МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Приамурский государственный университет имени Шолом-Алейхема»**

ФАКУЛЬТЕТ Математики, информационных технологий и техники
КАФЕДРА Информационных систем, математики и правовой информатики

Направление 09.04.02 Информационные системы и технологии
Направленность Модели информационных процессов

ВКР ДОПУЩЕНА К ЗАЩИТЕ
Заведующий кафедрой
_____ Р.И. Баженов
«_____»._____.2020 г.

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(магистерская диссертация)

на тему «Разработка имитационной модели системы эффективных блокировок в централизованных базах данных»

Студент 2 курса магистратуры
А.В. Ленкин
Научный руководитель
ведущий научный сотрудник, д.т.н., доцент
А.Н.Родионов

ВКР защищена с оценкой

_____

Технический секретарь ГЭК

_____ П.А. Козич

г. Биробиджан, 20

## Оглавление

## Введение

На текущий день повсеместно используются различные базы данных, различающиеся функционалом и набором инструментария, а также методами размещения. Существуют распределенные и централизованные базы данных. В последнем случае база данных размещается на одном носителе или ЭВМ.

В работе с централизованными базами данных часто возникают задачи увеличения скорости доступа к их ресурсам или возможности параллельной работы нескольких пользователей.

При обычной работе транзакции в базах данных используют стандартные механизмы блокировки ресурсов. Такой подход действительно помогает транзакциям не влиять на работу друг друга и обеспечивает некоторую параллельность. Но также, ко всему прочему, при использовании стандартных механизмов блокировок (или уровней изоляции транзакций) в больших и/или сложных структурах базах данных такой подход лишь замедлит работу с ней.

СУБД включает механизмы управления блокировками. Последние могут быть адаптированы под конкретные задачи, обращающиеся к базам данных. Так как можно вручную изменять и настраивать необходимые блокировки, то делать это следует только удостоверившись в стабильности своего метода, так как изменение стандартных алгоритмов блокировок может вызвать некорректное выполнение запросов.

Поэтому актуальной научно-практической задачей является разработка системы эффективных блокировок в централизованных базах данных, которая бы обеспечивала максимальный параллелизм и уменьшая уровень эскалации блокировки, при условии сохранения согласованности баз данных.

Цель работы – разработать алгоритм собственного механизма блокировок в централизованных базах данных.

Гипотеза исследования: достижения максимального параллелизма при работе запросов и транзакций в базе данных будет достигнуто, если:

- на блокируемые запросом ресурсы будет накладываться минимально возможный уровень эскалации блокировки;

- запросы и транзакции не будут вызывать взаимные блокировки;

- длительность блокировки ресурсов будет минимальной.

Для реализации поставленной цели необходимо выполнить следующие задачи:

- проанализировать современные подходы к блокировкам в базах данных;

- выявить ключевую задачу для блокировки ресурсов базы данных;

- разработать механизм адаптивных блокировок для централизованных баз данных;

- провести эксперимент по интеграции собственного механизма блокировок в работу крупной базы данных.

Объектом исследования является централизованная база данных с ручным механизмом блокировки.

Предметом исследования является механизм эффективных блокировок в централизованной базе данных.

Практическим результатом магистерской диссертации является сформулированный алгоритм механизма эффективных блокировок для централизованных баз данных, который отличается от стандартных автоматических блокировок тем, что базируется на использовании в определенных структурах организации данных, что намного эффективнее.

Структура диссертации: работа состоит из введения, двух глав, заключения и списка литературы.

В первой главе рассмотрены проблемы использования блокировок в базах данных, описана технология блокировок в различных СУБД, рассмотрены примеры реализаций блокировок.

Вторая глава посвящена описанию собственного механизма блокировок в централизованных базах данных, а также описание проведенного эксперимента по внедрению такого алгоритма и выявления его преимуществ.

# 1 Проблемы управления параллельными транзакциями в централизованных базах данных

## 1.1 Блокировка как способ обеспечения одновременного доступа к ресурсам баз данных

Блокировки в базах данных – это метод предотвращения одновременного доступа к одним и тем же данным в БД для предотвращения создания противоречивых результатов [25].

Классический пример демонстрируется двумя банковскими клерками, пытающимися обновить один и тот же банковский счет для двух разных транзакций. Клерки 1 и 2 получают (т.е. копируют) запись по счету. Клерк 1 применяет и сохраняет операцию. Клерк 2 применяет к своей сохраненной копии другую транзакцию и сохраняет результат, основываясь на оригинале записи и его изменениях, перезаписывая транзакцию, введенную клерком 1. Запись больше не отражает первую транзакцию, как если бы она никогда не происходила.

Простой способ предотвратить это – заблокировать файл всякий раз, когда запись изменяется любым пользователем, так чтобы никакой другой пользователь не смог сохранить данные. Это предотвращает неправильное перезаписывание записей, но позволяет одновременно обрабатывать только одну запись, блокируя других пользователей, которым необходимо редактировать записи одновременно.

Для того чтобы несколько пользователей могли одновременно редактировать таблицу базы данных, а также для предотвращения несоответствий, создаваемых неограниченным доступом, одна запись может быть заблокирована при ее извлечении для редактирования или

обновления. Любому, кто пытается получить одну и ту же запись для редактирования, запрещается доступ на запись из-за блокировки (хотя, в зависимости от реализации, он может иметь возможность просмотреть запись без ее редактирования). Как только запись сохраняется или редактирование отменяется, блокировка освобождается. Запись никогда не может быть сохранена таким образом, чтобы перезаписать другие изменения, сохраняя целостность данных.

Концепт использования транзакций и, следовательно, блокировок в СУБД впервые возник в 1993 году. Он был описан автором San Mateo в его книге «Transaction processing: concepts and techniques» [35]. И в дальнейшем эта область лишь развивалась, адаптируясь для конкретных баз данных.

Механизмы блокировок в базах данных развивались каждым разработчиком СУБД под конкретные задачи. При этом у каждой СУБД есть как общие со всеми механизмы блокировок, так и созданные конкретно для определенных нужд различные механизмы.

На данный момент в связи с очень быстрым развитием информационных технологий, увеличением количества обрабатываемой информации, наблюдается замедление в работе очень крупных баз данных. Такое происходит чаще всего из-за неправильно настроенных блокировок в компании, ведь обычно все используют механизмы блокировки, выставленные по умолчанию при развёртывании СУБД.

Также стоит учесть, что разработка собственного механизма блокировок в централизованных базах данных могут помешать следующие факторы:

1.Централизованность. Вся информация базы данных сервера находится в одном месте и не распределена, следовательно, осуществлять параллельный доступ к таким данным будет намного сложнее, чем в распределенных базах данных.

2.Автоматические алгоритмы блокировок. Использование в базе данных только автоматической блокировки может существенно замедлить процесс обмена данными в системе «пользователи – БД».

3.Ошибки в работе базы данных. При использовании ручного алгоритма блокировок возможны появления таких ошибок как: грязное чтение, невоспроизводимое чтение, фантомные данные.

## 1.2 Задачи использования блокировок в базах данных

Согласно теме выпускной магистерской диссертации, нами были рассмотрены наиболее актуальные тематике исследования научные работы и изыскания отечественных и зарубежных авторов.

Григорьев Ю.А. [10] провёл организацию базы данных в программном комплексе анализа характеристик производительности распределённых систем обработки данных. Он попытался рассмотреть проблемы при разработке базы данных, а также установил необходимое распределение транзакций. Рассмотрели определение временных интервалов в алгоритмах управления Ларкин Е.В., Ивутин А.Н. [13], где описали как правильно регулировать выполнение транзакций в базах данных. Васильев А.П., Степанов С.Н. [6] занимались исследованием математической модели с динамическим распределением канального ресурса при групповом поступление запросов на передачу. В своей работе они продемонстрировали как создать математическую модель распределения транзакций, описали сравнение их распределения в зависимости от размера выполняемых запросов и транзакций. Распределение потока запросов в параллельных СУБД на платформе вычислительных кластеров было продемонстрировано Минязевым Р.Ш [16]. Автор постарался сравнить все возможные алгоритмы, которые могут обеспечивать распределение транзакций и при этом осуществлять минимальное время её выполнения, описал теоретическую систему с непрерывной работой.

Лебеденко Е.В. [14] показал, как осуществлял разработку алгоритма оптимального планирования распределения запросов в системах распределенного моделирования. Он рассмотрел, какие из возможных алгоритмов распределения транзакций могут максимально повысить отказоустойчивость СУБД. Многопутевым резервированным распределением через сеть критичных к задержкам запросов занимались Богатырев В.А. и Паршутина С.А. [5]. Их задачей было показать эффективность передачи запросов на основе многопутевой маршрутизации и описать возможные ошибки при работе с ней. Богатырев В.А. [2] исследовал отказоустойчивость распределенных вычислительных систем динамического распределения запросов и размещение функциональных ресурсов. Он показал, как распределены ресурсы в децентрализованных базах данных и каким образом следует размещать там ресурсы для отказоустойчивости.

Выбор вариантов организации распределения запросов в системах предоставления информационных услуг был исследован Богатыревым В.А., Голубевым И.Ю., Нестеровым Д.А. [4]. В этой работе показано как распределять запросы максимально эффективно в сфере информационного сервиса, максимизировать производительность и минимизировать время выполнения запросов. Голубев И.Ю., Богатырев В.А. [9] предложили метод оптимизации распределения запросов в системе кластеров при сочетании аналитического и имитационного моделирования. Здесь же была изложена процедура распределения запросов с помощью аналитического и имитационного моделирования.

Оптимизация резервированного распределения запросов в кластерных системах реального времени была описана Богатыревым В.А., Богатыревым А.В. [3]. Авторы продемонстрировали как работать с распределением запросов в реальном времени, как обрабатывать ошибки и отказы, как оптимизировать такую систему. Аникин Н.А. [1] показал свой метод определения порядка сериализации транзакций в системах управления базами данных, использующих протокол строгой двухфазной блокировки. Также были рассмотрены способы обеспечения параллельного доступа в распределенной гетерогенной базе данных.

Математическая модель функционирования распределённой информационной системы на базе архитектуры "файл – сервер" с учётом влияния блокировок была описана Скоба А.Н., Логанчук М.Л. [23]. В данной статье было описано создание математической модели для решения задач получения интегральных показателей распределенной информационной системы с учётом блокировок на уровне базы данных. Также получены оценки её среднего времени выполнения запросов при таких блокировках. Пулатов Ю.Г. [21] провел сравнение методов оптимистической и пессимистической блокировки при параллельном доступе к ресурсу базы данных. Автор рассматривает проблему параллельного доступа к ресурсам базы данных, представляет способы решения в качестве использования оптимистичной и пессимистичной блокировки, сравнивает их. Мониторинг блокировок в Oracle был продемонстрирован Михеичевым В. [18]. Было показано

на примере организации как DML-блокировки происходят в СУБД Oracle.

