Рисунок 3 - Обработка очереди во втором терминале

В результате обработки с помощью специальных SQL запросов, представленных на рисунке 4, были получены сведения о результатах обработки, которые показывают, что были единожды обработаны все сообщения, а также не осталось сообщений в очереди.

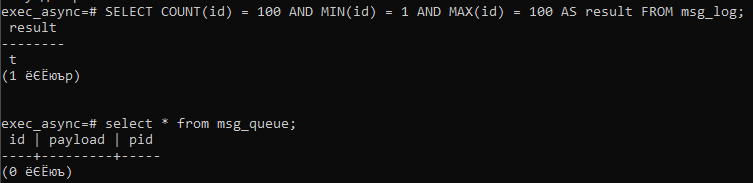


Рисунок 4 - Результат обработки очереди

Для проверки работоспособности очереди без команды FOR UPDATE SKIP LOCKED, код обработки очереди был изменён на представленный в листинге 6 приложения А. Перед выполнением все таблицы были восстановлены до изначального состояния, а также заново была заполнена таблица сообщений, что представлено на рисунке 5.



Рисунок 5 - заполнение таблицы сообщений

После заполнения была повторно запущена обработка очереди в двух терминалах, что представлено на рисунках 6 - 7.



Рисунок 6 - Обработка очереди в первом терминале



Рисунок 7 - Обработка очереди во втором терминале

В результате, после обработки видно, что часть сообщений была обработана повторно, что представлено на рисунке 8.

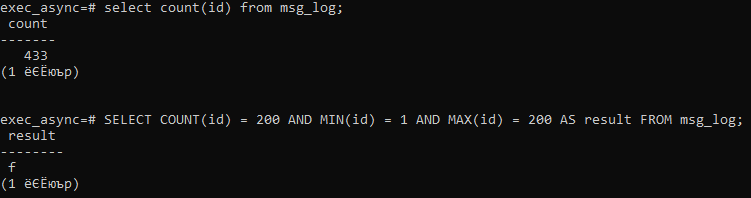


Рисунок 8 - Результат обработки очереди

* 1. **Выполнение задания 2**

Перед выполнением задания 2 база данных была приведена в изначальное состояния и код обработки очереди был изменён на SQL код, представленный в листинге 7 приложения А. Была добавлена возможность обрабатывать сообщения, обработчик которых не действует в настоящее время. Перед проверкой была заполнена таблица msg\_queue сотней строк, что показано на рисунке 9. А также был запущен процесс обработки очереди, что показано на рисунке 10.



Рисунок 9 - Заполнение таблицы сообщений



Рисунок 10 - Запуск обработки очереди в первом терминале

Для появления зависшего сообщения, из другого терминала была заблокирована работа обработчика очереди, что представлено на рисунке 11.

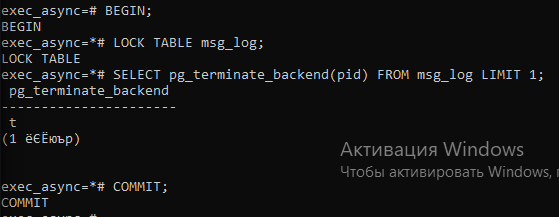


Рисунок 11 - Прерывание обработки очереди в терминале 2

В результате обработка была прервана с ошибкой, представленной на рисунке 12.

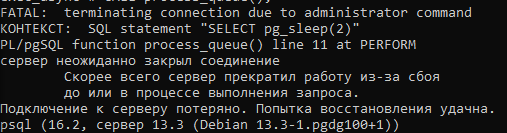


Рисунок 12 - Результат прерывания очереди

После остановки обработки появились зависшие сообщения, что представлено на рисунке 13.

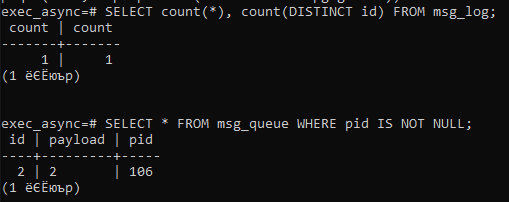


Рисунок 13 - Проверка наличия зависших сообщений

Так как данная реализация очередь обладает возможностью обработки зависших сообщений, процесс обработки был запущен заново, после чего зависшее сообщение также было обработано что видно на рисунке 14.

