Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация существующих методов параллельного выполнения запросов к СУБД PostgreSQL в пределах одного соединения»

Студент __ИУ7-75Б__ _____ ____О. С. Платонова__
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ ____М. В. Филиппов__
(Подпись, дата) (И.О.Фамилия)

Консультант _____ ____Ю. М. Гаврилова__
(Подпись, дата) (И.О.Фамилия)

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
_____ И. В. Рудаков
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме классификация существующих методов параллельного выполнения запросов к _____

СУБД PostgreSQL в пределах одного соединения._____

Студент группы ИУ7-75Б_____

_____Платонова Ольга Сергеевна_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

_____учебная_____

Источник тематики (кафедра, предприятие, НИР) ___НИР_____

График выполнения НИР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Техническое задание *Необходимо выполнить анализ существующих СУБД и привести аргументацию выбора PostgreSQL, изучить основные положения архитектуры PostgreSQL, реализации соединения; рассмотреть существующие методы и алгоритмы реализации, а также выполнить сравнительный анализ исследуемых методов._____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 15-25 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Презентация на 8-10 слайдах.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель НИР _____ М. В. Филиппов
(Подпись, дата) (И.О.Фамилия)

Консультант НИР _____ Ю. М. Гаврилова
(Подпись, дата) (И.О.Фамилия)

Студент _____ О. С. Платонова
(Подпись, дата) (И.О.Фамилия)

РЕФЕРАТ

Расчетно-пояснительная записка 15 страниц, 1 таблица, 6 рисунков, 18 источников.

БАЗА ДАННЫХ, POSTGRESQL, МНОГОПОТОЧНЫЕ СУБД

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

База данных (БД) — собрание данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, причем такое собрание данных, которое поддерживает одну или более областей применения [1].

Система управления базой данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [2].

Массивно-параллельная архитектура (massive parallel processing, MPP) — класс архитектур параллельных вычислительных систем. Главная особенность такой архитектуры состоит в том, что память физически разделена [3].

Введение

В XXI веке человечество владеет невообразимым объемом данных. Знания, передававшиеся из поколения в поколение в течение многих тысячелетий, продолжают увеличиваться каждый день. Так, ежегодный прирост информации составляет 30% [4].

С появлением письменности, будь то шумерские таблички или берестяные грамоты, перед человечеством возникает вопрос хранения и обработки данных. Причем с развитием цивилизации, и, как следствие, увеличением документооборота, проблема хранения информации требует систематического решения. Например, в конце XX века данные крупной компании могли занимать несколько этажей, что требовало дополнительных кадров для работы с ними.

Первым этапом решения этого вопроса стало внедрение компьютеров. Многие операции с данными были упрощены, а быстрый рост информационных технологий привел к увеличению скорости работы над данными. Однако хранение информации в виде файлов на одном компьютере стало неэффективным. Во-первых, поиск файла в файловой системе был долгим. Во-вторых, хранение информации в одном файле затрудняло поиск необходимых данных.

Решение проблемы разрозненного хранения данных впервые было представлено на симпозиуме в 1964 году в Санта-Монике [5]. Хотя речь шла о внедрении баз данных в военные приложения, этот момент считается точкой отсчета истории базы данных. Их применение в работе компаний привело к увеличению скорости работы. А автоматизация основных процессов базы данных, таких как создание, просмотр, удаление данных привело к созданию системы управления базы данных.

В 2021 году ни одна сфера жизни не обходится без компьютеризации. Организации используют базы и СУБД для перевода данных в электронный вид. Необходимость перевода заключается не столько в потребности сократить временные и материальные (сокращение кадров) расходы, сколько в

поддержании конкурентоспособности. Переход компании в электронный вид дает возможность приобретения принципиально новых качеств, позволяющих иметь существенные преимущества над другими.

Из-за высокой популярности СУБД возникает вопрос об оптимизации ее работы. Так как один из самых распространенных способов увеличения производительности — параллельное выполнение, следует рассмотреть оптимизацию многопоточной программы [6]. Поскольку операция соединения с базой данных является одной из самых дорогостоящих, следует минимизировать количество соединений.

