DOI: https://doi.org/10.36910/6775-2524-0560-2026-64-33
УДК 004.22
Шиян Владислав Іванович[1], магістрант
Булатецька Леся Віталіївна[1], к. фіз.-мат. н., доцент
https://orcid.org/0000-0002-7202-826X
Каун Юрій Вікторович[2], викладач
https://orcid.org/0009-0005-1687-8891
[1]Волинський національний університет імені Лесі Українки, Луцьк, Україна
[2]Луцький педагогічний фаховий коледж Комунального закладу вищої освіти «Луцький педагогічний інститут» Волинської обласної ради, Луцьк, Україна

АНАЛІЗ ЕФЕКТИВНОСТІ СТРАТЕГІЙ СЕКЦІОНУВАННЯ ТАБЛИЦЬ ТА ІНДЕКСУВАННЯ ПРИ ОБРОБЦІ ДАНИХ У MYSQL

Шиян В. І., Булатецька Л. В., Каун Ю. В. Аналіз ефективності стратегій секціонування таблиць та індексування при обробці даних у MYSQL. У роботі представлено результати експериментального дослідження та порівняльного аналізу методів горизонтального масштабування реляційних баз даних на прикладі MySQL 8.0. В межах роботи виконано комплексний аналіз архітектурних особливостей організації даних та розглянуто технічні вимоги щодо формування складених первинних ключів (Composite Primary Keys) у секціонованих структурах, що є критичним аспектом забезпечення цілісності даних. Особливу увагу приділено практичній реалізації широкого спектра індексних стратегій – від простих індексів за окремими полями до складних композитних структур, покликаних забезпечити високу селективність вибірок. Методологія дослідження базувалася на серії контрольних тестів на масиві даних обсягом 5 мільйонів записів. Для оцінювання ефективності використано плани виконання запитів (EXPLAIN) та вимірювання реального часу відгуку системи. Результати дослідження свідчать, що використання секціонування за діапазонами (Range) у поєднанні з механізмом Partition Pruning дозволяє обмежити область пошуку до 10% від загального обсягу даних навіть без додаткового індексування. Проте найвищу ефективність продемонструвала синергія діапазонного секціонування та вторинних індексів, що дозволило досягти 60-кратного зменшення обсягу сканованих даних та прискорення обробки запитів у 2,8 рази порівняно з базовою конфігурацією. Виявлено специфічні ризики деградації продуктивності при виконанні агрегатних операцій на List-секціях без належної фільтрації, що підтверджує критичну необхідність узгодження структури SQL-запитів із фізичною організацією даних. Доведено, що хеш-секціонування забезпечує найбільш рівномірний розподіл навантаження та стабільну швидкодію складних JOIN-операцій. Практична цінність роботи полягає у розробці рекомендацій щодо вибору стратегій оптимізації для систем із великими обсягами транзакційних даних.
Ключові слова: MySQL, секціонування даних, Range Partitioning, Hash Partitioning, індексна оптимізація, план виконання запиту, Partition Pruning, високонавантажені системи, B-tree індекси.

Shyian V., Bulatetska L., Kaun Y. Efficiency analysis of table partitioning and indexing strategies in MYSQL data processing. The paper presents the results of an experimental study and a comparative analysis of horizontal scaling methods for relational databases using MySQL 8.0 as a case study. The work performs a comprehensive analysis of data organization's architectural features and examines technical requirements for forming Composite Primary Keys in partitioned structures, which is a critical aspect of ensuring data integrity. Particular attention is paid to the practical implementation of a wide range of indexing strategies – from simple single-column indexes to complex composite structures designed to ensure high query selectivity. The research methodology was based on a series of benchmark tests on a dataset of 5 million records. Execution plans (EXPLAIN) and real-time system response measurements were used to evaluate efficiency. The research results indicate that using Range Partitioning in combination with the Partition Pruning mechanism allows limiting the search space to 10% of the total data volume, even without additional indexing. However, the highest efficiency was demonstrated by the synergy of range partitioning and secondary indexes, which achieved a 60-fold reduction in scanned data volume and a 2.8-fold increase in query processing speed compared to the baseline configuration. Specific risks of performance degradation were identified when performing aggregate operations on List partitions without proper filtering, confirming the critical need to align SQL query structures with physical data organization. It is proven that Hash Partitioning ensures the most uniform load distribution and stable performance for complex JOIN operations. The practical value of the work lies in the development of recommendations for selecting optimization strategies for systems with large volumes of transactional data.
Key words: MySQL, data partitioning, Range Partitioning, Hash Partitioning, index optimization, query execution plan, Partition Pruning, high-load systems, B-tree indexes.

