ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/IRJ.2025.162.73

РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ДЛЯ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ПРОСТРАНСТВЕННЫХ ЗАПРОСОВ В POSTGRESQL С ИСПОЛЬЗОВАНИЕМ .NET И POSTGIS

Научная статья

Нуриев М.Г.[1, *], Пикулев А.Н.[2], Панченко О.В.[3], Лаптева М.Г.[4]
[1] ORCID : 0009-0003-0741-1734;
[1, 2, 3, 4] Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация

* Корреспондирующий автор (mgnuriev[at]kai.ru)

Аннотация

В статье представлена разработка программного модуля для параллельной обработки пространственных запросов в системе управления базами данных PostgreSQL с расширением PostGIS, реализованного на платформе .NET. Решение основано на паттерне «Производитель-Потребитель» и механизме кэширования результатов в формате GeoJSON, что позволяет значительно снизить нагрузку на систему управления базами данных и ускорить время отклика при повторных запросах. Описан процесс нормализации SQL-запросов, генерации хэш-кодов с использованием алгоритма SHA-256 и формирования выходных файлов. Проведён анализ существующих решений, выявлены их ограничения в области параллельной обработки запросов. Представлены результаты тестирования на различных пространственных базах данных, подтвердившие повышение производительности и масштабируемости системы. Рассмотрены перспективы дальнейшего развития модуля, включая распределённую архитектуру, интеллектуальное прогнозирование частоты запросов и расширение функциональности веб-интерфейса.

Ключевые слова: PostgreSQL, PostGIS, параллельная обработка, пространственные данные, .NET, кэширование, GeoJSON, производитель-потребитель, нормализация запросов, SHA-256, GIS, многопоточность.

DEVELOPMENT OF A SOFTWARE MODULE FOR PARALLEL PROCESSING OF SPATIAL QUERIES IN POSTGRESQL USING .NET AND POSTGIS

Research article

Nuriev M.G.[1, *], Pikulev A.N.[2], Panchenko O.V.[3], Lapteva M.G.[4]
[1] ORCID : 0009-0003-0741-1734;
[1, 2, 3, 4] Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation

* Corresponding author (mgnuriev[at]kai.ru)

Abstract

The article presents the development of a software module for parallel processing of spatial queries in the PostgreSQL database management system with the PostGIS extension, implemented on the .NET platform. The solution is based on the Producer-Consumer pattern and a mechanism for caching results in GeoJSON format, which significantly reduces the load on the database management system and speeds up response times for repeated queries. The process of normalising SQL queries, generating hash codes using the SHA-256 algorithm, and forming output files is described. An analysis of existing solutions was carried out, and their limitations in the area of parallel query processing were identified. The results of testing on various spatial databases are presented, confirming the increase in system performance and scalability. The prospects for further development of the module are discussed, including distributed architecture, intelligent prediction of query frequency, and expansion of web interface functionality.

Keywords: PostgreSQL, PostGIS, parallel processing, spatial data, .NET, caching, GeoJSON, producer — consumer, query normalisation, SHA-256, GIS, multithreading.

Введение

В современном мире геопространственные данные играют ключевую роль в различных областях, таких как транспорт, экология, урбанистика, логистика и управление природными ресурсами. С развитием технологий дистанционного зондирования Земли, GPS и Интернета вещей (IoT) объемы геоданных увеличиваются экспоненциально. Это создает необходимость в эффективных методах их обработки и анализа. Одной из наиболее популярных систем для работы с геопространственными данными является PostgreSQL с расширением PostGIS, которое предоставляет широкий набор функций для выполнения пространственных запросов [1], [2], [3].

PostGIS соответствует стандартам Open Geospatial Consortium (OGC) и поддерживает сложные операции: буферизацию, пересечения, вычисления площадей и расстояний [4], [5]. Преимуществами PostGIS являются полноценная поддержка требований ACID и транзакций, возможна интеграция с такими GIS-инструментами, как QGIS, GRASS, GeoServer.

Однако, несмотря на свои возможности, традиционные подходы к обработке запросов могут не справляться с увеличивающимися нагрузками, особенно в условиях многопользовательской среды. Разработка модуля, который позволяет обрабатывать несколько запросов параллельно, может значительно повысить эффективность работы с пространственными данными.

Новизна работы заключается в создании программного компонента на базе .NET для интеграции с PostgreSQL, обеспечивающего параллельную обработку пространственных запросов. Данный подход позволит оптимизировать использование ресурсов системы, а также сократить время отклика на запросы, что особенно важно для пользователей, работающих с большими объемами пространственной информации. В то же время разработка программного компонента на базе .NET обеспечит простоту внедрения в существующие программные комплексы, что существенно расширяет круг потенциальных пользователей решения.

Целью данной работы является разработка программного модуля, обеспечивающего параллельную обработку множества пространственных запросов в системе управления базами данных PostgreSQL.

Задачи работы:

- исследование существующих аналогов;
- разработка программного модуля;
- тестирование и отладка разработанного модуля на реальных базах данных.

Внедрение разрабатываемого модуля позволит значительно увеличить пропускную способность существующих геоинформационных систем без замены системы управления базами данных, что особенно важно в условиях ограниченных бюджетов. Решение найдет применение в транспортной логистике, градостроительном планировании, экологическом мониторинге и других областях, где требуется обработка больших объемов пространственных данных в режиме реального времени. В связи с развитием технологий искусственного интеллекта, которые основаны на обработке больших данных, эти исследования и разработки особенно актуальны.

Анализ аналогов

При выборе подхода к параллельной обработке запросов важно понимать, какие решения уже существуют и какие ограничения они имеют. Давайте рассмотрим три ключевых инструмента, которые часто используются в связке с PostgreSQL, но не решают проблему параллелизма в полной мере.

1. Поддержка параллелизма в PostgreSQL. Начиная с версии 9.6 СУБД PostgreSQL стала поддерживать параллельное выполнение запроса, что значительно улучшает производительность при обработке больших объемов данных, однако архитектурные ограничения не позволяют выполнять параллельную обработку нескольких независимых запросов в рамках одного однопользовательского соединения. Это фундаментальное ограничение связано с моделью выполнения запросов, где каждое соединение обслуживается отдельным процессом, обрабатывающим запросы строго последовательно, для обеспечения согласованности данных и предсказуемости работы. Это обеспечивает стабильность и соблюдение ACID-свойств, но накладывает определенные ограничения на параллелизм [6], [7].

2. Pgpool-II — это промежуточное программное обеспечение для PostgreSQL, который позволяет администраторам управлять пулами соединений БД и реализовывать репликацию данных между серверами БД. Pgpool-II работает как прокси-сервер между клиентскими приложениями и серверами PostgreSQL, перехватывая запросы от клиентов и направляя их к соответствующим серверам БД согласно настроенным правилам и политикам [8], [9]. Однако PgPool-II имеет и некоторые минусы, такие как:

- Дополнительный слой сложности: PgPool-II требует отдельной настройки и обслуживания как промежуточного прокси-сервера, что увеличивает инфраструктурные затраты. В то время как прямое управление соединениями через Npgsql в коде C# даёт полный контроль без зависимостей от внешних компонентов.
- Задержки для коротких запросов: для высоконагруженных сценариев с миллионами быстрых запросов PgPool-II может стать узким местом из-за накладных расходов на маршрутизацию. Прямые подключения через Npgsql минимизируют задержки.

3. PgAdmin и Dbeaver. Данные инструменты предоставляют удобные интерфейсы для работы с PostgreSQL с расширением PostGIS для работы с пространственными данными, но не поддерживают асинхронную параллельную обработку запросов. Данные программы в основном предназначены для синхронного выполнения запросов [10], [11], что означает, что каждый запрос выполняется последовательно, и вы ждете завершения одного запроса, прежде чем отправить следующий.

Как показал наш анализ, ни в одной из этих СУБД не реализованы технологии параллельной обработки запросов к базам данных.

Проектирование модуля

Проектная часть работы направлена на создание программного модуля, обеспечивающего параллельную обработку пространственных запросов в базу данных PostgreSQL. Основная цель — преодоление ограничений стандартных подходов к выполнению запросов в условиях высокой нагрузки и многопользовательского доступа.

Для полноценного представления работы программного модуля также будет проектироваться и разрабатываться упрощённая версия клиентской части системы «клиент-сервер», отображающая алгоритм работы со стороны обычного пользователя этой базы данных. Система «клиент-сервер» подразумевает использование клиент-серверной архитектуры — это модель организации вычислительных систем, в которой задачи распределены между клиентами и серверами. Основные компоненты:

- Клиенты — это устройства или приложения, которые запрашивают информацию или услугу у сервера. Клиенты могут быть как программными приложениями (например, веб-браузеры, мобильные приложения), так и аппаратными устройствами (например, смартфоны, планшеты, терминалы). В нашем случае клиентская часть системы будет отвечать за отправление запросов и визуализацию полученных ответов.
- Сервер — это компьютер, который предоставляет запрашиваемую информацию или услугу клиентам. Серверы могут выполнять различные функции, такие как хранение данных, обработка запросов, вычисления и т. д.

