

УДК 004.65

АНАЛИЗ СПОСОБОВ ОПТИМИЗАЦИИ РАБОТЫ С SQL-БАЗАМИ ДАННЫХ В НАГРУЖЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

Медведев Н.Е.

студент,

*Поволжский государственный университет телекоммуникаций и
информатики (ПГУТИ),*

Самара, Россия

Куваева Е.Н.

старший преподаватель кафедры информационных систем и технологий,

*Поволжский государственный университет телекоммуникаций и
информатики (ПГУТИ),*

Самара, Россия

Аннотация

Работа рассматривает методы оптимизации SQL-баз данных под высокой нагрузкой, распределяя их по четырём уровням — запросы, структура данных, СУБД, инфраструктура. Отдельное внимание уделено индексированию, кэшированию, репликации, партиционированию и шардированию: именно эти инструменты снижают время отклика и наращивают пропускную способность системы. Поставленный модельный эксперимент показывает, насколько сокращается среднее время выполнения запроса при последовательном подключении индексирования и кэширования. На основе полученных данных сформулированы практические рекомендации по выбору сочетания методов сообразно профилю нагрузки — OLTP или OLAP, read-heavy или write-heavy. Материал рассчитан на разработчиков и администраторов баз данных, проектирующих высоконагруженные информационные системы.

Ключевые слова: SQL, база данных, оптимизация запросов, индексирование, кэширование, шардирование, репликация, высоконагруженные системы, масштабирование, СУБД.

ANALYSIS OF METHODS FOR OPTIMIZING SQL DATABASE OPERATIONS IN HIGH-LOAD INFORMATION SYSTEMS

Medvedev N.E.

student,

Povolzhskiy State University of Telecommunications and Informatics (PSUTI),

Samara, Russia

Kuvaeva E.N.

Senior Lecturer, Department of Information Systems and Technologies,

Povolzhskiy State University of Telecommunications and Informatics (PSUTI),

Samara, Russia

Abstract

The article examines methods for optimizing SQL database operations under high load on an information system. The methods are systematized into four levels: the query level, the data structure level, the DBMS level, and the infrastructure level. Indexing, caching, replication, partitioning, and sharding are considered as the main tools for reducing response time and increasing system throughput. A model experiment is carried out, demonstrating the reduction of average query execution time when indexing and caching are applied sequentially. Based on the analysis, practical recommendations are formulated for selecting a combination of optimization methods depending on the load profile (OLTP/OLAP, read-heavy/write-heavy). The results may be useful to developers and database administrators designing high-load information systems.

Keywords: SQL, database, query optimization, indexing, caching, sharding, replication, high-load systems, scalability, DBMS.

Данные копируются быстрее, чем прежде, и вместе с их объёмом растёт интенсивность обращений к информационным системам. Тысячи, а нередко и десятки тысяч запросов в секунду — уже не редкость для реляционных СУБД, обслуживающих электронную коммерцию, биллинг, банковский и государственный секторы. Задержка здесь не абстракция: от неё зависят и удовлетворённость пользователя, и итоговые цифры в отчётности организации. Одно лишь наращивание аппаратных мощностей сервера проблему не решает — у вертикального масштабирования есть потолок, технический и экономический одновременно [5, 6].

Именно поэтому методы оптимизации SQL-баз данных нуждаются в систематизации применительно к разным уровням архитектуры информационной системы — от отдельного запроса до распределённой инфраструктуры хранения [4, 7, 10, 18]. Целиком эту проблему отечественные публикации не охватывают: кто-то оценивает время выполнения запросов [2, 3], кто-то разбирает индексирование и структурную оптимизацию [11, 13, 15], кто-то ограничивается организацией кэширования [8, 12]. А вот сочетанный выбор методов под конкретный профиль нагрузки остаётся изученным лишь фрагментарно — в этом и состоит актуальность настоящей работы.

Цель работы двоякая: систематизировать методы оптимизации SQL-баз данных, работающих под нагрузкой, и на модельном примере оценить, насколько они эффективны в сравнении друг с другом. Задачи, вытекающие отсюда, довольно конкретны — классифицировать методы по уровням архитектуры; разобрать оптимизацию запросов и индексирование; отдельно остановиться на кэшировании; проанализировать репликацию, партиционирование и шардирование; поставить эксперимент, иллюстрирующий

эффект индексирования и кэширования; и, наконец, сформулировать рекомендации по выбору методов сообразно профилю нагрузки.