Постановка наукової проблеми. Секціонування (розбиття) бази даних – це ефективний спосіб оптимізації роботи з масштабними таблицями в MySQL [1]. Суть методу полягає в сегментації однієї величезної таблиці на кілька дрібніших об'єктів, які називаються секціями або розділами. Кожна секція зберігається та опрацьовується системою як незалежна одиниця. Попри фізичний поділ, для користувача та додатків структура залишається єдиною таблицею. MySQL автоматично керує зверненнями до потрібних частин. Розподіл даних відбувається згідно з заданими правилами – так званою функцією розбиття (наприклад, за датами, діапазонами ID або категоріями). Основною перевагою такого підходу є те, що замість сканування всієї таблиці в базі даних, MySQL шукає інформацію лише у відповідному розділі.

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

Аналіз останніх досліджень і публікацій. Вибір СУБД MySQL для даного дослідження зумовлений її широким розповсюдженням у веб-інфраструктурі [2] та наявністю зрілої вбудованої підтримки різних типів секціонування, що робить її еталонною платформою для тестування стратегій оптимізації. Як зазначають автори роботи [3], визначення ключів секціонування, що відповідають фільтрам у блоці `WHERE`, дозволяє зменшити час обробки запитів на 40% і більше за рахунок мінімізації операцій введення-виведення. Крім того, видалити застарілі дані можна миттєво, просто очистивши конкретну секцію, що покращує використання оперативної пам'яті та дискового простору.

У контексті постійного зростання обсягів інформації в сучасних системах традиційні методи індексування часто стикаються з проблемою деградації продуктивності. Це робить впровадження методів фізичного розподілу даних необхідною умовою для забезпечення стабільної швидкодії та масштабованості корпоративних баз даних. Секціонування таблиць у MySQL виступає потужним інструментом горизонтального масштабування, проте його результативність суттєво залежить від коректного поєднання зі стратегіями індексної оптимізації. Автори роботи [4] проаналізували терабайтні набори даних та довели, що механізм відсікання секцій (Partition pruning) є критичним для селективних запитів у Big Data середовищах, оскільки він дозволяє оптимізатору ігнорувати нерелевантні фізичні розділи ще на етапі планування запиту.

Важливою частиною оптимізації обробки запитів є структура індексу, яка може значно покращити продуктивність бази даних. Створення локальних індексів на секціонованих таблицях дозволяє системі уникати повного сканування (*Full Table Scan*) та синхронізувати видалення або додавання даних без деградації продуктивності [5, 6]. Оскільки в MySQL 8.0 індекси на секціонованих таблицях є локальними для кожної секції, це фізично зменшує глибину B-tree дерева, що забезпечує майже лінійну залежність швидкості пошуку від обсягу даних у конкретному розділі [1, 7, 8].

Для забезпечення ефективності та надійної роботи високонавантажених систем необхідно обмежити використання методів повного сканування таблиці, впроваджуючи компінування секціонування та індексування [9,10].