Для обеспечения повышенной производительности по сравнению с последовательным режимом работы, узким местом которого являются задержка и ожидание завершения работы над текущей задачей перед переходом к следующей, необходима бесперебойная работа системы, а именно параллельная асинхронная обработка запросов. Для достижения этой цели предлагается использовать такой паттерн, как «Производитель-Потребитель» — это шаблон проектирования, который организует параллельную обработку задач, разделяя систему на два ключевых компонента: производителей (producers) и потребителей (consumers). Производители — отвечают за генерацию данных или задач, помещают элементы в общий буфер или очередь. Потребители — отвечают за обработку данных или задач, извлекают элементы из общего буфера или очереди. Буфер/очередь — общая структура данных, которая выступает посредником между производителем и потребителем, хранит произведённые элементы до их потребления, обеспечивая баланс между скоростью их создания и обработки.

Несмотря на индивидуальность каждого пользователя и его причины для написания того или иного запроса к базе данных, есть не малая вероятность поступления одинаковых или же повторяющихся запросов, не обязательно идущих подряд. Они могут приходить с некими временными интервалами, будь то считанные секунды, минуты, а может и часы. По этой причине, а также и для достижения ещё большей производительности возникает необходимость кэширования результатов запросов. Кэширование представляет собой процесс хранения копий данных во временном хранилище (кэше) для быстрого доступа к ним в будущем. Таким образом, результат каждого запроса будет храниться в кэш-памяти сервера на протяжении заранее установленного времени, после которого будет удалён из кэш-памяти. При поступлении же этого запроса повторно до окончания этого промежутка времени, уже обработанный результат этого запроса будет немедленно передан клиенту без запроса к базе данных, при этом внутренний таймер данного запроса обновиться и отсчёт времени начнётся заново.

Для описания работы программы следует построить UML-диаграмму последовательности, но для начала давайте разберёмся, что она из себя представляет и как её читать. Диаграмма последовательности или же взаимодействия (interaction diagram) описывает взаимодействие групп объектов в различных условиях их поведения [12], [13]. Для написания нашей диаграммы последовательности понадобятся следующие элементы:

- Объекты (участники): представлены прямоугольниками с названием. Каждый объект соответствует компоненту системы.
- Линии жизни: вертикальные пунктирные линии, показывающие период активности объекта.
- Сообщения: стрелки между линиями жизни, обозначающие передачу или вызов методов.
- Фрагменты взаимодействия: блоки, структурирующие логику. Мы будем использовать такие блоки как:
- «par» (параллельно) — обозначает параллельное выполнение операций.
- «alt» (альтернативно) — указывает на условные ветвления.

Исходя из этого мы можем построить UML диаграмму (рисунок 1).

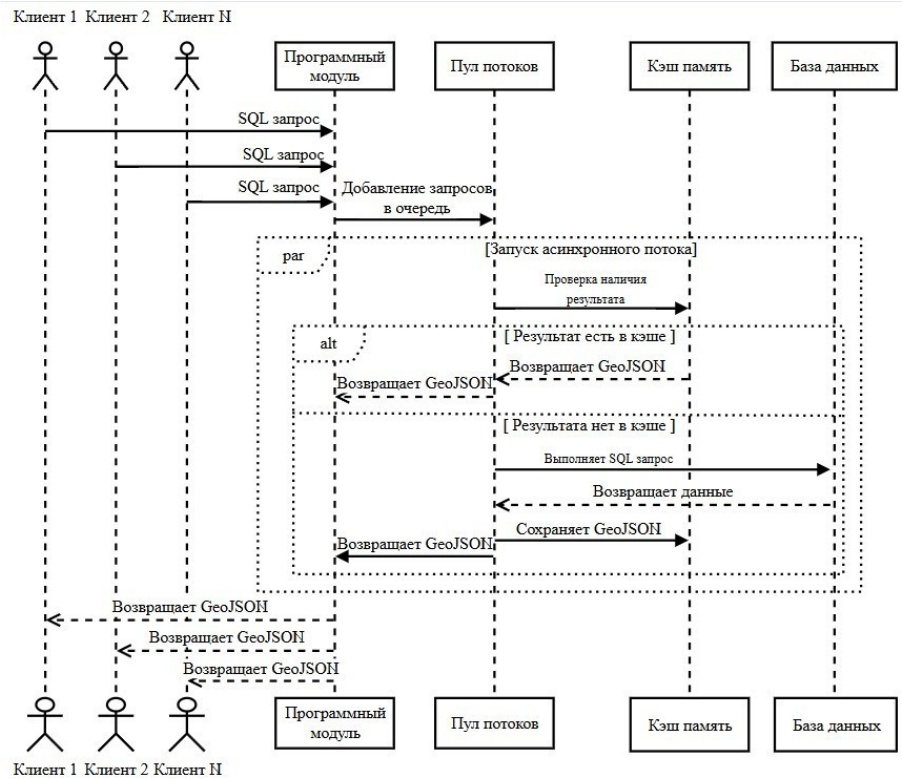

Рисунок 1 - Диаграмма последовательности алгоритма работы программы

DOI: https://doi.org/10.60797/IRJ.2025.162.73.1

Диаграмма последовательности демонстрирует архитектуру взаимодействия компонентов системы при параллельной обработке пространственных запросов. Этапы работы, представленные на диаграмме последовательности, протекают следующим образом.

Клиентские приложения (Клиент 1, Клиент 2 и Клиент N) инициируют работу системы, отправляя SQL запросы к пространственным данным. Пример пространственного SQL запроса представлен в листинге 1.

Листинг 1 — Пример пространственного SQL запроса

```
SELECT a.name AS city1,
      b.name AS city2,
      ST_Distance(a.geom::geography, b.geom::geography) / 1000 AS distance_km,
      ST_MakeLine(a.geom, b.geom) AS geom
FROM public.ne_10m_populated_placesz8 a,
    public.ne_10m_populated_placesz8 b
WHERE a.name < b.name AND a.name ='Kazan' AND b.name ='Moscow'
```

Подобные запросы поступают в программный модуль, который выступает центральным координатором системы. Важно отметить, что модуль принимает запросы от неограниченного количества клиентов одновременно, что подчёркивает масштабируемость решения.

Программный модуль выполняет критически важную функцию диспетчеризации — он не обрабатывает запросы самостоятельно, а распределяет их в пул потоков. Схематичное представление принципа работы пула потоков приведено на рисунке 2. Такой подход позволяет оптимально использовать вычислительные ресурсы сервера. Пул потоков обеспечивает истинную параллельность обработки — каждый запрос выполняется в отдельном потоке, при этом система автоматически балансирует нагрузку между доступными ядрами процессора.

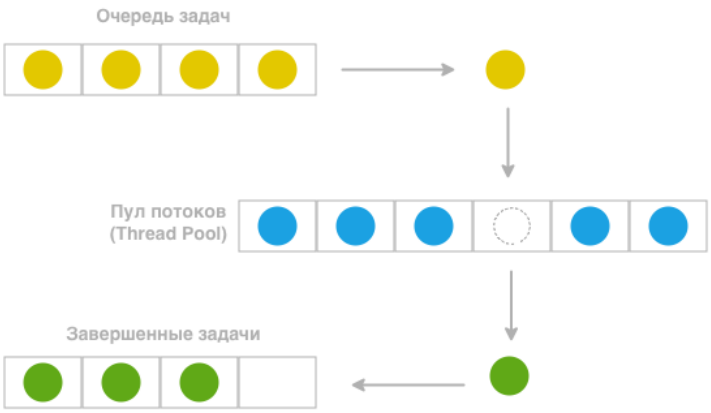

Рисунок 2 - Схема работы пула потоков
DOI: https://doi.org/10.60797/IRJ.2025.162.73.2

На следующем этапе каждый поток из пула читает свой SQL запрос, нормализует его, то есть приводит запрос к единому стандартному виду, чтобы устранить различия в форматировании (лишние пробелы, переносы строк, регистр). На основе нормализованного запроса формируется хэш-код [14], [15]. Примеры хэш-кодов, полученных кодированием по алгоритму SHA-256 представлены в таблице.