Черноморов Г.А. [28 ] получил рабочую модель функционирования корпоративной информационной системы централизованной архитектуры с учетом блокировок транзакций. В статье описаны этапы разработки концептуальной модели корпоративной информационной системы с учетом блокировок. Модель взаимодействия потоков транзакций на SQL-сервере подробно была описана Евсеевым Г.С., Лесниковым О.С. [12]. Авторы показывают модель того как в SQL сервере взаимодействуют два потока транзакций с учётом различных блокировок. Мониторинг блокировок в Oracle, а также методы предупреждения и автоматического устранения кратко были изложены Михеичев В. [19]. В статье описан опыт автора по устранению в базе данных системы организации блокировок, которые приводят к отказу работоспособности.

Головинский И.А. [8] объяснил значение блокировки автоматов групп и взаимных блокировок. Исследование проходит над группами автоматов, которые образованы взаимными блокировками перестановочных автоматов, также обнаружены максимальные взаимные блокировки, объяснён алгоритм выхода из взаимных блокировок. Анализ процессов согласования версий записей в базах данных NOSQL был исследован Григорьевым Ю.А., Плутенко А.Д., Бурдаковым А.В., Цвященко Е.В. [11]. Они показывают, как в базах данных NOSQL, которые работают без явных механизмов транзакций и блокировок ввести возможность оценки числа версий записей и времени их согласованности, что является

актуальным в таких не реляционных базах данных. Михеичев В. [17] демонстрирует взаимоблокировки Deadlock-сессий в Oracle. В статье изложены практический опыт диагностики взаимоблокировок deadlock-сессий, причины возникновения взаимоблокировок, методы борьбы с ними и возможности предупреждения.

Решением проблем параллельной обработки транзакций и выходом из тупиковых ситуаций в базах данных занимались Омельяненко М.В., Папинашвили В.Г. [20]. Авторы рассматривают вопрос такого понятия как «тупиковая ситуация», причины её появления и пути её решения при параллельной обработке транзакций, а также проблему параллелизма. Причины блокировок в системе 1С: Предприятие и методы борьбы с ними тщательно рассмотрены Мазеиным К.В., Поняшовой А.С., Безносовым П.П., Козырь А.А., Храмовым С.В. [15]. В статье рассматриваются причины блокировок в конкретной СУБД 1С: Предприятие, описана правильная настройка оборудования для методы их минимизирования, указаны рекомендации для уменьшения блокировок разработчиками. Форд Т. [27] проанализировал продолжительность ожидания блокировок. В своей статье он показал, как использование статистики работы индекса и общие рекомендации могут помочь создать высокопроизводительную среду SQL Server.

Проектирование транзакционной изолированности в различных типах приложений на примере базы данных Microsoft SQL Server приводится Герасимовым А.А., Тихомировой А.Н. [7]. В данной работе представлено проектирование транзакционной изолированности в

различных типах приложений, показаны случаи применения транзакции, собственные стратегии изоляции, их преимущества и недостатки, а также различные виды транзакций. Ризаев И.С., Кладиев А.В., Клевин А.С. [22] показали, как осуществить повышение эффективности выполнения транзакций при распределенной обработке данных. В этой работе авторы рассмотрели проблемы параллельного доступа нескольких пользователей к одним и тем же данным, описали понятие транзакции как единицы взаимодействия с базой данных, предложили решение проблемы конфликтов при параллельном доступе. Занимались эффективной совместной обработкой запросов в распределенной среде Wenjie Liu, Zhanhuai Li [44]. Они описали в своей работе методы фильтрации для тета-соединений, направленные на снижение времени их выполнения в распределенной сети.

Weipeng Jing, Dongxue Tian [43] занимались улучшенным распределенным хранением запросов к данным. В своей работе они использовали технологии открытых больших данных. Их метод может эффективно повысить скорость записи данных и выполнения запросов. Генерация оптимальных планов запросов для распределенной обработки запросов с использованием оптимизации на основе метода TLBO была описана Vikash Mishra, Vikram Singh [41]. Авторы данной статьи сообщают о проблемах распределенных баз данных и рассказывают об экспериментах по применению оптимизации построения запросов на основе обучения TLBO, а также его сравнение с GA. Emanuele Carlini, Alessandro Lulli, Laura Ricci dragon [33] проводили исследования

использования технологии Dragon – запросов на многомерные диапазоны в распределенных деревьях агрегации. Ими была объяснена эта технология, подробно проанализированы различные стратегии агрегирования и распределения запросов в широком спектре экспериментальных настроек.

Распределенные запросы в среде электронного обучения были показаны Iacob (Ciobanu) Nicoleta – Magdalena [37]. Целью данной работы являлось представление стратегий оптимизации выполнения запросов в системах электронного обучения, используемых в основном университетами с территориально распределенным положением для получения как можно более низкого времени отклика системы и минимизации общих затрат на их реализацию. Подход к оптимизации интеллектуальных запросов в распределенной среде был исследован авторами Hassen Fadoua, Touzi Grissa Amel [36]. Данные авторы показали, как в масштабах распределенной базы данных осуществлять поиск информации не отдельно, а все сразу, используя слой доступа высокого уровня с возможностью семантического анализа (индексное семантическое обобщение). Xiaoyong Li, Yijie Wang, Xiaoling Li, Xiaowei Wang, Jie Yu [46] выяснили многие нюансы в своей работе «GDPS: Эффективный подход для горизонтальных запросов по распределенным неопределенным данным». В данной работе они подробно исследовали проблему распределенного вероятностного горизонтального запроса и предложили эффективный подход GDPS для решения проблемы с оптимизированным

итеративным механизмом обратной связи, на основе сводной таблицы.

Планирование множественных запросов для распределенных семантических кэшей стала темой для статьи авторов Beomseok Nam, Minho Shin, Henrique Andrade, Alan Sussman [30]. Они экспериментально продемонстрировали полезность политик планирования с помощью MQO, который является распределенной межплатформенной системой обработки множества запросов с поддержкой Grid-совместимости, разработанной ими для оптимизации обработки запросов в приложениях для анализа и визуализации данных. Yongluan Zhou, Beng Chin Ooi, Kian-Lee Tan, Wee Hyong Tok [48] была выведена и описана адаптируемая архитектура распределенной обработки запросов. В данной статье они представили новую высоко адаптивную архитектуру распределенной обработки запросов. Их архитектура способна быстро обнаруживать колебания в селективности операций, а также в скорости передачи и нагрузки серверов, и соответственно изменять порядок выполнения плана распределенных запросов во время выполнения. Планирование многофакторных блокировок в реальном времени в объектно-ориентированных системах баз данных было тщательно продумано Qin Xiao, Pang Li-Ping, Liu Jie, Li Sheng-Li [39]. В этой статье представлена модель многофакторной блокировки в реальном времени (RTMGL) для управления параллелизмом в объектно-ориентированной системе баз данных, а также предложены алгоритмы планирования в реальном времени для RTMGL. Как показано в экспериментах, алгоритмы планирования в

реальном времени ведут себя лучше, чем алгоритмы не в реальном времени в среде реального времени.

D. Agrawal, A. Elabbadi [31] занимались такой проблемой как «Ограниченные общие блокировки для повышения параллелизма в базах данных». Новая взаимосвязь между разделяемыми и не разделяемыми блокировками позволило им разработать семейство протоколов блокировки, наименее разрешающим из которых является двухфазная блокировка, в то время как наиболее разрешающее допускает все сохраняющие порядок конфликтные сериализуемые запросы. «Адаптивные блокировки: объединение транзакций и блокировок для эффективного параллелизма» были описаны Takayuki Usui, Reimer Behrends, Jacob Evans, Yannis Smaragdakis [40]. В данной работе предложен метод адаптивной блокировки, который динамически отслеживает, будет ли критическая секция выполняться наилучшим образом транзакционно или при удержании мьютексной блокировки. Yonggoo Choi, Songchun Moon [47] занимались совместным исследованием облегченной мультигранулярной блокировкой для управления транзакциями в системах баз данных XML. Для разработки схемы параллельного управления без фантомного феномена они предложили легкую схему блокировки с несколькими гранулами (LWMGL), которая представляет собой гибридный механизм блокировки на основе дерева и блокировки с несколькими гранулами. Целью этой схемы является реализация блокировки на уровне точных элементов в XML-базе данных при одновременном предотвращении проблем с фантомами. «Заказанный обмен: Новый примитив блокировки для систем

баз данных» был темой исследования авторов Divyakant Agrawal, Amr El Abbadi [32]. Данные авторы в 1995 году первыми предложили новый примитив блокировки под названием упорядоченный обмен, который позволяет повысить параллелизм в системах баз данных. Разработан обобщенный протокол двухфазной блокировки, который может использовать блокировки со стандартными общими и нераспространенными отношениями, а также предлагаемыми упорядоченными общими отношениями между блокировками.

W Jun [42] исследовал метод семантических блокировок в объектно-ориентированных базах данных. В этой статье представлен элемент управления параллелизмом на основе блокировок, в котором рассматриваются три важных вопроса в объектно-ориентированных базах данных: семантика методов, вызов вложенных методов и объект со ссылками на общий доступ. Блокировка в DAG-структурированных базах данных является темой изучения авторов Wojciech Cellary, Waldemar Wieczerzycki [45]. Предлагается способ блокировки штампа, который предназначен для баз данных со структурой DAG, например, объектно-ориентированные базы данных. Метод рассматривает все семантические отношения между объектами в базе данных единообразным, иерархическим способом. Предложенная иерархическая блокировка не требует преднамеренных блокировок. G. Lausen, E. Soisalonsoininen [34] занимались исследованием безопасности в базах данных при помощью неинтерпретированных блокировок. В работе было представлено новое условие для безопасных блокировок. Ими

также было установлено, что существуют безопасные блокировки, которые не могут быть приняты безопасной блокировкой в модели блокировки сущностей. Нормализация больших данных для массивно параллельных вычислительных баз данных была целью исследования авторов Nikolay Golov, Lars Rönnbäck [38]. В этой статье представлен новый метод с высокой нормализацией больших данных, использующий моделирование привязки, который обеспечивает эффективный способ хранения информации и использования ресурсов, тем самым впервые обеспечивая специальные запросы с высокой производительностью в базах данных с массовой параллельной обработкой.