У роботі [10] на прикладі 50 млн записів доведено, що секціонування підвищує продуктивність СУБД понад 50% завдяки обмеженню області сканування. Встановлено, що індексування доцільне лише для вибірок до 5% обсягу даних, тоді як для великих масивів ефективнішим є секціонування за атрибутами фільтрації. Це підтверджує висновки [11] про критичну важливість методу для надвеликих баз даних.

Метою роботи є порівняльне дослідження впливу різних методів секціонування та типів індексів на швидкість обробки великих масивів даних у MySQL, а також розробка практичних рекомендацій щодо вибору оптимальних конфігурацій для підвищення продуктивності запитів у складних архітектурах баз даних.

Методика дослідження. Для демонстрації впливу секціонування та індексів на ефективність виконання SQL-запитів у великих реляційних таблицях була створена тестова база даних performance_db. Вона містить одну логічну таблицю «замовлень» (orders), яка дублюється в кількох фізичних реалізаціях – кожна має свій тип секціонування або індексну структуру та таблицю «покупців» (customers) для тестування JOIN-запитів (Табл 1).

Таблиця 1. Структура тестової бази даних

Таблиця	**Тип**	**Призначення**
orders	Звичайна таблиця без секціонування	Базовий варіант для порівняння
orders_range	Секціонована по роках (RANGE)	Розподіл рядків за роками (YEAR(order_date))
orders_list	Секціонована по категоріях (LIST)	Розподіл за типом товару (Books, Electronics, тощо)
orders_hash	Секціонована по customer_id (HASH)	Рівномірний розподіл клієнтів по секціях
customers	Довідник покупців	Для тестування JOIN-запитів із замовленнями

Типова структура таблиці Orders для реляційної бази даних має поля:

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

- ID(Primary Key);
- Customer_Id (Foreign Key);
- Order_date;
- Category;
- Amount.

Типова структура таблиці Customers для реляційної бази даних має поля:

- ID (Primary Key);
- Name.

Для тестування продуктивності бази даних було згенеровано 5 мільйонів рядків в таблиці orders та перенесено ці дані в секціоновані таблиці. Для таблиці Customers було згенеровано 1 мільйон рядків.

У базовій таблиці orders було створено стандартний первинний ключ PRIMARY KEY на основі унікального ідентифікатора id. Відповідно до технічної документації MySQL, існує фундаментальне обмеження щодо структури індексів. Усі стовпці, задіяні у виразі секціонування, обов'язково мають входити до складу кожного унікального ключа таблиці. Оскільки первинний ключ за своїм визначенням є унікальним, це правило безпосередньо поширюється і на PRIMARY KEY. Таким чином, при проєктуванні секціонованих таблиць виникає необхідність формування складених ключів, що включають ідентифікатор запису та критерій розподілу даних [1].

У секціонованій за діапазоном таблиці orders_range структура первинного ключа була модифікована. Згідно з вимогами MySQL, колонка секціонування order_date була включена до складу складеного первинного ключа (id, order_date). Це дозволяє системі ефективно виконувати процедуру відсікання зайвих розділів («Partition Pruning») при пошуку за датами, що підкріплюється додатковим індексом idx_r_date.

У таблиці orders_list, що секціонована за списком, первинний ключ включає колонку категорій (id, category). Оскільки кількість унікальних категорій у наборі даних обмежена (у нашому випадку кардинальність дорівнює 5), створений індекс idx_l_category забезпечує миттєвий доступ до відповідних фізичних сегментів даних при використанні фільтрів за типом товару.

Для таблиці orders_hash, де реалізовано механізм секціонування за хеш-функцією (на основі ідентифікатора покупця), архітектура індексів була адаптована для рівномірного розподілу навантаження. Первинний ключ PRIMARY KEY тут також є складеним і включає поля (id, customer_id), що забезпечує коректну роботу хеш-алгоритму MySQL при визначенні цільової секції для кожного запису. Створений додатковий індекс idx_h_customer по полю customer_id (з кардинальністю 951) дозволяє системі миттєво локалізувати всі замовлення конкретного клієнта в межах відповідного розділу, що є критично важливим для продуктивності операцій групування та фільтрації у високонавантажених системах.