В одном пункте публикации [4, 6, 7, 10, 18] единодушны: методы оптимизации SQL-СУБД в нагруженных информационных системах естественно раскладываются по четырём взаимосвязанным уровням — запросов, структуры данных, СУБД, инфраструктуры. Нижние обходятся дешевле при внедрении, поэтому к ним обращаются первыми — как только замечены первые признаки деградации производительности. Верхние, кэширование и масштабирование, подключаются позже: когда ресурс нижних исчерпан или нагрузка попросту превысила ёмкость одного сервера [17, 18]. Схема этих уровней в обобщённом виде — на рисунке 1.



Рис. 1 – Уровни оптимизации SQL-баз данных в нагруженных информационных системах

Источник: составлено автором на основе [4, 6, 7, 10, 18]

Что именно относится к каждому уровню и как это выглядит на практике, с примерами реализации, сведено в таблице 1.

Таблица 1 – Классификация методов оптимизации работы с SQL-базами данных

Уровень	Основные методы	Примеры реализации
1. Запросы	Переписывание запросов; анализ планов выполнения	EXPLAIN / EXPLAIN ANALYZE; отказ от SELECT *; устранение N+1-запросов
2. Структура данных	Индексирование; нормализация / денормализация	B-tree и составные индексы; покрывающие индексы
3. СУБД	Кэширование; пул соединений; материализованные представления	Redis, Memcached; connection pooling; materialized views
4. Инфраструктура	Репликация; партиционирование; шардирование	master-slave, master-master; горизонтальное / вертикальное партиционирование; range-, hash-шардирование

Начинать разумнее всего с запросов и структуры данных — из всех способов поднять производительность SQL-СУБД этот стоит дешевле остальных. Ошибки здесь довольно типовые: то тянут лишние столбцы без всякой на то нужды (SELECT *), то прячут запрос внутри цикла обработки результатов другого запроса — классическая проблема N+1, то забывают об индексах на столбцах из WHERE и JOIN, то используют коррелированные подзапросы там, где куда уместнее было бы простое соединение таблиц [3, 6, 15].

Обойти последовательное сканирование таблицы и тем самым ускорить выборку строк чаще всего помогает вспомогательная структура типа B-tree — индекс. Построенные сразу по нескольким столбцам составные индексы неплохо показывают себя при фильтрации по совокупности условий, а покрывающие идут ещё дальше: запрос обходится вовсе без обращения к самой таблице [11, 13].

Оптимизации запроса не мешает предшествовать объективная оценка его стоимости. В работах [2, 3] такие аналитические методы уже предложены — они учитывают селективность условий и стоимость соединения таблиц, а значит, эффект от добавления индекса можно спрогнозировать заранее, ещё до полноценного нагрузочного тестирования.

EXPLAIN и EXPLAIN ANALYZE вскрывают узкие места без особого труда: полное сканирование таблицы там, где следовало обратиться к индексу, неудачный порядок соединения, сортировку промежуточных результатов на диске [6, 9]. Вместе с индексированием такой анализ способен сократить время отклика в десятки раз — и для этого не нужно менять ни строчки в аппаратной конфигурации [9, 13].

Среди механизмов, поднимающих производительность без радикальной переделки алгоритмов обработки данных, кэширование остаётся, пожалуй, самым старым и универсальным [8]. Между приложением и базой размещается буфер, где оседают часто запрашиваемые либо редко меняющиеся данные, а число обращений к СУБД падает — вместе с ним снижается и среднее время отклика.

По месту размещения кэш бывает трёх видов: уровня приложения — прямо в оперативной памяти процесса; распределённый, вроде Redis или Memcached, — общий сразу для нескольких экземпляров приложения; и уровня самой СУБД, где трудятся буферный пул и кэш планов запросов. Паттернов на практике тоже три: cache-aside, где приложение само читает и обновляет кэш, write-through с одновременной записью в кэш и базу, и write-behind, откладывающий эту запись на потом [8, 12].

Требования к согласованности данных и определяют выбор паттерна. За более высокую актуальность данных cache-aside и write-through расплачиваются задержкой записи, тогда как write-behind жертвует согласованностью ради выигрыша в пропускной способности [8, 12]. Данные, впрочем, можно загрузить и заранее, до обращения к ним, — такой приём, упреждающее кэширование, в распределённых средах заметно сокращает задержку при выполнении связанных запросов [12].

Рано или поздно вертикальное масштабирование и тонкая настройка отдельного сервера упрутся в потолок, и тогда за дело берётся горизонтальное масштабирование инфраструктуры хранения данных [1, 17, 18].

