УДК 004.65

Н. Б. Бактыбекова, nonaor17@gmail.com
С. В. Корякин, srgkoryakin1@gmail.com
Э.А. Джалбиев, emirbekdzhalbiev@gmail.com
Кыргызско-Германский институт прикладной информатики

МЕТОДЫ ОБНАРУЖЕНИЯ И УСТРАНЕНИЯ ПРОБЛЕМ ПРОИЗВОДИТЕЛЬНОСТИ ДЛЯ СИСТЕМ МОНИТОРИНГА И АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ ЗАПРОСОВ POSTGRESQL

Данная работа посвящена разработке системы мониторинга и автоматической оптимизации баз данных PostgreSQL. Рассмотрены существующие инструменты мониторинга PostgreSQL, проведён анализ их возможностей и ограничений. Особое внимание уделено методам обнаружения медленных запросов, анализу использования индексов, выявлению проблем с производительностью, а также современным подходам к оптимизации запросов и настройке параметров СУБД. Представлена разработанная автоматизированная система, включающая модуль сбора метрик, анализатор производительности и генератор рекомендаций по оптимизации. Проведено тестирование системы на реальных данных, продемонстрировавшее повышение производительности на 35–40% после применения автоматических рекомендаций. В заключении приведены рекомендации по внедрению системы мониторинга и автоматической оптимизации PostgreSQL в производственные среды.

Ключевые слова: PostgreSQL, мониторинг производительности, автоматическая оптимизация, pg_stat_statements, медленные запросы, индексация, анализ производительности, система мониторинга, метрики базы данных, оптимизация запросов, EXPLAIN ANALYZE, настройка параметров, анализ планов выполнения

Введение

PostgreSQL является одной из наиболее популярных систем управления базами данных с открытым исходным кодом. По данным DB-Engines за 2025 год, PostgreSQL занимает 4-е место среди всех СУБД с рыночной долей 16.85% и продолжает демонстрировать стабильный рост популярности [1]. Согласно исследованию Stack Overflow 2025, PostgreSQL используется 45.55% разработчиков, опередив MySQL (41.09%) [2]. Система применяется в крупных компаниях, таких как Instagram, Reddit, Spotify и NASA, при этом около 11.9% компаний с доходом более $200 миллионов используют PostgreSQL в производственных системах [3].

Несмотря на высокую надёжность и производительность PostgreSQL, многие организации сталкиваются с проблемами производительности из-за отсутствия систематического мониторинга и оптимизации. Проблема производительности баз данных становится особенно критичной при росте объёмов данных и увеличении нагрузки. Согласно исследованиям, до 80% проблем производительности приложений связаны с неэффективными запросами к базе данных. При этом многие организации обнаруживают проблемы только после того, как они начинают влиять на работу пользователей.

В связи с этим возрастает актуальность разработки автоматизированных систем мониторинга и оптимизации, способных выявлять проблемы на ранних стадиях и предлагать рекомендации по их устранению. В данной работе представлена разработанная автоматизированная система мониторинга и оптимизации PostgreSQL, включающая модуль сбора метрик, анализатор производительности и генератор рекомендаций. Система позволяет снизить нагрузку на администраторов баз данных, предотвратить деградацию производительности и обеспечить стабильную работу приложений. В последующих разделах описана архитектура системы, алгоритм её работы, методика тестирования и полученные результаты, демонстрирующие улучшение производительности на 35–40%.

Методы мониторинга PostgreSQL. PostgreSQL предоставляет обширный набор встроенных инструментов для мониторинга производительности [4]. Основными источниками информации о состоянии базы данных являются системные представления (system views) и расширения, собирающие статистику выполнения запросов.

Встроенные инструменты мониторинга

pg_stat_statements. Это расширение является ключевым инструментом для мониторинга производительности запросов в PostgreSQL [5]. Оно отслеживает статистику планирования и выполнения всех SQL-запросов, выполняемых сервером. pg_stat_statements предоставляет информацию о количестве вызовов каждого запроса, общем времени выполнения, среднем времени выполнения, минимальном и максимальном времени, стандартном отклонении, количестве прочитанных и записанных блоков данных, а также статистику по использованию кэша [6].

Для активации расширения необходимо добавить его в параметр shared_preload_libraries в файле postgresql.conf и перезапустить сервер. После этого расширение создаётся командой CREATE EXTENSION pg_stat_statements в нужной базе данных. По умолчанию система хранит статистику по 5000 уникальным запросам, но этот параметр может быть увеличен при необходимости [7].

pg_stat_activity. Это представление показывает информацию о текущих процессах базы данных в режиме реального времени [8]. Оно отображает активные соединения, выполняемые запросы, состояние каждого процесса (active, idle, idle in transaction), время начала выполнения запроса, PID процесса, имя пользователя и базы данных. pg_stat_activity особенно полезно для выявления долго выполняющихся запросов, блокировок и проблем с подключениями.

pg_stat_database. Представление содержит статистику на уровне баз данных, включая количество транзакций (commits и rollbacks), количество возвращённых строк, количество блокировок, размер кэша, статистику по операциям чтения/записи, количество deadlocks, размер temporary файлов и время последнего выполнения VACUUM и ANALYZE [9].

pg_stat_user_tables и pg_stat_user_indexes. Эти представления предоставляют детальную статистику по таблицам и индексам: количество sequential scans и index scans, количество вставленных, обновлённых и удалённых строк, количество live и dead tuples, информацию о последнем VACUUM и ANALYZE, размер bloat [10]. Данная информация критична для оценки эффективности использования индексов и необходимости выполнения обслуживания таблиц.

EXPLAIN и EXPLAIN ANALYZE. Команда EXPLAIN показывает план выполнения запроса, выбранный оптимизатором PostgreSQL, включая типы операций (Sequential Scan, Index Scan, Bitmap Heap Scan и др.), оценочную стоимость и количество строк [11]. EXPLAIN ANALYZE дополнительно выполняет запрос и предоставляет фактическое время выполнения каждой операции и реальное количество обработанных строк, что позволяет сравнить оценки планировщика с реальностью и выявить неоптимальные планы выполнения.