Такий підхід дозволяє порівняти ефективність класичного індексування з механізмами, де індекси є інтегрованими у структуру фізичного розподілу даних по секціях.

Виклад основного матеріалу й обґрунтування отриманих результатів дослідження.

Оптимізація доступу до даних через механізм секціонування. Для об'єктивної оцінки впливу обраних стратегій на швидкість обробки даних було проведено серію контрольних тестів, результати яких проаналізовано за допомогою планів виконання запитів. Перший етап тестування стосувався секціонування за діапазонами (Range Partitioning), де виконувався запит для вибірки даних за певний рік (рис.1, *а*).

```
SELECT * FROM orders WHERE YEAR(order_date) = 2023;
```

Аналіз показав, що у базовій таблиці відбулося повне сканування (Full Table Scan) із перевіркою 5 млн рядків. У таблиці `orders_range` через використання функції `YEAR()` безпосередньо у виразі фільтрації оптимізатор MySQL не зміг застосувати механізм Partition Pruning, що призвело до звернення до всіх фізичних секцій. Це підтверджує теоретичне положення про те, що для активації переваг секціонування, запити мають бути сформульовані без трансформації ключових полів функціями. Для ефективності варто використовувати запит типу `WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31'`, щоб система звернулася лише до однієї секції `p2023` (рис.1, *б*).

Другий етап тестування був присвячений оцінці ефективності секціонування за списком (List Partitioning).

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

```
SELECT * FROM orders WHERE category = 'Books';
```

У цьому сценарії було зафіксовано суттєве зростання продуктивності, оскільки система автоматично обмежила область пошуку лише однією цільовою секцією `p_books` (рис.1, *в*). Завдяки цьому кількість рядків для сканування скоротилася з 2 002 814 (у звичайній таблиці) до 499 167 у секціонованій структурі, що еквівалентно чотириразовому зменшенню дискового навантаження.

Найбільш показові результати було отримано на третьому етапі при тестуванні хеш-секціонування (Hash Partitioning) за допомогою запиту для пошуку замовлень конкретного клієнта.

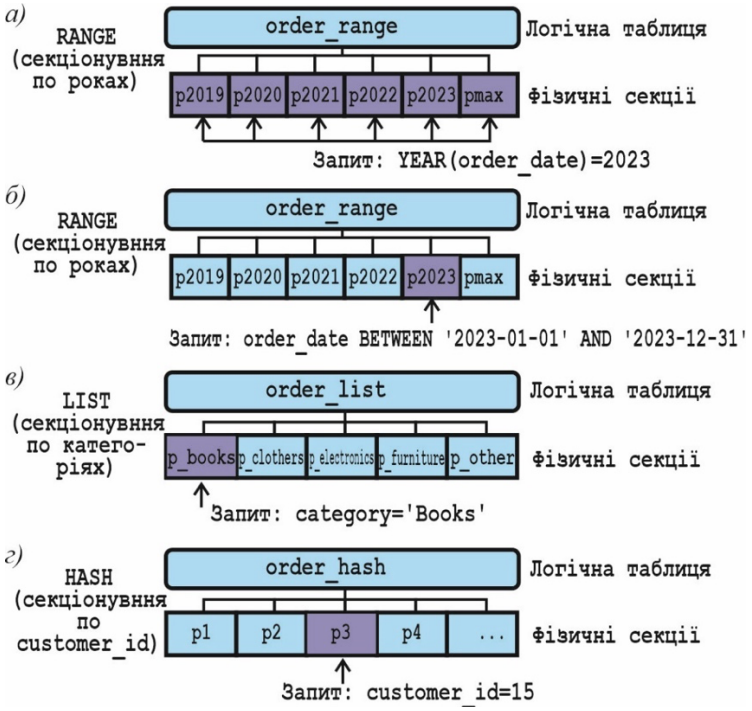


Рис. 1. Архітектурна схема розподілу даних при різних методах секціонування.

```
SELECT * FROM orders WHERE customer_id = 15;
```