Целью данной работы является разработка и реализация метода параллельного выполнения запросов к СУБД PostgreSQL в пределах одного соединения. Для достижения поставленной цели необходимо решить следующие задачи:

1. Анализ предметной области и существующих методов реализации многопоточного доступа в MPP системах;
2. Разработка метода параллельного выполнения запросов к СУБД PostgreSQL в пределах одного соединения;
3. Реализация программного модуля для СУБД PostgreSQL;
4. Проведение сравнительного анализа стандартного метода обработки запросов к СУБД PostgreSQL с реализуемым методом.

1 Анализ предметной области

В данном разделе будет выполнен анализ существующих СУБД и приведена аргументация выбора PostgreSQL, рассмотрены основные положения архитектуры. Также будут описаны инструменты соединения в PostgreSQL и обоснована реализация многопоточности.

1.1 Анализ СУБД

В таблице 1.1 представлен рейтинг популярности СУБД, составленный компанией «DB-Engines» по состоянию на конец 2021 года [7].

Таблица 1 — Рейтинг популярности СУБД.

Рейтинг	**СУБД**	**Модель БД**
1.	Oracle	Реляционная
2.	MySQL	Реляционная
3.	Microsoft SQL Server	Реляционная
4.	PostgreSQL	Реляционная
5.	MongoDB	Документная
6.	Redis	«Ключ-значение»
7.	IBM Db2	Реляционная
8.	Elasticsearch	Поисковая система
9.	SQLite	Реляционная
10.	Cassandra	Широкие столбцы

Согласно рейтингу, лидирующие позиции занимают реляционные модели баз данных. Данная работа будет основываться на объектно-реляционной СУБД PostgreSQL 12-ой версии, занимающей 4-ое место в рейтинге [7]. Выбор аргументирован такими преимуществами как доступность исходного кода и кроссплатформенность.

1.2 Архитектура PostgreSQL

Одной из наиболее сильных сторон PostgreSQL является архитектура, основанная на модели «клиент-сервер». Выделяют 3 основные подсистемы: клиентская часть, серверная часть и хранилище данных (рисунок 1.1) [8].

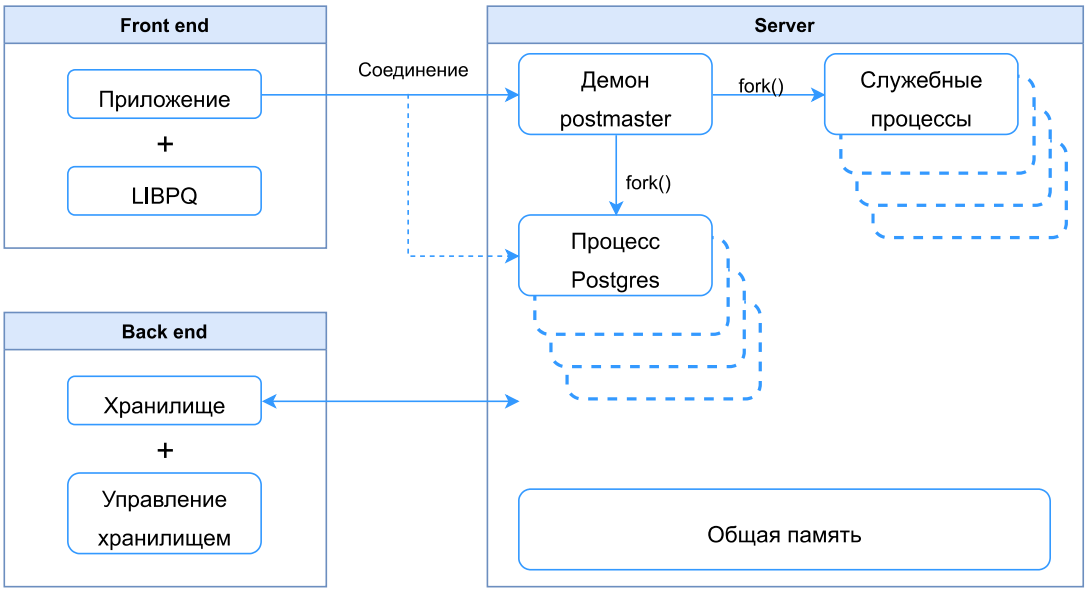

Рисунок 1.1 — «Клиент-серверная» архитектура PostgreSQL.

