УДК 004.65

Исследование влияния фрагментации данных на производительность чтения/записи и методы ее устранения в MySQL (InnoDB)

Черкасов Сергей Дмитриевич[1] — студент бакалавриата 2-го курса направления 38.03.05 «Бизнес-информатика». Научные интересы: информационные системы, разработка и исследование характеристик приложений для работы с базами данных. E-mail: cherkasovsergeidx@mail.ru

Хомоненко Анатолий Дмитриевич[1, 2] — доктор техн. наук, профессор, профессор кафедры «Информационные и вычислительные системы»; профессор кафедры математического и программного обеспечения. Научные интересы: информационные системы, обработка больших данных, вероятностное моделирование геоинформационных систем, генетические алгоритмы, информационная безопасность. E-mail: khomon@mail.ru

[1]Петербургский государственный университет путей сообщения Императора Александра I, Россия, 190031, Санкт-Петербург, Московский пр., 9

[2]Военно-космическая академия имени А. Ф. Можайского, Россия, 197198, Санкт-Петербург, ул. Ждановская, 13

Для цитирования: Черкасов С. Д., Хомоненко А. Д. Исследование влияния фрагментации данных на производительность чтения/записи и методы ее устранения в MySQL (InnoDB) // Интеллектуальные технологии на транспорте. 2026. № 2 (46). С. 64–74. DOI: 10.20295/2413-2527-2026-246-64-74

Аннотация. *Предлагается экспериментально обоснованный подход к оценке влияния фрагментации данных на производительность СУБД MySQL (InnoDB) и выбору метода ее устранения.* ***Цель:*** *исследование зависимости времени выполнения операций чтения и записи от степени фрагментации данных и определение наиболее эффективного способа дефрагментации.* ***Методы:*** *анализ архитектуры InnoDB (B+-деревья, page split); создание экспериментального стенда на базе MySQL 8.0 в Docker; генерация тестовой таблицы объемом 500 тыс. записей; моделирование фрагментации через массовые DELETE, UPDATE и хаотичные INSERT; нагрузочное тестирование sysbench; измерение времени запросов, латентности, IOPS и data_free.* ***Результаты:*** *фрагментация замедляет SELECT в 2,5–3,5 раза, INSERT и UPDATE — в 2–3 раза. Рост data_free с 2 до 38 Мб коррелирует с падением TPS на 58%. OPTIMIZE TABLE восстанавливает производительность до 95–98% от исходного уровня.* ***Практическая значимость:*** *методики мониторинга и плановой дефрагментации применимы для транспортных информационных систем с высокой интенсивностью обновлений данных (трекинг, логистика).*

Ключевые слова: *фрагментация данных, MySQL, InnoDB, производительность, page split, optimize table, B+-дерево, data_free, интеллектуальные транспортные системы*

2.3.5 *— математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей (технические науки);* **2.9.8** *— интеллектуальные транспортные системы (технические науки)*

Введение

Современные информационные системы, особенно в сфере транспорта и логистики, обрабатывают огромные объемы данных в режиме реального времени. Системы мониторинга подвижного состава, трекинга грузов, управления перевозками генерируют терабайты записей ежедневно.

При этом производительность системы управления базами данных (СУБД) становится критическим фактором, определяющим качество работы всего приложения.

Одной из скрытых, но существенных причин деградации производительности является *фрагментация данных* [1]. В отличие от явных ошибок (неоптимальные запросы, отсутствие индексов) фрагментация накапливается незаметно в процессе эксплуатации. Массовые удаления (например, очистка устаревших трекинговых данных), частые обновления, хаотичные вставки записей приводят к тому, что физическая структура данных на диске перестает соответствовать логической.

Движок InnoDB — наиболее распространенный в MySQL (по умолчанию с версии 5.5) [2]. Он используется в высоконагруженных проектах, включая онлайн-торговлю, банковские системы, а также в транспортных информационных системах. Однако, несмотря на его популярность, влияние фрагментации на реальную эксплуатационную производительность изучено недостаточно, а методы борьбы с ней часто применяются хаотично или игнорируются.

Особенно остро проблема фрагментации стоит в OLTP-системах (online transaction processing), где важна высокая скорость выполнения операций в режиме реального времени. В статье исследуется влияние фрагментации данных на производительность операций чтения и записи в MySQL (InnoDB) для определения эффективных способов ее устранения, в том числе применительно к системам обработки данных на транспорте. Для этого решались следующие задачи:

- рассмотрение архитектуры движка InnoDB, механизмов хранения и индексации [1, 3];
- анализ видов и причины возникновения фрагментации данных и индексов [2, 4, 5];
- разработка экспериментального стенда и тестовой базы данных;
- проведение контролируемого эксперимента с фиксацией ключевых метрик производительности в трех состояниях (без фрагментации, с фрагментацией, после оптимизации);
- сравнение полученных метрик и интерпретация результатов [6, 7];
- формулирование практических рекомендаций для эксплуатации высоконагруженных систем, включая транспортные [8–10].