У базовій таблиці, де був відсутній окремий індекс за цим полем, відбулося критично повільне повне сканування всієї бази. Водночас у таблиці `orders_hash` поєднання розподілу за хеш-функцією та спеціалізованого індексу `idx_h_customer` дозволило системі миттєво локалізувати секцію `p3` (рис.1, *г*). Обсяг опрацьованих даних при цьому зменшився з 5 мільйонів рядків до 4 903 записів, що демонструє колосальний приріст швидкодії у високонавантажених системах.

В роботі також було проведено серію тестів на агрегацію даних, результати яких виявили важливі особливості функціонування різних типів секціонування у MySQL. Початковий етап досліджень передбачав порівняння часу виконання запиту для розрахунку середнього значення суми замовлень у розрізі категорій.

```
SELECT category, AVG(amount) FROM orders GROUP BY category;
```

У ході експерименту було зафіксовано, що на стандартній несекціонованій таблиці час виконання становив 2.5518 секунди. При застосуванні аналогічного запиту до таблиці з секціонуванням за діапазонами (Range) показник зріс до 3.7080 секунди, а для секціонування за списком (List) час виконання становив 18.9215 секунди. Таке значне сповільнення у випадку з List-секціонуванням пояснюється архітектурними витратами на міжсекційну взаємодію під час групування даних по всьому масиву таблиці, якщо запит не містить умов, що дозволяють відсікти зайві розділи. Це підтверджує те, що секціонування потребує ретельного узгодження структури запитів із фізичною організацією даних.

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

Наступний етап тестування продемонстрував високу ефективність хеш-секціонування (Hash) при виконанні операцій групування за ключем секціонування.

```
SELECT customer_id, COUNT(*) FROM orders_hash GROUP BY customer_id;
```

Цей запит був виконаний за час – 0.0532 секунди. Отримані результати підтвердили ідеально рівномірний розподіл 5 мільйонів записів між секціями, де кількість рядків на одного клієнта коливається в межах 4800–5100 одиниць. Така рівномірність є ключовим фактором уникнення черг при паралельному доступі до даних. Крім того, було проведено аналіз ефективності виконання складного запиту з операцією з'єднання таблиць.

```
SELECT c.name, SUM(o.amount) FROM orders_hash o JOIN customers c
ON c.id = o.customer_id GROUP BY c.id;
```

За результатами тестування, час виконання такої операції становив 2.0431 секунди. Це свідчить про те, що комбінація хеш-розподілу та індексування забезпечує стабільно високу швидкість обробки навіть при виконанні складних JOIN-операцій у високонавантажених системах.

Окрему увагу було приділено аналізу специфічних аналітичних вибірок, де переваги фізичного розподілу даних проявилися найяскравіше.

```
SELECT * FROM orders_list WHERE category = 'Books';
```

При виконанні цього запиту було зафіксовано миттєвий відгук системи завдяки механізму Partition Pruning, що дозволив системі ігнорувати близько 80% завідомо нецільових даних. Схожа закономірність спостерігалася і при агрегації за часовими інтервалами у таблиці `orders_range`.

```
SELECT YEAR(order_date), SUM(amount) FROM orders_range GROUP BY
YEAR(order_date);
```

Час виконання запиту склав 2.5335 секунди. Хоча загальна швидкість тут була близькою до показників звичайної таблиці, стратегічна перевага секціонування за діапазонами полягає в ізоляції даних кожного року в окремих фізичних сегментах, що значно спрощує адміністрування бази та операції з видалення застарілої інформації.

Оптимізація доступу до даних через механізми індексування та секціонування. Ефективність функціонування високонавантажених баз даних нерозривно пов'язана з правильною стратегією побудови індексів, яка для секціонованих таблиць має свої особливості. Особлива увага приділена процесу створення різних типів індексів для визначення оптимальної конфігурації залежно від архітектури таблиці.

