ВІСНИК ПРИАЗОВСЬКОГО ДЕРЖАВНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
2026р. Серія: Технічні науки Вип. 53. Т. 1
p-ISSN: 2225-6733; e-ISSN: 2519-271X

УДК 004.65:004.45 DOI: 10.31498/2225-6733.53.1.2026.359786

ПІДХОДИ ТА МЕТОДИ ДО ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ СИСТЕМ УПРАВЛІННЯ БАЗАМИ ДАНИХ У СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

Воротнікова З.Є. *канд. техн. наук, доцент, ДВНЗ «Приазовський державний технічний університет», м. Дніпро, ORCID: https://orcid.org/0000-0002-0746-5981, e-mail: vorotnikova_z_j@pstu.edu*

У статті розглянуто методи підвищення продуктивності систем управління базами даних в умовах зростання обсягів даних і навантаження. Показано, що ефективність СКБД визначається архітектурою системи, структурою даних, способами доступу до них, а також ефективністю обробки запитів і транзакцій. Проаналізовано основні підходи до оптимізації, зокрема індексування, партиціонування, шардинг, кешування та реплікацію. Розглянуто вплив нормалізації та денормалізації на швидкодію системи; вибір підходу залежить від типу навантаження. Підкреслено, що продумане проєктування схеми бази даних дозволяє зменшити обсяг обробки та скоротити час виконання запитів. Окрему увагу приділено оптимізації SQL-запитів і аналізу планів їх виконання. Використання індексів (B-дерев, хеш- та повнотекстових) пришвидшує доступ до даних, але потребує врахування витрат під час запису. Зазначено роль матеріалізованих уявлень у прискоренні аналітичних запитів. Розглянуто вертикальне і горизонтальне масштабування та балансування навантаження як засобів ефективного використання ресурсів. Кешування зменшує навантаження на базу даних і прискорює доступ до часто використовуваних даних, за умови правильної стратегії оновлення. Реплікація підвищує доступність системи та дозволяє масштабувати читання; актуальною залишається проблема затримок у розподілених середовищах. Показано, що вибір між реляційними та нереляційними СКБД залежить від вимог до узгодженості, масштабованості та гнучкості, що обґрунтовує використання поліглотного підходу. Окремо відзначено застосування методів машинного навчання для автоматизації оптимізації. Отримані результати можуть бути використані для підвищення ефективності інформаційних систем.

Ключові слова: *продуктивність СКБД; оптимізація запитів; індексування; шардинг; реплікація; кешування; масштабування; розподілені системи; NoSQL; машинне навчання.*

Постановка проблеми

У сучасній цифровій економіці дані стали одним із ключових ресурсів, а їх обсяг постійно зростає. Це призводить до підвищених вимог до ефективності систем керування базами даних (СКБД), які забезпечують зберігання, обробку та доступ до інформації. Швидкість бізнес-процесів, стабільність інформаційних систем та рівень задоволеності користувачів безпосередньо залежать від продуктивності СКБД.

Проблема підвищення продуктивності СКБД полягає в необхідності забезпечення швидкої обробки запитів та транзакцій в умовах зростаючого навантаження та обмежених ресурсів. Продуктивність у цьому контексті розглядається як здатність системи мінімізувати час відгуку та ефективно обробляти значну кількість операцій. Водночас вона залежить від сукупності факторів: архітектури системи, характеристик апаратного забезпечення, структури даних, методів доступу до них та ефективності реалізації механізмів обробки запитів.

Ключовими аспектами проблеми є оптимізація структури бази даних, використання індексів, методів секціонування, шардування, кешування та реплікації, а також оптимізація SQL-запитів та проектування схем даних. Додаткову складність створює необхідність постійного моніторингу стану системи, виявлення вузьких місць та адаптації до змін навантаження. Для цього використовуються як вбудовані засоби СКБД, так і зовнішні засоби аналізу та візуалізації.

Незважаючи на наявність значної кількості підходів та інструментів оптимізації, питання їх ефективного поєднання з урахуванням конкретних умов використання залишається відкритим. Це визначає актуальність досліджень, спрямованих на аналіз методів підвищення продуктивності СКБД та оцінку їх практичної доцільності.

Таким чином, основними завданнями є виявлення факторів, які найбільше впливають на продуктивність СКБД, аналіз існуючих методів оптимізації та обґрунтування підходів до їх комплексного застосування. Вирішення цих завдань є важливим як для розробки теоретичних підходів до управління даними, так і для підвищення ефективності сучасних інформаційних систем.

Аналіз останніх досліджень та публікацій

За останнє десятиліття дослідження методів оптимізації продуктивності систем керування базами даних суттєво змістили фокус від класичних евристичних підходів до інтелектуальних, адаптивних і апаратно-орієнтованих рішень. Сучасні публікації відображають зміщення уваги з ізольованої оптимізації окремих компонентів СКБД до комплексного підходу, що охоплює всі рівні системи, від фізичного зберігання до планування виконання запитів і управління ресурсами.

Сучасні підходи до підвищення продуктивності СКБД орієнтовані на інтеграцію інтелектуальних методів, розподілених архітектур та ефективного використання апаратних ресурсів, див табл. 1.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Таблиця 1

Сучасні тенденції підвищення продуктивності СУБД

Напрям	Суть підходу	Переваги	Недоліки	Джерела
Інтелектуальна оптимізація запитів (AI/ML)	Використання ML-моделей для побудови планів виконання та оцінки вартості запитів	Вища точність, автоматизація	Потреба в даних, складність впровадження	[4–8]
Автономні СУБД	Самоналаштування параметрів, індексів, планів виконання	Зменшення ручного адміністрування	Недостатня зрілість технологій	[9], [10]
Прогнозний моніторинг	Аналіз і прогноз навантаження та вузьких місць	Проактивна оптимізація	Складність моделей	[1–3]
Cloud-native архітектури	Розділення обчислень і зберігання, масштабування у хмарі	Висока гнучкість і відмовостійкість	Залежність від мережі	[11], [12]
Оптимізація пам’яті та кешу	Використання in-memory, кешування, memory-aware алгоритмів	Значне прискорення доступу	Вартість пам’яті	[13], [14], [19]
Оптимізація транзакцій	MVCC, lock-free структури, детерміновані СУБД	Підвищення паралелізму	Складність реалізації	[15], [16]
Розподілені системи та консенсус	Використання Raft, Paxos для узгодженості	Масштабованість і надійність	Затримки, складність	[21], [22]
Апаратно-орієнтована оптимізація	Використання CPU cache, NUMA, GPU	Висока продуктивність	Залежність від архітектури	[17], [18]
Денормалізація та адаптивні моделі даних	Баланс між узгодженістю та швидкодією	Прискорення аналітики	Надмірність даних	[20]