Теоретическая основа: архитектура InnoDB и механизмы фрагментации

Для понимания природы фрагментации приведем кратко основные архитектурные решения InnoDB [1, 3]. Схема архитектуры хранения данных InnoDB показана на рис. 1.

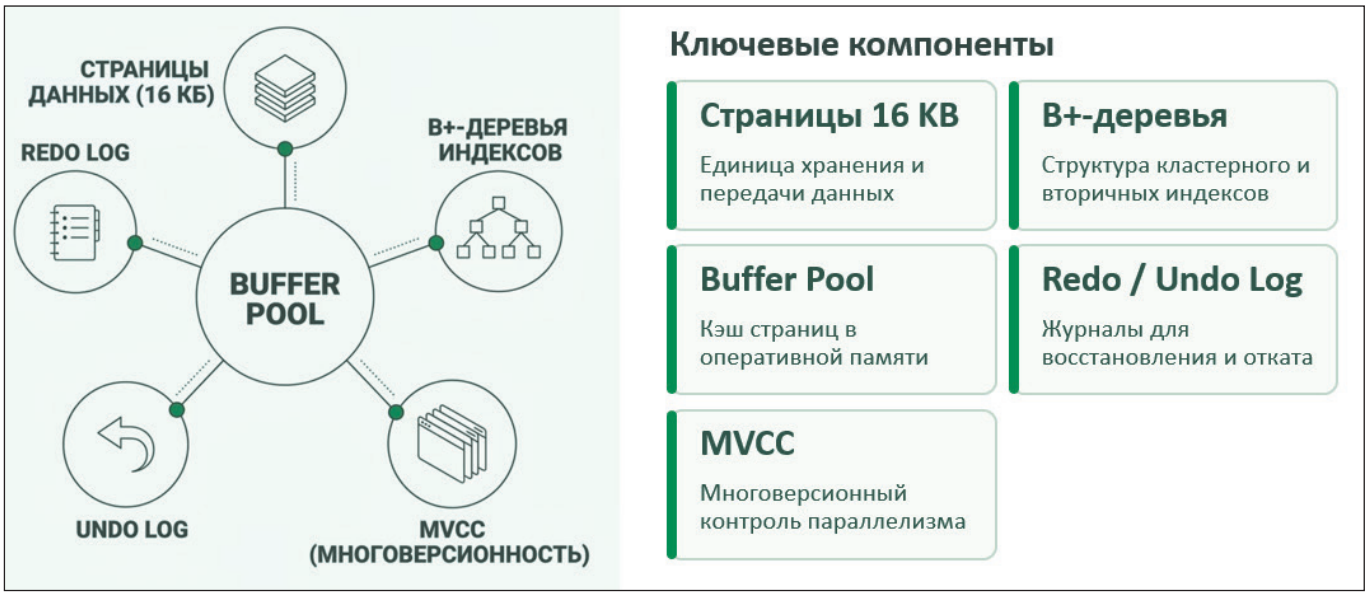

Рис. 1. Схема архитектуры хранения данных InnoDB

Полная схема архитектуры включает табличное пространство, сегменты, экстенты, страницы, буферный пул и журналы Redo/Undo [1].

Страничная организация данных

InnoDB хранит данные и индексы на диске блоками фиксированного размера — страницами (pages), по умолчанию 16 Кб [1, 2]. Каждая страница содержит записи таблицы или узлы индексов. При выполнении запросов страницы подгружаются в буферный пул (Buffer Pool) в оперативной памяти [3]. Эффективность работы системы напрямую зависит от того, насколько плотно заполнены страницы и насколько последовательно они расположены на диске [5].

B+-деревья

И первичный ключ (кластерный индекс), и вторичные индексы в InnoDB реализованы в виде B+-деревьев [1, 2]. Листовые страницы кластерного индекса содержат полные строки данных. Листовые страницы вторичных индексов содержат значения индексированных полей и значения первичного ключа [11]. B+-деревья обеспечивают логарифмическую сложность поиска, но чувствительны к степени заполнения страниц [3, 5].

Кластерный индекс и последовательная вставка

Если первичный ключ генерируется как AUTO_INCREMENT, новые записи вставляются в конец B+-дерева — в правую крайнюю страницу [2, 4]. Это минимизирует разбиения страниц. Если же первичный ключ не является последовательным (например, UUID, случайные числа, хеши), новые записи могут попадать в середину дерева, вызывая частые разбиения [5]. Для транспортных систем, использующих распределенные идентификаторы (GUID) для синхронизации между филиалами, это особенно актуально [8, 10].

Page split (разбиение страницы) — ключевой механизм фрагментации

Когда вставка новой записи происходит в уже заполненную страницу, InnoDB выполняет разбиение страницы: примерно половина записей перемещается на новую страницу [1, 4]. В результате:

- образуется пустое пространство (фрагментация внутри страниц) — страница становится заполненной менее чем на 50–60 % [2];
- нарушается физическая смежность логически последовательных записей [5];
- увеличивается глубина B+-дерева и количество операций ввода-вывода [6, 11];
- снижается эффективность буферного пула, так как в кэш попадает больше пустых страниц [3].