Представленная концепция взаимодействия позволяет исключать произвольный доступ клиентов к данным, тем самым поддерживая их целостность и повышая уровень безопасности. Также она дает возможность создания распределенных систем.

1.2.1 Клиентская часть

Клиентская часть состоит из пользовательского приложения и библиотеки LIBPQ. Данная библиотека содержит набор функций, с помощью которых клиентские программы могут передавать запросы серверу PostgreSQL и принимать результаты этих запросов [9].

1.2.2 Серверная часть

Серверная часть, включающая в себя процесс-демон *postmaster*, серверные и служебные процессы, выполняет обработку запросов. Соединение, установленное клиентом, принимается демоном *postmaster*, который в дальнейшем с помощью системного вызова *fork()* создаст новый серверный процесс для обслуживания соединения данного клиента.

Между собой серверные процессы взаимодействуют с помощью семафоров и разделяемой памяти. Для всех серверных процессов доступна общая память, большую часть которой занимает буферный кэш. Он необходим для ускорения работы с данными на диске, обращение к которым реализовано через операционную систему [10]. Описанные механизмы позволяют обеспечить целостность данных при одновременном обращении к ним [9].

Помимо создания серверного процесса процесс-демон также порождает ряд служебных процессов.

1.2.3 Хранилище данных

Третья часть сформирована из хранилища данных и средств его управления. Хранение данных на диске организовано с помощью табличных пространств, указывающих расположение данных. Допускается обращение нескольких серверных процессов к информации хранилища одновременно. На рисунке 1.2 представлена организация хранилища [10].

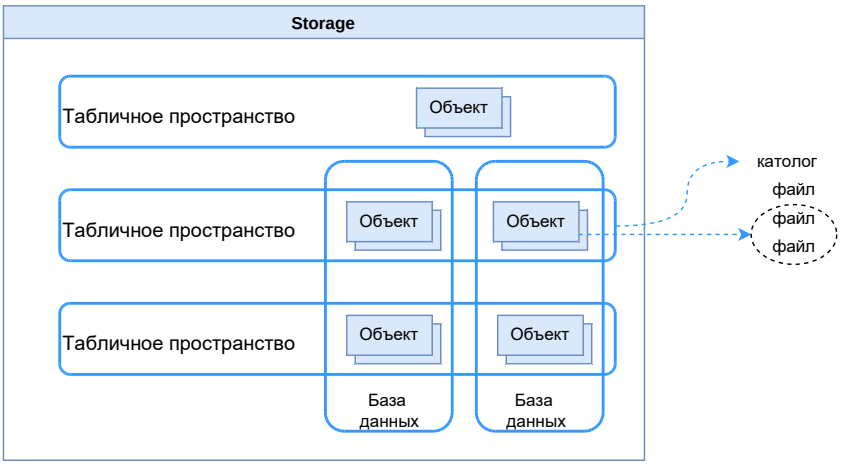

Рисунок 1.2. – Организация хранилища PostgreSQL.

1.3 Соединение в PostgreSQL

Как было рассмотрено выше, пользователь устанавливает соединение и посылает его серверному процессу *postmaster*. Каждое соединение представляется объектом PGconn, который можно получить от функций PQconnectdb, PQconnectdbParams или PQsetdbLogin.

PostgreSQL содержит инструменты для реализации многопоточности. Один из них — библиотека libpq, которая по умолчанию поддерживает повторные вызовы. Однако при реализации многопоточности существует ограничение: «два потока не должны пытаться одновременно работать с одним объектом PGconn. В частности, не допускается параллельное выполнение команд из разных потоков через один объект соединения.» [9].

1.4 Многопоточность

Поскольку на сегодняшний день конкуренция происходит в области затрат и скорости, работа многих приложений основана на многопоточности, которая рассматривается как один из способов увеличения производительности.

С ростом объема БД наблюдается преимущество по времени многопоточной реализации. Так, при работе с базой данных, состоящей из 100000 записей, время выполнения запросов примерно в 1000 раз выше у

однопоточной программы. Также однопоточная модель показывает нестабильную работу на больших данных (ошибка OutOfMemory). На рисунке 1.3 приведены результаты сравнения работы однопоточной и многопоточной программ на больших данных [11].