Сторонние инструменты мониторинга

Помимо встроенных средств PostgreSQL, существует множество сторонних инструментов для мониторинга производительности, предоставляющих расширенные возможности визуализации, анализа и алертинга [12].

pgAdmin. Наиболее популярный графический инструмент управления PostgreSQL. Предоставляет базовые возможности мониторинга, включая просмотр активных соединений, выполняемых запросов, статистики по таблицам и индексам, а также графический интерфейс для анализа планов запросов. Является бесплатным, имеет кроссплатформенную поддержку и активно развивается сообществом.

pgBadger. Специализированный инструмент для анализа лог-файлов PostgreSQL [13]. Генерирует подробные HTML-отчёты с информацией о самых медленных запросах, наиболее частых ошибках, пиковых нагрузках, распределении типов запросов и других метриках производительности. Поддерживает анализ больших лог-файлов (гигабайты), работает с различными форматами логирования и позволяет строить исторические графики.

Grafana + Prometheus. Комбинация инструментов для визуализации метрик в реальном времени [14]. Prometheus собирает метрики PostgreSQL через специальный экспортер

(postgres_exporter), который опрашивает системные представления PostgreSQL с заданным интервалом. Grafana отображает собранные метрики в виде графиков, дашбордов и позволяет настраивать алерты при превышении пороговых значений.

pgWatch2. Самостоятельное решение для мониторинга PostgreSQL с предустановленными дашбордами Grafana и встроенным хранилищем метрик. Не требует установки расширений или прав суперпользователя для базовой функциональности. Поддерживает мониторинг множества инстансов PostgreSQL из единого интерфейса [15].

pgDash. Коммерческое SaaS-решение или self-hosted вариант для комплексного мониторинга PostgreSQL. Предоставляет детальную информацию о каждом запросе, включая планы выполнения, временные ряды, рекомендации по оптимизации [16].

Методы оптимизации производительности PostgreSQL. Оптимизация производительности PostgreSQL включает широкий спектр подходов на различных уровнях: от оптимизации отдельных запросов до настройки параметров сервера и архитектурных изменений [17]. Эффективная оптимизация требует комплексного подхода и глубокого понимания работы СУБД.

Оптимизация SQL-запросов

Анализ планов выполнения. Первым шагом оптимизации запроса является анализ его плана выполнения с помощью EXPLAIN ANALYZE [18]. План показывает, какие операции выполняет PostgreSQL: Sequential Scan (полное сканирование таблицы), Index Scan (использование индекса), Bitmap Heap Scan (комбинированное использование индекса и таблицы), Hash Join или Merge Join (соединения таблиц). Сравнение estimated rows с actual rows позволяет выявить неточности статистики, которые могут приводить к выбору неоптимальных планов.

Переписывание запросов. Часто запросы можно оптимизировать путём переписывания: использование EXISTS вместо IN для больших подзапросов, замена подзапросов на JOIN, использование UNION ALL вместо UNION, когда дубликаты не критичны, избежание SELECT * и выборка только необходимых колонок, использование LIMIT для ограничения результатов, применение фильтров WHERE как можно раньше в запросе, избежание функций в WHERE для индексируемых колонок.

При оптимизации запросов важно учитывать эффективность различных подходов к проверке существования записей. В таблице 1 представлено сравнение основных методов проверки существования данных в PostgreSQL, их производительность и рекомендации по применению в различных сценариях.

Таблица 1 – Сравнение методов проверки существования

Метод	Производительность	Применение	Примечание
EXISTS	Высокая	Проверка существования	Останавливается при первом совпадении
IN с подзапросом	Средняя	Небольшой список значений	Материализует весь подзапрос
IN со списком	Высокая	Константный список	Преобразуется в OR
JOIN	Средняя-Высокая	Нужны данные из обеих таблиц	Может вернуть дубликаты

Как видно из таблицы 1, оператор EXISTS демонстрирует наивысшую производительность при проверке существования записей, поскольку прекращает выполнение при обнаружении первого совпадения, в то время как конструкция IN с подзапросом вынуждена материализовать полный результирующий набор перед проверкой [18].

Стратегии индексирования. Правильное использование индексов является ключевым фактором производительности PostgreSQL [19]. Индексы представляют собой вспомогательные структуры данных, позволяющие существенно ускорить операции поиска и фильтрации за счёт дополнительных накладных расходов на обслуживание при

модификации данных. В таблице 2 представлена сравнительная характеристика основных типов индексов, поддерживаемых PostgreSQL, их операторов и областей применения.

Таблица 2 – Сравнительная характеристика типов индексов PostgreSQL

Тип индекса	Поддерживаемые операторы	Типичное применение	Размер индекса	Скорость создания
B-tree	<, <=, =, >=, >, BETWEEN, IN	Универсальный, числа, текст, даты	Средний	Средняя
Hash	=	Точное совпадение, длинные строки	Средний	Быстрая
GIN	@>, ?, ?&, ?|, @@	Массивы, JSONB, полнотекстовый поиск	Большой	Медленная
GiST	<<, &<, &>, >>, @>, <@	Геометрия, диапазоны, полнотекстовый поиск	Средний	Средняя
BRIN	<, <=, =, >=, >	Очень большие таблицы с корреляцией	Очень малый	Очень быстрая

Анализ таблицы 2 показывает, что выбор типа индекса критически влияет на производительность запросов. B-tree индексы, являющиеся стандартным типом в PostgreSQL, обеспечивают оптимальную производительность для большинства операций сравнения и сортировки. GIN индексы, несмотря на больший размер и более медленное создание, незаменимы для полнотекстового поиска и работы с JSON-документами. BRIN индексы демонстрируют исключительную эффективность для очень больших таблиц с естественной корреляцией данных, занимая на порядок меньше дискового пространства по сравнению с B-tree индексами.

Настройка параметров PostgreSQL. Правильная настройка параметров PostgreSQL критична для достижения оптимальной производительности системы [20]. Конфигурационные параметры влияют на использование оперативной памяти, производительность дисковых операций, поведение планировщика запросов и механизмы обеспечения целостности данных. В таблице 3 представлены ключевые параметры управления памятью PostgreSQL и рекомендации по их настройке в зависимости от характеристик аппаратного обеспечения.