Копии базы данных на разных серверах — вот чем по сути занимается репликация. В схеме master–slave запись идёт на основной сервер, а чтение распределяется по репликам; в master–master запись возможна сразу на нескольких узлах. Отказоустойчивость растёт, нагрузка на чтение рассредоточивается, но одну проблему репликация так и не решает — ограниченную ёмкость при записи больших объёмов данных [1].

Партиционирование действует иначе: большую таблицу оно делит на логические части в пределах одного сервера — по диапазону, списку значений или хэшу ключа. Администрировать становится проще, операции с частью данных идут быстрее, а вот ограничение ресурсов одного узла как было, так и остаётся [17, 18].

Шардирование заходит дальше партиционирования — части данных, шарды, разносятся уже по разным физическим серверам, и вместе с этим растут вычислительные ресурсы, объём хранимых данных и пропускная способность разом. Распределять данные умеют по-разному: range-, hash-, directory-based. Платить за это приходится усложнением архитектуры — соединением данных из разных шардов, поддержанием транзакционной целостности, ресардингом при перераспределении, и каждая из этих задач требует отдельного решения. Прибегать к шардированию стоит лишь тогда, когда индексирование, кэширование и репликация уже исчерпали себя [1, 17].

Сведены и сопоставлены друг с другом эти методы масштабирования в [таблице 2](#).

Таблица 2 – Сравнительная характеристика методов горизонтального масштабирования SQL-СУБД

Критерий	Репликация	Партиционирование	Шардирование
Цель применения	Отказоустойчивость, распределение чтения	Упрощение структуры больших таблиц	Распределение нагрузки записи и объёма данных
Сложность внедрения	Средняя	Средняя	Высокая

Критерий	Репликация	Партиционирование	Шардирование
Влияние на консистентность	Возможна задержка репликации	Не влияет	Требует согласования транзакций между шардами
Типичная область применения	Read-heavy нагрузка	Крупные таблицы на одном сервере	Системы с очень большим объёмом данных и записи

Насколько ощутим совместный эффект индексирования и кэширования, демонстрирует модельный эксперимент на синтетическом наборе данных. Все приведённые далее цифры получены на модельных данных — они показывают относительный эффект методов, а вовсе не абсолютные отраслевые показатели какой-то конкретной промышленной системы.

Устроен эксперимент так. В PostgreSQL сгенерирована синтетическая таблица заказов на 1 000 000 записей — с полями `customer_id`, `order_date`, `status`, `amount`. В роли тестового запроса выступила выборка заказов клиента за заданный период, соединённая с таблицей клиентов, — типичный запрос для биллинговых и e-commerce систем. Время выполнения усредняли по 100 запускам для каждой из трёх конфигураций: без индекса и кэширования вовсе; с составным B-tree индексом по столбцам `customer_id` и `order_date`; и с этим же индексом плюс распределённый кэш результатов повторяющихся запросов — паттерн `cache-aside`.

Что получилось в итоге, собрано в [таблице 3](#) и показано на [рисунке 2](#).

Таблица 3 – Результаты модельного эксперимента (иллюстративные данные)

Конфигурация	Среднее время выполнения запроса, мс	Изменение относительно baseline
Без индекса (baseline)	480	—
С B-tree индексом	42	ускорение примерно в 11 раз
С индексом и кэшированием	3	ускорение примерно в 160 раз