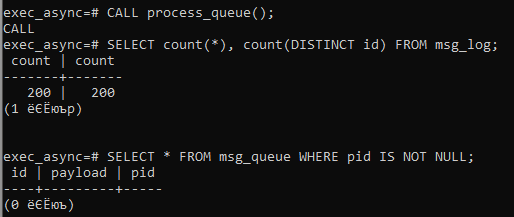


Рисунок 14 - Результат повторного вызова обработки очереди

* 1. **Выполнение задания 3**

Перед выполнением задания 3, база данных была приведена к изначальному состоянию. При выполнении задания использовалась реализация очереди представленная в листинге 7 приложения А. Для проверки работоспособности обработчика очереди на большом количестве данных, таблица msg\_queue была заполнена 3000 записями, что представлено на рисунке 15.



Рисунок 15 - Заполнение очереди большим количеством сообщений

После заполнения был начат процесс обработки с включённым секундомером в двух различных терминалах, что представлено на рисунках 16 -17.

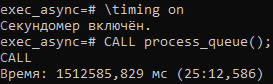


Рисунок 16 - Обработка очереди в первом терминале



Рисунок 17 - Обработка очереди во втором терминале

В результате обработки очереди были единожды обработаны все сообщения, что представлено на рисунке 18.

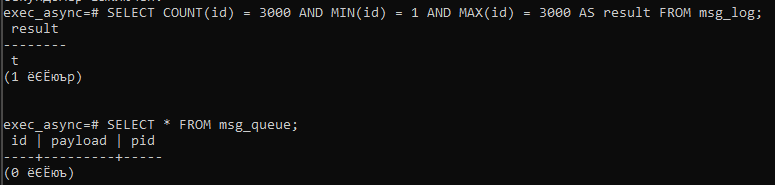


Рисунок 18 - Результат обработки очереди

* 1. **Выполнение задания 4**

Перед выполнением задания база данных была приведена в изначальное состояние. В качестве обработчика очереди использовались процедуры и функции представленные в листинге 8 приложения А, в которых была убрана строка ожидания 1 секунды для уменьшения задержки и увеличения конкурентности между процессами. Перед тестированием в таблицу msg\_queue было внесено 3000 записей, что представлено на рисунке 19.



Рисунок 19 - Заполнение таблицы сообщений большим количеством сообщений

После заполнения таблицы был начат процесс обработки в двух терминалах, что представлено на рисунках 20 - 21.

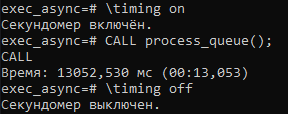


Рисунок 20 - Обработки очереди в первом терминале



Рисунок 21 - Обработки очереди во втором терминале

В результате обработки были единожды обработаны все сообщения, что представлено на рисунке 22.

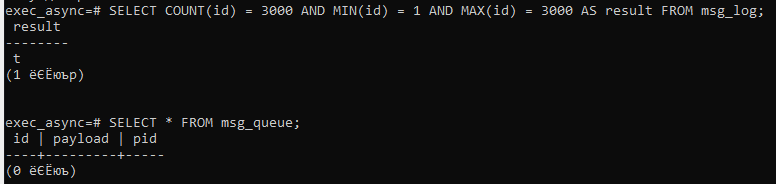


Рисунок 22 - Результат обработки очереди

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были получены практические навыки по реализации асинхронных механизмов обработки транзакций для повышения эффективности системы, а также знания о проблемах 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

--Terminal 1

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

\timing on

CALL process\_queue();

--Terminal 2

CALL process\_queue();

--Terminal 1

\timing off

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

SELECT \* FROM msg\_queue;

--Terminal 1

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

CALL process\_queue();

--Terminal 2

CALL process\_queue();

--Terminal 1

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

--Terminal 1

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

CALL process\_queue();

--Terminal 2

BEGIN;

LOCK TABLE msg\_log;

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

COMMIT;

--Terminal 1

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

CALL process\_queue();

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

Листинг 3 - SQL код для выполнения задания 3 практической работы

--Terminal 1

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

\timing on

CALL process\_queue();

--Terminal 2

CALL process\_queue();

--Terminal 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

--Terminal 1

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

\timing on

CALL process\_queue();

--Terminal 2

CALL process\_queue();

--Terminal 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 первичной реализации очереди

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 без FOR UPDATE SKIP LOCKED

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

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 с обработкой зависших сообщений

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(2);

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 без задержки

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;