Таблица 1 - Примеры хэширования

DOI: https://doi.org/10.60797/IRJ.2025.162.73.3

Входные данные	Выход хэша
Меня зовут Тоби	cacb5418163039b016be9746818a2926f68fd1e4bad1b04f6791f6aabb5e8c52
Меня зовут Тони	9cd2444dc56929bdb97123add1f007643effa88bf1ed061eee1eead4e15ac7f9
Меня зовут Тоби, и это мой проект	9abbaa0c54fcd028ac51bede2608d06e8d3a026784e34adfac14fadd143d212c

На основе этого хэш-кода, как по ключу, происходит поиск готового результата в кэше. В результате этого поиска возможны два исхода:

1. Если результат найден, он выгружается из кэша и сохраняется в виде GeoJSON файла, пропуская этап с обращением к базе данных PostgreSQL + PostGIS и обработкой запроса.

2. Если совпадений по поиску нет — запрос выполняется в СУБД. Программа независимо взаимодействуют с СУБД PostgreSQL, расширенной функционалом PostGIS для работы с геоданными. Это взаимодействие включает выполнение пространственных запросов, например таких как расчёты расстояний, анализ пересечений географических объектов или выборки данных по пространственным критериям. После выполнения каждого запроса PostgreSQL возвращает результаты в табличном формате, которые дальше претерпевают преобразование полученных табличных данных в стандартизированный формат GeoJSON [16], [17], которые пул потоков передаёт обратно в основной программный модуль. Пример GeoJSON файла представлен в листинге 2.

Листинг 2 — Файл NewYork.geojson

```
{
 "type": "FeatureCollection",
 "features": [
  {
   "type": "Feature",
   "properties": {
    "name": "New York City"
   },
   "geometry": {
    "type": "Point",
    "coordinates": [
     -73.99571754361698,
     40.72156174972766
    ]
   }
  }
 ]
}
```

Завершающая стадия работы системы включает кэширование полученного GeoJSON файла. Использование кэширования сокращает время отклика для повторяющихся запросов и снижает нагрузку на базу данных. После чего готовые результаты в удобном для клиентов виде возвращаются каждому из отправителей запросов. Выбор GeoJSON формата обусловлен тем, что он поддерживается большинством ГИС-платформ, веб-картографических сервисов (таких как Leaflet, OpenLayers, Mapbox) и мобильных приложений [18], [19]. Это обеспечивает простую интеграцию модуля с существующими системами без необходимости дополнительных преобразований данных на стороне клиента.

Важно подчеркнуть, что вся обработка в потоках происходит асинхронно – клиенты получают результаты по мере готовности, без блокировки основного потока выполнения.

Реализация проекта

Основной задачей модуля является эффективная обработки множества запросов одновременно. Для этого была реализована система, основанная на паттерне «Производитель-Потребитель», где:

- производитель — мониторит директорию с SQL запросами и добавляет их в очередь;
- потребитель — представляет собой пул потоков, выполняющий запросы к базе данных.

Данный паттерн был реализован через механизмы библиотеки Task Parallel Library (TPL) и «BlockingCollection<Task>» в .NET, что обеспечивает потокобезопасную организацию очереди задач, а также позволяет эффективно распределять задачи между потоками, используя ресурсы процессора и задавать границы, ограничивая максимальную ёмкость коллекции [20], [21]. В рамках нашей реализации было ограничено количество одновременно выполняемых задач до 4. Производитель (листинг 3) реализован как цикл для мониторинга директории, который каждые 5 секунд вызывает функцию «ProcessAvailableQueriesAsync».

Листинг 3 — Цикл для проверки новых запросов

```
try {
    while (!_cts.Token.IsCancellationRequested) {
        await ProcessAvailableQueriesAsync(connectionString, queriesDirectory, outputDirectory);
        await Task.Delay(5000, _cts.Token);
    }
}
catch (OperationCanceledException) {
    Console.WriteLine("Работа приложения прервана...");
}
finally {
    _taskQueue.CompleteAdding();
    await Task.WhenAll(consumers);
}
```

Метод «ProcessAvailableQueriesAsync» (листинг 4) в свою очередь сканирует директорию с SQL запросами и создаёт задачи обработки, которые добавляются в очередь через «_taskQueue.Add()».

Листинг 4 — Метод «ProcessAvailableQueriesAsync»

```
static async Task ProcessAvailableQueriesAsync(string connectionString, string queriesDirectory, string
outputDirectory)
{
   var queryFiles = Directory.GetFiles(queriesDirectory, "*.sql");
   if (queryFiles.Length == 0) return;
   Console.WriteLine([__rj__content__placeholder__]quot;\nНайдено {queryFiles.Length} SQL-файлов для
обработки");
   foreach (var filePath in queryFiles)
   {
      _taskQueue.Add(ProcessSingleQueryFileAsync(connectionString, filePath, outputDirectory));
   }
}
```

Метод «.GetFiles()» в нашем коде принимает два параметра:

- путь к директории, который является обязательным параметром;
- шаблон поиска или же фильтр — параметр, позволяет фильтровать файлы по конкретным критериям, в данном случае по расширению файла «.sql».

Потребители (листинг 5) реализованы как пул из определённого числа фоновых задач (число задач устанавливается администратором), запускаемых через «Task.Run», которые, в свою очередь, непрерывно извлекают и выполняют задачи из очереди с помощью метода «GetConsumingEnumerable()», обеспечивая параллельное выполнение заданного числа запросов одновременно.

Листинг 5 — Потребитель задач

```
consumers[i] = Task.Run(async () =>
{
   foreach (var task in _taskQueue.GetConsumingEnumerable())
   {
      try
      {
         await task;
      }
      catch (Exception ex)
      {
         Console.WriteLine([__rj__content__placeholder__]quot;Ошибка при выполнении задачи:
{ex.Message}");
      }
   }
});
```

Нормализация SQL запросов и генерация хэша — критически важные этапы для корректной работы кэширования. Давайте рассмотрим их более подробно.

Нормализация запросов — это процесс приведения SQL запроса к унифицированному формату, который позволяет идентифицировать семантически идентичные запросы, даже если они имеют синтаксические различия [22], [23]. Это критически важно для корректной работ кэша, так как два логически одинаковых запроса с разными пробелами, регистром или переносами строк должны генерировать одинаковый ключ, а именно хэш-код.

Процесс нормализацию в свою очередь состоит из нескольких ключевых операций:

4.1. Удаление лишних пробелов и переносов строк

Любой SQL запрос может содержать случайные пробелы, табуляции или переносы строк, которые не оказывают никакого влияния на его смысл, но делают его уникальным для системы. Для примера рассмотрим два SQL запроса с незначащими пробелами и без незначащих пробелов (листинг 6):

Листинг 6 — Примеры запросов

```
-- запрос с незначащими пробелами
SELECT  *  FROM  cities  WHERE  population  >  1000000;

-- запрос без незначащих пробелов
SELECT * FROM cities WHERE population > 1000000;
```

Фактически это два одинаковых запроса, результаты обработки которых будут полностью идентичны, однако технически эти запросы различаются. Что приводит к излишней нагрузке системы и соответственно снижению производительности, которой можно было бы избежать. Для этого следует предпринять следующие действия:

- все последовательности пробелов заменить на один пробел;
- специальные управляющие последовательности (escape sequences), которые используются для записи пробельных (непечатных) символов (whitespace characters), а именно символы переноса строк «\r\n», «\n» и табуляции «\t» заменяются на один пробел;
- удаляются пробелы в начале и конце строки (для выполнения данной операции существует метод «.Trim()»).

Код реализующий эти действия представлен в листинге 7, где используется метод «Replace» класса «Regex», а также одноимённый метод, но и из класса «String». Использование метода «Regex.Replace()» с регулярным выражением «\s+» гарантирует, что между словами всегда будет не более одного пробела.

Листинг 7 — Удаление лишних пробелов и переносов строк

```
string normalizedQuery = Regex.Replace(
   query
      .Replace("\r\n", " ")
      .Replace("\n", " ")
      .Replace("\t", " "),  // Замена табуляции
   @"\s+",                  // Шаблон для любых пробельных символов
   " "                    // Замена на один пробел
).Trim();                   // Удаление пробелов по краям
```

4.2. Приведение к нижнему регистру

SQL запросы нечувствительны к регистру ключевых слов, таких как SELECT, FROM, WHERE и другие, то есть такие слова как SELECT и select являются эквивалентными. Приведение к нижнему регистру стандартизирует запросы, уменьшая количество всевозможных уникальных запросов и тем самым значительно снижая количество хранимых в кэш-памяти файлов-дубликатов с результатами одних и тех же запросов [24], [25]. Для примера рассмотрим 2 запроса представленных в листинге 8, в нём приведён SQL запрос до приведения к нижнему регистру и после.