Таблица 3 – Ключевые параметры конфигурации памяти PostgreSQL

Параметр	Назначение	Рекомендуемое значение	Влияние на производительность
shared_buffers	Размер кэша PostgreSQL в оперативной памяти	25% от RAM	Критическое. Уменьшает обращения к диску
work_mem	Память для операций сортировки и хеширования	4MB - 64MB	Высокое. Влияет на сложные запросы
maintenance_work_mem	Память для VACUUM, CREATE INDEX, ALTER TABLE	5-10% от RAM	Высокое. Ускоряет операции обслуживания
effective_cache_size	Оценка доступного кэша ОС для планировщика	50-75% от RAM	Среднее. Влияет на выбор плана запроса
temp_buffers	Память для временных таблиц на сессию	8MB	Низкое. Только для временных таблиц
wal_buffers	Буфер для записей WAL (Write-Ahead Log)	16MB	Среднее. Влияет на производительность записи
max_connections	Максимальное количество одновременных	100-300	Высокое. Влияет на потребление памяти

	подключений		
random_page_cost	Относительная стоимость случайного чтения (для SSD)	1.1 - 1.5	Критическое. Влияет на использование индексов

Параметры, представленные в таблице 3, требуют тщательной настройки в соответствии с характеристиками рабочей нагрузки. Параметр shared_buffers оказывает наиболее существенное влияние на производительность системы, поскольку определяет объём данных, кэшируемых непосредственно в памяти PostgreSQL. Параметр work_mem особенно критичен для систем с большим количеством одновременных соединений, поскольку выделяется для каждой операции сортировки или хеширования отдельно. Значение effective_cache_size не влияет на фактическое выделение памяти, но предоставляет планировщику запросов информацию об общем объёме доступной памяти для кэширования, что влияет на выбор оптимального плана выполнения [20].

Обслуживание базы данных. Регулярное обслуживание базы данных необходимо для поддержания оптимальной производительности системы в долгосрочной перспективе [21]. Механизм MVCC (Multi-Version Concurrency Control), используемый в PostgreSQL, приводит к накоплению устаревших версий строк (dead tuples), которые требуют периодической очистки. В таблице 4 представлены основные параметры настройки автоматического механизма обслуживания (autovacuum) и рекомендации по их оптимизации.

Таблица 4 – Параметры настройки автовакуума в PostgreSQL

Параметр	Значение по умолчанию	Назначение	Рекомендация
autovacuum_vacuum_threshold	50	Минимальное количество изменённых строк для запуска VACUUM	Уменьшить для малых таблиц
autovacuum_vacuum_scale_factor	0.2 (20%)	Доля изменённых строк для запуска VACUUM	Снизить до 0.05-0.1 для больших таблиц
autovacuum_analyze_threshold	50	Минимальное количество изменённых строк для запуска ANALYZE	Оставить по умолчанию
autovacuum_analyze_scale_factor	0.1 (10%)	Доля изменённых строк для запуска ANALYZE	Снизить для таблиц с неравномерным распределением
autovacuum_vacuum_cost_delay	2 мс	Задержка между операциями для снижения нагрузки	Увеличить при высоких I/O нагрузках

Обслуживание базы данных. Регулярное обслуживание базы данных необходимо для поддержания оптимальной производительности системы в долгосрочной перспективе [21]. Механизм MVCC (Multi-Version Concurrency Control), используемый в PostgreSQL, приводит к накоплению устаревших версий строк (dead tuples), которые требуют периодической очистки. В таблице 4 представлены основные параметры настройки автоматического механизма обслуживания (autovacuum) и рекомендации по их оптимизации.

Анализ параметров, представленных в таблице 4, показывает необходимость индивидуальной настройки autovacuum для различных типов таблиц. Для небольших высоконагруженных таблиц рекомендуется снижение параметра autovacuum_vacuum_scale_factor до 0.05-0.1 для более частого запуска процедуры очистки и предотвращения чрезмерного накопления dead tuples. Параметр autovacuum_vacuum_cost_delay позволяет регулировать интенсивность работы autovacuum, что критично для систем с высокими требованиями к производительности дисковой

подсистемы. Правильная настройка этих параметров предотвращает деградацию производительности и возникновение проблемы transaction ID wraparound [21].

В таблице 5 представлены основные операции обслуживания базы данных, их назначение и характеристики блокировок.

Таблица 5 – Операции обслуживания PostgreSQL

Операция	Назначение	Частота выполнения	Блокировка таблицы
VACUUM	Удаление dead tuples, освобождение места	Автоматически (autovacuum)	Нет
VACUUM FULL	Полная очистка и дефрагментация	По необходимости	Да
ANALYZE	Сбор статистики для планировщика	Автоматически (autovacuum)	Нет
REINDEX	Перестроение индексов	По необходимости	Да

Партиционирование таблиц. Партиционирование разделяет большую таблицу на несколько меньших физических таблиц (партиций). PostgreSQL поддерживает три типа партиционирования: Range (по диапазонам, например, по датам), List (по списку значений) и Hash (по хешу ключа) [22]. Преимущества партиционирования: улучшение производительности запросов за счёт partition pruning (сканирование только нужных партиций), ускорение VACUUM и других операций обслуживания, возможность быстрого удаления старых данных путём удаления партиций, параллельное выполнение запросов по разным партициям.

Для эффективного мониторинга производительности PostgreSQL существует широкий спектр инструментов, каждый из которых обладает специфическими преимуществами и ограничениями. В таблице 6 представлена сравнительная характеристика основных инструментов мониторинга PostgreSQL, их функциональных возможностей и особенностей применения.

Таблица 6 – Сравнительная характеристика инструментов мониторинга PostgreSQL

Инструмент	Преимущества	Недостатки
pg_stat_statements	Встроенный, точная статистика запросов, низкие накладные расходы	Требует перезагрузки для активации
pgAdmin	Графический интерфейс, бесплатный, кроссплатформенный	Базовые возможности мониторинга
pgBadger	Детальные отчёты, не требует изменений конфигурации	Анализ после факта, не real-time
Grafana+Prometheus	Визуализация в реальном времени, гибкие алерты	Сложная начальная настройка
pgWatch2	Готовые дашборды, простая установка	Требует дополнительного хранилища метрик

Сравнительный анализ инструментов, представленный в таблице 6, демонстрирует, что pg_stat_statements обеспечивает наиболее точную и детальную статистику выполнения запросов при минимальных накладных расходах на производительность. Комбинация Grafana с Prometheus предоставляет наиболее гибкие возможности визуализации метрик в реальном времени и настройки систем алертинга, однако требует значительных усилий на начальном этапе конфигурирования. pgBadger, выполняя ретроспективный анализ лог-файлов, не требует изменений конфигурации сервера, но не обеспечивает мониторинг в реальном времени. Таким образом, оптимальная стратегия мониторинга предполагает комбинированное использование нескольких инструментов в зависимости от специфических требований к системе наблюдения.