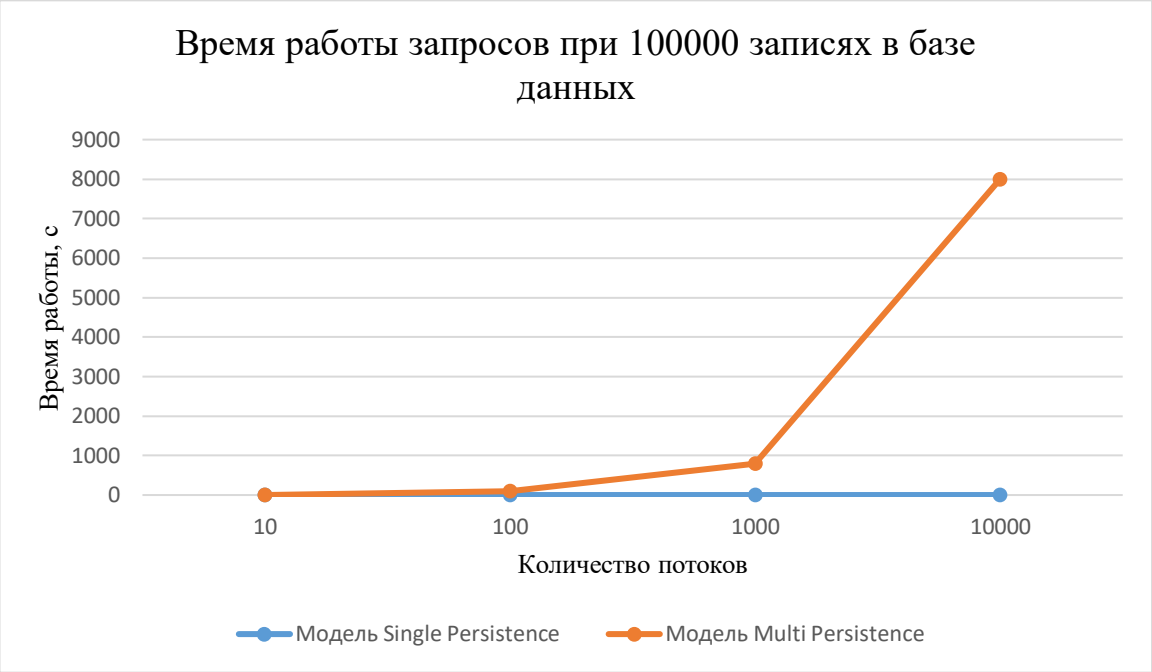

Рисунок 1.2 — Сравнение однопоточной и многопоточной программ, выполняющих запросы чтения БД.

Однако многопоточная реализация имеет свои недостатки. Многопоточные приложения, использующие PostgreSQL, вынуждены открывать новое соединение в каждом потоке.

Поскольку операция подключения — одна из самых дорогостоящих (процесса подключения к БД занимает от 2 до 3 МБ [12]), рост количества потоков может привести к замедлению работы программы: повышенная нагрузка на системные ресурсы и значительное снижение производительности, особенно на многоядерных системах. Увеличение конкуренции при обращении множества процессов к ресурсам PostgreSQL также способно замедлять работу программы.

Открытие соединений на разных потоках может привести к проблеме превышения количества подключений на сервере, что приведет к долгому ожиданию дальнейших запросов или их отклонению.

2 Классификация существующих решений

В данном разделе будут представлены существующие методы решения поставленной задачи. Также будет выполнен сравнительный анализ решений с указанием достоинств и недостатков.

2.1 Пул соединений

Объектный пул представляет из себя набор инициализированных и готовых к использованию объектов. При необходимости системы обратиться к объекту, вместо его создания будет взят экземпляр из пула. Особенно заметно повышение производительности, когда стоимость и скорость инициализации экземпляра высоки, а количество одновременно используемых объектов в любой момент времени является низким [13].

Из-за того, что PostgreSQL не имеет встроенного пула подключений [9], большинство клиентских программ вынуждены реализовывать свой собственный.

2.1.1 Frontend pool

При инициализации пула выполняется установка необходимого количества соединений. Предельный размер определяется пользователем в зависимости от контекста задачи. После успешной инициализации из пула может быть извлечено свободное соединение, для выполнения необходимых запросов к БД. После выполнения запросов соединение должно быть возвращено в пул. Если соединение было закрыто, его следует удалить из пула, и вместо него создать новое [14].