Именно page split является ключевым механизмом, порождающим фрагментацию индексов и данных [4, 5]. Особенно интенсивно page split происходят:

- при массовых INSERT с непоследовательными ключами [2];
- при UPDATE, увеличивающих размер строки (например, заполнение поля VARCHAR) [11];
- при DELETE, создающих дыры, которые затем хаотично заполняются [4].

Виды и причины фрагментации в InnoDB

В контексте InnoDB выделяют два основных типа фрагментации: данных и индексов [2, 4].

Фрагментация данных возникает в кластерном индексе из-за частых операций DELETE и UPDATE, изменяющих размер строк [1, 5]. Удаленные строки помечаются как мертвые (внутренний флаг DELETE_MARK), но занимаемое ими физическое пространство не возвращается операционной системе немедленно [2]. При последующих вставках InnoDB может переиспользовать эти дыры, однако это приводит к несвязному расположению данных — строки, логически принадлежащие одному диапазону, физически разбросаны по разным страницам [4, 6].

Фрагментация индексов характерна для вторичных индексов и также возникает при page split, обновлениях ключей и удалениях [1, 11]. Фрагментированный индекс имеет много частично заполненных страниц (коэффициент заполнения может падать до 30–40 %), что увеличивает количество страниц, которые нужно прочитать при сканировании индекса [4, 7]. Для запросов, использую-

щих вторичный индекс с последующим доступом к кластерному (так называемый обратный поиск), фрагментация особенно критична [2, 6].

На рис. 2 для наглядности приведены три варианта фрагментации: внутренняя, внешняя и индексная.

Внутренняя фрагментация возникает внутри выделенного блока памяти, когда выделенное пространство используется не полностью, оставляя пустоты (гэпы), что снижает плотность хранения. Внешняя фрагментация проявляется на уровне расположения данных на диске, когда строки таблицы или блоки данных разбросаны по несмежным страницам, что приводит к избыточным перемещениям головок диска. Индексная фрагментация характерна для структур B-tree: при вставке записей не по порядку.

Основные причины фрагментации в реальных системах

1. Массовые DELETE (особенно 30% и более записей), например, периодическая очистка устаревших трекинговых данных или удаление завершенных заказов [8, 9].

2. UPDATE индексируемых полей фактически выполняется как DELETE + INSERT, что порождает фрагментацию сразу в двух индексах [2, 5].

3. Хаотичные INSERT — вставка записей с неупорядоченными значениями первичного ключа (характерно для распределенных транспортных систем, где ID генерируются на разных узлах) [8, 10].

4. Отсутствие плановой дефрагментации в течение длительной эксплуатации (месяцы и годы) [4, 7].

Экспериментальная методика

Для получения объективных результатов разработан контролируемый эксперимент. Эксперимент проводился в трех состояниях: без фрагментации (эталон), с искусственно созданной фрагментацией и после выполнения оптимизации [6].

Окружение

Контролируемый эксперимент проводился с использованием в окружении следующих составляющих и характеристик:

- СУБД: MySQL 8.0.33 (движок InnoDB) [1];
- развертывание: Docker (изолированная среда, контейнер с 4 vCPU и 8 Гб RAM);
- тестовая база данных: fragmentation_test;
- основная таблица: orders (моделирует таблицу заказов или записей трекинга) [8];
- объем данных: более 500 тыс. записей.

Структура таблицы:

```
sql
CREATE TABLE orders (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  product_id INT NOT NULL,
  amount DECIMAL(10,2),
  status VARCHAR(20),
  created_at TIMESTAMP DEFAULT CURRENT_
TIMESTAMP,
  INDEX idx_user(user_id),
  INDEX idx_product(product_id)
) ENGINE=InnoDB;
```

Выбор такой структуры обусловлен тем, что она типична для OLTP-систем [2, 3]: есть первичный

Рис. 2. Варианты фрагментации

ключ-автоинкремент, два внешних логических поля для фильтрации, числовой атрибут (amount) и статус. Вторичные индексы позволяют оценить влияние фрагментации на индексные сканирования [6, 11].

Инструменты нагрузочного тестирования и анализа:

- sysbench — для генерации смешанной OLTP-нагрузки [6] (подготовка, запуск, очистка);
- INFORMATION_SCHEMA — для мониторинга data_length, index_length, data_free [1];
- EXPLAIN ANALYZE — для анализа реального плана выполнения запросов и оценки стоимости [2];
- системные утилиты: iostat, vmstat — для оценки I/O [5];
- MySQL Workbench — для выполнения SQL-скриптов и визуализации.