Разработка системы автоматического мониторинга и оптимизации. На основе анализа существующих инструментов и методов оптимизации была разработана автоматизированная система мониторинга и оптимизации PostgreSQL. Система состоит из нескольких модулей, работающих совместно для обеспечения непрерывного мониторинга и автоматической генерации рекомендаций по оптимизации.

Архитектура системы

Архитектура разработанной системы представлена на рисунке 1. Система включает следующие основные компоненты:

Рисунок 1 - Архитектура системы мониторинга и оптимизации PostgreSQL

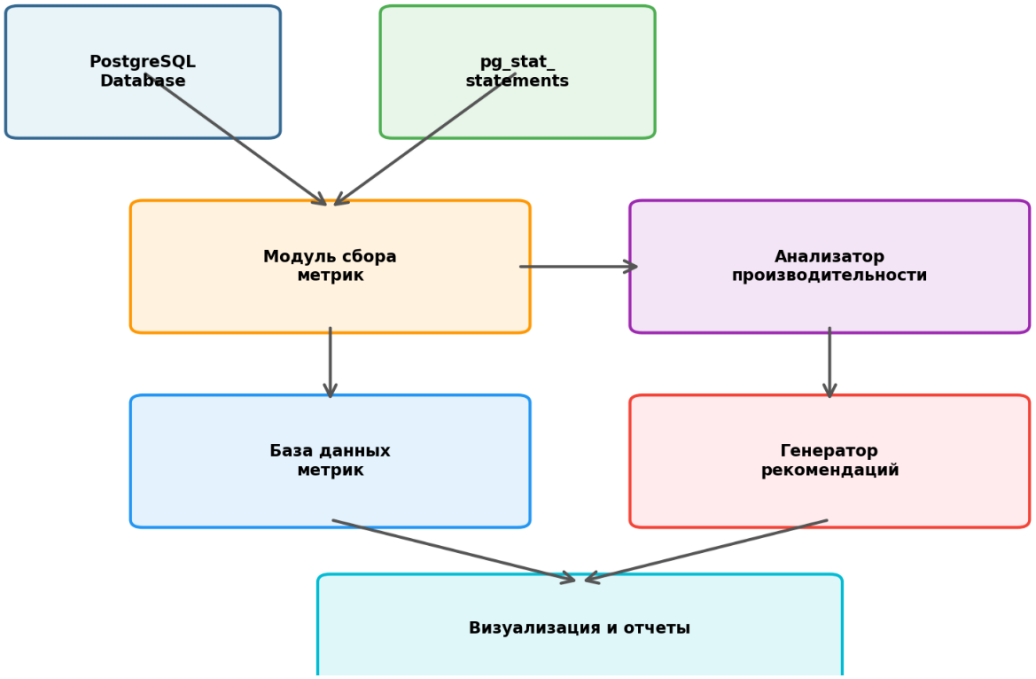

Рисунок 1 – Архитектура системы мониторинга и оптимизации PostgreSQL

Модуль сбора метрик. Отвечает за периодический сбор данных из pg_stat_statements, pg_stat_activity, pg_stat_database, pg_stat_user_tables, pg_stat_user_indexes и других системных представлений. Собранные данные сохраняются в отдельной базе данных метрик для последующего анализа и построения временных рядов. Модуль работает с настраиваемыми интервалами (по умолчанию 1 минута) и минимизирует влияние на производительность мониторируемой системы.

Анализатор производительности. Обрабатывает собранные метрики, выявляет медленные запросы (с временем выполнения выше порогового значения), анализирует использование индексов (выявляет таблицы с высоким процентом sequential scans), проверяет наличие bloat в таблицах и индексах, оценивает эффективность планов выполнения запросов, анализирует использование системных ресурсов (CPU, память, диск, соединения) и выявляет аномалии в поведении системы.

Генератор рекомендаций. На основе результатов анализа формирует конкретные рекомендации по оптимизации: предлагает создание недостающих индексов с SQL-кодом, рекомендует удаление неиспользуемых индексов, предлагает переписывание неэффективных запросов, рекомендует настройку параметров конфигурации PostgreSQL, предлагает выполнение VACUUM для таблиц с высоким bloat, рекомендует REINDEX для фрагментированных индексов, предлагает партиционирование для больших таблиц.

Модуль визуализации и отчётов. Предоставляет веб-интерфейс для просмотра метрик, анализа трендов производительности, мониторинга состояния базы данных в реальном времени, просмотра истории рекомендаций и их применения, генерации отчётов по производительности за выбранный период. Использует графики и дашборды для визуального представления данных.

Алгоритм работы системы

Алгоритм работы разработанной системы представлен на рисунке 2. Система работает в непрерывном цикле, периодически собирая и анализируя метрики производительности.