Листинг 8 — Примеры запросов с разными регистрами

```
-- Пример запроса до приведения к нижнему регистру
SELECT Name FROM Cities;

-- Пример запроса после приведения к нижнему регистру
Select name from cities;
```

Однако стоит учесть, что это не затрагивает строковые литералы и идентификаторы в кавычках (например WHERE name = «Москва»), что важно для сохранения семантики. То есть запросы, где текст внутри кавычек написан в нижнем регистре и в верхнем регистре не будут считаться идентичными, а значит и результаты этих запросов могут отличаться, в зависимости от того был ли предусмотрен данный аспект в базе данных (листинг 9).

Листинг 9 — Пример запроса с нижним регистром

```
-- Пример запроса с нижним регистром в кавычках
SELECT * FROM Cities WHERE name = «Kazan»;

-- Пример запроса с верхним регистром в кавычках
SELECT * FROM Cities WHERE name = «KAZAN»;
```

4.3. Удаление комментариев

Комментарии в SQL запросах — это текстовые пояснения, которые игнорируются СУБД при выполнении, но могут влиять на уникальность хэша запроса. Например, два семантически одинаковых запроса с разными комментариями будут считаться разными, что приведёт к промахам кэша и избыточной нагрузке на БД. Удаление комментариев – важный этап нормализации для обеспечения корректной работы системы кэширования.

Примеры запросов до нормализации и после нормализации представлены в листингах 10 и 11 соответственно.

Листинг 10 — Пример запроса до нормализации

```
SELECT  a.name AS city1, -- Москва
      b.name AS city2, -- Нью Йорк
```

```
    ST_Distance(a.geom::geography, b.geom::geography) / 1000 AS distance_km, -- подсчёт расстояния
между двумя объектами a и b
    ST_MakeLine(a.geom, b.geom) AS geom /* для создания линии
                                    между этими двумя точками */
FROM   public.ne_10m_populated_placesz8 a,
    public.ne_10m_populated_placesz8     b

WHERE a.name < b.name AND   a.name ='Moscow'   AND b.name ='New York'
```

Листинг 11 – Запрос после нормализации

```
select_a.name_as_city1,_b.name_as_city2,_st_distance(a.geomgeography,_b.geomgeography)_1000_as_dista
nce_km,_st_makeline(a.geom,_b.geom)_as_geom_from_publi
c.ne_10m_populated_placesz8_a,_public.ne_10m_populated_placesz8_b_where_a.name_b.name_and_a.name
_='moscow'_and_b.name_='new_york'
```

Далее будет описан процесс по генерации хэш-кода запроса.

Генерация хэш-кода — завершающий этап подготовки SQL запроса к кэшированию. Этот процесс преобразует нормализованную строку запроса в уникальный идентификатор, который используется для быстрого поиска результатов в кэше. Основная задача хэш-функции — обеспечить минимальную вероятность коллизий (ситуаций, когда разные запросы получают одинаковый хэш) при сохранении высокой производительности.

Для кодирования был выбран алгоритм SHA-256 в качестве основы для генерации хэш-кода. Причина такого выбора обусловлена криптографической устойчивостью и широкому применению в современных системах. SHA-256 преобразует входные данные в 256-битный (32-байтовый) хэш, который практически исключает вероятность коллизии для различных входных строк. Такой подход сокращает объём полезных данных и ускоряет сравнение ключей, сохраняя при этом достаточный уровень уникальности.

Сам же процесс генерации хэш-кода начинается с преобразования полученной ранее нормализованной строки запроса в массив байтов с использованием кодировки UTF-8. Для выполнения данного преобразования используется метод «Encoding.UTF8.GetBytes(normalizedQuery)», где «normalizedQuery» — это строка полученная после всех этапов нормализации. Затем для полученного массива байтов вычисляется хэш-сумма с помощью метода «ComputeHash» экземпляра «sha256». Полученный массив «hashBytes»преобразуется в строку с помощью метода «BitConverter.ToString(hashBytes, 0, 8)». Входными параметрами данного метода являются:

- «hashBytes» — массив байтов;
- «0» — индекс символа начала строки;
- «8» — длина строки, которую мы хотим получить.

Выходным параметром является строка в шестнадцатеричном формате, разделённая дефисами. В дальнейшем для удаления этих дефисов мы применим метод с заданными входными параметрами «Replace(“-”, “”)», в результате чего получим непрерывную шестнадцатеричную строку, однако, чтобы уменьшить количество возможных вариантов и тем самым оптимизировать скорость проверки кэш на поиск файлов с этим же хэш-кодом, мы приведём все символы к нижнему регистру с помощью метода «ToLower()». В результате всех этих преобразований мы получим уникальный идентификатор для каждого запроса — хэш-код. Полный листинг функции по генерации хэш-кода представлен в листинге 12.

Листинг 12 — Генерация хэш-кода

```
string hashString;
using (var sha256 = SHA256.Create())
{
   byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(normalizedQuery));
   hashString = BitConverter.ToString(hashBytes, 0, 8).Replace("-", "").ToLower();
}
```

Сгенерированный хэш-код используется как часть имени файла для сохранения результатов в формате GeoJSON, что гарантирует, что повторяющиеся запросы будут ссылаться на один и тот же файл, даже если исходный текст запроса отличался незначительными синтаксическими вариациями. А с учётом того, что данный GeoJSON также будет храниться в кэше, то хэш-код, записанный в имени этого файла, будет служить ключом для проверки был ли уже выполнен данный запрос ранее.

Что же касается остальной части имени каждого GeoJSON файла, то она формируется из уже полученной нами нормализованной строки. Данная строка обрезается до 100 символов, из неё удаляются недопустимые символы, которые нельзя использовать при переименовании файла, а пробелы заменяются на подчёркивания. Описанное преобразование обеспечивает читаемость имени файла и добавляет дополнительный уровень защиты от конфликтов, например «использование недопустимых символов при переименовании файла». Для примера рассмотрим запрос, представленный в листинге 13.

Листинг 13 — Пример SQL запроса

```
select *
from   public.ne_10m_populated_placesz8
where  public.ne_10m_populated_placesz8.name='Kazan'
```

Этот запрос выбирает все поля для объекта из таблицы «ne_10m_populated_placesz8» из схемы «public», где поле «name» принимает значение «Kazan». Путём описанных выше преобразований мы получаем следующее имя для этого файла:

select_from_public.ne_10m_populated_placesz8_where_public.ne_10m_populated_placesz8.name_=_'Kazan'_51f3ca9b6 9a80c00

Полный код этого ключевого элемента отвечающий и за нормализацию и за генерацию хэш-кода, а также за формирование имени файла представлен в листинге 14. В этом элементе также используется функция «RemoveCommentsOptimized», отвечающая за удаление комментариев всех видов. На этом моменте реализация второго ключевого элемента отвечающего за нормализацию и генерацию хэша, а также за формирование имени выходного файла завершается.

Листинг 14 — Функция нормализации и генерации хэша

```
static string GenerateGeoJsonFileName(string query, string outputDirectory) {
   string withoutComments = RemoveCommentsOptimized(query);
   string normalizedQuery = Regex.Replace(
      withoutComments.Replace("\r\n", " ").Replace("\n", " ").Replace("\t", " "),@"\s+"," ").Trim().ToLower();
   string hashString;
   using (var sha256 = SHA256.Create())
   {
      byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(normalizedQuery));
      hashString = BitConverter.ToString(hashBytes, 0, 8).Replace("-", "").ToLower();
   }
   string safeQueryName = new string(normalizedQuery.Where(c => !
Path.GetInvalidFileNameChars().Contains(c)).ToArray()).Replace(" ", "_").Replace("__", "_").Trim('_');
   safeQueryName = safeQueryName.Length > 100 ? safeQueryName.Substring(0, 100) : safeQueryName;
   string fileName = [__rj__content__placeholder__]quot;{safeQueryName}_{hashString}.geojson";
   return Path.Combine(outputDirectory, fileName);
}
```

Взаимодействие с базой данных обязательный элемент данного программного модуля. Осуществляться оно будет с помощью библиотеки Npgsql, которая предоставляет асинхронные методы для выполнения запросов и работы с геопространственными данными. Также будет использоваться расширение «NetTopologySuite», интегрированное в «Npgsql». Данное расширение позволяет преобразовывать результаты запросов в пространственные объекты типа «Geometry» (точки, линии, полигоны), которые затем сериализуются в формат GeoJSON. Выполняться весь процесс взаимодействия будет в рамках метода «ExecuteQueryToGeoJsonAsync», входными параметрами которого являются строка подключения к базе данных и сам SQL запрос, а выходным параметром результант запроса в формате GeoJSON файла.

Настройка подключения начинается с создания экземпляра «NpgsqlDataSourceBuilder», где указывается строка подключения к базе данных:

Host=localhost;Username=postgres;Password=123;Database=DatabaseName

В данной строке указывается адрес сервера, в данном случае это локальный хост, имя пользователя для подключения к базе данных — «postgres», пароль пользователя — «123» и название конкретной базы данных — «DatabaseName», к которой происходит подключение.