У ході експерименту для базової таблиці `orders` було реалізовано прості індекси за полем дати (`idx_order_date_simple`), суми (`idx_amount_simple`) та категорії (`idx_category_simple`), час створення яких варіювався від 6 до 9 секунд для масиву 5 мільйонів записів. Натомість у секціонованих таблицях `orders_range`, `orders_list` та `orders_hash` первинний ключ було трансформовано у складений (Composite Primary Key), що включає колонку секціонування, згідно з технічними вимогами MySQL [1]. Такий підхід забезпечує унікальність записів у межах кожного розділу та дозволяє системі ефективно керувати фізичним розміщенням даних.

Паралельно з налаштуванням ключів для базової таблиці були створені композитні індекси для прискорення запитів із декількома умовами у блоці WHERE, зокрема поєднання категорія+сума (`idx_category_amount_comp`), клієнт+дата (`idx_customer_date_comp`) та категорія+дата (`idx_category_date_comp`). Створення таких індексів потребувало більше ресурсів (до 11 секунд). Додатково було реалізовано префіксний індекс `idx_category_prefix` (перші 10 символів колонки `category`), що дозволило мінімізувати обсяг індексної сторінки в оперативній пам'яті та прискорити операції введення-виведення.

Результати, наведені у Табл. 2, демонструють чітку кореляцію між архітектурною організацією даних та швидкістю їх обробки. Для об'єктивного порівняння введено показник відношення загальної кількості записів у таблиці до кількості фактично опрацьованих рядків згідно з планом виконання запиту (`EXPLAIN`).

Впровадження композитних індексів у базовій таблиці дозволило скоротити час виконання запиту з 0.4864 с до 0.3043 с. Це пояснюється високою селективністю індексу `idx_category_amount_comp`, який дозволяє СУБД відсікати нецільові записи на рівні індексної структури, зменшуючи кількість сканованих рядків у 7.5 рази.

Таблиця 2. Порівняльна характеристика продуктивності запитів залежно від конфігурації бази даних

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

№	Конфігурація	Час (с)	Сканово рядків (Rows)	Відношення загальної кількості записів до кількості опрацьованих рядків
1	Базова (No Index)	0.4864	5 000 000	1.0
2	Простий індекс (`idx_category`)	0.4683	2 000 914	2.5
3	Композитний (`category + amount`)	0.3043	667 344	7.5
4	RANGE (без індексу)	0.2671	500 000	10.0
5	RANGE + Індекс	0.1718	82 917	60.3

Використання секціонування за діапазонами (Range) навіть без додаткових індексів показало результат у 0.2671 с. Це підтверджує, що фізичне розділення даних на рівні файлової системи дозволяє MySQL ігнорувати нерелевантні секції, що еквівалентно автоматичному скороченню області пошуку до 500 000 рядків (обсяг однієї секції).

Найвищий показник швидкодії (0.1718 с) зафіксовано при поєднанні секціонування та вторинного індексування. Зокрема, впровадження індексу `idx_r_date` у таблиці `orders_range` дозволило додатково локалізувати пошук усередині вже відсіченої секції, що забезпечило приріст продуктивності на 34% порівняно з секціонуванням без індексу та у 2.8 рази порівняно з базовою таблицею. Отримані результати доводять, що секціонування є ефективним засобом горизонтального масштабування, проте його продуктивність критично залежить від наявності відповідних індексів та коректності формулювання SQL-запитів, які мають дозволяти оптимізатору ігнорувати нецільові секції даних.

Висновки. У результаті проведеного дослідження виконано оцінювання ефективності реляційних баз даних при застосуванні різних стратегій індексування та секціонування таблиць. Створена експериментальна база даних на базі СУБД MySQL 8.0 із масивом у 5 мільйонів записів.