Раніше домінували дослідження, спрямовані на вдосконалення класичних cost-based оптимізаторів запитів, однак із часом стало очевидно, що традиційні моделі оцінки вартості не здатні адекватно працювати в умовах складних і динамічних навантажень. Це призвело до активного розвитку підходів із використанням Machine Learning, зокрема learned cost models та learned query optimization. У роботах [4–6] показано, що моделі машинного навчання можуть значно точніше прогнозувати вартість виконання запитів порівняно з традиційними методами. Оптимізатори, що навчаються, досягають прискорення в 1,5–3 рази по геометричному середньому на стандартних тестових завданнях, одночасно знижуючи помилки оцінки кардинальності на порядок. Однак залишаються проблеми в узагальненні, інтерпретованості, вимогах до навчальних даних та розгортанні у виробничому середовищі. Значна кількість досліджень також спрямована на пошук більш ефективних методів оптимізації планів виконання запитів. Гібридні архітектури, що поєднують традиційні та навчені компоненти, пропонують найбільш перспективний шлях до впровадження у виробництво [7, 8]. Паралельно розвивається концепція автономних баз даних, популяризована Michael Stonebraker, яка передбачає автоматичне налаштування індексів, параметрів і стратегій виконання без втручання адміністратора [9, 10].

Розвиток розподілених систем і хмарних технологій істотно вплинув на підходи до оптимізації продуктивності. У контексті cloud-native СКБД, які розгортаються на платформах на кшталт Amazon Web Services, Google Cloud і Microsoft Azure, дослідження фокусуються на розділенні обчислювальних ресурсів і зберігання, а також на оптимізації мережевих взаємодій. У роботах [11, 12] для Amazon Aurora та інших сучасних систем показано, що така архітектура дозволяє досягти високої масштабованості та відмовостійкості, але потребує нових методів оптимізації доступу до даних і кешування. Проблемам кешування присвячені роботи [13, 14].

Окремий значний напрям становлять дослідження конкурентності та транзакцій [15, 16]. Протягом останніх 10 років активно вдосконалювалися методи Multiversion Concurrency Control, що дозволяють зменшити блокування і підвищити паралелізм. Паралельно розвиваються детерміновані СКБД, такі як Calvin, де порядок виконання транзакцій визначається наперед, що знижує витрати на синхронізацію. Також досліджуються lock-free та wait-free структури даних, які дозволяють ефективно працювати на багатоядерних системах.

У розподілених СКБД значна увага приділяється оптимізації розміщення даних і виконання запитів з урахуванням locality. Протягом десятиліття було вдосконалено алгоритми обробки розподілених запитів, а також протоколи узгодження, зокрема Raft Consensus Algorithm і Paxos Algorithm, які залишаються основою для забезпечення узгодженості в кластерах. Дослідження спрямовані на зменшення затримок, використання батчінгу та адаптивне керування реплікацією.

Ще одним важливим трендом є врахування особливостей апаратного забезпечення. Сучасні дослідження показують, що продуктивність СКБД значною мірою залежить від ефективного використання CPU кешів, NUMA-архітектур і навіть GPU. У роботах [17, 18] доведено, що апаратно-орієнтована оптимізація може забезпечити кратне зростання продуктивності, особливо для аналітичних навантажень. In-memory бази даних стали стандартом для високопродуктивних систем, мінімізуючи залежність від дискових операцій [19].

Таким чином, аналіз та систематизація методів оптимізації продуктивності СКБД є актуальним,

p-ISSN: 2225-6733; e-ISSN: 2519-271X

оскільки наявність великої кількості різнорідних підходів і технологій зумовлює необхідність їх систематизації та порівняльного аналізу.

Мета статті

Метою роботи є систематизація сучасних підходів та методів підвищення продуктивності СКБД, аналіз їх ефективності, а також пошук ключових напрямів подальших досліджень.

Виклад основного матеріалу

Важливим аспектом аналізу є розуміння природи робочого навантаження на базу даних (read-heavy, write-heavy, mixed), характеру запитів (OLTP, OLAP) та прогнозування майбутнього зростання. Це дозволяє вибрати найбільш відповідні стратегії оптимізації та масштабування бази даних.

Продуктивність систем управління базами даних формується під впливом сукупності апаратних, архітектурних та програмних факторів. Їх урахування є необхідною умовою для забезпечення ефективної обробки даних і стабільної роботи інформаційних систем. У табл. 2 приведені головні фактори, що впливають на продуктивність СКБД.

Архітектура бази даних є фундаментом її продуктивності. Правильне проектування, денормалізація, шардинг та вибір відповідної моделі даних можуть кардинально покращити ефективність роботи СКБД. Одним із основних принципів проектування баз даних для досягнення високої продуктивності є ретельне планування структури даних з урахуванням конкретних завдань та навантажень. Це включає визначення оптимальної кількості таблиць і їх взаємозв'язків, а також вибір відповідних типів даних для кожного стовпця (наприклад, INT замість BIGINT, якщо діапазон значень дозволяє; TIMESTAMP WITH TIME ZONE vs TIMESTAMP). Такий підхід дозволяє мінімізувати надмірність даних та прискорити виконання запитів.

Нормалізація даних є важливим етапом проектування, спрямованим на усунення надмірності, запобігання аномаліям даних та спрощення структури бази даних. Нормалізація включає поділ великих таблиць на менші та встановлення зв'язків між ними для підтримки цілісності даних. Хоча нормалізація сприяє підвищенню логічної узгодженості даних, у деяких випадках сильна нормалізація може призвести до збільшення складності запитів (більше JOIN операцій), що негативно впливає на продуктивність читання.

Денормалізація даних – це процес додавання надмірності до бази даних для прискорення читання даних за рахунок скорочення кількості необхідних операцій з'єднання таблиць. Денормалізацію слід застосовувати з обережністю, коли продуктивність читання критична, а операції запису менш інтенсивні. Дослідження показують, що виборча денормалізація може значно скоротити час виконання запитів у OLAP-системах та сховищах даних [20].

Підвищення продуктивності СКБД досягається шляхом застосування різноманітних методів, які охоплюють як логічний рівень організації даних, так і фізичні механізми їх обробки та зберігання. У табл. 3 наведено основні методи оптимізації продуктивності СКБД.

Шардинг (Sharding) та поділ даних (Partitioning) використовуються для горизонтального масштабування та розподілу великих обсягів даних. Поділ даних дозволяє розподілити дані за різними таблицями або базами даних на основі певних критеріїв, наприклад, за часом або географічною ознакою, що полегшує керування даними та підвищує продуктивність за рахунок паралельної обробки запитів.