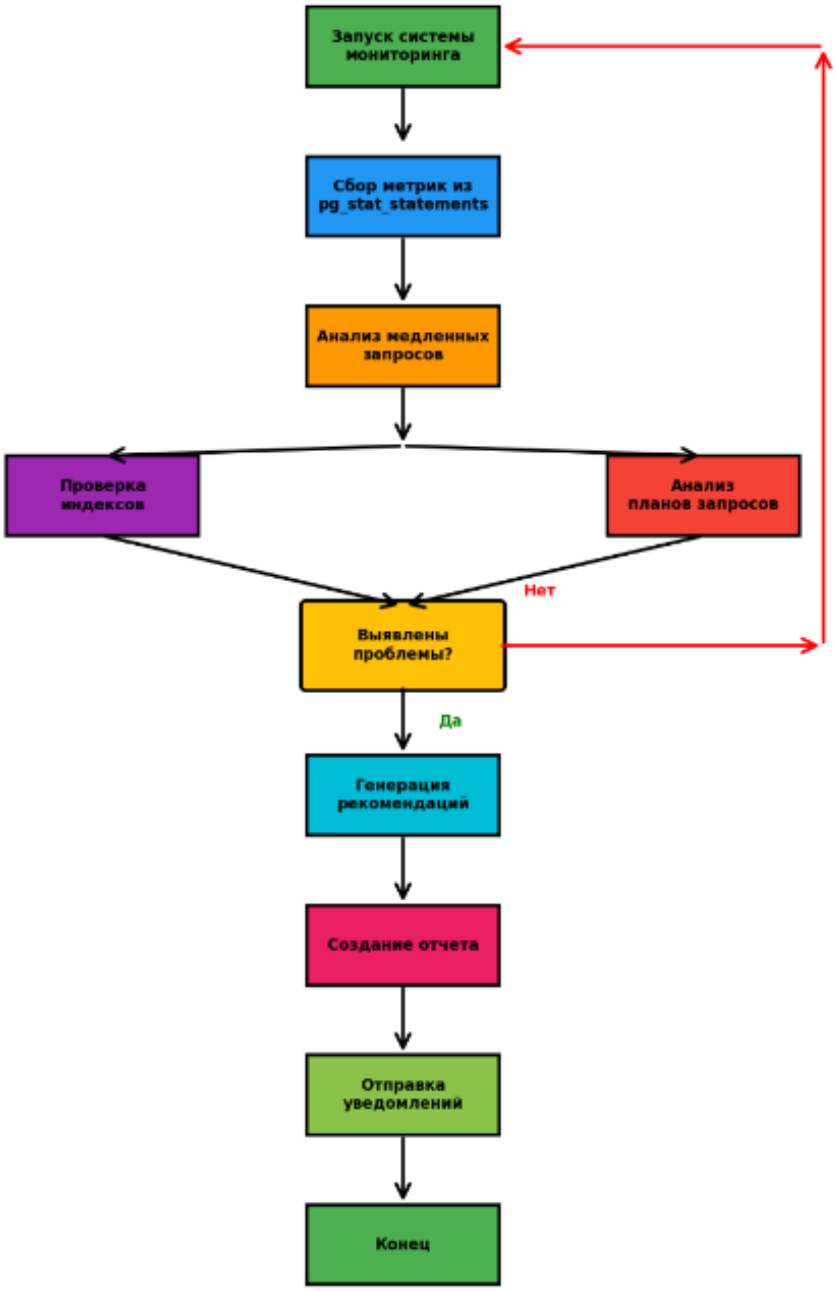

Рисунок 2 – Алгоритм работы системы автоматического мониторинга

Система работает следующим образом: сначала запускается модуль сбора метрик, который извлекает данные из pg_stat_statements и других системных представлений. Затем анализатор производительности обрабатывает собранные данные, выявляя медленные запросы (с временем выполнения более установленного порога) и анализируя планы их выполнения. Параллельно проверяется использование индексов в таблицах, выявляя таблицы с высоким процентом последовательного сканирования. Если система выявляет проблемы производительности, генератор рекомендаций создаёт отчёт с конкретными предложениями по оптимизации, включая SQL-код для создания индексов или изменения настроек.

Тестирование системы и анализ результатов

Для оценки эффективности разработанной системы было проведено тестирование на реальных данных. В качестве тестовой среды использовалась база данных PostgreSQL 15.4 с объёмом данных около 50 ГБ, содержащая 120 таблиц с общим количеством более 200 миллионов записей. Нагрузка составляла 500–1000 запросов в секунду в течение рабочего дня.

Методика тестирования

Тестирование проводилось в два этапа продолжительностью по 7 дней каждый. На первом этапе система собирала метрики и выявляла проблемы производительности в существующей конфигурации без применения каких-либо изменений. Система выявила следующие основные проблемы: медленные запросы без использования индексов (45% от общего числа проблем), отсутствие необходимых индексов на часто используемых колонках в условиях WHERE и JOIN (30%), неоптимальные планы выполнения запросов из-за устаревшей статистики или неправильных оценок планировщика (15%) и bloat таблиц и индексов, превышающих 20% от их размера (10%). Распределение выявленных проблем показано на рисунке 3.