В результате проведенного анализа исследований были выделены основные проблемы разработки системы эффективных блокировок в централизованных базах данных:

1.Существенные различия в методах и целях исследования блокировок в работах других ученых и исследователей по теме диссертации. Эта проблема существенно усложняет задачу, так как нельзя все цело опираться на чужой опыт, но при этом данное исследование может считаться инновационным.

2.Необходимость следить за возникающими ошибками в базе данных. При нарушении записи или чтения из базы данных исследуемая система эффективных блокировок будет считаться некорректно работающей, а, следовательно, и неполноценной.

3.Малое количество официального руководства по блокировкам в базах данных. Найденный материал касательно использование ручных блокировок носит

исключительно ознакомительный формат и не рекомендуется к использованию разработчиками СУБД.

### 1.3. Транзакции и блокировки в базах данных

В базах данных MS SQL поддерживаются и работают с общими типами блокировок.

Режим блокировки учитывает различные типы блокировок, которые могут быть применены к ресурсу, который должен быть заблокирован [29]:

1. Исключительная (X).
2. Разделяемая (S).
3. Обновления (U).
4. Намерения (I).
5. Схемы (Sch).
6. Групповое обновление (BU).

Исключительная блокировка (X) – это тип блокировки, когда он установлен, гарантирует, что страница или строка будут зарезервированы исключительно для транзакции, которая наложила исключительную блокировку, до тех пор, пока транзакция держит блокировку.

Разделяемая блокировка (S) – этот тип блокировки, когда он установлен, зарезервирует страницу или строку, которая будет доступна только для чтения, что означает, что любая другая транзакция будет запрещена для изменения заблокированной записи, пока блокировка активна. Однако, разделяемая блокировка может быть наложена несколькими транзакциями одновременно на одну и ту же страницу или строку, и таким образом, несколько транзакций могут совместно использовать возможность чтения данных,

поскольку сам процесс чтения никак не повлияет на фактические данные страницы или строки. Кроме того, разделяемая блокировка позволит выполнять операции записи, но изменения не будут разрешены.

Блокировка обновления (U) – эта блокировка похожа на исключительную, но разработана таким образом, чтобы быть более гибкой. Блокировка обновления может быть наложена на запись, которая уже имеет общую. В этом случае блокировка обновления наложит еще одну разделяемую блокировку на целевую строку. Как только транзакция, содержащая блокировку обновления, будет готова к изменению данных, блокировка обновления (U) будет преобразована в исключительную блокировку (X). Важно понимать, что блокировка обновления асимметрична по отношению к разделяемым. В то время как блокировка обновления может быть наложена на запись, которая имеет разделяемую блокировку, разделяемая блокировка не может быть наложена на запись, которая уже имеет блокировку обновления

Блокировка намерения (I) – является средством, используемым транзакцией для информирования другой транзакции о своем намерении осуществить блокировку. Целью такой блокировки является обеспечение корректного выполнения модификации данных, предотвращая получение блокировки на следующем объекте иерархии другой транзакцией. На практике, когда транзакция хочет получить блокировку на строке, она приобретает намеренную блокировку на таблице, которая является объектом более высокой иерархии. Приобретая намеренную блокировку,

транзакция не позволит другим транзакциям приобрести исключительную блокировку на этой таблице.

Блокировка схемы (Sch) – механизм базы данных SQL Server распознает два типа блокировок схемы: Блокировка изменения схемы (Sch-M) и блокировка устойчивости схемы (Sch-S)

Блокировка модификации схемы (Sch-M) будет получена при выполнении DDL-запроса, что предотвратит доступ к данным заблокированного объекта при изменении структуры объекта. SQL Server позволяет блокировку изменения одной схемы (Sch-M) на любом заблокированном объекте. Для того чтобы изменить таблицу, транзакция должна дождаться, пока получится получить Sch-M-блокировку на целевом объекте. После приобретения блокировки изменения схемы (Sch-M) транзакция может изменить объект, а после ее завершения и разблокировки блокировка будет снята. Типичным примером блокировки Sch-M является перестройка индекса, поскольку перестройка индекса – это процесс модификации таблицы. После выдачи идентификатора перестройки индекса на этой таблице будет получена блокировка модификации схемы (Sch-M), которая будет выпущена только после завершения процесса перестройки индекса.

Блокировка стабильности схемы (Sch-S) будет получена во время компиляции и выполнения запроса, зависящего от схемы, и генерации плана выполнения. Данная блокировка не блокирует другие транзакции для доступа к объектным данным и совместима со всеми режимами блокировки, кроме блокировки модификации схемы (Sch-M). По сути, блокировки стабильности схемы будут собираться каждым

DML-кодом и выбирать запросы, чтобы обеспечить целостность структуры таблицы (убедиться, что таблица не изменяется во время выполнения запросов).

Блокировка группового обновления (BU) – предназначена для использования в операциях массового импорта при выдаче аргумента/подсказки TABLOCK. При получении блокировки группового обновления другие процессы не смогут получить доступ к таблице во время выполнения групповой загрузки.

Для ручного управления блокировками в MS SQL существует механизм уровня изоляции, который присваивается нужной транзакции и выдает при её работе соответствующие блокировки. Уровни изоляции указаны в таблице 1.1 [26].

Таблица 1.1 – Уровни изоляций транзакций в базах данных MS SQL.

| **Уровень изоляции** | **Грязное чтение** | **Невоспроизводимое чтение** | **Фантомные данные** | **Параллелизм (несколько пользователей)** |
|---|---|---|---|---|
| ReadUncommitted | Да | Да | Да | Наилучший |
| ReadCommitted | Нет | Да | Да | Хороший |
| Snapshot | Нет | Нет | Нет | Хороший |
| RepeatableRead | Нет | Нет | Да | Слабый |
| Serializable | Нет | Нет | Нет | Очень слабый |

ReadUncommitted – полное отсутствие исключительных и разделяемых блокировок. Максимальная производительность работы с такими данными, но они могут быть изменены любой транзакцией и вызвать недействительное чтение.

ReadCommitted – разделяемые блокировки присутствуют, пока считываются данные. Режим по умолчанию в MS SQL.

Snapshot – с данных, к которым обращается транзакция, берется копия. Каждая транзакция может делать это повторно и работать только со своей копией данных. Отключена по умолчанию.

RepeatableRead – разделяемые блокировки накладываются на все данные, с которыми работает транзакция.

Serializable – блокировки диапазона данных с которыми работает транзакция. Во время такой блокировки запрещены вставки и обновления записей другими пользователями.

Также для баз данных MS SQL существует другой способ использование блокировок вручную – табличные указания. [24]

Табличные указания (табличные подсказки) – переопределяют поведение оптимизатора запросов по умолчанию на время выполнения инструкции языка обработки данных (DML). Для этого указываются способ блокировки, один или более индексов, операция обработки запроса, например сканирования таблицы или поиска в индексе, или другие параметры. Табличные указания задаются в предложении FROM инструкции DML и относятся

только к таблицам и представлениям, на которые ссылается это предложение.

Табличные указания задаются к запросу с помощью конструкции WITH (<table_hint>), где <table_hint> – одно из указаний выбранного уровня блокировки, опишем самые важные из них:

1. PAGLOCK – блокировка страницы, вместо блокировки строки или индексов по умолчанию.
2. ROWLOCK – блокировка строк, вместо блокировки страниц или таблиц.
3. TABLOCK – блокировка всей таблицы.
4. NOLOCK – отсутствие любых блокировок.

## Выводы

В данной главе были рассмотрены причины внедрения собственной эффективной системы блокировок в централизованной базе данных. Рассмотрены основные проблемы, возникающие при использовании блокировок вручную, а также приведены существующие средства для управления блокировками в СУБД MS SQL.

Проведен анализ подходов к внедрению и использованию ручных блокировок и состояние исследований на данный момент, в результате чего был выведен общий подход авторов к созданию блокировок, заключающийся в простом использовании уровня изоляции транзакций или увеличение максимально возможного уровня параллельной работы для пользователей. Такой подход никак не учитывает структуру базы данных, так как описанные методы блокировок являются обобщенными, а также не ясно насколько их исследования могут обеспечить стабильную работу базы данных.

В первую очередь необходимо решить будет ли правильным решением использовать уровень изоляций транзакций или же табличные указания могут обеспечить куда более полный контроль над выполнением запросов в базе данных и самих блокировок данных.

Следует выяснить, эффективны ли такие подходы исследователей в данной проблеме и требуется ли более тонкая настройка механизма блокировок.

Основной целью при этом будет получение результата того, какая именно блокировка может обеспечить

целостность базы данных одновременно с увеличением уровня параллелизма для пользователей.

# 2 Стратегии, методы и алгоритмы тестирования и проектирования системы блокировок баз данных

## 2.1 Использование уровней изоляции транзакций для изменения блокировок

Первоначально для создания ручного алгоритма блокировок решено было проверить эффективность уровня изоляций транзакций в базах данных MS SQL.

Было описано две транзакции

SET TRANSACTION ISOLATION

LEVEL READ UNCOMMITTED

BEGIN TRANSACTION

INSERT INTO dbo.Kurs (Name_krs) VALUES ('8-й курс')»

И

BEGIN TRANSACTION

INSERT INTO dbo.Kurs (Name_krs) VALUES ('8-й курс')»

В результате выполнения обеих транзакций в SQL Profiler был зарегистрирован один режим блокировки (рис. 2.1).

| | | |
|---|---|---|
| Lock:Acquired | 2 - DATABASE | 3 - S |
| Lock:Acquired | 5 - OBJECT | 8 - IX |
| Lock:Acquired | 6 - PAGE | 8 - IX |
| Lock:Acquired | 7 - KEY | 15 - RangeI-N |
| Lock:Acquired | 7 - KEY | 5 - X |
| Lock:Released | 7 - KEY | 5 - X |
| Lock:Released | 6 - PAGE | 8 - IX |
| Lock:Released | 5 - OBJECT | 8 - IX |
| Lock:Released | 2 - DATABASE | 3 - S |

Рис. 2.1. Блокировка базы данных в SQL Profiler при выполнении транзакции

Такой результат можно интерпретировать двумя способами:

1. Уровень изоляции READ UNCOMMITTED не работает в базах данных MS SQL.

2. При выполнении транзакции на вставку уровень изоляции не учитывается и используются автоматические алгоритмы блокировок.

Поэтому данный способ управления блокировками не подходит для проведения исследования, так как из него невозможно получить какие-либо значимые результаты.