Таблиця 2

Фактори, що впливають на продуктивність СУБД

Тип факторів	Складові	Вплив
Апаратні ресурси	CPU, RAM, дискова підсистема, мережа	Визначають базову швидкість обробки даних
Архітектура СУБД	Тип СУБД, розподіленість, модель даних	Впливає на масштабованість і ефективність
Структура даних	Схема БД, нормалізація/денормалізація	Зменшує надмірність і складність запитів
Обробка запитів	SQL-запити, плани виконання	Визначає час виконання операцій
Методи оптимізації	Індекси, кешування, реплікація, шардинг	Зменшують затримки та навантаження

Таблиця 3

Методи оптимізації продуктивності

Метод	Вміст	Переваги	Недоліки
Індексування	Створення структур для швидкого пошуку	Швидкий доступ	Уповільнення запису
Партиціонування	Розподіл даних на частини	Прискорення запитів	Складність управління
Шардинг	Розподіл по серверах	Масштабованість	Складність транзакцій
Кешування	Збереження часто використовуваних даних	Зменшення затримок	Потреба інвалідації
Реплікація	Копіювання даних	Висока доступність	Затримки синхронізації

Поділ може базуватися на діапазоні значень (Range Partitioning), хеш-функції (Hash Partitioning) або списках значень (List Partitioning). Це дозволяє оптимізувати запити, звертаючись лише до релевантних розділів (partition pruning), а також полегшує обслуговування (наприклад, видалення старих даних). PostgreSQL, Oracle, SQL Server надають потужні засоби для партиціонування.

Більш агресивний підхід полягає у розподілі даних між кількома серверами (шардами). Кожен шард містить підмножину даних, що дозволяє обробляти запити паралельно у різних вузлах, значно збільшуючи пропускну здатність і знижуючи навантаження на деякі машини.

Стратегії шардингу, або розподілу даних, включають кілька підходів, кожен з яких має свої особливості і області застосування.

Діапазонний шардинг (Range Sharding), при якому дані розподіляються по діапазонах ключів, дозволяє легко розділяти дані за певними інтервалами, що зручно для послідовних або часових даних. Однак, якщо один з діапазонів стає більш навантаженим, ніж інші, можуть виникати труднощі з балансуванням навантаження.

При хеш-шардінгу (Hash Sharding) розподіл даних заснований на хеш-функції від ключа. Цей спосіб забезпечує рівномірний розподіл даних по шардгам, що сприяє балансуванню навантаження та підвищенню масштабованості системи.

Метод, заснований на використанні директорії (Directory Based Sharding), передбачає наявність окремої таблиці або служби, яка зберігає інформацію про те, до якої кулі належить кожен запис і дозволяє швидко визначити розташування даних.

Незважаючи на переваги, шардинг стикається із низкою викликів. Насамперед ускладняються виконання запитів, оскільки для отримання повної інформації може знадобитися звернутися до кількох шардгів, що збільшує затримки та складність обробки. Також зростає складність управління системою, оскільки необхідно забезпечити правильне розподілення даних, балансування навантаження та моніторинг. Особливі труднощі виникають при реалізації розподілених транзакцій, оскільки необхідно зберігати цілісність даних та дотримуватись властивостей ACID (атомарність, узгодженість, ізоляція, довговічність). Для вирішення цих проблем сучасні системи використовують різні інструменти та рішення. Наприклад, Vitess для MySQL та Citus Data для PostgreSQL надають механізми для управління шардингом, автоматизації процесів та забезпечення надійності. В останні роки дослідники та розробники зосередилися на сучасних трендах, таких як автоматизація шардингу, динамічне управління шардовими сегментами та підвищення відмовостійкості систем, що дозволяє системам більш гнучко адаптуватися до змін навантаження та забезпечувати високий рівень доступності та надійності даних [21].

Ефективне індексування дозволяє СКБД швидше знаходити дані без необхідності сканувати всю таблицю, що значно скорочує час виконання запитів. Оптимізація SQL-запитів і використання планів виконання запитів також відіграють важливу роль у підвищенні ефективності роботи з даними. Стратегії індексування починаються з аналізу найчастіше використовуваних запитів та визначення ключових стовпців, за якими відбувається пошук. Створення індексів цих стовпців може значно прискорити виконання запитів.

Існує кілька основних типів індексів, кожен з яких призначений для певних завдань і має свої особливості. Одним з найбільш поширених є B-дерево, або збалансоване дерево, яке широко використовується в більшості систем управління базами даних завдяки своїй ефективності при пошуку діапазону, виконанні точних збігів і сортуванні даних. Такий індекс дозволяє швидко знаходити потрібні записи, підтримуючи баланс між висотою дерева та кількістю елементів, що забезпечує високу продуктивність під час роботи з великими обсягами даних. Наступним популярним типом є хеш-індекс, який підходить для швидкого пошуку точного збігу ключа, проте неефективний для пошуку по діапазону або сортування даних, оскільки не зберігає порядок елементів.

Для пошуку за текстовими даними призначений повнотекстовий індекс, який оптимізований для швидкого пошуку за вмістом текстових полів, що особливо важливо при роботі з великими масивами текстової інформації. У географічних системах або під час роботи з просторовими даними широко застосовуються просторові індекси, такі як R-дерево або GiST (Generalized Search Tree). Вони забезпечують ефективний пошук та обробку географічних об'єктів та просторових даних.

У PostgreSQL використовуються такі гнучкі типи індексів, як GiST та GIN (Generalized Inverted Index). GiST дозволяє індексувати складні типи даних, такі як масиви та JSONB, а також реалізовувати повнотекстовий пошук. GIN, у свою чергу, особливо ефективний для індексування структур даних, таких як масиви та повнотекстові пошукові системи, забезпечуючи швидкий доступ до окремих елементів усередині складних структур.

У дискових системах часто застосовують варіацію B-дерева, де всі дані зберігаються у листі, а внутрішні вузли служать для навігації. Такий підхід забезпечує швидкий доступ до даних та зручний для роботи з великими обсягами інформації, особливо за необхідності послідовного читання.

При виборі стовпців для індексування важливо враховувати, як часто вони беруть участь в умовах пошуку, з'єднаннях або сортуваннях, а також їх селективність, тобто унікальність значень в стовпці. Зазвичай рекомендується індексувати ті стовпці, які часто використовуються в умовах WHERE, JOIN ON або ORDER BY, особливо якщо вони мають високу селективність, тобто більшість значень у них унікальні або рідкісні.