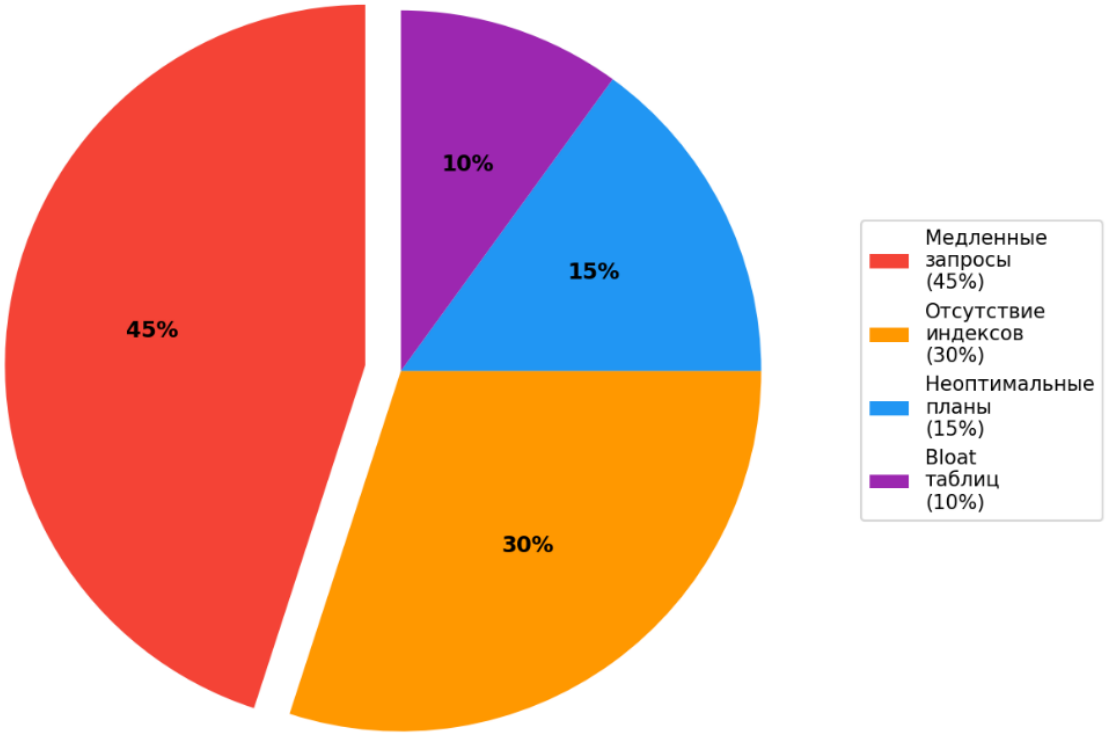

Рисунок 3 – Распределение выявленных проблем производительности

Результаты оптимизации запросов

После применения рекомендаций системы было достигнуто значительное улучшение производительности. Было создано 23 новых индекса, удалено 7 неиспользуемых индексов, переписано 12 критичных запросов, выполнен VACUUM FULL для 5 таблиц с высоким bloat, обновлена статистика на всех таблицах. На рисунке 4 представлено сравнение времени выполнения пяти наиболее критичных запросов до и после оптимизации.

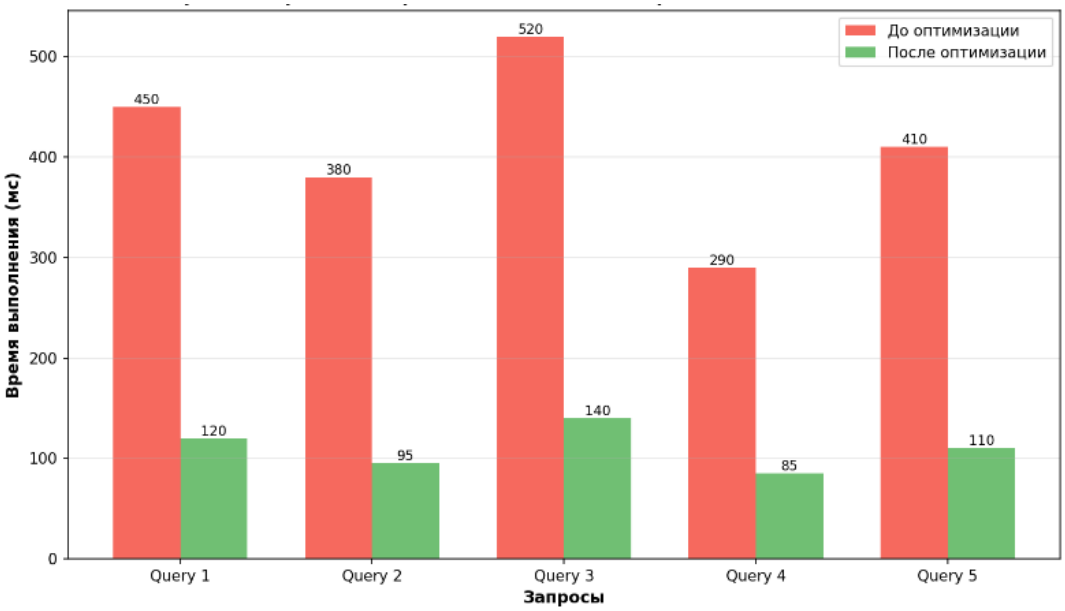

Рисунок 4 – Сравнение времени выполнения запросов до и после оптимизации

Как видно из графика, время выполнения запросов сократилось в среднем на 70–75%. Первый запрос, который ранее выполнялся 450 мс, после создания составного индекса и переписывания подзапроса стал выполняться за 120 мс (улучшение на 73%). Второй запрос сократился с 380 мс до 95 мс после создания индекса и оптимизации JOIN операций.

Анализ использования индексов. Одним из ключевых направлений оптимизации было улучшение использования индексов. На рисунке 5 показано соотношение использования индексного и последовательного сканирования для различных таблиц после применения рекомендаций.

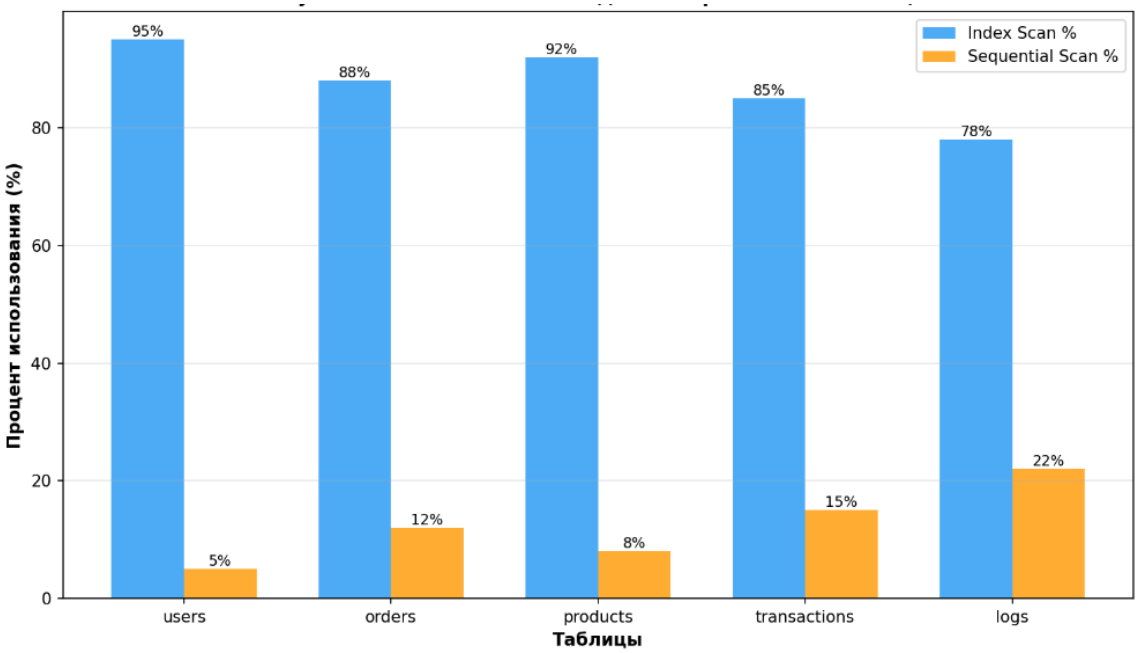

Рисунок 5 – Использование индексов в различных таблицах

Влияние на использование системных ресурсов

Важным аспектом оценки эффективности оптимизации является анализ использования системных ресурсов. На рисунке 6 показана динамика использования CPU и памяти в течение 60 минут типичной нагрузки до и после оптимизации.