Преимущество данного метода наглядно демонстрирует следующий пример. Устанавливается соединение для 10 клиентов, каждый из которых выполняет 10.000 запросов в БД. Если в среднем выполняется 486 транзакций в секунду, то реализация пула соединений (размером 25) позволяет увеличить это значение примерно на 60% – до 566 транзакций в секунду [15].

Пул соединений имеет несколько недостатков, один из которых заключается в ограничении максимального количества одновременных подключений к БД. В зависимости от реализации, пользователь может задать размер пула, а также количество соединений, которое может быть добавлено в пул. К другому существенному недостатку следует отнести сложность реализации, а также встраиваемость кода (особенно в крупных компаниях). Также следует обратить внимание на расчет следующих параметров: минимальное количество соединений, максимальное количество пулов соединений, максимальное время простоя, время ожидания соединения, количество попыток после тайм-аута. От корректной конфигурации пула зависит то, насколько увеличится пропускная способность транзакции.

2.1.2 Server pool

Хотя PostgreSQL не имеет встроенного пула подключений, он был реализован в коммерческой системе Postgres Pro Enterprise. Это объектно-реляционная СУБД, разработанная Postgres Professional в рамках проекта Postgres Pro на основе PostgreSQL [16]. В отличие от внешнего, встроенный пул не требует дополнительного обслуживания и не налагает на клиента никаких ограничений.

Работа встроенного пула аналогична работе внешнего. Число обслуживающих процессов, которые могут использоваться для отдельно взятой БД ограничивается размером пула. При достижении этого значения, процесс-демон Postgres перестает запускать новые процессы, а передает последующее подключение запущенному процессу. Так как один процесс может работать только с одной БД, возникает необходимость поддержки отдельного пула соединения для каждой БД. При появлении подключения к новой БД, добавляется новой пул. В параметрах могут быть указаны пользователи и БД, для которых не требуется реализация пула. Пулы функционируют только на уровне

транзакций, т.е. процесс может переключиться на обслуживание нового соединения только после завершения транзакции [16].

На рисунках 1.3 – 1.4 представлен цикл соединения с БД без пула и с его использованием соответственно [17].

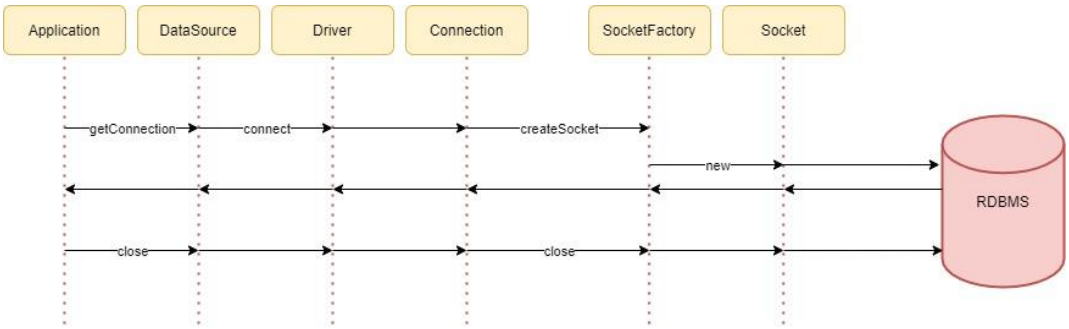

Рисунок 1.3 — Цикл соединения с БД.

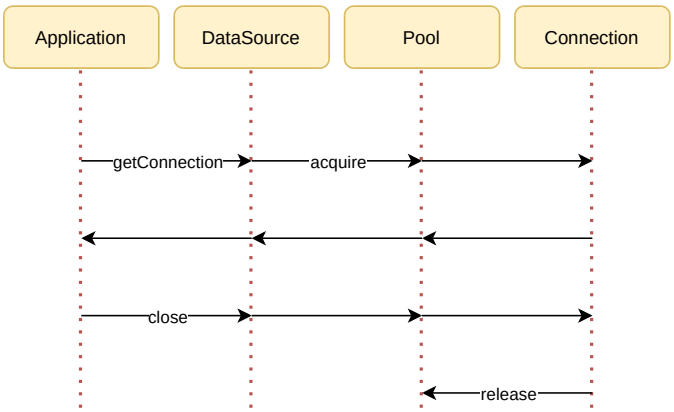

Рисунок 1.4 — Цикл соединения с БД с использованием пула.