Для підвищення ефективності запитів у випадках,

p-ISSN: 2225-6733; e-ISSN: 2519-271X

коли потрібна фільтрація кількома стовпцями, використовують складові (композитні) індекси. Порядок стовпців у такому індексі має велике значення: зазвичай на початок ставляться найбільш селективні стовпці, щоб максимально знизити обсяг даних, що перевіряються при виконанні запитів.

Крім того, існують так звані індекси, що покривають (Covering Indexes), які містять всі стовпці, необхідні для виконання конкретного запиту. У такому разі система управління базою даних може отримати всі необхідні дані прямо з індексу, уникаючи звернення до таблиці, що значно прискорює виконання запитів і знижує навантаження на систему.

Кожен індекс збільшує накладні витрати при операціях запису (INSERT, UPDATE, DELETE), оскільки індекс також має бути оновлено. Надмірне індексування або те, що не використовується, може уповільнити запис і відібрати дисковий простір.

Навіть із хорошими індексами, неефективно написаний запит може працювати погано. Аналіз та оптимізація SQL-запитів потребують глибокого розуміння того, як запити впливають на продуктивність. Використання EXPLAIN або аналогічних команд у СКБД дозволяє розробникам бачити план виконання запиту та розуміти, які операції, наприклад, сканування таблиці чи індексу, виконуються для обробки. На основі цього аналізу можна модифікувати запити, щоб скоротити кількість даних, використовувати індексне сканування замість повного сканування таблиць або оптимізувати з'єднання таблиць. Використання планів виконання запитів не тільки допомагає в аналізі поточних запитів, а й у прогнозуванні поведінки бази даних при змінах структури даних або обсягу даних. Це дозволяє заздалегідь адаптувати стратегії індексування та оптимізувати запити для підтримки високої продуктивності системи.

Матеріалізовані уявлення також важливі для підвищення ефективності аналітичних систем і потребують акуратного управління. Їх застосування особливо актуальне під час роботи з великими обсягами даних та складними запитами. Матеріалізовані уявлення – це об'єкти бази даних, які зберігають попередньо обчислені результати запитів для прискорення доступу та зниження навантаження. На відміну від звичайних уявлень, що обчислюються при кожному зверненні, вони зберігають дані фізично та потребують механізму оновлення. Основні переваги використання включають швидке виконання аналітичних запитів та підтримку звітності, а недоліки – необхідність оновлення та додаткових ресурсів для зберігання.

Коли оптимізація на рівні запитів та індексів досягає своєї межі, у гру вступають стратегії масштабування, кешування та реплікації.

Кешування даних полягає у зберіганні копій даних, що часто використовуються, у більш швидкодоступній пам'яті, що дозволяє знизити кількість звернень до основного сховища даних і зменшити час відгуку на запити. Механізми кешування працюють на різних рівнях архітектури програми, включаючи кешування на стороні клієнта, веб-сервера, програми та бази даних. Ефективність кешування залежить від правильного вибору даних для кешування, а також від алгоритмів керування кешем, таких як LRU (Least Recently Used) або FIFO (First In, First Out), які визначають які дані слід видалити з кешу при необхідності звільнити місце. Вплив кешування на продуктивність проявляється у зменшенні навантаження на базу даних та прискоренні доступу до даних. Кешування знижує кількість дискових операцій, необхідних для обробки запитів, зменшуючи час відгуку і збільшуючи пропускну здатність системи.

Стратегії кешування включають визначення оптимального розміру кешу, вибір відповідного розташування кешу (наприклад, на стороні клієнта чи сервера) та визначення життєвого циклу даних у кеші. Важливим аспектом є також інвалідність кешу, коли дані в основному сховище змінюються, щоб запобігти наданню застарілих даних користувачам. Рішення для кешування даних можуть бути реалізовані на рівні апаратного забезпечення, наприклад з використанням розподілених кеш-систем, таких як Memcached або Redis, які надають високопродуктивні механізми для зберігання тимчасових даних в оперативній пам'яті. Ці системи дозволяють легко масштабувати кеш, розподіляючи його на кількох вузлах.

Впровадження стратегій кешування потребує ретельного планування та тестування, оскільки неправильно налаштований кеш може призвести до зниження продуктивності через часті операції інвалідності або неефективне використання ресурсів. Однак при правильному підході кешування стає потужним інструментом для оптимізації продуктивності, що дозволяє забезпечити швидкий доступ до даних та підвищити загальну ефективність роботи системи.

Реплікація (Replication) є процес створення копій даних на різних серверах для забезпечення надійності та ефективності роботи інформаційних систем. Її основне призначення включає забезпечення високої доступності (High Availability), що досягається автоматичним перемиканням на резервну копію (failover) у разі збою основного сервера, а також підвищення читальної масштабованості (Read Scalability) за рахунок перенаправлення запитів на читання до основного серверу, що знижує навантаження. Крім того, реплікація служить для резервного копіювання (Backup), дозволяючи створювати резервні копії з реплік та мінімізувати вплив на основний сервер. Використання реплікації для аналітичної обробки (Analytical Processing) дозволяє обробляти ресурсоємні аналітичні запити на окремих репліках, не погіршуючи продуктивність транзакційної обробки.

Існує кілька типів реплікації, кожен із яких відрізняється особливостями роботи. Синхронна реплікація передбачає, що дані записуються спочатку на основний сервер, а підтвердження запису відбувається лише після успішного запису на всі (або задану кількість)

реплік, що гарантує високу консистентність, але збільшує затримку запису. Асинхронна реплікація передбачає негайне підтвердження запису на основний сервер, а реплікація даних здійснюється у фоновому режимі, що підвищує продуктивність запису, але допускає можливу невелику втрату даних у разі збою основного сервера до завершення реплікації. Напівсинхронна реплікація є компромісним варіантом: підтвердження запису відбувається після її завершення на основному сервері та отримання підтвердження хоча б від однієї репліки, що забезпечує баланс між швидкістю та консистентністю. Сучасні дослідження в галузі розподілених систем зосереджені на розробці більш просунутих протоколів консенсусу, таких як Raft та Paxos, які забезпечують сувору консистентність при реплікації. Також велика увага приділяється зниженню затримок реплікації у географічно розподілених системах, що особливо важливо для глобальних додатків та сервісів [22].

Масштабуванням (Scaling) називають процес збільшення можливостей системи для обробки більшого обсягу даних та запитів. Існує два основних підходи до масштабування: вертикальне масштабування (Scaling Up) та горизонтальне масштабування (Scaling Out). Вертикальне масштабування передбачає збільшення потужності існуючого сервера через додавання ресурсів, як-от CPU, RAM чи швидших дисків. Цей метод простіше в реалізації і не вимагає зміни архітектури системи, однак він має апаратні обмеження і може бути досить дорогим, оскільки можливості збільшення ресурсів обмежено і рано чи пізно вичерпаються. У свою чергу, горизонтальне масштабування полягає в розширенні системи за рахунок додавання більшої кількості серверів, що забезпечує високу пропускну здатність та стійкість до відмов. Такий підхід більш гнучкий і економічний, особливо при роботі з великими обсягами даних, проте вимагає більш складної архітектури системи, наприклад, реалізації шардування і балансування навантаження, щоб ефективно розподіляти запити між безліччю серверів і уникати вузьких місць.