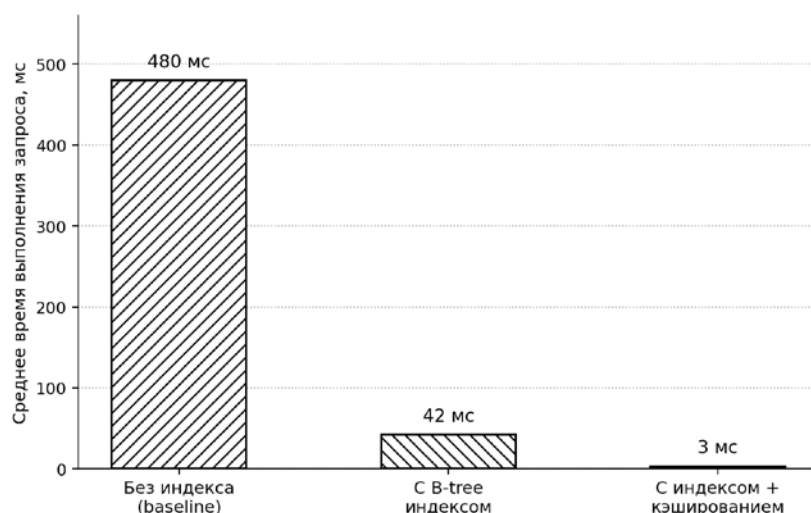


Рис. 2 – Среднее время выполнения запроса в модельном эксперименте, мс

Источник: составлено автором по результатам модельного эксперимента

С данными источников [3, 9, 13] результаты вполне согласуются. Больше всего выигрывает индекс, построенный на столбцах фильтрации; кэширование повторяющихся запросов добавляет кратный прирост, но по большей части в read-heavy сценариях. А вот переносить эти цифры на произвольную промышленную систему без повторного тестирования на реальных данных не стоит: эффект индексирования зависит от селективности условий, эффект кэширования — от доли повторяющихся запросов в общем потоке обращений.

Проведённый анализ и результаты эксперимента подсказывают вполне определённую последовательность действий при оптимизации нагруженной SQL-СУБД. Первым делом — «тяжёлые» запросы: EXPLAIN, EXPLAIN ANALYZE, устранение антипаттернов вроде SELECT *, проблемы N+1, отсутствующих индексов на WHERE и JOIN [3, 6, 9]. Затем — индексы, составные и покрывающие, добавленные и пересмотренные с оглядкой на скорость записи [11, 13, 15]. После — кэширование для read-heavy нагрузок, с паттерном, подобранным под требования к согласованности данных [8, 12]. И только когда эти уровни исчерпаны, а нагрузка всё ещё растёт, — очередь репликации, которая распределяет чтение, а при необходимости и партиционирования с шардированием [1, 17, 18].

За этой последовательностью стоит принцип минимальной достаточности: вперёд пускают методы, почти не трогающие архитектуру, а те, что усложняют её всерьёз, шардирование в первую очередь, подключают лишь при действительной необходимости [17, 18].

Методы оптимизации SQL-баз данных, применяемые в нагруженных информационных системах, в итоге удалось разложить по четырём уровням — запросов, структуры данных, СУБД, инфраструктуры, а индексирование и оптимизацию запросов, кэширование, репликацию, партиционирование и шардирование — сопоставить друг с другом.

Эксперимент же показал: применённые последовательно, индексирование и кэширование сокращают среднее время выполнения типового запроса на несколько порядков. Основную работу здесь делает индексирование, а кэширование добавляет выигрыш там, где нагрузка read-heavy и запросы повторяются часто.

Применить сформулированные рекомендации можно при проектировании и сопровождении нагруженных информационных систем — там, где выбираются архитектурные решения. Дальше есть смысл двигаться в сторону количественной методики выбора стратегии шардирования и анализа того, насколько применимы гибридные SQL/NoSQL-архитектуры для систем со смешанным профилем нагрузки.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Горизонтальное масштабирование базы данных. Репликация. Партиционирование. Шардирование [Электронный ресурс] // Хабр. — 2025. — URL: <https://habr.com/ru/articles/875708/> (дата обращения: 14.09.2026).
2. Григорьев, Ю. А. Метод оценки времени выполнения запросов SQL к реляционным базам данных / Ю. А. Григорьев, А. Д. Плутенко [Электронный ресурс] // Радіоелектроніка, інформатика, управління. — 2002. — URL:

<https://cyberleninka.ru/article/n/metod-otsenki-vremeni-vypolneniya-zaprosov-sql-k-relyatsionnym-bazam-dannyh> (дата обращения: 14.09.2026).