После создания экземпляра и подключения к базе данных вызывается метод «UseNetTopologySuite()» из ранее упомянутого расширения для активации поддержки геопространственных типов данных PostGIS. Далее нужно обеспечить выполнение асинхронного подключения к базе данных, для этого будет использован метод «OpenConnectionAsync()», что позволяет не блокировать основной поток при ожидании ответа от СУБД. Установка «CommandTimeout = 300» гарантирует, что запросы длительностью до 300 секунд не будут прерваны раньше времени. Данную настройку следует изменить в зависимости от размеров базы данных и сложности запросов, в связи с чем она может настраиваться индивидуально для каждой базы данных.

После успешного выполнения запроса происходит чтение результата с помощью «NpgsqlDataReader», который последовательно обрабатывает каждую строку результата, и в цикле с помощью «await reader.ReadAsync()» извлекаются значения из столбцов. Если столбец содержит геометрию, то есть является типом «Geometry», то значения данного столбца сохраняются отдельно для последующего преобразования в GeoJSON, все остальные атрибуты добавляются в коллекцию свойств «properties» объекта.

Преобразование результатов в GeoJSON выполняется с использованием метода «Wrtie(geometry)» класса «GeoJsonWriter» из «NetTopologySuite». Данный метод сериализует геометрию в JSON-совместимый формат, а

благодаря «JsonSerializer» из класса «Systems.Text.Json» собирает полный объект «FeatureCollection» – объект JSON, содержащий коллекцию функцию GeoJSON.

Настройка «UnsafeRelaxedJsonEscaping» отключает экранирование символов, то есть замена в тексте управляющих (служебных) символов на соответствующие им последовательности символов, что важно для корректного отображения координат в веб-клиентах.

Также для представления полной картины взаимодействия с базой данных с клиентской стороны был реализован небольшой сайт способный отправлять SQL запросы и принимать GeoJSON результаты, при этом визуализируя их на карте мира.

Алгоритм действий для работы с сайтом следующий:

1. При первом заходе на сайт вас встречает домашняя страница (рисунок 3). На ней есть 2 кнопки «Login» и «Register», при нажатии на которые вы перейдёте на страницу для входа или регистрации соответственно. Мы же для примера перейдём по кнопке «Register».

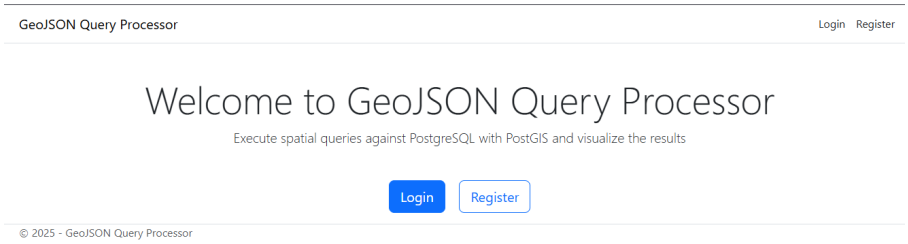

Рисунок 3 - Домашняя страница сайта
DOI: https://doi.org/10.60797/IRJ.2025.162.73.4

2. На странице регистрации нас встречают 3 поля, которые нужно заполнить для прохождения регистрации (рисунок 4). После успешной регистрации мы перейдём на страницу входа.

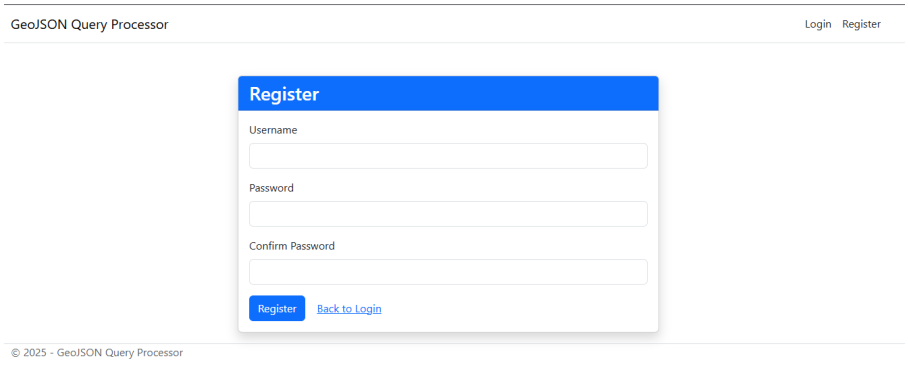

Рисунок 4 - Страница регистрации
DOI: https://doi.org/10.60797/IRJ.2025.162.73.5

3. На странице входа в профиль мы должны повторить введённые нами при регистрации данные и успешно войти в аккаунт (рисунок 5). Все данные об аккаунтах сохраняются в отдельный JSON файл, а их пароли шифруются, что позволяет избежать их потери даже при перезапуске сервера.

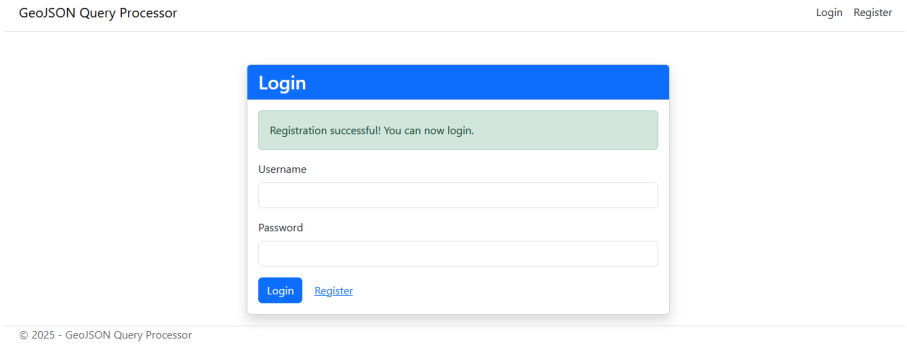

Рисунок 5 - Страница для входа в аккаунт
DOI: https://doi.org/10.60797/IRJ.2025.162.73.6

4. После входа в аккаунт вас встречает поле для ввода SQL запроса и кнопка «Execute Query», то есть выполнить запрос (рисунок 6). Для примера введём следующий запрос (листинг 15) и нажмём кнопку.

Листинг 15 — Пример пространственного SQL запроса

```
select *
from public."ne_10m_admin_1 z8"
where public."ne_10m_admin_1 z8".name='Grodno'
```

Данный запрос ищет в базе данных «NaturalEarth» в таблице «ne_10m_admin_1 z8» объект, у которого в поле «name» содержится запись «Grodno».

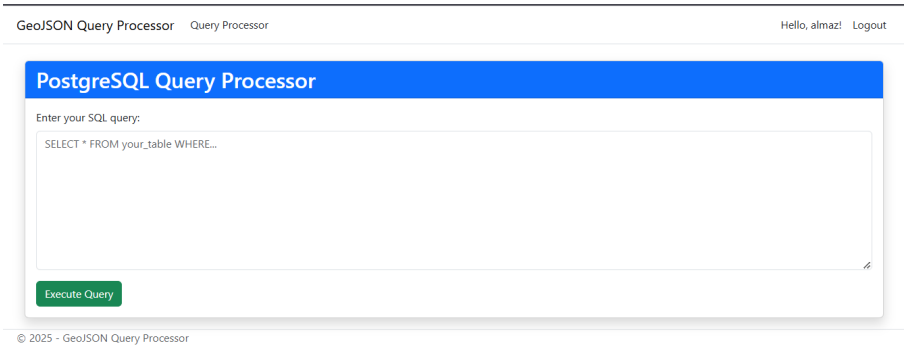

Рисунок 6 - Страница для обработки запросов
DOI: https://doi.org/10.60797/IRJ.2025.162.73.7

5. По прошествии короткого периода времени на нашей странице появится карта Земли содержащая поверх которой выделен результат нашего запроса (рисунок 7). Для отображения результата на карте внутри веб-сайта была использована библиотека «Leaflet».