2.2 Распараллеливание запросов

Другой стратегией увеличения производительности в рамках одного потока является распараллеливание запросов. Распараллеливание – это возможность построения таких планов запросов, которые будут задействовать несколько ядер. Наибольший выигрыш при использовании данного метода можно получить в случае работы с запросами, обрабатывающими большой объемом данных, но возвращающих пользователю всего несколько строк [9].

2.2.1 Планировщик

Сервер, получив запрос от прикладной программы, выполняет его разбор. Разбор подразумевает синтаксическую проверку запроса и построение дерева запроса. Задача планировщика заключается в создании плана запроса на основании полученного дерева. Выбор плана состоит из двух этапов: рассмотрение всевозможных вариантов для получения одного и того же результата и оценка каждого для выбора самого дешевого.

В случае, когда планировщик принимает решение о том, что параллельное выполнение запроса будет наилучшим вариантом, он создает план запроса. Данный план будет состоять из узла *Gather*(Сбор) или *Gather Merge*(Сбор со слиянием) [9]. Если один из описанных узлов будет располагаться в вершине дерева, то весь запрос будет выполняться параллельно. Иначе – частично.

2.2.2 Алгоритм работы параллельно выполняемых процессов

При достижении одного из узлов *Gather* или *Gather Merge* процесс, обслуживающий текущее подключение, выполняет запрос фоновых рабочих процессов. Каждый успешно запущенный фоновый процесс будет выполнять параллельную часть плана. При этом ведущий процесс помимо выполнения части плана будет выполнять контроль фоновых процессов. Его задача состоит в чтении кортежей, полученных от фоновых процессов, и выполнении других шагов обработки, связанных с узлами плана, расположенных выше *Gather* или *Gather Merge.*

На рисунке 1.5 представлена схема реализации параллельного запроса.

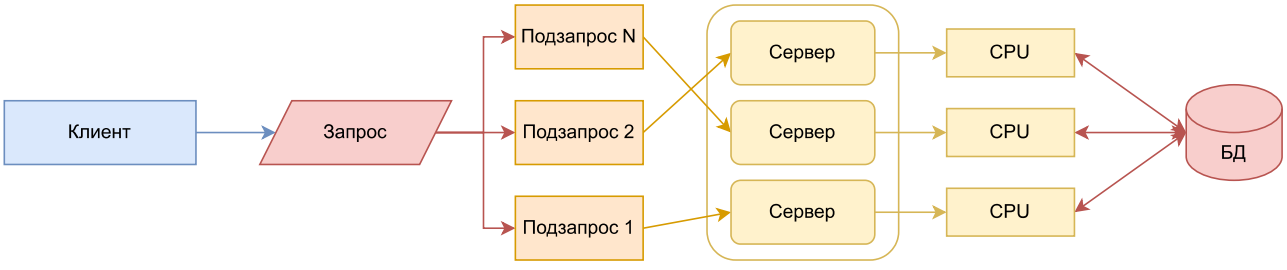

Рисунок 1.5 – Схема реализации параллельного запроса.

2.2.3 Безопасность метода

Планировщик классифицирует запросы следующим образом:

- Безопасно распараллеливаемая. Не мешает выполнению запроса.
- Ограниченно распараллеливаемая. Не может выполняться в параллельном рабочем процессе, но может выполняться в ведущем.
- Небезопасно распараллеливаемая. Не может выполняться параллельно даже в ведущем процессе. Для такого запроса параллельное выполнение полностью исключается.

К ограниченно распараллеливаемым запросам относятся такие запросы, как сканирование временных, сторонних таблиц, общих табличных выражений [9].

2.2.4 Недостатки метода

Для большинства запросов параллельное выполнение не дает никаких преимуществ во времени. Это может быть связано с тем, что последовательный план быстрее параллельного или с ограничением текущей транзакции. Также число рабочих процессов может не соответствовать ожидаемому, поскольку число исполнителей и число фоновых процессов ограничено. А нехватка рабочих процессов ведет к снижению производительности.

Описанный метод применим только к ограниченному числу запросов (большие данные, короткий ответ).