Балансування навантаження є процесом розподілу вхідних запитів до програми або бази даних між декількома обробниками (наприклад, серверами або процесами), щоб жоден з них не був перевантажений. Вона особливо актуальна у системах з реплікованими базами даних (наприклад, read replicas) або шардованим кластерами. Основне завдання балансування – запобігти перевантаженню окремих серверів та забезпечити стабільну роботу системи. Існує кілька методів балансування навантаження.

Метод Round Robin дозволяє послідовно розподіляти між серверами запити по черзі незалежно від поточного навантаження. Цей спосіб простий у реалізації, але не враховує реальні показники завантаженості серверів.

Метод Least Connections спрямований на те, щоб нові запити надсилалися на сервер із найменшою кількістю активних з'єднань, що сприяє більш рівномірному розподілу навантаження.

Метод IP Hash використовує хеш-функцію від IP-адреси клієнта, що дозволяє забезпечити, щоб запити одного користувача завжди оброблялися одним сервером, що особливо важливо для збереження сеансів і даних користувача.

Географічне балансування розподіляє запити залежно від географічного положення користувача, що допомагає знизити затримки та підвищити швидкість відгуку.

Просунуті системи використовують динамічне балансування, яке в реальному часі аналізує поточне навантаження та параметри серверів, такі як час відгуку, завантаження процесора та доступна пам'ять, і на основі цих даних оптимально розподіляє вхідні запити, чим забезпечує максимальну ефективність роботи всієї системи.

Вибір моделі даних важливий етап при проектуванні системи зберігання інформації. Тип моделі даних істотно впливає на продуктивність системи, масштабованість і зручність управління. Існує два основних підходи до моделювання баз даних: реляційна модель (SQL) та нереляційна модель (NoSQL). Рішення на користь того чи іншого варіанта залежить від конкретних вимог проекту, особливостей даних та сценаріїв використання.

Реляційні системи управління базами даних, до яких належать такі популярні рішення, як PostgreSQL, MySQL, Oracle та SQL Server, характеризуються строгою структурою даних та підтримкою транзакційної цілісності, що базується на принципах ACID (атомарність, узгодженість, ізоляція, довговічність). Вони підходять для зберігання структурованих даних, де важливим є високий ступінь узгодженості та можливість виконання складних запитів з використанням операторів JOIN, агрегування та інших механізмів. У таких системах ключовими напрямами оптимізації є створення індексів для прискорення пошуку, проведення нормалізації чи денормалізації даних залежно від ситуації, і навіть партиціонування таблиць обробки великих обсягів інформації. Крім того, реляційні бази даних традиційно використовують вертикальне масштабування, тобто збільшення ресурсів одного сервера, що в певних випадках може обмежувати їх масштабованість.

Системи NoSQL охоплюють широкий спектр технологій, таких як MongoDB, Cassandra, Redis та Neo4j. Вони відрізняються гнучкістю схеми, що дозволяє динамічно змінювати структуру даних без необхідності складної міграції. Залежно від типу бази даних, їх переваги виявляються у різних сценаріях. Документоорієнтовані бази, наприклад MongoDB, добре працюють з неструктурованими або напівструктурованими даними, забезпечуючи швидкий доступ і масштабованість. Колонкові системи, такі як Cassandra, оптимізовані для обробки великих об'ємів даних та високої швидкості запису, поширюючи навантаження горизонтального масштабування. Ключ-значення бази, зокрема Redis, забезпечують надшвидкі операції читання

p-ISSN: 2225-6733; e-ISSN: 2519-271X

та запису в пам'яті, що актуально для кешування та обробки високонавантажених систем. Графові бази даних типу Neo4j спеціалізуються на моделюванні та аналізі складних взаємозв'язків між об'єктами. Оптимізація роботи з NoSQL системами залежить від їх типу. Для MongoDB важливим є правильний вибір структури документів та ефективне індексування, що дозволяє прискорити запити. У системах типу Cassandra необхідно грамотно спроектувати ключі шардування та кластеризації даних, щоб забезпечити рівномірний розподіл навантаження та мінімізувати затримки під час виконання запитів. Redis потребує оптимізації структури зберігання даних та налаштування пам'яті для підтримки високої швидкості операцій.

Основні відмінності в оптимізації продуктивності між NoSQL та реляційними СКБД полягають у наступному:

• Масштабованість. NoSQL бази даних краще підходять для горизонтального масштабування за рахунок шардингу та реплікації, що дозволяє обробляти великі обсяги даних та високі навантаження. Реляційні СКБД зазвичай орієнтовані на вертикальне масштабування, що може обмежувати їхню здатність до обробки дуже великих обсягів даних.

• Схема даних. Реляційні СКБД вимагають суворої схеми даних, що впливає на продуктивність при змінах структури даних. NoSQL бази забезпечують велику гнучкість в управлінні схемами, що дозволяє ефективніше адаптуватися до вимог додатків, що змінюються.

• Транзакції. Реляційні СКБД забезпечують сильну підтримку транзакцій з гарантіями ACID (атомарність, узгодженість, ізоляція, довговічність), що може впливати на продуктивність при високих навантаженнях. Багато NoSQL систем пропонують більш гнучкі моделі узгодженості (наприклад, BASE – Basically Available, Soft state, Eventual consistency), що дозволяє збільшити продуктивність за рахунок зменшення гарантій узгодженості.

• Оптимізація запитів. У реляційних СКБД існує складна оптимізація запитів з використанням планів виконання, індексів та статистики для покращення продуктивності. У NoSQL оптимізація часто залежить від специфіки моделі даних та типу бази даних (ключ-значення, документо-орієнтована, колонкова, графова), що потребує індивідуального підходу до кожної системи.

Вибір між NoSQL та реляційними СКБД для оптимізації продуктивності залежить від конкретних вимог додатку, обсягів даних, необхідності масштабування та специфіки робочих навантажень.

Сучасні програми часто використовують гібридні підходи, що поєднують у собі можливості реляційних і нереляційних баз даних. Такий підхід називається поліглотним зберіганням даних (polyglot persistence), він дозволяє застосовувати найбільш підходящу модель для кожної конкретної задачі, наприклад, зберігати критично важливі структуровані дані в реляційній базі, а неструктуровані або швидкозростаючі дані в NoSQL системах. Такий підхід потребує грамотної інтеграції та управління кількома системами, що у свою чергу висуває додаткові вимоги до архітектури та забезпечення узгодженості даних.