## 2.2 Структура тестового приложения для эмуляции запросов с блокировками

Чтобы создать клиент-серверную систему тестировании необходимо прежде всего понять, как работает приложение базы данных. Рассмотрим стандартную структуру любого приложения баз данных (рис. 2.2). Для создания тестовой среды необходимо: создать сервер баз данных, разработать имитатор запросов) клиента, настроить сервер так, чтобы мониторить время выполнения запросов от клиента. Общая схема тестовой среды будет выглядеть как на рисунке 2.3.

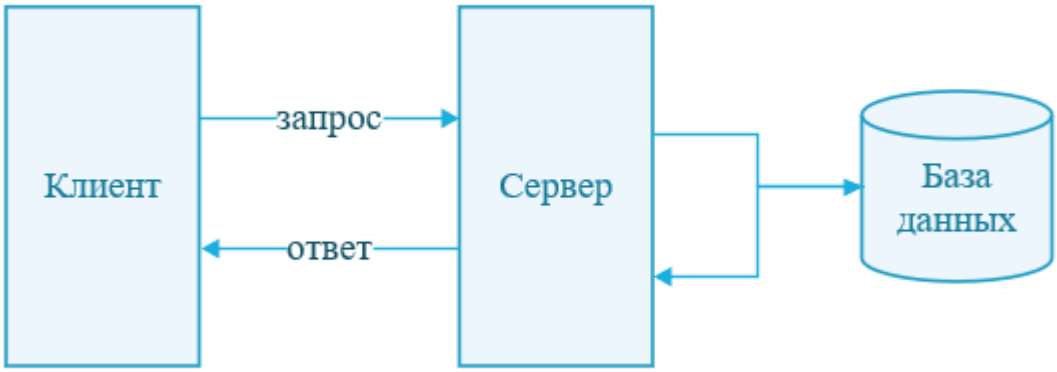


Рис. 2.2. Структура приложения баз данных.

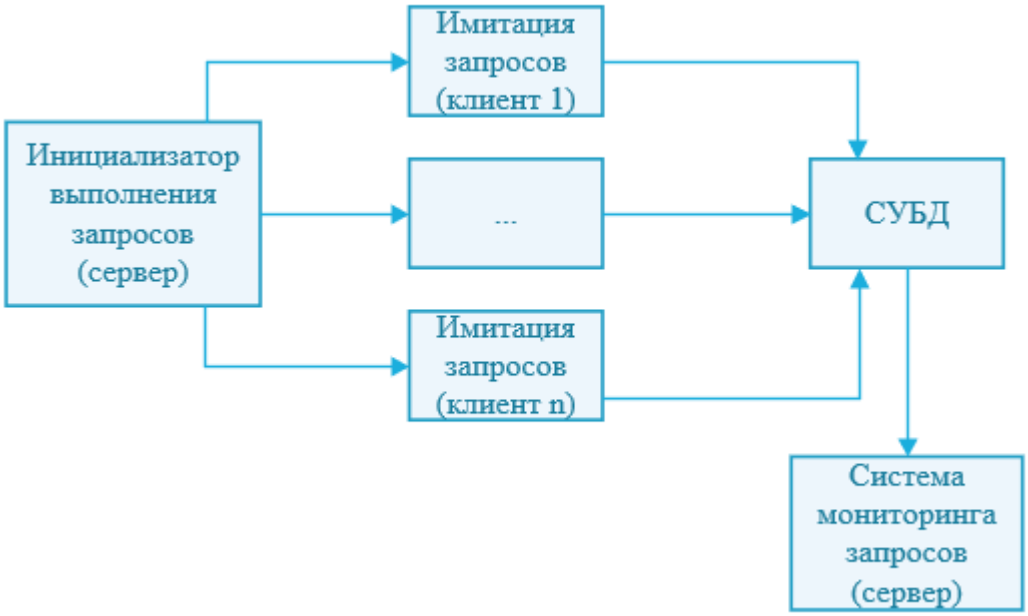


Рис. 2.3. Структура тестового стенда для проведения эксперимента

Для более точных симуляций работы реальной системы наилучшим решением будет расположить сервер и СУБД на одной главной рабочей станции, а клиенты по остальным рабочим станциям, но все рабочие станции должны обладать

одинаковыми характеристиками. Таким образом схема тестового стенда будет выглядеть так (рис. 2.4.).

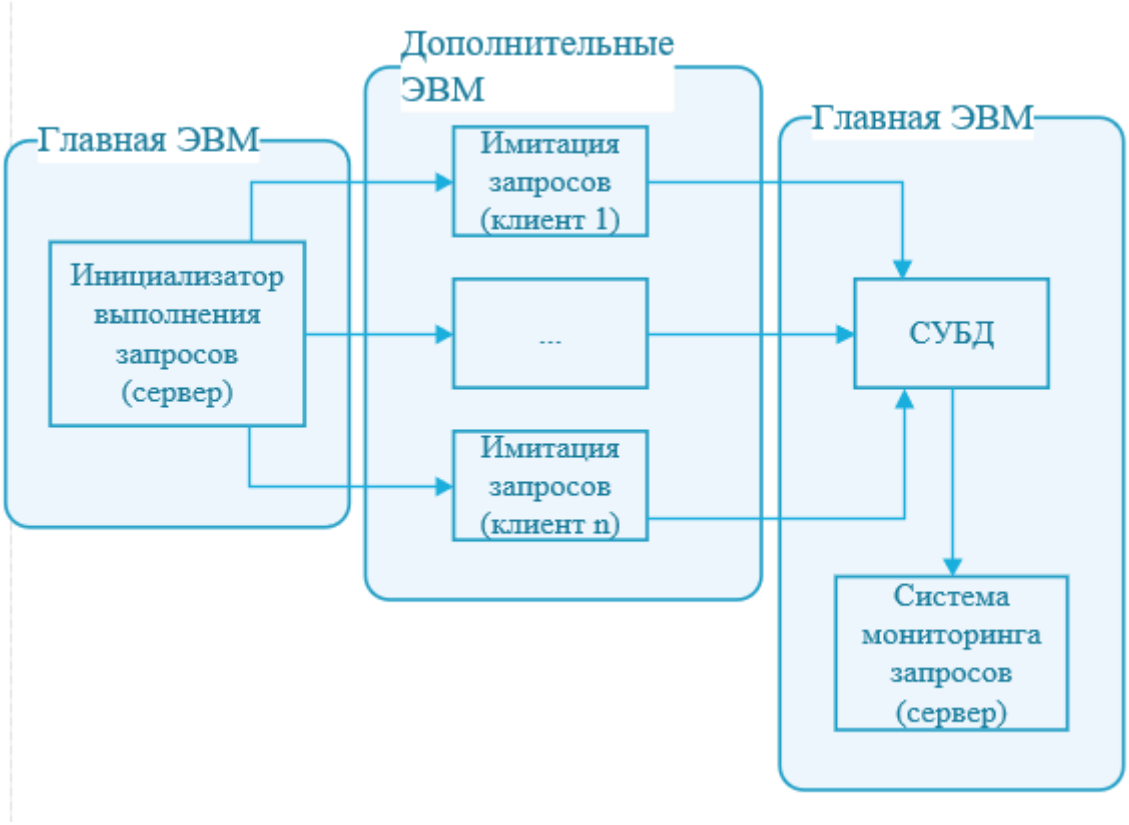


Рис. 2.4. Схема окончательного размещения тестового стенда.

Теперь рассмотрим, как реализована данная архитектура на конкретном языке программирования. Для начала опишем характеристики рабочих машин тестового стенда. Для тестирования использовались ЭВМ со следующими параметрами: процессор Intel Core i3-2120 3.30GHz, ОЗУ объёмом 2GB, HDD 250GB и установленной операционной системой Windows 7 x32.

На клиенты был установлен Microsoft .NET framework версии 4.8, так как клиентские приложения генератора запросов создавались с использованием этой технологии. У клиентов также прописан функционал для мониторинга

выполнения запросов. Техническое исполнение будет описано дальше.

На главную ЭВМ был установлен СУБД Microsoft SQL Server 2014, так как это последний СУБД поддерживающий 32-битные системы. Для мониторинга используется сбор ответов с клиентов, а также Microsoft SQL Server Profiler – встроенный инструментарий для мониторинга в СУБД Microsoft SQL Server.

В СУБД была развёрнута тестовая база данных University. Обращение к базе данных будет выполняться только к одной её части. Схема тестовой базы данных изображена на рисунке 2.5.

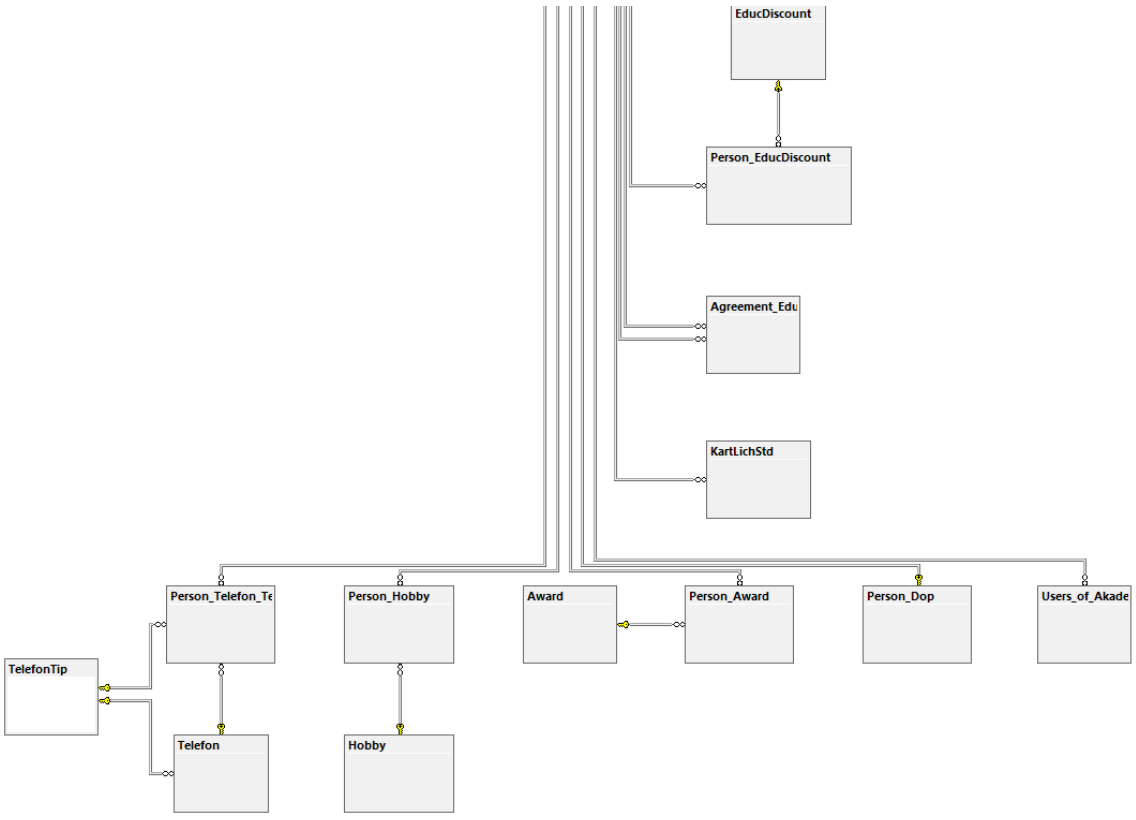


Рис. 2.5. Схема тестовой базы данных

Также были определены отправляемые клиентом процедуры для имитации запросов. Код одной из процедур изображен на рисунке 2.6. В нём видно, что запросы будут выполняться последовательно с очищением кэша плана запросов. Это сделано для того, чтобы провести эксперимент учитывая только возможности, получаемые от механизма блокировок, без учёта кэширования и оптимизации запросов.

```
DBCC FREEPROCCACHE;
DBCC DROPCLEANBUFFERS;
  DECLARE @i INT;
  SET @i=0;
  WHILE @i<" + valuetext+@"
  BEGIN
      INSERT INTO University.dbo.Hobby (Name_hb) VALUES ('Test string" + data.Substring(0, data.IndexOf(" ")) + @"_'+CONVERT(NVARCHAR, @i));
      SET @i=@i+1;
  END
```