Этапы эксперимента

Этап 1. Создание эталонного состояния (без фрагментации):

- создание таблицы и индексов [1];
- заполнение таблицы (500 тыс. строк) с помощью хранимой процедуры generate_orders();
- выполнение OPTIMIZE TABLE для гарантии отсутствия фрагментации [4];
- замер метрик производительности [6].

Этап 2. Моделирование фрагментации:

- последовательное выполнение операций:
  - удаление 33 % записей (DELETE FROM orders WHERE id % 3 = 0);
  - обновление 20 % записей (UPDATE orders SET status = ‘archived’ WHERE id % 5 = 0);
  - вставка 100 тыс. новых записей (частично заполняют образовавшиеся дыры, создавая хаотичную структуру) [4, 5];
- замер метрик производительности [6].

Этап 3. Оптимизация (устранение фрагментации) [4, 7]:

- выполнение OPTIMIZE TABLE orders;
- повторный замер метрик [6].

Измеряемые метрики:

- время выполнения SELECT (точечные запросы по индексу user_id и диапазонные по id) [2, 6];
- скорость INSERT (операций в секунду) [5];
- скорость UPDATE неиндексируемого поля (amount) и индексируемого поля (status) [11];
- латентность (p95, p99 по данным sysbench) [6];
- количество операций чтения / записи на диск (IOPS) [5];
- размер таблицы и индексов на диске (через INFORMATION_SCHEMA) [1];
- показатель data_free (незанятое пространство внутри таблицы) [4].

Результаты эксперимента

Состояние таблицы по данным INFORMATION_SCHEMA

На рис. 3 приведен результат запроса к INFORMATION_SCHEMA.TABLES.

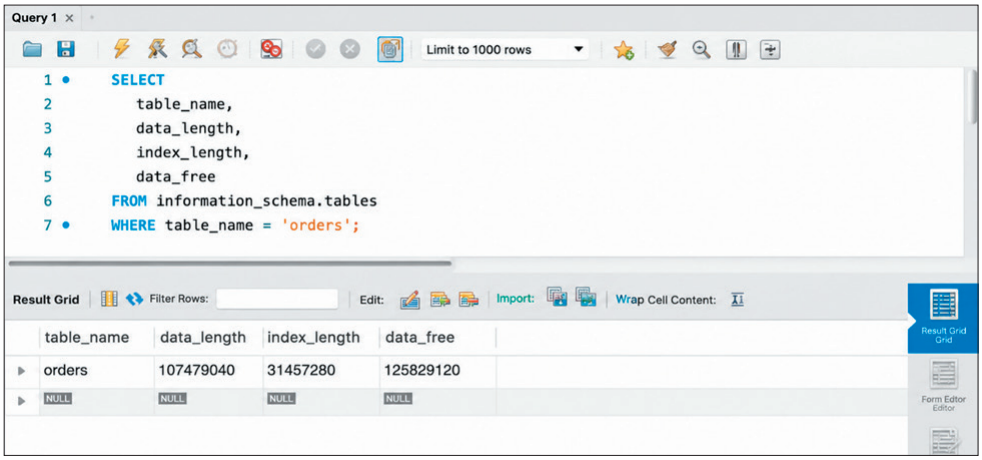

Рис. 3. Проверка состояния таблицы с помощью запроса

Ключевые показатели данных и индексов [1, 4] приведены в табл. 1.

Рост data_free с 2 до 38 Мб свидетельствует о появлении значительного количества пустого пространства внутри страниц — это прямой индикатор фрагментации [2, 4].

Влияние на время выполнения запросов

Измерения проводились для трех типовых запросов:

- запрос 1 (SELECT по вторичному индексу):

sql
SELECT * FROM orders WHERE user_id = 100;

- запрос 2 (INSERT новой записи):

sql
INSERT INTO orders (user_id, product_id, amount, status) VALUES (1001, 501, 250.50, ‘new’);

- запрос 3 (UPDATE статуса):

sql
UPDATE orders SET status = ‘processed’ WHERE id = 1000;

Результаты экспериментального оценивания среднего времени выполнения запросов приведены в табл. 2.

Нагрузочное тестирование (sysbench)

С помощью sysbench проводилось смешанное тестирование oltp_read_write с 8 потоками в течение 60 с. [6]. Результаты приведены в табл. 3.

Интерпретация:

- TPS упал в 2,37 раза при фрагментации [5, 6];
- латенция P95 выросла в 2,4 раза [6];
- IOPS выросли в 2,6 раза — система вынуждена читать больше разрозненных страниц с диска, несмотря на кэширование [2, 5].

Сравнительный анализ и интерпретация

Основные метрики, используемые для сравнительного анализа и интерпретации результатов, приведены в табл. 4.

Ключевые выводы из измерений [4–6]:

1. SELECT страдает сильнее всего — замедление в 3,2 раза. Причина: сканирование фрагментированного индекса idx_user требует чтения большего количества страниц, многие из которых заполнены менее чем на 50 % [2, 4].