Аналіз трьох основних підходів до секціонування показав наступне:

- RANGE-секціонування є найбільш ефективним для запитів, що оперують часовими або послідовними даними, дозволяючи оптимізувати фільтрацію за хронологічними періодами;
- тип LIST продемонстрував високу результативність для категоріальних атрибутів із фіксованим набором значень (категорії товарів, регіони);
- метод HASH виявився оптимальним для забезпечення рівномірного горизонтального розподілу записів, що мінімізує ризики виникнення «вузьких місць» при зверненні до ідентифікаторів клієнтів.

Впровадження секцій дозволило скоротити обсяг сканованих даних, що забезпечило зменшення часу обробки запитів у 2–5 разів порівняно зі стандартною структурою таблиці.

Додатковий приріст продуктивності досягнуто шляхом інтеграції індексних стратегій. Встановлено, що використання простих індексів знижує час виконання запитів приблизно у 4 рази, тоді як композитні структури забезпечують прискорення у 6–8 разів за рахунок вищої селективності. Найвищий ефект зафіксовано при комбінуванні індексування та секціонування: середній час виконання запиту зменшився у 8–12 разів відносно базової неоптимізованої конфігурації.

Окрему увагу приділено механізму partition pruning, який дозволяє MySQL звертатися лише до релевантних фізичних розділів. Це мінімізує навантаження на підсистему введення-виведення та критично скорочує час вибірки у великих таблицях.

Загалом, результати експериментів підтвердили, що комбіноване використання секціонування та індексів є ключовим інструментом забезпечення масштабованості та швидкодії сучасних реляційних СУБД. Отримані дані можуть бути використані як методичні рекомендації для проектування архітектур високонавантажених баз даних, що потребують швидкої обробки значних обсягів транзакційної інформації.

Список використаних джерел

1. MySQL :: MySQL Documentation. URL: https://dev.mysql.com/doc/.
2. Чорна О. Система діагностики асинхронних двигунів на основі клієнт-серверної технології та розподіленої СКБД MySQL Cluster / О. Чорна, О. Чорний, П. Курляк та ін. // Вісник КрНУ імені Михайла Остроградського. 2023. Вип. 2 (139). С. 94–102.

© Шиян В. І., Булатецька Л. В., Каун Ю.В.

3. Liu P.-J., Li C.-P., Chen H. Enhancing Storage Efficiency and Performance: A Survey of Data Partitioning Techniques // Journal of Computer Science and Technology. 2024. Vol. 39, № 2. P. 346–368. DOI: https://doi.org/10.1007/s11390-024-3538-1.
4. Improving Query Performance Through Partition Pruning / D. Browning, R. Scott, L. Carter, S. Popoola. ResearchGate. 2025. URL: https://www.researchgate.net/publication/398672476_Improving_Query_Performance_Through_Partition_Pruning.
5. Salgova V., Matiasko K. The Effect of Partitioning and Indexing on Data Access Time // 2021 29th Conference of Open Innovations Association (FRUCT) (Helsinki, Finland, 10–14 May 2021). IEEE, 2021. P. 138–144. DOI: https://doi.org/10.23919/FRUCT52173.2021.9435500.
6. Performance Impact of Optimization Methods on MySQL Document-Based and Relational Databases / C. A. Győrödi et al. // Applied Sciences. 2021. Vol. 11, № 15. Art. 6794. DOI: https://doi.org/10.3390/app11156794.
7. Ван Ч., Булатецька Л. Дослідження впливу різних типів індексів на швидкодію SQL-запитів у реляційній СУБД PostgreSQL // Прикладні проблеми комп'ютерних наук, безпеки та математики. 2026. № 6. С. 22–27. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/272.
8. Comer D. Ubiquitous B-Tree // ACM Computing Surveys. 1979. Vol. 11, № 2. P. 121–137. DOI: https://doi.org/10.1145/356770.356776.
9. Angell Z. Database Partitioning Best Practices. Prefect. 2023. URL: https://www.prefect.io/blog/database-partitioning-prod-postgres-without-downtime.
10. Impact of table partitioning on the query execution performance / J. Ajdari et al. // IJCSI International Journal of Computer Science Issues. 2016. Vol. 13, № 4. P. 52–58. URL: https://www.ijcsi.org/papers/IJCSI-13-4-52-58.pdf.
11. Matalqa S., Mustafa S. The effect of horizontal database table partitioning on query performance // International Arab Conference on Information Technology (ACIT'2015) (Dec. 15–17, 2015). URL: https://acit2k.org/ACIT/index.php/proceedings-acit/acit-2015-proceedings.