Рис. 2.6. Один из кодов эмуляции запросов

Мониторинг будет осуществляться на серверном приложении для тестирования блокировок, который будет получать от клиентов их затраченное время на выполнения запросов.

Все рабочие станции расположены в одной сети и находятся в одном кабинете. Это было сделано для удобства управления и размещения клиентов. На рисунке ниже изображен кабинет с размещенным тестовым стендом (рис 2.7-2.8).

Рис. 2.7. Рабочие станции с размещенными клиентами и сервер

Рис. 2.8. Рабочие станции с размещенными клиентами

## 2.3 Описание программы для эмуляции тестовой среды

Для того чтобы проверить уникальность или преимущество собственного алгоритма блокировок, необходимо выполнять эмуляцию запросов с клиентов не последовательно, а параллельно, т.е. запускать клиенты одновременно. Для реализации этого необходимо использовать параллельные вычисления. Основные термины, которые будут использоваться:

- конкурентность (concurrency);
- параллельное исполнение (parallel execution);
- блокировка страницы (PAGLOCK).

Конкурентность (concurrency) – это обозначение того, что одновременно будет исполняться более одной задачи. Сама конкурентность будет выполняться за счёт того, что база данных будет накладывать блокировки на свои ресурсы, что будет вызывать при некоторых запросах конкурентные транзакции.

Параллельное исполнение (parallel execution) – значит, что будет выполняться одновременно несколько задач, которые будут работать одновременно благодаря наличию более одного вычислительного устройства.

Блокировка страницы (PAGLOCK) – это термин языка SQL, означающий что будет применена блокировка страницы в базе данных. Страница это основная единица хранения данных в SQL Server, размер страницы в SQL Server составляет 8 КБ.

Для создания приложения и реализации параллельных вычислений .Net Framework, с использованием одновременно всех клиентов будем использовать технологию параллельного программирования с использованием библиотеки Task Parallel Library (TPL), где в основе лежит класс Task, пространства имен System.Threading.Tasks, так как она легче других позволит отправить команду на старт запросов одновременно всем клиентам.

Цель разработки генератора запросов к базе данных, это отправлять запросы в базу данных одновременно, создавая конкурентные транзакции, чтобы сравнить как использование автоматического механизма блокировок или ручного механизма блокировок скажется на скорости выполнения последовательных запросов.

Тестовый стэнд проверки механизма блокировок в базах данных состоит из двух приложений: сервера с интерфейсом, для управление отправкой запросов, подключения клиентов и сбор информации о выполненных запросах; клиента для генерации запросов в базу данных являющийся консольным приложением. Такой подход обусловлен следующими фактами:

1. Консольное приложение легко разместить на других ЭВМ и оно не потребляет лишних ресурсов.

2. Визуальное представление сервера позволит нагляднее заниматься мониторингом запросов.

3. Наличие сервера будет обеспечивать точное одновременное исполнение клиентов.

У серверного приложение имеется два окна: главное (рис.2.9), где определяется тип передаваемой команды

клиентам, количество запросов и тест связи с клиентами; окно настроек (рис. 2.10) в котором устанавливаются IP адреса клиентов и их статус (вкл/выкл).

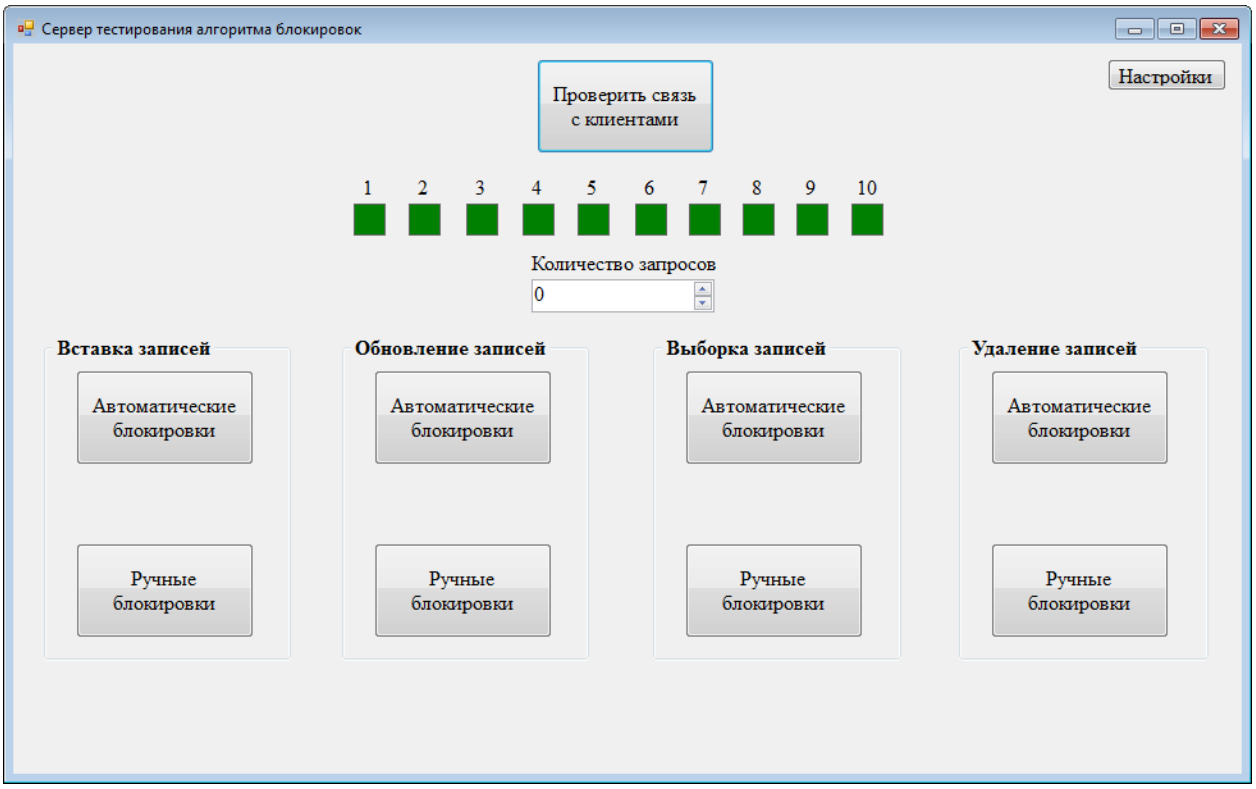


Рис. 2.9. Окно сервера тестирования алгоритма блокировок

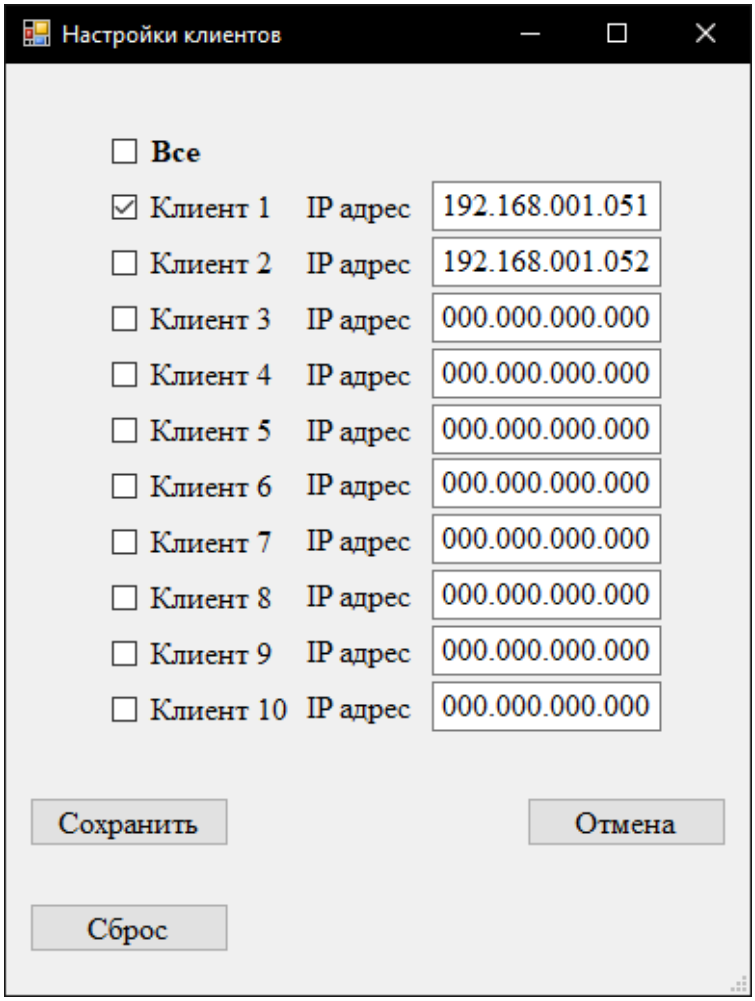


Рис. 2.10. Окно настройки клиентов

Клиентское же приложение является консольным (рис. 2.11) и сигнализирует о полученной команде от сервера, своём IP адресе, а также по клиенту можно отслеживать выполняет ли он ещё запросы.