2. INSERT и UPDATE замедляются в 2–3 раза**.** Основной вклад вносит механизм page split: при вставке новых записей в неплотные страницы

Таблица 1

Ключевые показатели данных и индексов

Состояние	data_length, Мб	index_length, Мб	data_free, Мб
Без фрагментации	~180	~45	~2
С фрагментацией	~195	~52	~38
После OPTIMIZE	~178	~44	~1

Таблица 2

Среднее время выполнения запросов

Состояние БД	SELECT (user_id), мс	INSERT, мс	UPDATE (status), мс
Без фрагментации	12	7	10
С фрагментацией	38	21	28
После OPTIMIZE	14	8	11

Таблица 3

Результаты нагрузочного тестирования

Состояние БД	TPS, транзакций/с	P95 latency, мс	IOPS (чтение)
Без фрагментации	185	28	340
С фрагментацией	78	67	890
После OPTIMIZE	179	30	365

Таблица 4

Основные метрики для сравнительного анализа и интерпретации результатов

Метрика	Без фрагментации	С фрагментацией	После OPTIMIZE
SELECT (по индексу), мс	12	38 (↑ 217%)	14
INSERT, мс	7	21 (↑ 200%)	8
UPDATE (индекс), мс	10	28 (↑ 180%)	11
TPS (sysbench)	185	78 (↓ 58%)	179
P95 latency, мс	28	67 (↑ 139%)	30
Размер данных + индексов, Мб	225	247 (↑ 10%)	222
data_free, Мб	2	38 (↑ 1800%)	1

InnoDB вынуждена выполнять дополнительные операции перераспределения [1, 5].

3. OPTIMIZE TABLE восстанавливает производительность до 95–98% от исходного уровня [4, 7]. Небольшое остаточное отставание связано с тем, что после оптимизации структура данных становится даже более плотной, чем в исходной (за счет устранения начальных дыр) [2].

4. data_free — наиболее наглядный индикатор фрагментации [4]. Его рост с 2 до 38 Мб коррелирует с падением TPS на 58 % [6]. Рекомендуется проводить дефрагментацию при data_free > 20 % от data_length.

На рис. 4 приведена диаграмма, визуализирующая замедление всех типов операций на фрагментированной таблице и полное восстановление после OPTIMIZE TABLE.

Наиболее критичное замедление наблюдается для операций чтения, что объясняется сканированием фрагментированных страниц вторичных индексов и ростом числа случайных операций ввода-вывода [1, 11]. В то же время выполнение команды OPTIMIZE TABLE позволяет практически полностью восстановить исходную производительность.

Методы устранения фрагментации

OPTIMIZE TABLE

Команда OPTIMIZE TABLE orders перестраивает таблицу и все ее индексы, освобождая незанятое пространство и уплотняя данные [1, 4]. Для InnoDB эта команда:

- создает новую пустую таблицу;
- копирует в нее данные из исходной в порядке первичного ключа;
- перестраивает все вторичные индексы;
- удаляет старую таблицу и переименовывает новую [2, 7].

В данном эксперименте OPTIMIZE TABLE сократил время выполнения SELECT в 2,7 раза по сравнению с фрагментированным состоянием, а размер таблицы уменьшил на 10% (с 247 до 222 Мб).

Недостаток метода: требует блокировки таблицы на время выполнения (для InnoDB — не полной блокировки, но значительной нагрузки на I/O) [4, 6]. Рекомендуется проводить в периоды низкой активности системы (например, ночью) [8, 10].

ALTER TABLE ... ENGINE=InnoDB

Альтернативный способ ALTER TABLE orders ENGINE=InnoDB [1] заставляет InnoDB пересоздать таблицу заново аналогично OPTIMIZE

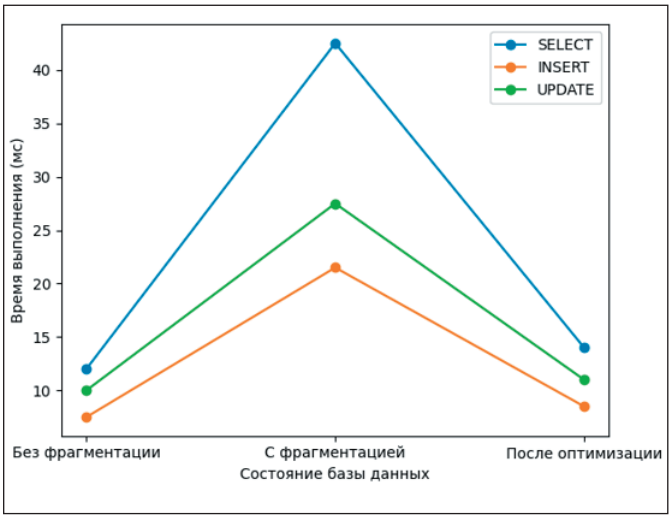

Рис. 4. Влияние фрагментации на производительность MySQL InnoDB

TABLE [4, 7]. По эффективности сопоставим. Может использоваться, если OPTIMIZE по каким-то причинам недоступен (например, в некоторых репликационных конфигурациях) [2].