2.3 Сравнительный анализ методов

Пул соединений и распараллеливание процессов являются принципиально разными способами увеличения производительности. Первый метод решает задачу оптимизации операции подключения, второй – оптимизацию выполнения запроса. В зависимости от задачи пользователь выбирает тот или иной подход. Однако стоит отметить, что пул соединений является более универсальным

методом, в то время как для распараллеливания требуется соблюдение многих ограничений.

Что касается сравнения внешнего и внутреннего пулов, то недостатками первого является ограничение максимального количества одновременных подключений к БД, сложность встраиваемости в код и настройка конфигурации. В то время как второй доступен только в коммерческой версии PostgreSQL.

Вывод

В данной работе был выполнен анализ существующих СУБД и приведена аргументация выбора PostgreSQL, были рассмотрены основные положения архитектуры, были описаны инструменты соединения в PostgreSQL. Также были представлены существующие методы и выполнен сравнительный анализ.

Список использованных источников

1. ГОСТ 34.320-96 Информационные технологии. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы.
2. ГОСТ Р ИСО/МЭК ТО 10032-2007: Эталонная модель управления данными.
3. Основные классы современных параллельных компьютеров. [Электронный ресурс]. Режим доступа: https://parallel.ru/computers/classes.html.
4. Lyman P., Varian H.R. How much information Архивная копия от 19 февраля 2018 на Wayback Machine. Release of the University of California. Oct.27, 2003.
5. Копцев П. Ю., Аникьева Э. Н. Зарождение и история развития баз данных // Наука и образование. – 2018. – Том 1 №1. Режим доступа: http://www.opusmgau.ru/index.php/see/article/view/288.
6. Дикарев Н. И., Шабанов Б. М., Шмелев А. С. Векторный потоковый процессор: оценка производительности // Известия ЮФУ. Технические науки. 2014. №12 (161). Режим доступа: https://cyberleninka.ru/article/n/vektornyy-potokovyy-protsessor-otsenka-proizvoditelnosti.
7. Knowledge Base of Relational and NoSQL Database Management Systems. [Электронный ресурс]. Режим доступа: https://db-engines.com/en/ranking.
8. Концепция архитектуры PostgreSQL. [Электронный ресурс]. Режим доступа: http://www.dataved.ru/2014/09/postgresql.html
9. PostgreSQL: Документация: 12.8. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql/12.
10. Тарасов, С. В. СУБД для программиста. Базы данных изнутри / С. В. Тарасов. — Москва: СОЛОН-Пресс, 2018. — 320 с. Режим доступа: https://www.iprbookshop.ru/90409.html.
11. Пан К. С., Цымблер М. Л. Разработка параллельной СУБД на основе последовательной СУБД PostgreSQL с открытым исходным кодом // Вестник ЮУрГУ. Сер. Математическое моделирование и программирование. – 2012. – №18. – С. 277. Режим доступа: https://cyberleninka.ru/article/n/razrabotka-

parallelnoy-subd-na-osnove-posledovatelnoy-subd-postgresql-s-otkrytym-ishodnym-kodom.

12. Воронова, Н. М. Алгоритмы оценки производительности модуля работы с данными / Н. М. Воронова, А. С. Кованова, Н. С. Корж // Инновации. Наука. Образование. – 2021. – № 38. – С. 647-658.
13. Shetty N. Everything you need to know about Connection Pooling in Postgres. – 2019. [Электронный ресурс]. Режим доступа: https://www.ashnik.com/everything-you-need-to-know-about-connection-pooling-in-postgres/.
14. Object Pool. [Электронный ресурс]. Режим доступа: https://www.oodesign.com/object-pool-pattern.html.
15. Шиндов Д. А. Разработка пула соединений для работы с СУБД MYSQL на языке программирования C++ // ББК 1 А28. – 2021. – С. 112-113.
16. Aboagye M. Improve database performance with connection pooling. – 2020. [Электронный ресурс]. Режим доступа: https://stackoverflow.blog/2020/10/14/improve-database-performance-with-connection-pooling/.
17. Документация к Postgres Pro Enterprise 12.8.1. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/enterprise/12.
18. Shaikh S. S., Pachghare V. K. A Comparative Study of Database Connection Pooling Strategy // International Research Journal of Engineering and Technology. – 2017.