Рис. 2.11. Окно клиентского приложения

При передаче команды на выполнения клиентам в качестве входных параметров передаются следующие значения:

1. Количество генерируемых запросов.
2. Тип запроса (вставка, обновление, выборка, удаление).
3. Номер клиента (чтобы визуально отмечать в базе данных записи от разных клиентов).
4. Автоматическая или ручная блокировка должна осуществляться при генерации запросов.

Клиент же возвращает серверу следующую информацию после выполнения:

1. Время, затраченное на выполнение запросов, в миллисекундах*100 (для более удобного отображения).
2. Номер клиента, закончившего выполнять запросы (для удобства мониторинга).

На языке C# были реализованы функции выполнения запросов как с автоматическими блокировками, так и с ручными. Ручными блокировками было решено выбрать блокировку страницы, так как это элементарная единица базы данных и её проще всего регулировать и добиться результатов отличных от автоматических блокировок.

Для того, чтобы сформировать команды клиентам для каждой кнопки была описана схожая функция, она представлена на рис. 2.12. При активации функции она считывает сохраненные настройки клиентов и в параллельном цикле одновременно выполняет класс SendMessageFromSocket для всех активных клиентов, передавая ему сформированное сообщение клиенту в формате: «IP адрес клиента» + «Количество запросов» +

«Вид выполняемого запроса». Для остальных кнопок функция схожа и отличается лишь передаваемой командой.

```
private void button1_Click(object sender, EventArgs e)
{

        string path = "setting.txt";
        using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))
        {
        sr.BaseStream.Position = 0;
        sr.DiscardBufferedData();
        string line, timeline, timeline1, timeline2;
            Parallel.For(1, 10, i =>
            {
                line = sr.ReadLine();
                timeline = line.Substring(line.IndexOf(" ") + 1);
                timeline = timeline.Replace(".00", ".");
                timeline = timeline.Replace(".0", ".");
                timeline1 = line.Substring(line.IndexOf("|") + 1, line.IndexOf(" ") - line.IndexOf("|") - 1);
                timeline2 = line.Substring(0, line.IndexOf("|"));
                if (timeline1 == "True")
                {
                    SendMessageFromSocket(4400, IPAddress.Parse(timeline), timeline2.ToString() +" "+ numericUpDown1.Value.ToString() + " insertauto");
                }
            });
        }

}
```

Рис. 2.12. Пример формирования команды клиентам

Для передачи команд клиентам, был создан класс SendMessageFromSocket (рис. 2.13). Он работает следующим образом, если полученное сообщение не состоит из одной лишь цифры (что сделано для проверки связи с клиентами), то отправляет сформированное сообщение клиенту по протоколу TCP на порт 4400 и IP клиента, после чего ожидает ответа.

```
private void SendMessageFromSocket(int port, IPAddress ip, string mycom)
{
    byte[] bytes = new byte[1024];
    IPAddress ipAddr = ip;
    IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);
    Socket sender = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
    sender.Connect(ipEndPoint);
    string message = mycom;
    int Num;
    if (int.TryParse(message, out Num))
    {
        byte[] msg = Encoding.UTF8.GetBytes(message);
        int bytesSent = sender.Send(msg);
        int bytesRec = sender.Receive(bytes);
        string answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);
        if (answer == "true")
            (this.Controls["status" + mycom] as PictureBox).BackColor = Color.Green;
    }
    else
    {
        Num = Convert.ToInt32(message.Substring(0, message.IndexOf(" ")));
        byte[] msg = Encoding.UTF8.GetBytes(message);
        int bytesSent = sender.Send(msg);
        int bytesRec = sender.Receive(bytes);
        string answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);
        (this.Controls["answer" + answer.Substring(0, answer.IndexOf("|"))] as Label).BeginInvoke((MethodInvoker)(() =>
        (this.Controls["answer" + answer.Substring(0, answer.IndexOf("|"))] as Label).Text = answer.Substring(answer.IndexOf("|") + 1)));
    }
    sender.Shutdown(SocketShutdown.Both);
    sender.Close();
}
```

Рис. 2.13. Код метода передачи

Рассмотрим, как происходит выполнение полученных команд сервера у клиента (рис.2.14). При старте клиент просит ввести имя сервера базы данных, после чего создаёт сервер TCP, чтобы получать команды. После чего ожидает их получение.

```
static void Main(string[] args)
{
    string ipserver = Console.ReadLine();
    var ipClient = Dns.GetHostEntry(Dns.GetHostName());
    IPAddress ipAddr = ipClient.AddressList[1];
    foreach (var ip in ipClient.AddressList)
    {
        if (ip.AddressFamily == AddressFamily.InterNetwork)
        {
            ipAddr = ip;
        }
    }
    IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 4400);
    Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

    try
    {
        sListener.Bind(ipEndPoint);
        sListener.Listen(10);
        while (true)
        {
            Console.WriteLine("Ожидаем соединение через порт {0}", ipEndPoint);
            Socket handler = sListener.Accept();
            string data = null;

            byte[] bytes = new byte[1024];
            int bytesRec = handler.Receive(bytes);
            data += Encoding.UTF8.GetString(bytes, 0, bytesRec);
            Console.Write("Полученный текст: " + data + "\n\n");
            int Num;
            string reply="";

            byte[] msg = Encoding.UTF8.GetBytes(reply);
            handler.Send(msg);
            handler.Shutdown(SocketShutdown.Both);
            handler.Close();
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    finally
    {
        Console.ReadLine();
    }
}
```

Рис. 2.14. Обработчик команд на клиенте и формирование ответа серверу

После получения команды он проверяет её содержание и выполняет соответствующую SQL команду (рис. 2.15). Клиент определяет на какую БД отправлять запросы, после чего начинает последовательно отправлять соответствующее количество запросов в цикле. Последовательная передача и

очистка кэша планов запросов сделана для того, чтобы максимально замедлить выполнение запросов, чтобы посчитать чистое время выполнения. Остальные запросы с автоматическими блокировками отличаются лишь видом команды SQL. В случае с ручной блокировкой в запрос добавляется конструкция WITH (PAGLOCK).

```
if (data.Contains("insertauto"))
{
    string valuetext = data.Substring(data.IndexOf(" ")+1, data.LastIndexOf(" ")-data.IndexOf(" "));
    string datatext = data.Substring(0, data.IndexOf(" "));
    string connectionString = "Data Source="+ipserver+"; User Id=client"+datatext+"; Password=1234; Initial Catalog=University";
    string sqlExpression = @"DBCC FREEPROCCACHE;
                             DBCC DROPCLEANBUFFERS;
            DECLARE @i INT;
            SET @i=0;
            WHILE @i<" + valuetext+@"
            BEGIN
                INSERT INTO University.dbo.Hobby (Name_hb) VALUES ('Test string" + data.Substring(0, data.IndexOf(" ")) + @"_'+CONVERT(NVARCHAR, @i));
                SET @i=@i+1;
            END
            ";
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.CommandTimeout = 3600;
        System.Diagnostics.Stopwatch sw = new Stopwatch();
        sw.Start();
        command.ExecuteNonQuery();
        sw.Stop();
        reply = datatext +"|"+(sw.ElapsedMilliseconds / 100.0).ToString();
    }
}
```

Рис. 2.15. Отправка запросов клиентом

Эксперименты для измерения скорости выполнения запросов выполнялся на 9000, 90000 запросах. На каждом было проведено 10 экспериментов, также первоначально использовалось 9 клиентов (1000 и 10000 запросов с каждого), после чего 5 клиентов (1800 с каждого), два клиента (4500 с клиента) и на одном клиенте с выполнением 9000 запросов. Всего было проведено 248 экспериментов с учётом каждого вида запроса и выбранного типа блокировки. Результаты для автоматического вида блокировок представлена в таблице 2.1.

Таблица 2.1 – Время выполнения 9000 запросов с автоматическими блокировками

| | Вставка | Обновление | Выборка | Удаление |
|---|---|---|---|---|
| 9 клиентов | 00:01,294 | 00:43,835 | 00:10,739 | 00:28,313 |
| 5 клиентов | 00:00,609 | 00:36,301 | 00:13,636 | 00:28,062 |
| 2 клиента | 00:01,837 | 00:41,025 | 00:16,573 | 00:29,578 |
| 1 клиент | 00:02,615 | 00:55,257 | 00:29,926 | 00:31,374 |

На рисунке 2.16 можно увидеть результат в графическом представлении. На нём видно, что при уменьшении количества клиентов время выполнения запросов существенно меняется лишь при выборке, а также, что с одним клиентом время выполнения всегда максимальное.

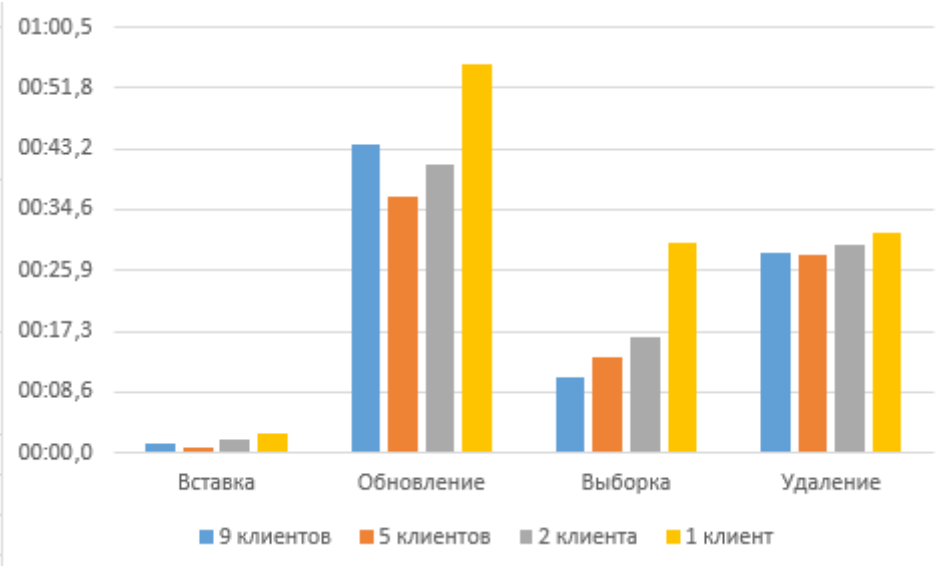


Рис. 2.16. Сравнение времени выполнения запросов при автоматических блокировках

Далее можно увидеть, как меняется время выполнения запросов при использовании ручных блокировок в таблице 2.2. А на рисунке 2.17 можно увидеть графическое представление этих данных. При ручных блокировках на

графике также видно, что при уменьшении клиентов скорость выполнения запросов увеличивается