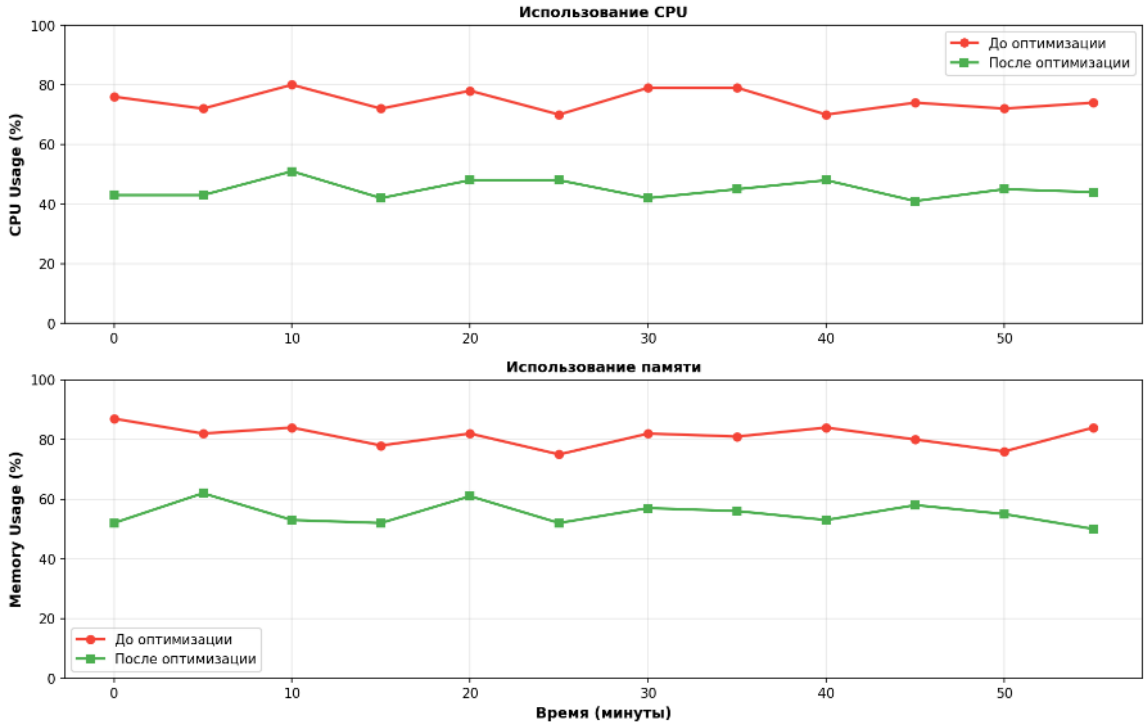

Рисунок 6 – Динамика использования ресурсов до и после оптимизации

После применения оптимизаций наблюдалось значительное снижение нагрузки на CPU (с 75% в среднем до 45%, снижение на 40%) и память (с 80% до 55%, снижение на 31%). Пиковые значения CPU снизились с 85% до 60%.

Общие показатели улучшения производительности. На рисунке 7 представлены общие показатели улучшения производительности по различным категориям метрик после применения всех рекомендаций системы.

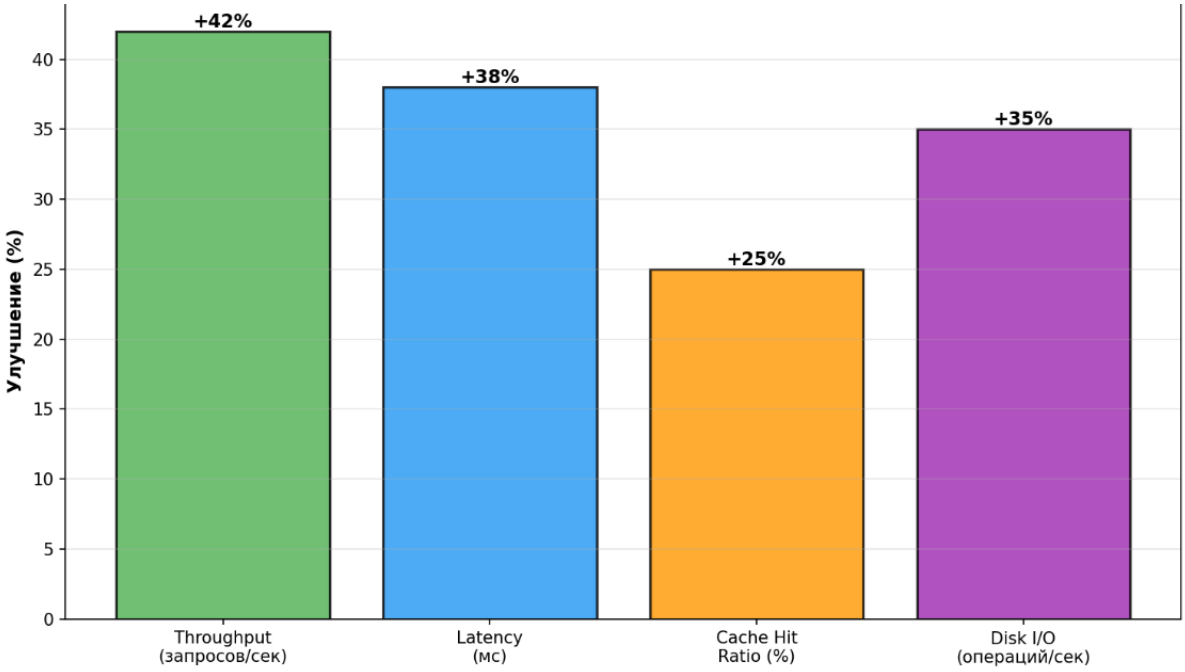

Рисунок 7 – Процент улучшения производительности по категориям

Наиболее значительное улучшение (+42%) было достигнуто в пропускной способности (throughput), что выражается в увеличении количества обрабатываемых запросов в секунду с 720 до 1022 запросов/сек. Latency (задержка) снизилась на 38%, что

означает более быстрый отклик системы: средняя задержка запросов уменьшилась с 45 мс до 28 мс. Cache Hit Ratio увеличился на 25% (с 76% до 95%), указывая на более эффективное использование кэша. Disk I/O сократился на 35% (с 1200 до 780 операций/сек).