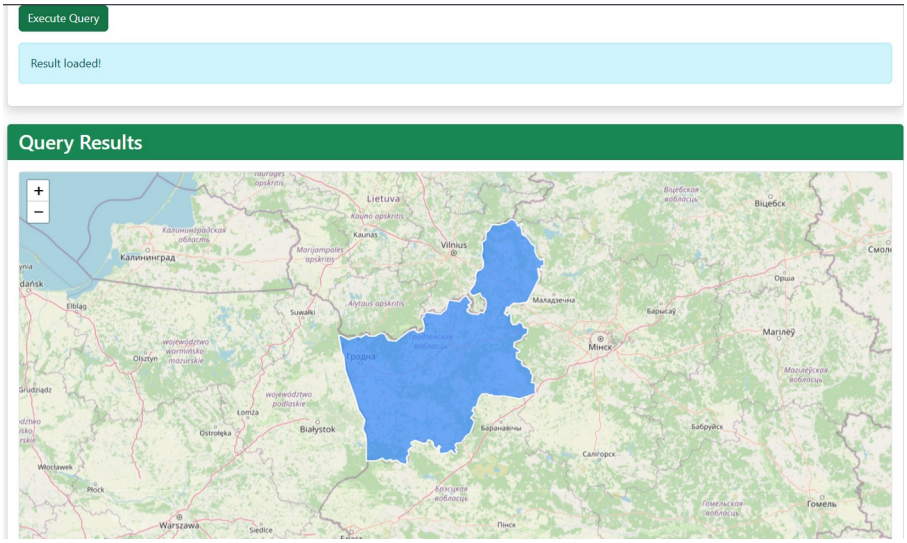

Рисунок 7 - Визуализация результата SQL запроса
DOI: https://doi.org/10.60797/IRJ.2025.162.73.8

Как видно из рисунка 7, данный объект является полигоном и представляет собой город Гродно, расположенный в Республике Беларусь. Если же мы нажмём мышкой на этот полигон появится список его свойств из базы данных (рисунок 8).

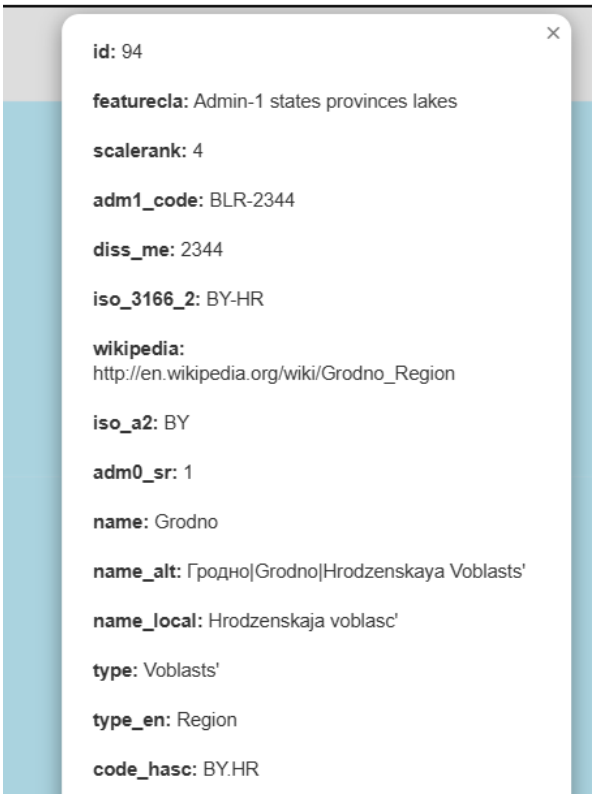

Рисунок 8 - Список свойств выбранного объекта
DOI: https://doi.org/10.60797/IRJ.2025.162.73.9

Тестирование

Разработанный программный модуль был протестирован на производительность и корректность работы ключевых компонентов. Основное внимание уделялось проверке параллельной обработке запросов и эффективности кэширования. Для тестирования были использованы различные базы данных PostgreSQL с расширением PostGIS, а именно:

- База данных «NaturalEarth» содержащая информацию о городах, водоёмах, островах и многих других объектах планеты Земля. Источником этих данных является электронный ресурс «Natural Earth». Общий вес базы данных составляет 193 МБ.

- База данных «Volga-fed-district» содержащая значительные объёмы пространственные данных Приволжского федерального округа. Источником этих данных является электронный ресурс «download.geofabrik.de» предоставляющий свободный доступ к пространственным данным OpenStreetMap. Общий вес базы данных составляет 4138 МБ.

Для тестирования производилась практически единовременная отправка 20 запросов, в которых было 3 разных повторяющихся запроса. Запросы относились к разным данным из разных таблиц для каждой базы данных. При этом перед тестированием был выполнен 1 запрос для установления корректного подключения к базе данных, данный запрос не повторялся ни с одним из запросов, которые использовались для тестов.

Результаты этих тестов следующие:

1. Для базы данных «NaturalEarth». Время выполнения уникальных запросов составляет в среднем 214 миллисекунд на запрос, что превышает показатели последовательного режима обработки запросов (в среднем 127 миллисекунд на запрос) из-за дополнительных операций нормализации, генерации хэш-кода и управления потоками. Однако следует учитывать, что общее время последовательных запросов равно сумме времени выполнения всех запросов, в то время как в параллельном режиме работы общее время обработки всех запросов значительно ниже и практически равна времени обработки самого долгого процесса. Таким образом, за время выполнения в среднем 2–3 последовательных запроса, в параллельном же режиме уже успеют обработаться большая часть всех запросов, а при повышении вероятности повторяющихся запросов этот процесс в параллельном режиме будет выполняться ещё быстрее, так как вместо тех же 214 миллисекунд на обработку запроса результат, которого имеется в кэше, затрачивается всего 5 миллисекунд.

2. Для базы данных «Volga-fed-district». Время выполнения 20 запросов разной степени сложности с 3 повторяющимися запросами в последовательном режиме суммарно составляет 14,802 секунды, в то время как выполнение этих же запросов в параллельном режиме, то есть при практически одновременной отправке этих запросов через сайт, в среднем происходит за 5,365 секунд. Это подтверждает эффективность реализации параллелизма и кэширования: повторяющиеся запросы, результаты которых уже находились в кэше, обрабатывались мгновенно, а распределение нагрузки между потоками позволило сократить общее время выполнения в 2,76 раза. Максимальное время обработки одного сложного запроса в параллельном режиме составило 5,565 секунды, что свидетельствует о корректной работе системы с ресурсоёмкими операциями.

Результаты демонстрируют, что модуль успешно масштабируется для работы с большими объёмами данных, обеспечивая стабильную производительность даже при высокой нагрузке.

Заключение

В ходе проведённой работы была разработана программная система для параллельной обработки пространственных запросов в системе управления базами данных PostgreSQL с использованием расширения PostGIS и платформы .NET. Предложенное решение основано на применении паттерна «Производитель-Потребитель» в сочетании с механизмом кэширования результатов в формате GeoJSON, что позволило существенно снизить нагрузку на базу данных и обеспечить высокую скорость отклика при повторных обращениях.

Проведённое тестирование показало, что разработанный модуль демонстрирует высокую масштабируемость и стабильную производительность даже при интенсивном потоке запросов. Для крупных наборов пространственных данных удалось сократить общее время обработки более чем в два раза по сравнению с последовательным режимом, а при высоком количестве повторяющихся запросов производительность возрастала ещё значительнее благодаря мгновенной выдаче уже кэшированных результатов.

К числу основных достоинств разработанной системы относится возможность интеграции в существующую инфраструктуру без замены используемой базы данных, поддержка большого числа одновременных клиентских подключений, использование универсального формата передачи данных GeoJSON, обеспечивающего совместимость с популярными геоинформационными платформами и веб-сервисами, а также оптимизация работы за счёт нормализации запросов и предотвращения избыточных вычислений.

Перспективы дальнейшего развития решения заключаются в создании распределённой архитектуры, обеспечивающей ещё более высокую пропускную способность и отказоустойчивость; внедрении алгоритмов интеллектуального анализа данных, позволяющих прогнозировать частоту запросов и заранее формировать кэш; расширении возможностей веб-интерфейса с реализацией визуального построения пространственных запросов; совершенствовании системы хранения кэша с применением высокопроизводительных технологий и механизмов репликации; внедрении гибкой приоритезации запросов в зависимости от их значимости; создании подсистемы мониторинга и аналитики, позволяющей отслеживать производительность, выявлять узкие места и адаптировать параметры работы модуля в зависимости от текущей нагрузки.

Таким образом, разработанное решение не только обеспечивает значительное ускорение обработки пространственных запросов, но и формирует основу для построения масштабируемых геоинформационных систем, способных эффективно функционировать в условиях постоянного роста объёмов геоданных и ужесточения требований к скорости их анализа.

Конфликт интересов

Не указан.

Рецензия

Рудой Е.М., ООО «ГК «Иннотех», Москва Российская Федерация

DOI: https://doi.org/10.60797/IRJ.2025.162.73.10

Conflict of Interest

None declared.

Review

Rudoi E.M., Innotech Group LLC, Moscow Russian Federation

DOI: https://doi.org/10.60797/IRJ.2025.162.73.10

Список литературы / References

1. Ельшин Н.И. Методы обработки пространственных данных / Н.И. Ельшин, В.В. Куралесин // Молодежь и наука: шаг к успеху : Сборник научных статей 8-й Всероссийской научной конференции перспективных разработок молодых ученых. В 4-х томах, Курск, 20–21 марта 2025 года. — Курск: Университетская книга, 2025. — С. 126–129.