Таблица 2.2 – Время выполнения 9000 запросов с ручными блокировками

| | Вставка | Обновление | Выборка | Удаление |
|---|---|---|---|---|
| 9 клиентов | 00:03,375 | 00:25,085 | 00:11,843 | 00:15,746 |
| 5 клиентов | 00:02,316 | 00:22,204 | 00:13,636 | 00:17,034 |
| 2 клиента | 00:02,493 | 00:24,884 | 00:13,636 | 00:18,030 |
| 1 клиент | 00:02,696 | 00:32,955 | 00:29,935 | 00:18,004 |

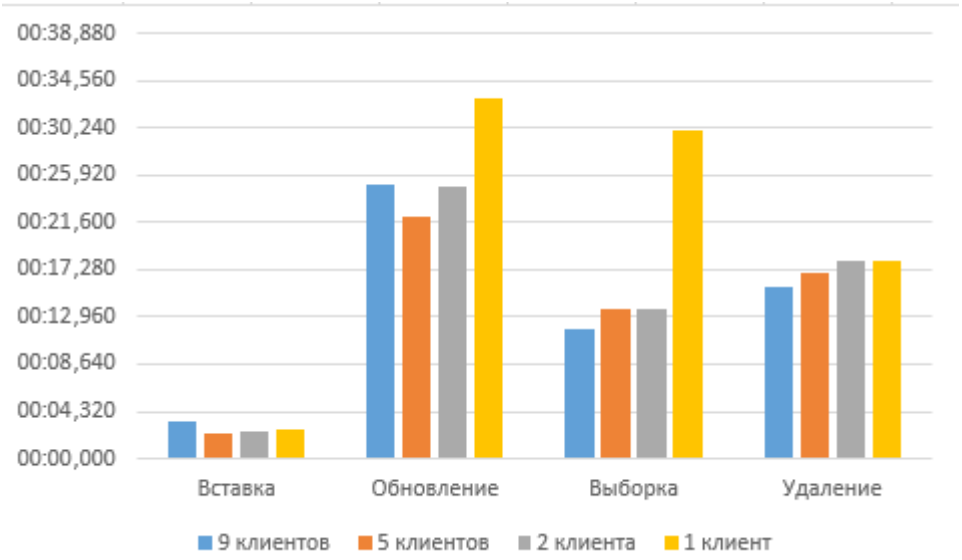


Рис. 2.17. Сравнение времени выполнения 9000 запросов при ручных блокировках

Таким образом, по результатам можно сказать, что при любых блокировках сохраняется параллельное исполнение, которое уменьшается при уменьшении соответственно клиентов. Также данный алгоритм генератора запросов очень медленный и предназначен только для небольших значений числа запросов.

## 2.4 Интерпретация результатов проведенного эксперимента

Используя созданный экспериментальный стэнд теперь есть две таблицы данных с разными блокировками. Основной рассматриваемый показатель – это время выполнения запроса. Поэтому теперь проведём эксперимент с большим количеством запросов и сравним автоматические и ручные блокировки.

Был проведен эксперимент по выполнению 90000 запросов. Данные о нём записаны в таблицу 2.3. А графическое представление на рисунке 2.18. Во-первых, здесь заметно существенное увеличение времени выполнения запросов, более чем в 100 раз, кроме вставки. А также, что ручные блокировки осуществляют выполнение запросов на удаление и обновление записей почти в два раза быстрее, но скорость вставки записей падает на 36%.

Таблица 2.3 – Время выполнения 90000 запросов с автоматическими и ручными блокировками

| | Вставка | Обновление | Выборка | Удаление |
|---|---|---|---|---|
| Автоматические | 00:14,506 | 56:37,627 | 21:45,119 | 59:53,002 |
| Ручные | 00:22,756 | 34:26,674 | 21:47,311 | 29:42,922 |

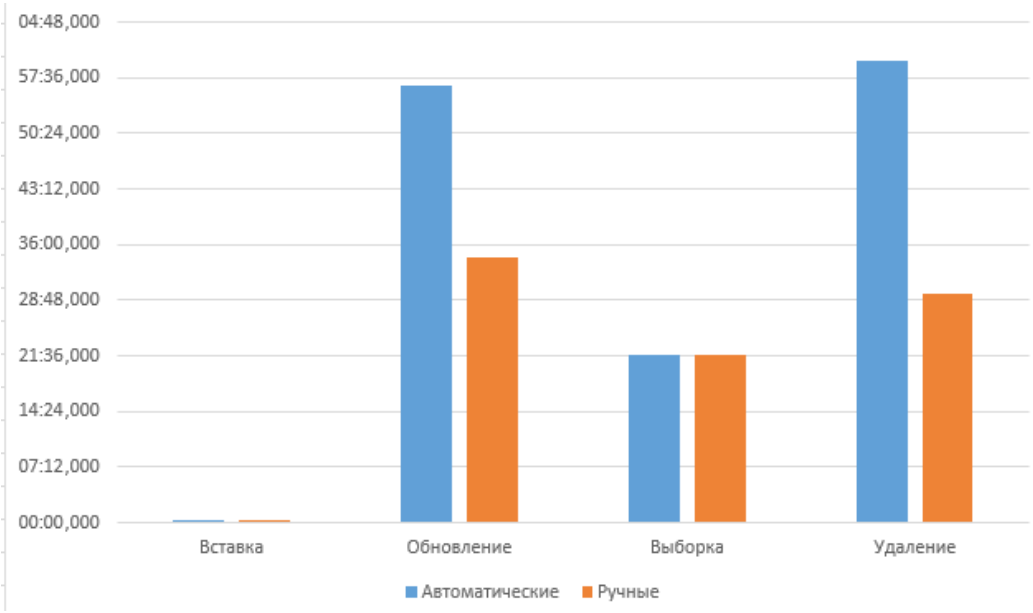


Рис. 2.18. Сравнение времени выполнения 9000 запросов при ручных и автоматических блокировках

На основе данных из таблицы 2.1 и 2.2 составим график (рис.2.19). На примере 9000 запросов

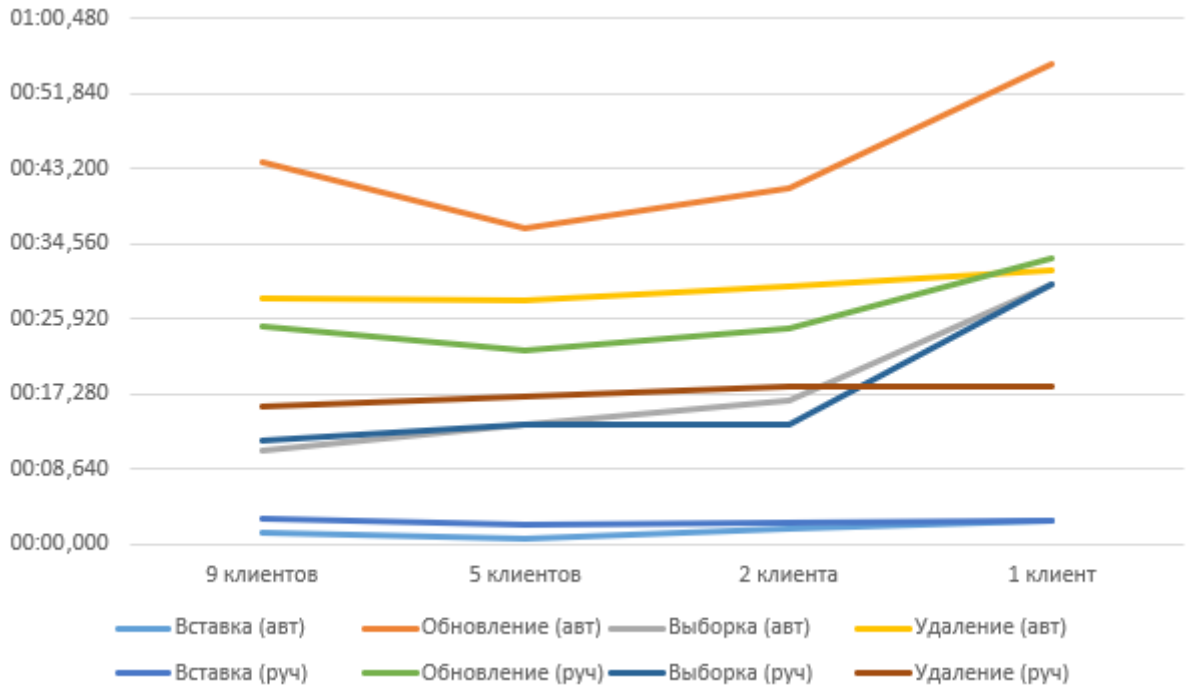


Рис. 2.19. Сравнение времени выполнения запросов между ручными и автоматическими блокировками

Использование ручных блокировок как видно из графика почти не меняет время выполнения запросов на вставку и выборку, но почти в два раза меньше потрачено времени на обновление и удаление записей. Так как эта закономерность остаётся и при 90000 запросах, можно сказать, что использование ручного механизма блокировок оправдано, так как ни одна запись при такой блокировке не пострадала и не было вызвано взаимных блокировок.

## Выводы

В данной главе была рассмотрена общая схема построения и условия создания адаптивного механизма ручных блокировок в СУБД MS SQL. Установлена ключевая цель использования конкретно ручных блокировок в централизованных базах данных, а именно увеличение уровня параллелизма при сохранении стабильности работы СУБД, а также предложен подход к решению этой проблемы, заключающийся в использовании в запросах табличного указания на блокировку страницы базы данных.

Разработан программный комплекс для тестирования эффективности адаптивного механизма ручных блокировок перед автоматическими в централизованной базе данных. Комплекс состоит из серверной и клиентской частей. Был проведен эксперимент, в ходе которого с помощью физических машин на сервер отправлялось множество запросов таких типов как выборка, вставка, обновление и удаление записей, как под автоматическими, так и с ручными блокировками.

Эксперимент проводился с различным количеством тестируемых машин, чтобы выявить влияние параллельной работы, и с разным количеством запросов, чтобы проверить стабильность работы базы данных и сохранение целостности данных. Запросы при этом выполнялись с отключением кэширования и максимальным возможным замедлением выполнения, чтобы создать как можно больше возможностей для конфликта блокировок и выявить его недостатки.

Полученные результаты времени выполнения запросов были записаны и проанализированы, а также

визуализированы на графиках для поиска закономерностей и выявления эффективности подхода с использованием механизма адаптивных ручных блокировок.

Результатом работы тестового стенда при различном количестве клиентов, генерирующих запросы, и при разном числе этих запросов стало получение того, что почти в два раза увеличивается скорость выполнения запросов на удаление и обновление записей при использовании блокировок страницы.