Профилактические меры (архитектурные решения)

1. Использование последовательных первичных ключей (AUTO_INCREMENT). Избегать случайных значений (UUID, хеши) в качестве PK [2, 5].

2. Настройка innodb_fill_factor (доступен в MariaDB и некоторых сборках MySQL) — резервирование места (например, 90 %) на страницах для будущих обновлений, что снижает частоту page split [4].

3. Регулярный мониторинг через INFORMATION_SCHEMA.TABLES — обращать внимание на поле data_free [1, 6].

4. Партиционирование больших таблиц — разделение данных на независимые физические сегменты [3, 10].

Мониторинг фрагментации в production

Простейший SQL-запрос для мониторинга (использовался в эксперименте):

```
sql
SELECT
  table_name,
  ROUND(data_length / 1024 / 1024, 2) AS data_mb,
  ROUND(index_length / 1024 / 1024, 2) AS index_mb,
  ROUND(data_free / 1024 / 1024, 2) AS free_mb,
  ROUND((data_free / data_length) * 100, 2) AS free_pct
FROM information_schema.tables
WHERE table_schema = ‘fragmentation_test’
 AND table_name = ‘orders’;
```

Эмпирическое правило [4]: если free_pct > 20 % — рекомендуется дефрагментация. При free_pct > 40 % дефрагментация обязательна, так как производительность уже существенно деградировала [6, 7].

Выводы

В результате проведенного экспериментального исследования установлено:

1. Фрагментация данных в InnoDB приводит к снижению производительности в 2–3,5 раза в зависимости от типа операции [4, 5]. Наиболее сильно страдают запросы чтения, использующие вторичные индексы (замедление до 3,2 раза) [2, 6].

2. Операции записи (INSERT, UPDATE индексируемых полей) замедляются в 2–3 раза из-за увеличения числа page split и большего размера индексов [1, 11].

3. Метод OPTIMIZE TABLE полностью восстанавливает производительность до 95–98 % от исходного уровня, а размер таблицы уменьшает на 10–15 % за счет уплотнения страниц [4, 7].

4. Показатель data_free из INFORMATION_SCHEMA.TABLES является простым и надежным индикатором фрагментации [1, 4]. Его рост с 2 до 38 Мб в эксперименте коррелирует с падением TPS на 58 % [6].

5. Фрагментация накапливается нелинейно — первые массовые DELETE и UPDATE создают дыры, а последующие хаотичные вставки закрепляют фрагментацию, делая структуру максимально неоптимальной [2, 5].

Для транспортных информационных систем, где интенсивность обновлений (статусы заказов, координаты трекинга) и удалений (очистка истории) высока, проблема фрагментации стоит особенно остро [8–10]. Игнорирование плановой дефрагментации приводит к постоянному росту времени отклика и, как следствие, нарушению SLA [6, 10].

Практические рекомендации для транспортных систем

На основе проведенного исследования сформулированы следующие практические рекомендации для администраторов баз данных и разработчиков:

Регламентные меры:

1. Настроить автоматический сбор метрик data_free для всех таблиц с интенсивными изменениями (таблицы заказов, трекинга, статусов) [4, 8].

2. Выполнять дефрагментацию (OPTIMIZE TABLE) в периоды минимальной нагрузки, например с 2:00 до 4:00, с частотой 1 раз в 1–2 недели для высоконагруженных таблиц [6, 10].

3. Использовать партиционирование для больших таблиц (более 10 млн записей), чтобы

дефрагментировать только изменяемые партиции, а не всю таблицу [3, 9].

Архитектурные решения:

1. Применять последовательные первичные ключи (AUTO_INCREMENT) [2, 5]. Если требуется глобальный уникальный идентификатор (например, для синхронизации между филиалами), добавить отдельное поле uuid с индексом, но PK оставить AUTO_INCREMENT [1, 4].

2. Минимизировать обновления индексируемых полей. Если поле часто меняется, рассмотреть возможность выноса его в отдельную таблицу или замены статусной модели на журналирование событий (журнал изменений) [7, 11].

3. Разделять горячие и холодные данные — старые записи перемещать в таблицы-архивы, что снижает фрагментацию в оперативной таблице [8, 10].

Операционный мониторинг:

1. Контролировать размер буферного пула (innodb_buffer_pool_size). При фрагментации эффективность кэша падает, поэтому увеличение буферного пула может временно компенсировать падение производительности, но не устраняет причину [3, 6].

2. Использовать EXPLAIN ANALYZE для запросов, которые стали выполняться дольше обычног. Часто проблема кроется во фрагментированном индексе, а не в самом запросе [2, 5].