2. Иванченкова А.М. Тенденции развития геоинформационных систем / А.М. Иванченкова, Е.А. Нартова // Молодежный вектор развития аграрной науки : Материалы 76-й национальной научно-практической конференции студентов и магистрантов, Воронеж, 14 февраля 2025 года. — Воронеж: Воронежский государственный аграрный университет имени Императора Петра I, 2025. — С. 121–124.

3. Екубджонов Д.И. Исследование производительности технологий объектно-реляционного отображения при взаимодействии с Microsoft SQL Server / Д.И. Екубджонов, Р.Ф. Гибадуллин // Международный научно-исследовательский журнал. — 2024. — № 10 (148). — DOI: 10.60797/IRJ.2024.148.145.

4. Беляков С.Л. Прецедентный анализ образов в интеллектуальных геоинформационных системах / С.Л. Беляков, М.Л. Белякова, М.Н. Савельева // Информационные технологии. — 2013. — № 7. — С. 22–25.

5. Vaslavskaya I. The Use of Blockchain Technology for Transport and Logistics Systems in the Digital Economy / I. Vaslavskaya, I. Koshkina, R. Zaripova [et al.] // Finance, Economics, and Industry for Sustainable Development. — Springer Cham, 2024. — DOI: 10.1007/978-3-031-56380-5_16.

6. Тимофеева Н.Е. Универсальный алгоритм обработки запросов с использованием технологии параллельных вычислений / Н.Е. Тимофеева, К.А. Дмитриева // Научно-технический вестник Брянского государственного университета. — 2018. — № 2. — С. 211–217. — DOI: 10.22281/2413-9920-2018-04-02-211-217.

7. Шигабетдинова Д.И. Обнаружение и локализация утечек на нефтедобывающих объектах с помощью компьютерного зрения / Д.И. Шигабетдинова, З.М. Гизатуллин, М.П. Шлеймович // Вестник Технологического университета. — 2025. — Т. 28. — № 5. — С. 123–128. — DOI: 10.55421/3034-4689_2025_28_5_123.

8. Супрунов С. PostgreSQL: функции и триггеры / С. Супрунов // Системный администратор. — 2004. — № 10 (23). — С. 42–47.

9. Шарипов Р.Р. Разработка программного комплекса потокового шифра RC4 для обучающихся по дисциплине «криптография» / Р.Р. Шарипов, С.П. Макаров, А.А. Кассирова // Международный научно-исследовательский журнал. — 2024. — № 9 (147). — DOI: 10.60797/IRJ.2024.147.15.

10. Пашинин О.В. Оптимизация запросов к базам данных / О.В. Пашинин // Математические структуры и моделирование. — 2007. — № 17. — С. 100–107.

11. Шкиндеров М.С. Моделирование помехоустойчивости системы контроля и управления доступом при воздействии электростатического разряда / М.С. Шкиндеров, Р.Р. Мубараков // Труды МАИ. — 2021. — № 120. — DOI: 10.34759/trd-2021-120-12.

12. Новиков Ф.А. Язык описания диаграмм / Ф.А. Новиков, К.Б. Степанян // Информационно-управляющие системы. — 2007. — № 4 (29). — С. 28–36.

13. Шурдилов И.С. Разработка системы распознавания эмоций по лицевым выражениям на основе машинного обучения / И.С. Шурдилов, М.Г. Нуриев, М.Г. Лаптева [и др.] // Международный научно-исследовательский журнал. — 2025. — № 6 (156). — DOI: 10.60797/IRJ.2025.156.52.

14. Корсунов Н.И. Защита информации баз данных, хранящихся на удаленных серверах / Н.И. Корсунов, А.И. Титов // Вопросы радиоэлектроники. — 2012. — Т. 4. — № 1. — С. 180–186.

15. Катасёв А.С. Нейронечеткая модель и программный комплекс автоматизации формирования нечетких правил для оценки состояния объектов / А.С. Катасёв // Автоматизация процессов управления. — 2019. — № 1 (55). — С. 21–29.

16. Лебедев А.Ю. Создание централизованного каталога алгоритмов обработки геоданных / А.Ю. Лебедев, А.Е. Березко // Геоинформатика. — 2010. — № 2. — С. 67–70.

17. Шакирзянов Р.М. Метод автоматического позиционирования беспилотных аппаратов на основе распознавания сигнальных радиально-симметричных маркеров подводных целей / Р.М. Шакирзянов, М.П. Шлеймович, С.В. Новикова // Автоматика и телемеханика. — 2023. — № 7. — С. 93–120. — DOI: 10.31857/S0005231023070061.

18. Гинзбург И.Б. Отказоустойчивые веб-интерфейсы для геоинформационных систем с использованием данных дистанционного зондирования Земли / И.Б. Гинзбург // Научно-технический вестник Поволжья. — 2016. — № 4. — С. 72–75.

19. Смирнов Ю.Н. Математическая модель оптимизации деятельности для цифровой системы управления предприятием / Ю.Н. Смирнов, А.В. Каляшина // Научно-технический вестник Поволжья. — 2023. — № 11. — С. 119–122.

20. Гибадуллин Р.Ф. Оптимизация асинхронных операций в .NET / Р.Ф. Гибадуллин, Д.А. Гашигуллин // Международный научно-исследовательский журнал. — 2025. — № 7(157). — DOI: 10.60797/IRJ.2025.157.40.

21. Хабибуллин Ф.Ф. Анализ параметров ошибки положения, перемещения идеального и реального механизма для робототехнических систем / Ф.Ф. Хабибуллин, Р.Т. Исламов, Л.Ф. Хабибуллина // Проблемы машиностроения и автоматизации. — 2024. — № 1. — С. 36–43.

22. Норенков И.П. Извлечение знаний из текстовых документов на основе концептно-ориентированной типизации запросов / И.П. Норенков, М.Ю. Уваров // Информационные технологии. — 2012. — № 4. — С. 14–17.

23. Brigida V. Technogenic Reservoirs Resources of Mine Methane When Implementing the Circular Waste Management Concept / V. Brigida, V.I. Golik, E.V. Voitovich [et al.] // Resources. — 2024. — Vol. 13. — Art. 33. — DOI: 10.3390/resources13020033.

24. Щеглов А.Ю. Механизм переадресации запросов к файловым объектам / А.Ю. Щеглов, К.А. Щеглов // Вопросы защиты информации. — 2004. — № 2 (65). — С. 41–43.

25. Райхлин В.А. Конструктивное моделирование систем информатики / В.А. Райхлин, И.С. Вершинин, Р.Ш. Минязев [и др.]; под ред. В.А. Райхлина. — Казань: Фэн, 2016. — 312 с.

Список литературы на английском языке / References in English

1. Yelshin N.I. Metodi obrabotki prostranstvennikh dannikh [Methods of spatial data processing] / N.I. Yelshin, V.V. Kuralesin // Molodezh i nauka: shag k uspekhu : Sbornik nauchnikh statei 8-i Vserossiiskoi nauchnoi konferentsii perspektivnikh razrabotok molodikh uchenikh. V 4-kh tomakh, Kursk, 20–21 marta 2025 goda [Youth and science: a step towards success: Collection of scientific articles from the 8th All-Russian Scientific Conference on Promising Developments by Young Scientists. In 4 volumes, Kursk, 20–21 March 2025]. — Kursk: University Book, 2025. — P. 126–129. [in Russian]

2. Ivanchenkova A.M. Tendentsii razvitiya geoinformatsionnikh sistem [Trends in the development of geographic information systems] / A.M. Ivanchenkova, Ye.A. Nartova // Molodezhnii vektor razvitiya agrarnoi nauki : Materiali 76-i natsionalnoi nauchno-prakticheskoi konferentsii studentov i magistrantov, Voronezh, 14 fevralya 2025 goda [Youth vector of agricultural science development: Proceedings of the 76th National Scientific and Practical Conference of Students and Master's Students, Voronezh, 14 February 2025]. — Voronezh: Voronezh State Agrarian University named after Emperor Peter I, 2025. — P. 121–124. [in Russian]

3. Yekubdzhonov D.I. Issledovanie proizvoditelnosti tekhnologii obektno-relyatsionnogo otobrazheniya pri vzaimodeistvii s Microsoft SQL Server [Study of the performance of object-relational mapping technologies when interacting with Microsoft SQL Server] / D.I. Yekubdzhonov, R.F. Gibadullin // Mezhdunarodnii nauchno-issledovatelskii zhurnal [International Research Journal]. — 2024. — № 10 (148). — DOI: 10.60797/IRJ.2024.148.145. [in Russian]

4. Belyakov S.L. Pretsedentnii analiz obrazov v intellektualnikh geoinformatsionnikh sistemakh [Precedent analysis of images in intelligent geographic information systems] / S.L. Belyakov, M.L. Belyakova, M.N. Saveleva // Informatsionnie tekhnologii [Information Technologies]. — 2013. — № 7. — P. 22–25. [in Russian]

5. Vaslavskaya I. The Use of Blockchain Technology for Transport and Logistics Systems in the Digital Economy / I. Vaslavskaya, I. Koshkina, R. Zaripova [et al.] // Finance, Economics, and Industry for Sustainable Development. — Springer Cham, 2024. — DOI: 10.1007/978-3-031-56380-5_16.