## Заключение

В результате данного исследования для проведенных экспериментов была разработана тестовая среда, используя современные методы тестирования и разработки приложений. Созданные генераторы могли выполняться параллельно с помощью технологии .Net framework языка программирования C#. Были протестированы генераторы запросов, которые успешно справились с задачей – имитацией наплыва запросов. Были проведены серии экспериментов, чтобы выявить отличия во времени выполнения автоматических и ручных блокировок. А также при разном количестве клиентов.

Итогом стало создание полностью рабочего стенда для проведения экспериментов для блокировок. И результатом работы стэнда стало получение результата того, что ручные блокировки при использовании блокировки страницы существенно уменьшают в два раза время выполнения запросов на обновление и удаление данных.

В дальнейшем исследование можно использовать для поиска более сложных механизмов ручных блокировок, адаптируемых к конкретной структуре баз данных, а также выявление возможных ошибок в уже найденном алгоритме ручной блокировки на большем количестве запросов.

**Библиографический список**

1.Аникин Н.А. Метод определения порядка сериализации транзакций в системах управления базами данных, использующих протокол строгой двухфазной блокировки // Труды МАИ. 2011. № 42. С. 19.

2.Богатырев В.А. Отказоустойчивость распределенных вычислительных систем динамического распределения запросов и размещение функциональных ресурсов // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. 2006. № 1. С. 1.

3.Богатырев В.А., Богатырев А.В. Оптимизация резервированного распределения запросов в кластерных системах реального времени // Информационные технологии. 2015. Т. 21. № 7. С. 495-502.

4.Богатырев В.А., Голубев И.Ю., Нестеров Д.А. Выбор вариантов организации распределения запросов в системах предоставления информационных услуг // Технико-технологические проблемы сервиса. 2013. № 1 (23). С. 43-46.

5.Богатырев В.А., Паршутина С.А. Многопутевое резервированное распределение через сеть критичных к задержкам запросов // Вестник компьютерных и информационных технологий. 2016. № 10 (148). С. 41-46.

6.Васильев А.П., Степанов С.Н. Исследование математической модели с динамическим распределением канального ресурса при групповом поступление запросов на передачу данных // В сборнике: Технологии информационного общества X Международная отраслевая научно-техническая конференция: сборник трудов. 2016. С. 16.

7.Герасимов А.А., Тихомирова А.Н. Проектирование транзакционной изолированности в различных типах приложений на примере базы данных Microsoft SQL Server // Молодежный научный вестник. 2017. № 6 (18). С. 145-154.

8.Головинский И.А. Блокировки автоматов групп. II. Взаимные блокировки // Известия Российской академии наук. Теория и системы управления. 2009. № 2. С. 72-83.

9.Голубев И.Ю., Богатырев В.А. Оптимизация распределения запросов в системе кластеров при сочетании аналитического и имитационного моделирования // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 5 (81). С. 79-83.

10. Григорьев Ю.А. Организация базы данных в программном комплексе анализа характеристик производительности распределённых систем обработки данных // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. 2012. № 2. С. 39.

11. Григорьев Ю.А., Плутенко А.Д., Бурдаков А.В., Цвященко Е.В. Анализ процессов согласования версий записей в базах данных NOSQL // Радиопромышленность. 2017. № 4. С. 125-134.

12. Евсеев Г.С., Лесникова О.С. Модель взаимодействия потоков транзакций на SQL-сервере // В сборнике: НАУЧНАЯ СЕССИЯ ГУАП Сборник докладов: в 3 частях. Под общей редакцией Ю. А. Антохиной. 2015. С. 197-200.

13. Ларкин Е.В., Ивутин А.Н. Определение временных интервалов в алгоритмах управления // Известия Томского политехнического университета. 2014. Т. 324. № 5. С. 6-12.

14. Лебеденко Е.В. Разработка алгоритма оптимального планирования распределения запросов в системах распределенного моделирования // Системы управления и информационные технологии. 2007. Т. 27. № 1-2. С. 240-243.

15. Мазеин К.В., Поняшова А.С., Безносов П.П., Козырь А.А., Храмов С.В. Причины блокировок в системе 1С: Предприятие и методы борьбы с ними // Аллея науки. 2017. Т. 1. № 12. С. 247-254.

16. Минязев Р.Ш. Распределение потока запросов в параллельных субд на платформе вычислительных кластеров // Нелинейный мир. 2012. Т. 10. № 3. С. 173-179.

17. Михеичев В. Взаимоблокировки Deadlock-сессий в Oracle // Системный администратор. 2016. № 3 (160). С. 38-42.

18. Михеичев В. Мониторинг блокировок в Oracle часть 2. Практический опыт диагностики блокировок // Системный администратор. 2015. № 12 (157). С. 41-43.

19. Михеичев В. Мониторинг блокировок в Oracle. Методы предупреждения и автоматического устранения // Системный администратор. 2015. № 4 (149). С. 30-34.

20. Омельяненко М.В., Папинашвили В.Г. Решение проблем параллельной обработки транзакций и выход из тупиковых ситуаций в базах данных // Молодой ученый. 2017. № 9 (143). С. 31-34.

21. Пулатов Ю.Г. Сравнение методов оптимистической и пессимистической блокировки при параллельном доступе к ресурсу базы данных // Решетневские чтения. 2018. Т. 2. С. 289-290.

22. Ризаев И.С., Кладиев А.В., Клевин А.С. Повышение эффективности выполнения транзакций при распределенной

обработке данных // Вестник Казанского государственного технического университета им. А.Н. Туполева. 2004. № 4. С. 41-45.

23. Скоба А.Н., Логанчук М.Л. Математическая модель функционирования распределённой информационной системы на базе архитектуры "файл – сервер" с учётом влияния блокировок // Инженерный вестник Дона. 2015. № 3 (37). С. 68.

24. Табличные указания (Transact-SQL) // Microsoft. Документация по SQL URL: https://docs.microsoft.com/ru-ru/sql/t-sql/queries/hints-transact-sql-table?view=sql-server-ver15 (дата обращения: 26.03.20).

25. Транзакции и блокировки // Технологии баз данных: SQL, T-SQL, PL/SQL, реляционные БД URL: http://datasql.ru/basesql/16.htm (дата обращения: 16.03.2020).

26. Уровни изоляции // Professor Web URL: https://professorweb.ru/my/sql-server/2012/level3/3_16.php (дата обращения: 26.03.20).

27. Форд Т. Анализ времени ожидания блокировок // Windows IT Pro/ RE. 2016. № 12. С. 46.

28. Черноморов Г.А. Модель функционирования корпоративной информационной системы централизованной архитектуры с учетом блокировок транзакций // Известия высших учебных заведений. Северо-Кавказский регион. Технические науки. 2003. № 2. С. 33-40.

29. All about locking in SQL Server // SQLShack URL: https://www.sqlshack.com/locking-sql-server/ (дата обращения: 26.03.20).

30. Beomseok Nam, Minho Shin, Henrique Andrade, Alan Sussman Multiple query scheduling for distributed semantic caches // Journal of Parallel and Distributed Computing, Volume 70, Issue 5, May 2010, Pages 598-611

31. D. Agrawal, A. Elabbadi Constrained Shared Locks for Increasing Concurrency in Databases // Journal of Computer and System Sciences Volume 51, Issue 1August 1995 Pages 53-63

32. Divyakant Agrawal, Amr El Abbadi Ordered sharing: A new lock primitive for database systems // Information Systems Volume 20, Issue 5 July 1995 Pages 361-392

33. Emanuele Carlini, Alessandro Lulli, Laura Ricci dragon: Multidimensional range queries on distributed aggregation trees // Future Generation Computer Systems, Volume 55, February 2016, Pages 101-115

34. G. Lausen, E. Soisalonsoininen Safety by Uninterpreted Locks // Information and Computation Volume 117, Issue 115 February 1995 Pages 37-49

35. Gray, Jim & Reuter, Andreas Distributed Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993. 375–437 c.

36. Hassen Fadoua, Touzi Grissa Amel Smart Query Optimization Approach In Distributed Environment // Procedia Computer Science, Volume 126, 2018, Pages 355-362

37. Iacob (Ciobanu) Nicoleta – Magdalena Distributed Queries in the E-learning Environment // Procedia – Social and Behavioral Sciences, Volume 28, 2011, Pages 241-245

38. Nikolay Golov, Lars Rönnbäck Big Data normalization for massively parallel processing databases // Computer

Standards & Interfaces Volume 54, Part 2 November 2017 Pages 86-93

39. Qin Xiao, Pang Li-Ping, Liu Jie, Li Sheng-Li Scheduling Real-Time Multi-Granularity Locks in Object-Oriented Database Systems // IFAC Proceedings Volumes Volume 31, Issue 14 June 1998 Pages 159-160

40. Takayuki Usui, Reimer Behrends, Jacob Evans, Yannis Smaragdakis Adaptive locks: Combining transactions and locks for efficient concurrency // Journal of Parallel and Distributed Computing Volume 70, Issue 10 October 2010 Pages 1009-1023

41. Vikash Mishra, Vikram Singh Generating Optimal Query Plans for Distributed Query Processing using Teacher-Learner Based Optimization // Procedia Computer Science, Volume 54, 2015, Pages 281-290

42. W Jun Semantic-based locking technique in object-oriented databases // Information and Software Technology Volume 42, Issue 815 May 2000 Pages 523-531

43. Weipeng Jing, Dongxue Tian An improved distributed storage and query for remote sensing data // Procedia Computer Science, Volume 129, 2018, Pages 238-247

44. Wenjie Liu, Zhanhuai Li An efficient theta-join query processing in distributed environment // Journal of Parallel and Distributed Computing, Volume 121, November 2018, Pages 42-52

45. Wojciech Cellary, Waldemar Wieczerzycki Locking in DAG-structured databases // Microprocessing and Microprogramming Volume 39, Issues 2–5 December 1993 Pages 161-164

46. Xiaoyong Li, Yijie Wang, Xiaoling Li, Xiaowei Wang, Jie Yu GDPS: An Efficient Approach for Skyline Queries over Distributed Uncertain Data // Big Data Research, Volume 1, August 2014, Pages 23-36

47. Yonggoo Choi, Songchun Moon Lightweight multigranularity locking for transaction management in XML database systems // Journal of Systems and Software Volume 78, Issue 1 October 2005 Pages 37-46

48. Yongluan Zhou, Beng Chin Ooi, Kian-Lee Tan, Wee Hyong Tok An adaptable distributed query processing architecture // Data & Knowledge Engineering, Volume 53, Issue 3, June 2005, Pages 283-309