Застосування машинного навчання (ML) та штучного інтелекту (AI) в оптимізації систем управління базами даних стає все більш актуальним та перспективним напрямом розвитку. Ці технології дозволяють значно підвищити ефективність роботи СУБД за рахунок автоматизації складних процесів конфігурування, моніторингу та оптимізації, які раніше вимагали високого рівня експертизи та ручних зусиль. Зокрема, автоматичне індексування та налаштування параметрів системи за допомогою ML дозволяють системам самостійно аналізувати журнали запитів, виявляти шаблони використання та пропонувати або автоматично впроваджувати зміни у структурі індексів та конфігурації, що сприяє підвищенню швидкості виконання запитів та зниженню витрат ресурсів. Методи конфігурації, що навчаються на даних про продуктивність, допомагають адаптувати параметри системи до поточних умов роботи, забезпечуючи оптимальне функціонування без необхідності постійного ручного втручання.

Крім того, технології машинного навчання активно використовуються для прогнозування майбутнього навантаження та виявлення аномалій у роботі баз даних. ML-моделі здатні на основі історичних даних передбачати можливі вузькі місця та потенційні збої системи, що дозволяє заздалегідь вживати заходів щодо масштабування ресурсів або оптимізації системних налаштувань. Це значно скорочує час реакції на проблеми, що виникають, і підвищує стійкість IT-інфраструктури. Важливим напрямом є інтелектуальна оптимізація запитів. ML-рішення покращують традиційні методи планування виконання SQL-запитів, навчаючись на реальних даних про виконання операцій та передбачаючи їхню вартість з більшою точністю. Такі підходи дозволяють створювати ефективніші плани виконання, що особливо важливо для складних та розподілених систем, де оптимізація маршрутів передачі даних та розподіл обчислювальних завдань відіграють ключову роль.

Також широкого застосування знаходять системи інтелектуального управління ресурсами. Хмарні платформи використовують машинне навчання для автоматичного масштабування ресурсів, таких як CPU, пам'ять та дискова I/O, що дозволяє динамічно реагувати на зміни навантаження та забезпечувати необхідний рівень продуктивності. У розподілених та шардованих системах ML допомагає приймати рішення про найбільш раціональне розміщення даних, мінімізуючи затримки та підвищуючи пропускну спроможність системи.

Рис. 1 – Етапи оптимізації продуктивності СУБД

Оптимізація продуктивності СКБД є поетапним процесом, що охоплює аналіз, виявлення проблем, впровадження рішень і подальший контроль ефективності (дивись рис. 1).

Отже, підвищення продуктивності СКБД потребує комплексного підходу, що поєднує класичні методи оптимізації з новітніми інтелектуальними технологіями. Отримані результати можуть бути використані для вдосконалення інформаційних систем і підвищення ефективності управління даними в умовах зростаючих вимог до швидкодії та масштабованості.

Висновки

У роботі досліджено підходи до підвищення продуктивності систем управління базами даних в умовах зростання обсягів даних і навантаження. Показано, що ефективність СКБД визначається сукупністю факторів, зокрема архітектурою системи, якістю проєктування схеми даних, рівнем оптимізації запитів і використанням апаратних ресурсів.

Порівняння реляційних і нереляційних СКБД свідчить, що вибір моделі даних залежить від специфіки задач. Реляційні системи доцільно застосовувати там, де критичною є узгодженість даних, тоді як NoSQL рішення краще підходять для систем із високими вимогами до масштабованості та гнучкості. У зв'язку з цим обґрунтовано використання поліглотного підходу, який дозволяє комбінувати різні типи СКБД.

Встановлено, що одним із ключових напрямів розвитку є застосування методів машинного навчання для автоматизації оптимізації. Такі підходи дають змогу адаптивно налаштовувати параметри системи, прогнозувати навантаження та виявляти вузькі місця.

Отже, підвищення продуктивності СКБД потребує комплексного підходу, що поєднує традиційні методи оптимізації з сучасними інтелектуальними рішеннями. Отримані результати можуть бути використані при розробці та вдосконаленні інформаційних систем.

Перелік використаних джерел

[1] Căuniac D. A., Niculae A. M. Monitoring and optimizing the database performance. *Proceedings of 22nd International Conference on Informatics in Economy (IE 2023)*, Bucharest, Romania, 25–26 May 2023. Singapore: Springer Nature, 2023. Pp. 39–49. DOI: https://doi.org/10.1007/978-981-99-6529-8_4.

[2] Ravi Kumar Y. V., Samayam A. K., Kadambari P. PostgreSQL monitoring using PgAdmin and Grafana. *Mastering PostgreSQL Administration: Internals, Operations, Monitoring, and Oracle Migration Strategies*. Berkeley: Apress, 2025. Pp. 405–484. DOI: https://doi.org/10.1007/979-8-8688-1507-2_7.

[3] Lingala B. Predictive monitoring for distributed and relational database management systems (RDBMS): A comprehensive analysis. *International Journal of Research and Applied Innovations*. 2025. Vol. 8, № 6. Pp. 13070–13082. DOI: https://doi.org/10.52783/jisem.v10i63s.13857.

[4] Marcus R., Papaemmanouil O. Towards a hands-free query optimizer through deep learning. arXiv preprint. arXiv:1809.10212. 2018. DOI: https://doi.org/10.48550/arXiv.1809.10212.

[5] AI-augmented query optimization: A technical survey of foundations, systems, and open challenges / I. Khazrak et al. SSRN preprint. 2026. DOI: http://dx.doi.org/10.2139/ssrn.6190260.

[6] A review of query optimization techniques: From traditional to reinforcement learning approaches / Shajeena S., Shiny R. M., Gayathri Devi K., Mary

Vespa M. *2026 International Conference on Electronics and Renewable Systems (ICEARS)*, Tuticorin, India, 11–13 February 2026. 2026. Pp. 1185–1189. DOI: https://doi.org/10.1109/ICEARS67481.2026.11416629.

[7] Weng L., Liu D., Zhu W. та ін. Learned query optimizer in Alibaba MaxCompute: Challenges, analysis, and solutions. arXiv preprint. arXiv:2602.07336. 2026. DOI: https://doi.org/10.48550/arXiv.2602.07336.

[8] Sassi N., Jaziri W. Efficient AI-driven query optimization in large-scale databases: A reinforcement learning and graph-based approach. *Mathematics*. 2025. Vol. 13, № 11. Article 1700. DOI: https://doi.org/10.3390/math13111700.