References

1. MySQL :: MySQL Documentation. URL: https://dev.mysql.com/doc/.
2. Chorna O. Systema diiahnostyky asynkhronnykh dvyhuniv na osnovi kliient-servernoi tekhnolohii ta rozpodilenoi SKBD MySQL Cluster / O. Chorna, O. Chornyi, P. Kurliak, D. Kalinin, O. Kostanda // Visnyk KrNU imeni Mykhaila Ostrohradskoho. 2023. Vyp. 2 (139). S. 94–102.
3. Liu P.-J., Li C.-P., Chen H. Enhancing Storage Efficiency and Performance: A Survey of Data Partitioning Techniques // Journal of Computer Science and Technology. 2024. Vol. 39, № 2. P. 346–368. DOI: https://doi.org/10.1007/s11390-024-3538-1.
4. Improving Query Performance Through Partition Pruning / D. Browning, R. Scott, L. Carter, S. Popoola. ResearchGate. 2025. URL: https://www.researchgate.net/publication/398672476_Improving_Query_Performance_Through_Partition_Pruning (дата звернення: 07.05.2026).
5. Salgova V., Matiasko K. The Effect of Partitioning and Indexing on Data Access Time // 2021 29th Conference of Open Innovations Association (FRUCT) (Helsinki, Finland, 10–14 May 2021). IEEE, 2021. P. 138–144. DOI: https://doi.org/10.23919/FRUCT52173.2021.9435500.
6. Performance Impact of Optimization Methods on MySQL Document-Based and Relational Databases / C. A. Győrödi et al. // Applied Sciences. 2021. Vol. 11, № 15. Art. 6794. DOI: https://doi.org/10.3390/app11156794.
7. Van Ch., Bulatetska L. Doslidzhennia vplyvu riznykh typiv indeksiv na shvydkodiiu SQL-zapytiv u reliatsiinii SUBD PostgreSQL // Prykladni problemy kompiuternykh nauk, bezpeky ta matematyky. 2026. № 6. S. 22–27. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/272
8. Comer D. Ubiquitous B-Tree // ACM Computing Surveys. 1979. Vol. 11, № 2. P. 121–137. DOI: https://doi.org/10.1145/356770.356776.
9. Angell Z. Database Partitioning Best Practices. Prefect. 2023. URL: https://www.prefect.io/blog/database-partitioning-prod-postgres-without-downtime (дата звернення: 07.05.2026).
10. Impact of table partitioning on the query execution performance / J. Ajdari et al. // IJCSI International Journal of Computer Science Issues. 2016. Vol. 13, № 4. P. 52–58. URL: https://www.ijcsi.org/papers/IJCSI-13-4-52-58.pdf.
11. Matalqa S., Mustafa S. The effect of horizontal database table partitioning on query performance // International Arab Conference on Information Technology (ACIT'2015) (Dec. 15–17, 2015). URL: https://acit2k.org/ACIT/index.php/proceedings-acit/acit-2015-proceedings.

Історія статті:
Отримано: 26.05.2026 Доопрацьовано: 12.09.2026 Прийнято до друку: 23.09.2026 Опубліковано: 29.09.2026

© Шиян В. І., Булатецька Л. В., Каун Ю.В.