3. Григорьев, Ю. А. Оценка времени выполнения SQL-запросов к базам данных / Ю. А. Григорьев // Программные продукты и системы. — 2012. — № 1. — URL: <https://cyberleninka.ru/article/n/otsenka-vremeni-vypolneniya-sql-zaprosov-k-bazam-dannyh> (дата обращения: 14.09.2026).
4. Грофф, Дж. Р. SQL: полное руководство / Дж. Р. Грофф, П. Н. Вайнберг, Э. Дж. Оппель ; пер. с англ. — 3-е изд. — М. : ООО «И.Д. Вильямс», 2015. — 960 с.
5. Кампбелл, Л. Базы данных. Инжиниринг надёжности / Л. Кампбелл, Ч. Мейджорс ; пер. с англ. — СПб. : Питер, 2020. — 304 с.
6. Космодемьянский, И. Производительность запросов в PostgreSQL — шаг за шагом [Электронный ресурс] // Хабр. — URL: <https://habr.com/ru/companies/oleg-bunin/articles/319018> (дата обращения: 14.09.2026).
7. Кузнецов, Л. А. Оптимизация запросов к базам данных информационных систем / Л. А. Кузнецов, А. К. Погодаев, В. В. Овчинников [Электронный ресурс] // Управление большими системами: сборник трудов. — 2003. — URL: <https://cyberleninka.ru/article/n/optimizatsiya-zaprosov-k-bazam-dannyh-informatsionnyh-sistem> (дата обращения: 14.09.2026).
8. Грушин Д.А., Лазарев Д.О., Фомин С.А. Кэширование данных в мультиконтейнерных системах [Электронный ресурс] // Труды Института системного программирования РАН. — 2019. — URL: <https://cyberleninka.ru/article/n/keshirovanie-dannyh-v-multikonteyneryh-sistemah> (дата обращения: 14.09.2026).
9. Оптимизация запросов базы данных на примере B2B сервиса для строителей [Электронный ресурс] // Хабр. — 2019. — URL: <https://habr.com/ru/articles/461071/> (дата обращения: 14.09.2026).

10. Каримов Т.Ш. Оптимизация запросов в реляционных базах данных: подходы и инструменты [Электронный ресурс] // Вестник Науки. – 2025. — URL: <https://cyberleninka.ru/article/n/optimizatsiya-zaprosov-v-relyatsionnyh-bazah-dannyh-podhody-i-instrumenty> (дата обращения: 14.09.2026).
11. Чухраев И.В., Жукова И.В. Оптимизация работы с информацией в базах данных [Электронный ресурс] // Инновационная наука. – 2016. — URL: <https://cyberleninka.ru/article/n/optimizatsiya-raboty-s-informatsiey-v-bazah-dannyh> (дата обращения: 14.09.2026).
12. Организация упреждающего кэширования для приложений, исполняющихся в сетевой среде распределённых вычислений // Вестник Ивановского государственного энергетического университета [Электронный ресурс]. — URL: <https://cyberleninka.ru/article/n/organizatsiya-uprezhdayushchego-keshirovaniya-dlya-prilozheniy-ispolnyayuschih-sya-v-setevoy-srede-raspredeleennyh-vychisleniy> (дата обращения: 14.09.2026).
13. Набродова И.Н., Кузнецов Г.А., Повышение производительности баз данных [Электронный ресурс] // Известия Тульского государственного университета. Технические науки. – 2021. — URL: <https://cyberleninka.ru/article/n/povyshenie-proizvoditelnosti-baz-dannyh> (дата обращения: 14.09.2026).
14. Мартынов В.В., Прошин Е.Н. Подход к анализу характеристик производительности БД [Электронный ресурс] // Вестник УГАТУ. - 2009. — URL: <https://cyberleninka.ru/article/n/podhod-k-analizu-harakteristik-proizvoditelnosti-bd> (дата обращения: 14.09.2026).
15. Хайдарова С. Создание SQL-подзапросов в реляционных базах данных [Электронный ресурс] // Экономика и социум. – 2023. — URL: <https://cyberleninka.ru/article/n/sozdanie-sql-podzaprosov-v-relyatsionnyh-bazah-dannyh> (дата обращения: 14.09.2026).
16. Бородин А.М., Поршневу С.В. Сравнительный анализ возможностей и скорости обработки многомерных данных программными средствами бизнес-

аналитики на основе индексирующих структур основной памяти [Электронный ресурс] // Информатика, телекоммуникации и управление. – 2010. — URL: <https://cyberleninka.ru/article/n/sravnitelnyy-analiz-vozmozhnostey-i-skorosti-obrabotki-mnogomernyh-dannyh-programmnymi-sredstvami-biznes-analitiki-na-osnove> (дата обращения: 14.09.2026).

17. Шардирование баз данных: проблемы, альтернативы, практические рекомендации [Электронный ресурс] // Хабр. — 2025. — URL: <https://habr.com/ru/articles/916394/> (дата обращения: 14.09.2026).
18. Стукало Р.Е., Сафин М.А. Эффективная организация баз данных в промышленных системах [Электронный ресурс] // Проблемы современной науки и образования. — 2024. — URL: <https://cyberleninka.ru/article/n/effektivnaya-organizatsiya-baz-dannyh-v-promyshlennyh-sistemah> (дата обращения: 14.09.2026).