[9] Gabriska D., Pribilová K., Střelec P. Artificial intelligence in modern database systems: Applications, tools, and challenges for autonomous data management. *2025 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Stary Smokovec, Slovakia, 13–14 November 2025. 2025. Pp. 191–196. DOI: https://doi.org/10.1109/ICETA67772.2025.11280054.

[10] Oloruntoba O. AI-Driven autonomous database management: Self-tuning, predictive query optimization, and intelligent indexing in enterprise it environments. *World Journal of Advanced Research and Reviews*. 2025. Vol. 25, № 2. Pp. 1558–1580. DOI: https://doi.org/10.30574/wjarr.2025.25.2.0534.

[11] Sirimalla A. Performance optimization in Oracle and SQL Server on AWS & Azure: A comprehensive framework for enterprise database management. *Journal of Computer Science and Technology Studies*. 2025. Vol. 7, № 9. Pp. 150–160. DOI: https://doi.org/10.32996/jcsts.2025.7.9.19.

[12] Amazon Aurora: Design considerations for high throughput cloud-native relational databases / Verbitski A. et al. *Proceedings of the 2017 ACM SIGMOD International Conference*, Chicago, USA, 14–19 May 2017. Pp. 1041–1052. DOI: https://doi.org/10.1145/3035918.3056101.

[13] Memory-aware query optimization / H. Dong et al. *2025 IEEE International Conference on Big Data*, Macau, China, 08–11 December 2025. Pp. 7321–7329. DOI: https://doi.org/10.1109/BigData66926.2025.11400969.

[14] HiEngine: How to architect a cloud-native memory-optimized database engine / Y. Ma et al. *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, Philadelphia, USA, 12–17 June 2022. Pp. 2177–2190. DOI: https://doi.org/10.1145/3514221.3526043.

[15] In-memory transaction processing: Efficiency and scalability considerations / H. Hu et al. *Knowledge and Information Systems*. 2019. Vol. 61. Pp. 1209–1240. DOI: https://doi.org/10.1007/s10115-019-01340-7.

[16] Li J., Michael E., Ports D. R. K. Eris: Coordination-free consistent transactions using in-network concurrency control. *Proceedings of the 26th Symposium on Operating Systems Principles*, Shanghai, China, 28 October 2017. Pp. 104–120. DOI: https://doi.org/10.1145/3132747.3132751.

[17] Hardware-conscious stream processing: A survey / S. Zhang et al. *SIGMOD Record*. 2020. Vol. 48, № 4. Pp. 18–29. DOI: https://doi.org/10.1145/3385658.3385662.

[18] Teubner J. Data processing on modern hardware. *SummerSchool '23: 4th ACM Europe Summer School on Science: Towards building the Data Science Stack*, Athens, Greece, 10–14 July 2023. DOI: https://doi.org/10.1145/3673199.3673204.

[19] Elastic use of far memory for in-memory database management systems / D. Lee et al. *Proceedings of the 19th International Workshop on Data Management on New Hardware*, Seattle, USA, 18–23 June 2023. Pp. 35–43. DOI: https://doi.org/10.1145/3592980.3595311.

[20] Talakh M. V., Dvorzhak V. V., Ushenko Y. O. Denormalization techniques for IoT data warehouses: Balancing query performance and data redundancy. *Оптико-електронні інформаційно-енергетичні технології*. 2025. Том 49, № 1. Pp. 72–81. DOI: https://doi.org/10.31649/1681-7893-2025-49-1-72-81.

[21] Resilient and dependability management in distributed environments: A systematic review / Amiri Z., Heidari A., Navimipour N. J., Unal M. *Cluster Computing*. 2023. Vol. 26. Pp. 1565–1600. DOI: https://doi.org/10.1007/s10586-022-03738-5.

[22] Georgakakos G., Verginadis Y. A novel data store supporting decentralized data management in the cloud computing continuum. *Advanced Information Networking and Applications (AINA 2025)*. Cham: Springer, 2025. DOI: https://doi.org/10.1007/978-3-031-87778-0_40.

APPROACHES AND METHODS TO OPTIMIZING THE PRODUCTIVITY OF DATABASE MANAGEMENT SYSTEMS IN MODERN INFORMATION SYSTEMS

Vorotnikova Z.J. *PhD (Engineering), associate professor, SHEI «Priazovskyi state technical university», Dnipro, ORCID: https://orcid.org/0000-0002-0746-5981, e-mail: vorotnikova_z_j@pstu.edu*

The article discusses methods for improving the performance of database management systems (DBMSs) in the face of growing data volumes and workloads typical of modern information systems. The effectiveness of a DBMS is directly related to the ability to quickly process queries, maintain low response times, and provide stable throughput. The performance of a DBMS is affected by many factors, including system architecture, hardware resources, data structure, access methods, and the efficiency of transaction and query processing. Key optimization approaches are analysed, such as database schema design, indexing, partitioning, sharding, as well as caching and replication. The impact of normalization and denormalisation on system performance is considered, and the choice depends on the characteristics of the workload and query types. Proper schema design helps reduce data processing overhead and query execution time. Special attention is paid to SQL query optimization and execution plan analysis as important tools for improving performance. The use of various types of indexes (B-trees, hash indexes, and full-text indexes) significantly speeds up data access, but introduces additional write overhead. Materialized representations are also considered as a way to speed up analytical queries. Scaling approaches, including vertical and horizontal scaling, as well as load balancing mechanisms, are analysed as means of efficient use of resources. Caching is highlighted as an effective method to reduce the load on the database and increase access speed, provided that appropriate update and invalidation strategies are used. Replication increases system availability and allows scaling of read operations, although reducing replication latency remains a challenge, especially in distributed environments. A comparison of relational and non-relational databases shows that the choice depends on the system requirements. Relational DBMSs provide high consistency, while NoSQL solutions offer better scalability and flexibility, supporting the use of a polyglot persistence approach. The application of machine learning methods for automated optimization is also being considered. Such systems can analyse workloads, identify bottlenecks, and automatically tune DBMS parameters, moving towards autonomous databases. The results can be used to improve the efficiency of information systems and data management under increasing workloads.
Keywords: *DBMS performance; query optimization; indexing; sharding; replication; caching; scaling; distributed systems; NoSQL; machine learning.*

References

[1] D. A. Căuniac, and A. M. Niculae, "Monitoring and optimizing the database performance," in *Proc. of the Int. Conf. Informatics in Economy*, Bucharest, Romania, May 25–26, 2023, pp. 39–49. **doi: 10.1007/978-981-99-6529-8_4.**

[2] Y. V. Ravi Kumar, A. K. Samayam, and P. Kadambari, "PostgreSQL monitoring using PgAdmin and Grafana," *Mastering PostgreSQL Administration: Internals, Operations, Monitoring, and Oracle Migration Strategies*, pp. 405–484, 2025. **doi: 10.1007/979-8-8688-1507-2_7**.