Заключение

Фрагментация данных в MySQL (InnoDB) является скрытым, но управляемым фактором деградации производительности [4, 5]. Как показывает эксперимент, ее негативное влияние на чтение и запись может быть весьма существенным: замедление в 2–3,5 раза. Однако современные средства (OPTIMIZE TABLE, ALTER TABLE ... ENGINE=InnoDB) позволяют эффективно восстанавливать производительность [4, 7], а превентивные архитектурные решения (последовательный первичный ключ, мониторинг data_free) — долгое время поддерживать базу данных в оптимальном состоянии [2, 6].

Особую значимость исследование имеет для интеллектуальных транспортных систем, где высока интенсивность обновлений (GPS-треки, статусы заказов, логистические события) и удалений (очистка истории) [8–10]. Внедрение регулярной дефрагментации должно стать обязательной частью регламента эксплуатации таких систем [6].

Результаты работы могут быть полезны администраторам баз данных, разработчикам и техническим архитекторам, применяющим MySQL в производственных средах с высокой интенсивностью изменений данных [3, 4, 7].

СПИСОК ИСТОЧНИКОВ

1. The InnoDB Storage Engine Oracle // MySQL 8.0 Reference Manual. URL: http://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html (дата обращения: 13.05.2026).

2. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication. Third Edition. Sebastopol (CA): O'Reilly Media, 2012. 826 p.

3. Хомоненко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных: учебник для вузов / под ред. А. Д. Хомоненко. 6-е изд., доп. СПб.: Корона-Век, 2009. 736 с.

4. Zaitsev P. MySQL File System Fragmentation Benchmarks // Percona Blog. 22.03.2008. URL: http://www.percona.com/blog/mysql-file-system-fragmentation-benchmarks (дата обращения: 13.05.2026).

5. Huang S.-M., Chang L.-P. Exploiting Page Correlations for Write Buffering in Page-Mapping Multichannel SSDs // ACM Transactions on Embedded Computing Systems. 2016. Vol. 15, iss. 1. Art. no. 12. 25 p. DOI: 10.1145/2815622

6. Малютин А. Г., Лаврухин А. А., Окишев А. С. Архитектурные аспекты реализации корпоративной информационной системы мониторинга и учета ресурсов // Известия Транссиба. 2017. № 4 (32). С. 130–141.

7. Павлов Д. В. Реляционная распределенная система управления базами данных с автоматической масштабируемостью // Вестник Уфимского государственного авиационного технического университета. 2012. Т. 16, № 3 (48). С. 143–152.

8. Малыгин Д. С. Проблемы производительности реляционных баз данных в распределенных архитектурах и стратегии их решения // Современные наукоемкие технологии. 2024. № 10. С. 61–71. DOI: 10.17513/snt.40173

9. Харченко П. А. Совершенствование системы и технологии эксплуатации подвижного состава на основе анализа информации средств регистрации параметров движения, диагностики и мониторинга технического состояния локомотива // Вестник Ростовского государственного университета путей сообщения. 2025. № 1 (97). С. 201–210. DOI: 10.46973/0201-727X_2025_1_201

10. Безфамильный Д. Г. Концептуальные подходы к оптимизации жизненного цикла разработки программного обеспечения // Современные инновации, системы и технологии. 2026. Т. 6, № 2. С. 1027–1034. DOI: 10.47813/2782-2818-2026-6-2-1027-1034

11. Нихтер Д. Настройка производительности MySQL. Секреты и приемы / пер. с англ. СПб.: БХВ-Петербург, 2023. 340 с.

Дата поступления: 21.05.2026
Решение о публикации: 21.05.2026

Investigation of Data Fragmentation Impact on Read/Write Performance and Methods for Its Elimination in MySQL (InnoDB)

Sergey D. Cherkasov[1] — 2nd year Bachelor's Degree Student in 38.03.05 Business Informatics. Research interests: information systems, development and research of characteristics of applications for working with databases. E-mail: cherkasovsergeidx@mail.ru

Anatoly D. Khomonenko[1,2] — Dr. Sci. in Engineering, Full Professor, Professor of the "Information and Computing Systems" Department; Professor of the Department of Mathematical and Software Engineering. Research interests: information systems, big data processing, probabilistic modeling of geographic information systems, genetic algorithms, information security. E-mail: khomon@mail.ru

[1]Emperor Alexander I St. Petersburg State Transport University, 9 Moskovsky ave., Saint Petersburg, 190031, Russia

[2]Mozhaisky Military Aerospace Academy, 13 Zhdanovskaya str., Saint Petersburg, 197198, Russia

For citation: Cherkasov S. D., Khomonenko A. D. Investigation of Data Fragmentation Impact on Read/Write Performance and Methods for Its Elimination in MySQL (InnoDB), *Intellectual Technologies on Transport*, 2026, no. 2 (46), pp. 64–74. DOI: 10.20295/2413-2527-2026-246-64-74 (In Russian)