6. Timofeeva N.E. Universalnii algoritm obrabotki zaprosov s ispolzovaniem tekhnologii parallelnikh vichislenii [Universal algorithm for processing queries using parallel computing technology] / N.E. Timofeeva, K.A. Dmitrieva // Nauchno-tekhnicheskii vestnik Bryanskogo gosudarstvennogo universiteta [Scientific and Technical Bulletin of Bryansk State University]. — 2018. — № 2. — P. 211–217. — DOI: 10.22281/2413-9920-2018-04-02-211-217. [in Russian]

7. Shigabetdinova D.I. Obnaruzhenie i lokalizatsiya utechek na neftedobivayushchikh obektakh s pomoshchyu kompyuternogo zreniya [Detection and localisation of leaks at oil production facilities using computer vision] / D.I. Shigabetdinova, Z.M. Gizatullin, M.P. Shleimovich // Vestnik Tekhnologicheskogo universiteta [Bulletin of the Technological University]. — 2025. — Vol. 28. — № 5. — P. 123–128. — DOI: 10.55421/3034-4689_2025_28_5_123. [in Russian]

8. Suprunov S. PostgreSQL: funktsii i triggeri [PostgreSQL: Functions and Triggers] / S. Suprunov // Sistemnii administrator [System Administrator]. — 2004. — № 10 (23). — P. 42–47. [in Russian]

9. Sharipov R.R. Razrabotka programmnogo kompleksa potokovogo shifra RC4 dlya obuchayushchikhsya po distsipline «kriptografiya» [Development of the RC4 stream cipher software package for students studying cryptography] / R.R. Sharipov, S.P. Makarov, A.A. Kassirova // Mezhdunarodnii nauchno-issledovatelskii zhurnal [International Research Journal]. — 2024. — № 9 (147). — DOI: 10.60797/IRJ.2024.147.15. [in Russian]

10. Pashinin O.V. Optimizatsiya zaprosov k bazam dannikh [Optimisation of database queries] / O.V. Pashinin // Matematicheskie strukturi i modelirovanie [Mathematical structures and modelling]. — 2007. — № 17. — P. 100–107. [in Russian]

11. Shkinderov M.S. Modelirovanie pomekhoustoichivosti sistemi kontrolya i upravleniya dostupom pri vozdeistvii elektrostaticheskogo razryada [Modelling of noise immunity of access control and management systems under the influence of electrostatic discharge] / M.S. Shkinderov, R.R. Mubarakov // Trudi MAI [Proceedings of MAI]. — 2021. — № 120. — DOI: 10.34759/trd-2021-120-12. [in Russian]

12. Novikov F.A. Yazik opisaniya diagramm [Diagram description language] / F.A. Novikov, K.B. Stepanyan // Informatsionno-upravlyayushchie sistemi [Information and control systems]. — 2007. — № 4 (29). — P. 28–36. [in Russian]

13. Shurdilov I.S. Razrabotka sistemi raspoznavaniya emotsii po litsevim virazheniyam na osnove mashinnogo obucheniya [Development of a system for recognising emotions based on facial expressions using machine learning] / I.S. Shurdilov, M.G. Nuriev, M.G. Lapteva [et al.] // Mezhdunarodnii nauchno-issledovatelskii zhurnal [International Research Journal]. — 2025. — № 6 (156). — DOI: 10.60797/IRJ.2025.156.52. [in Russian]

14. Korsunov N.I. Zashchita informatsii baz dannikh, khranyashchikhsya na udalennikh serverakh [Protection of database information stored on remote servers] / N.I. Korsunov, A.I. Titov // Voprosi radioelektroniki [Issues of Radio Electronics]. — 2012. — Vol. 4. — № 1. — P. 180–186. [in Russian]

15. Katasyov A.S. Neironechetkaya model i programmnii kompleks avtomatizatsii formirovaniya nechetkikh pravil dlya otsenki sostoyaniya obektov [Neural fuzzy model and software package for automating the formation of fuzzy rules for assessing the condition of objects] / A.S. Katasyov // Avtomatizatsiya protsessov upravleniya [Automation of management processes]. — 2019. — № 1 (55). — P. 21–29. [in Russian]

16. Lebedev A.Yu. Sozdanie tsentralizovannogo kataloga algoritmov obrabotki geodannikh [Creation of a centralised catalogue of geodata processing algorithms] / A.Yu. Lebedev, A.E. Berezko // Geoinformatika [Geoinformatics]. — 2010. — № 2. — P. 67–70. [in Russian]

17. Shakirzyanov R.M. Metod avtomaticheskogo pozitsionirovaniya bespilotnikh apparatov na osnove raspoznavaniya signalnikh radialno-simmetrichnikh markerov podvodnikh tselei [Method for automatic positioning of unmanned aerial vehicles based on recognition of radial-symmetrical signal markers of underwater targets] / R.M. Shakirzyanov, M.P. Shleimovich, S.V. Novikova // Avtomatika i telemekhanika [Automation and Telemechanics]. — 2023. — № 7. — P. 93–120. — DOI: 10.31857/S0005231023070061. [in Russian]

18. Ginzburg I.B. Otkazoustoichivie veb-interfeisi dlya geoinformatsionnikh sistem s ispolzovaniem dannikh distantsionnogo zondirovaniya Zemli [Fault-tolerant web interfaces for geographic information systems using Earth remote sensing data] / I.B. Ginzburg // Nauchno-tekhnicheskii vestnik Povolzhya [Scientific and Technical Bulletin of the Volga Region]. — 2016. — № 4. — P. 72–75. [in Russian]

19. Smirnov Yu.N. Matematicheskaya model optimizatsii deyatelnosti dlya tsifrovoi sistemi upravleniya predpriyatiem [Mathematical model for optimising operations for a digital enterprise management system] / Yu.N. Smirnov, A.V. Kalyashina // Nauchno-tekhnicheskii vestnik Povolzhya [Scientific and Technical Bulletin of the Volga Region]. — 2023. — № 11. — P. 119–122. [in Russian]

20. Gibadullin R.F. Optimizatsiya asinkhronnikh operatsii v .NET [Optimisation of asynchronous operations in .NET] / R.F. Gibadullin, D.A. Gashigullin // Mezhdunarodnii nauchno-issledovatelskii zhurnal [International Research Journal]. — 2025. — № 7(157). — DOI: 10.60797/IRJ.2025.157.40. [in Russian]

21. Khabibullin F.F. Analiz parametrov oshibki polozheniya, peremeshcheniya idealnogo i realnogo mekhanizma dlya robototekhnicheskikh sistem [Analysis of position error parameters, movement of ideal and real mechanisms for robotic systems] / F.F. Khabibullin, R.T. Islamov, L.F. Khabibullina // Problemi mashinostroeniya i avtomatizatsii [Problems of Mechanical Engineering and Automation]. — 2024. — № 1. — P. 36–43. [in Russian]

22. Norenkov I.P. Izvlechenie znanii iz tekstovikh dokumentov na osnove kontseptno-orientirovannoi tipizatsii zaprosov [Extracting knowledge from text documents based on concept-oriented query typification] / I.P. Norenkov, M.Yu. Uvarov // Informatsionnie tekhnologii [Information Technologies]. — 2012. — № 4. — P. 14–17. [in Russian]

23. Brigida V. Technogenic Reservoirs Resources of Mine Methane When Implementing the Circular Waste Management Concept / V. Brigida, V.I. Golik, E.V. Voitovich [et al.] // Resources. — 2024. — Vol. 13. — Art. 33. — DOI: 10.3390/resources13020033.

24. Shcheglov A.Yu. Mekhanizm pereadresatsii zaprosov k failovim obektam [Mechanism for redirecting requests to file objects] / A.Yu. Shcheglov, K.A. Shcheglov // Voprosi zashchiti informatsii [Information Security Issues]. — 2004. — № 2 (65). — P. 41–43. [in Russian]

25. Raikhlin V.A. Konstruktivnoe modelirovanie sistem informatiki [Constructive modelling of information systems] / V.A. Raikhlin, I.S. Vershinin, R.Sh. Minyazev [et al.]; ed. by V.A. Raikhlin. — Kazan: Fen, 2016. — 312 p. [in Russian]