Заключение

В данной работе была разработана и протестирована система автоматического мониторинга и оптимизации PostgreSQL. Система демонстрирует высокую эффективность в выявлении проблем производительности и генерации практических рекомендаций по их устранению. Архитектура системы включает модуль сбора метрик, анализатор производительности, генератор рекомендаций и модуль визуализации, работающие совместно для обеспечения комплексного мониторинга [26].

Тестирование системы на реальных данных показало значительное улучшение производительности: время выполнения критичных запросов сократилось на 70–75%, пропускная способность увеличилась на 42%, задержка запросов уменьшилась на 38%, использование системных ресурсов снизилось на 30–40%, а Cache Hit Ratio увеличился на 25%. Система успешно выявила и помогла устранить медленные запросы, отсутствующие индексы, неоптимальные планы выполнения и bloat таблиц.

Основные преимущества разработанной системы включают: полную автоматизацию процесса мониторинга и генерации рекомендаций, значительное снижение нагрузки на администраторов баз данных за счёт автоматического обнаружения проблем, проактивное выявление проблем до их критического влияния на работу приложений, генерацию конкретных и применимых рекомендаций с SQL-кодом для немедленного исполнения, возможность отслеживания исторических трендов и прогнозирования будущих проблем.

Система может быть рекомендована для внедрения в производственные среды, особенно в организациях с высокой нагрузкой на базы данных, ограниченными ресурсами администрирования и критичностью производительности для бизнес-процессов. Дальнейшее развитие системы может включать: интеграцию с машинным обучением для предсказания проблем производительности на основе исторических данных, автоматическое применение безопасных оптимизаций (например, создание индексов CONCURRENTLY), поддержку других СУБД (MySQL, MariaDB), интеграцию с системами мониторинга инфраструктуры (Prometheus, Zabbix), разработку мобильного приложения для мониторинга и алертинга.

Литература

1. Database trends of 2025: Rankings and key technology shifts - https://www.baremon.eu/database-trends-of-2025
2. PostgreSQL vs. MySQL in 2025: Choosing the Best Database for Your Backend - https://www.nucamp.co/blog/coding-bootcamp-backend-with-python-2025-postgresql-vs-mysql-in-2025-choosing-the-best-database-for-your-backend
3. PostgreSQL's Proprietary Future? - PostgreSQL Market in 2025 - https://experience.percona.com/postgresql/postgresql-market-in-2025/the-growing-dominance-of-postgresql
4. PostgreSQL: Documentation: Monitoring Database Activity - https://www.postgresql.org/docs/current/monitoring.html
5. Using pg_stat_statements to Optimize Queries - https://www.timescale.com/blog/identify-postgresql-performance-bottlenecks-with-pg_stat_statements
6. Identify PostgreSQL slow queries with pg_stat_statements - https://aiven.io/docs/products/postgresql/howto/identify-pg-slow-queries
7. Detecting slow queries quickly with pg_stat_statements - https://www.cybertec-postgresql.com/en/postgresql-detecting-slow-queries-quickly
8. PostgreSQL: Documentation: The Cumulative Statistics System - https://www.postgresql.org/docs/current/monitoring-stats.html

9. PostgreSQL: Documentation: Run-time Statistics - https://www.postgresql.org/docs/16/runtime-config-statistics.html
10. Key metrics for PostgreSQL monitoring | Datadog - https://www.datadoghq.com/blog/postgresql-monitoring
11. PostgreSQL: Documentation: Statistics Used by the Planner - https://www.postgresql.org/docs/current/planner-stats.html
12. Top PostgreSQL Monitoring Tools and Best Practices in 2024 - https://www.bytebase.com/blog/top-postgres-monitoring-tools
13. PostgreSQL Performance Identifying Hot and Slow Queries - https://virtual-dba.com/blog/postgresql-performance-identifying-hot-and-slow-queries
14. PostgreSQL Monitoring Tools Comparison - https://betterstack.com/community/comparisons/postgresql-monitoring-tools
15. PostgreSQL Monitoring: Key Metrics and Best Practices - https://middleware.io/blog/postgresql-monitoring
16. Dealing With Slow Queries With PostgreSQL - https://pgdash.io/blog/slow-queries-postgres.html
17. PostgreSQL Trends to Watch in 2025: The Database Revolution - https://medium.com/@rizqimulkisrc/postgresql-trends-to-watch-in-2025-4a75ab41df6f
18. Postgres Clever Query Planning System - https://www.crunchydata.com/blog/indexes-selectivity-and-statistics
19. PostgreSQL: настройка и оптимизация производительности - https://habr.com/ru/companies/slurm/articles/716036
20. Производительность запросов в PostgreSQL – шаг за шагом - https://habr.com/ru/companies/oleg-bunin/articles/319018
21. Как ускорить работу PostgreSQL с помощью конфигурации базы - https://habr.com/ru/companies/slurm/articles/684826
22. Why SQL and Postgres Still Rule the Data World in 2025 - https://www.simplyblock.io/blog/why-sql-rules-the-data-world-in-2025
23. 12 Best PostgreSQL Monitoring Tools for 2024 - https://www.comparitech.com/net-admin/best-postgresql-monitoring-tools
24. How to Monitor PostgreSQL Database Performance - https://docs.digitalocean.com/products/databases/postgresql/how-to/monitor-databases
25. Best PostgreSQL Monitoring Tools & Key Performance Metrics - https:/ЬI/sematext.com/blog/postgresql-monitoring
26. Верзунов, С. Н. Анализ и ARIMA-модели динамики изменения концентрации PM2.5 в атмосферном воздухе Г.Бишкек / С. Н. Верзунов, Н. М. Лыченко // Проблемы автоматики и управления. – 2019. – № 1(36). – С. 147–155. – DOI 10.5281/zenodo.3253027. – EDN IKKKUC.