Abstract. *An experimentally substantiated approach to assessing the impact of data fragmentation on MySQL (InnoDB) performance and selecting a method for its elimination is proposed.* ***Purpose:*** *to study the dependence of read and write operation execution time on the degree of data fragmentation and to determine the most effective defragmentation method.* ***Methods:*** *analysis of InnoDB architecture (B+ trees, page split); creation of an experimental testbed based on MySQL 8.0 in Docker; generation of a test table with 500 000 records; simulation of fragmentation through massive DELETE, UPDATE, and chaotic INSERT operations; load testing using sysbench; measurement of query time, latency, IOPS, and data_free.* ***Results:*** *fragmentation slows down SELECT by 2,5–3,5 times, INSERT and UPDATE by 2–3 times. An increase in data_free from 2 to 38 MB correlates with a 58% drop in TPS. OPTIMIZE TABLE restores performance to 95–98% of the original level.* ***Practical significance:*** *monitoring and scheduled defragmentation techniques are applicable to transport information systems with high data update intensity (tracking, logistics).*

Keywords: *data fragmentation, MySQL, InnoDB, performance, page split, optimize table, B+ tree, data_free, intelligent transport systems*

REFERENCES

1. The InnoDB Storage Engine Oracle, *MySQL 8.0. Reference Manual*. Available at: http://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html (accessed: May 13, 2026).

2. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication. Third Edition. Sebastopol (CA), O'Reilly Media, 2012, 826 p.

3. Khomonenko A. D., Tsygankov V. M., Maltsev M. G. Bazy dannykh: uchebnik dlya vuzov [Databases: A Textbook for Universities]. Saint Petersburg, Korona-Vek Publishing House, 2009, 736 p. (In Russian)

4. Zaitsev P. MySQL File System Fragmentation Benchmarks, *Percona Blog*. Published online at March 22, 2008. Available at: http://www.percona.com/blog/mysql-file-system-fragmentation-benchmarks (accessed: May 13, 2026).

5. Huang S.-M., Chang L.-P. Exploiting Page Correlations for Write Buffering in Page-Mapping Multichannel SSDs, *ACM Transactions on Embedded Computing Systems*, 2016, vol. 15, iss. 1, art. 12, 25 p. DOI: 10.1145/2815622

6. Malyutin A. G., Lavrukhin A. A., Okishev A. S. Arkhitekturnye aspekty realizatsii korporativnoy informatsionnoy sistemy monitoringa i ucheta resursov [Architectural Aspects of the Implementation of the Corporate Information System for Monitoring and Resources Accounting], *Izvestiya Transsiba [Journal of Transsib Railway Studies]*, 2017, no. 4 (32), pp. 130–141. (In Russian)

7. Pavlov D. V. Relyatsionnaya raspredelennaya sistema upravleniya bazami dannykh s avtomaticheskoy masshtabiruemostyu [Distributed Relational Database Management System with Automatic Scalability], *Vestnik Ufimskogo gosudarstvennogo aviatsionnogo tekhnicheskogo Universiteta [Vestnik of Ufa State Aviation Technical University]*, 2012, Vol. 16, no. 3 (48), pp. 143–152. (In Russian)

8. Malygin D. S. Problemy proizvoditelnosti relyatsionnykh baz dannykh v raspredelennykh arkhitekturakh i strategii ikh resheniya [Performance Issues of Relational Databases in Distributed Architectures and Strategies for Resolution], *Sovremennye naukoemkie tekhnologii [Modern High Technologies]*, 2024, no. 10, pp. 61–71. DOI: 10.17513/snt.40173 (In Russian)

9. Kharchenko P. A. Sovershenstvovanie sistemy i tekhnologii ekspluatatsii podvizhnogo sostava na osnove analiza informatsii sredstv registratsii parametrov dvizheniya, diagnostiki i monitoringa tekhnicheskogo sostoyaniya lokomotiva [Improvement of the System and Technology of Rolling Stock Operation Based on the Analysis of Information from Motion Parameter Registration, Diagnostics and Monitoring of the Technical Condition of the Locomotive], *Vestnik Rostovskogo Gosudarstvennogo Universiteta Putej Soobshcheniya*, 2025, no. 1 (97), pp. 201–210. DOI: 10.46973/0201–727X_2025_1_201 (In Russian)

10. Bezfamilnyi D. G. Kontseptualnye podkhody k optimizatsii zhiznennogo tsikla razrabotki programmnogo obespecheniya [Conceptual Approaches to Optimizing the Software Development Lifecycle], *Sovremennye innovatsii, sistemy i tekhnologii [Modern Innovations, Systems and Technologies]*, 2026, vol. 6, no. 2, pp. 1027–1034. DOI: 10.47813/2782-2818-2026-6-2-1027-1034 (In Russian)

11. Nichter D. Nastroyka proizvoditelnosti MySQL. Sekrety i priemy [Efficient MySQL Performance: Best Practices and Techniques]. Saint Petersburg, BHV-Peterburg Publishing House, 2023, 340 p. (In Russian)

Received: May 21, 2026
Accepted: May 21, 2026