[3] B. Lingala, "Predictive monitoring for distributed and relational database management systems (RDBMS): A comprehensive analysis," *International Journal of Research and Applied Innovations*, vol. 8, no. 6, pp. 13070–13082, 2025. **doi: 10.52783/jisem.v10i63s.13857**.

[4] R. Marcus, and O. Papaemmanouil, "Towards a hands-free query optimizer through deep learning," 2018, *arXiv:1809.10212*. **doi: 10.48550/arXiv.1809.10212**.

[5] I. Khazrak, N. Mashhadi Nejad, M. Homaei, M. Rezaee, and R. Green, "AI-augmented query optimization: A technical survey of foundations, systems, and open challenges," 2026, SSRN preprint. **doi: 10.2139/ssrn.6190260**.

[6] S. Shajeena, R. M. Shiny, Gayathri K. Devi, and M. Mary Vespa, "A review of query optimization techniques: From traditional to reinforcement learning approaches," in *Proc. of the 2026 Int. Conf. on Electronics and Renewable Systems (ICEARS)*, Tuticorin, India, February 11–13, 2026, pp. 1185–1189. **doi: 10.1109/ICEARS67481.2026.11416629**.

[7] L. Weng et al., "Learned query optimizer in Alibaba MaxCompute: Challenges, analysis, and solutions," 2026, *arXiv:2602.07336*. **doi: 10.48550/arXiv.2602.07336**.

[8] N. Sassi, and W. Jaziri, "Efficient AI-driven query optimization in large-scale databases: A reinforcement learning and graph-based approach," *Mathematics*, vol. 13, no. 11, article 1700, 2025, **doi: 10.3390/math13111700**.

[9] D. Gabriska, K. Pribilová, and P. Střelec, "Artificial intelligence in modern database systems: Applications, tools, and challenges for autonomous data management," in *Proc. of the 2025 Int. Conf. on Emerging eLearning Technologies and Applications (ICETA)*, Stary Smokovec, Slovakia, November 13–14, 2025, pp. 191–196. **doi: 10.1109/ICETA67772.2025.11280054**.

[10] O. Oloruntoba, "AI-based autonomous database management: Self-tuning, predictive query optimization and intelligent indexing in enterprise IT environments," *World Journal of Advanced Research and Reviews*, vol. 25, no. 2, pp. 1558–1580, 2025. **doi: 10.30574/wjarr.2025.25.2.0534**.

[11] A. Sirimalla, "Performance optimization in Oracle and SQL Server on AWS & Azure: A comprehensive framework for enterprise database management," *Journal of Computer Science and Technology Studies*, vol. 7, no. 9, pp. 150–160, 2025. **doi: 10.32996/jcsts.2025.7.9.19**.

[12] A. Verbitski et al., "Amazon Aurora: Design considerations for high throughput cloud-native relational databases," in *Proc. of the 2017 ACM SIGMOD*

International Conf., Chicago, USA, May 14–19, 2017, pp. 1041–1052. **doi: 10.1145/3035918.3056101**.

[13] H. Dong, Z. Hu, C. Lu, S. Weng, Q. Ruan, and R. Zhang, "Memory-aware query optimization," in *2025 IEEE International Conference on Big Data*, Macau, China, December 08–11, 2025, pp. 7321–7329. **doi: 10.1109/BigData66926.2025.11400969**.

[14] Y. Ma, S. Xie, H. Zhong, L. Lee, and K. Lv, "HiEngine: How to architect a cloud-native memory-optimized database engine," in *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, Philadelphia, USA, June 12–17, 2022, pp. 2177–2190. **doi: 10.1145/3514221.3526043**.

[15] H. Hu, X. Zhou, T. Zhu, W. Qian, and A. Zhou, "In-memory transaction processing: Efficiency and scalability considerations," *Knowledge and Information Systems*, vol. 61, pp. 1209–1240, 2019. **doi: 10.1007/s10115-019-01340-7**.

[16] J. Li, E. Michael, and D. R. K. Ports, "Eris: Coordination-free consistent transactions using in-network concurrency control," in *Proc. of the 26th Symposium on Operating Systems Principles*, Shanghai, China, October 28, 2017, pp. 104–120. **doi: 10.1145/3132747.3132751**.

[17] S. Zhang, F. Zhang, Y. Wu, B. He, and P. Johns, "Hardware-conscious stream processing: A survey," *SIGMOD Record*, vol. 48, no. 4, pp. 18–29, 2020. **doi: 10.1145/3385658.3385662**.

[18] J. Teubner, "Data processing on modern hardware," in Proc. of the *SummerSchool '23: 4th ACM Europe Summer School on Science: Towards building the Data Science Stack*, Athens, Greece, July 10–14, 2023. **doi: 10.1145/3673199.3673204**.

[19] D. Lee et al., "Elastic use of far memory for in-memory database management systems," in *Proc. of the 19th Int. Workshop on Data Management on New Hardware*, Seattle, USA, June 18–23, 2023, pp. 35–43. **doi: 10.1145/3592980.3595311**.

[20] M. V. Talakh, V. V. Dvorzhak, and Y. O. Ushenko, "Denormalization techniques for IoT data warehouses: Balancing query performance and data redundancy," *Optyko-elektronni informatsiino-enerhetychni tekhnolohii – Optoelectronic information–power technologies*, no. 1, pp. 72–81, 2025. **doi: 10.31649/1681-7893-2025-49-1-72-81**.

[21] Z. Amiri, A. Heidari, N. J. Navimipour, and M. Unal, "Resilient and dependability management in distributed environments: A systematic review," *Cluster Computing*, vol. 26, pp. 1565–1600, 2023. **doi: 10.1007/s10586-022-03738-5**.

[22] G. Georgakakos, and Y. Verginadis, "A novel data store supporting decentralized data management in the cloud computing continuum," *Advanced Information Networking and Applications (AINA 2025)*, Cham: Springer, 2025. **doi: 10.1007/978-3-031-87778-0_40**.

Стаття надійшла 12.12.2025
Стаття прийнята 18.01.2026
Стаття опублікована 26.03.2026

Цитуйте цю статтю як: Воротнікова З. Є. Підходи та методи до оптимізації продуктивності систем управління базами даних у сучасних інформаційних системах. *Вісник Приазовського державного технічного університету. Серія: Технічні науки*. 2026. Вип. 53, том 1. С. 109–119. DOI: https://doi.org/10.31498/2225-6733.53.1.2026.359786.