https://doi.org/10.31891/2307-5732-2025-359-62
УДК 004.6

СОЛОГУБ ВОЛОДИМИР
Національний університет «Львівська політехніка»
https://orcid.org/0000-0003-1553-530X
e-mail: volodymyr.r.solohub@lpnu.ua
ПАШКЕВИЧ ВОЛОДИМИР
Національний університет «Львівська політехніка»
https://orcid.org/0000-0002-6849-652X
e-mail: volodymyr.z.pashkevych@lpnu.ua

КОМБІНОВАНИЙ МЕТОД ПАРТИЦІЮВАННЯ ТА ІНДЕКСУВАННЯ ДАНИХ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ OLTP/OLAP СИСТЕМ

Стаття присвячена вирішенню однієї з ключових проблем сучасних інформаційних систем — забезпеченню ефективної одночасної обробки транзакційних (OLTP) та аналітичних (OLAP) навантажень. Така потреба є критично важливою для бізнесу, що вимагає прийняття рішень на основі даних у реальному часі. Проведено глибокий аналіз обмежень традиційних підходів. З одного боку, системи, що покладаються виключно на індексування, демонструють високу продуктивність для точкових OLTP-запитів, але суттєво деградують при виконанні складних аналітичних запитів через значні накладні витрати на підтримку індексів та неефективне сканування даних. З іншого боку, стратегії, засновані лише на партиціюванні, хоч і прискорюють OLAP-запити шляхом відсікання зайвих блоків даних, часто є недостатньо гнучкими для швидкого доступу до окремих рядків, що є типовим для OLTP-операцій. Для подолання цих недоліків розроблено комбінований метод, що поєднує дворівневе партиціювання даних з адаптивною стратегією індексування. Перший рівень партиціювання виконується за часовим критерієм, що дозволяє логічно відокремити «гарячі» (актуальні) дані від «холодних» (історичних). Другий рівень, що застосовується в межах кожної часової партиції, базується на категоріальному критерії (наприклад, регіон, тип продукту), що додатково локалізує дані. Ключовою особливістю методу є адаптивне індексування: для активних, часто змінюваних партицій застосовуються легкі некластеризовані індекси, що мінімізують накладні витрати на операції запису (INSERT, UPDATE, DELETE). У той же час, для історичних партицій, які переважно використовуються для читання, індекси автоматично трансформуються у стовпцеві (column-store), що ідеально підходять для OLAP-запитів, оскільки забезпечують високий рівень компресії та мінімізують обсяг зчитування з диска. Для автоматизації життєвого циклу даних розроблено алгоритм, що динамічно управляє створенням нових партицій та зміною типів індексів. Алгоритм працює на основі заданих граничних значень, таких як обсяг даних у партиції та інтенсивність звернень, автоматично переводячи партиції зі стану «активних» у «історичні». Ефективність розробленого підходу була підтверджена експериментальним аналізом на синтетичній моделі даних, що імітувала діяльність великої торгової мережі. Результати показали, що комбінований метод дозволив досягти кількаразового прискорення виконання складних OLAP-запитів у порівнянні з системами, що використовують лише індекси або лише партиціювання, при цьому зберігши високу пропускну здатність для транзакційних операцій на рівні спеціалізованих OLTP-систем. Таким чином, розроблений метод забезпечує оптимальний баланс між швидкодією та масштабованістю, усуваючи конфлікт між двома типами навантажень, і може бути рекомендований для впровадження у сучасних високопродуктивних гібридних системах, де аналітика в реальному часі є бізнес-вимогою.

Ключові слова: *OLTP, OLAP, підвищення ефективності SQL-запитів, партиціювання даних, бази даних, індекси.*

SOLOHUB VOLODYMYR
PASHKEVYCH VOLODYMYR
Lviv Polytechnic National University

COMBINED DATA PARTITIONING AND INDEXING METHOD TO IMPROVE OLTP/OLAP SYSTEM EFFICIENCY

The article is devoted to solving one of the key problems of modern information systems - ensuring efficient simultaneous processing of transactional (OLTP) and analytical (OLAP) loads. Such a need is critically important for businesses that require decision-making based on real-time data. An in-depth analysis of the limitations of traditional approaches is conducted. On the one hand, systems that rely exclusively on indexing demonstrate high performance for point OLTP queries, but significantly degrade when executing complex analytical queries due to significant overhead for index maintenance and inefficient data scanning. On the other hand, strategies based only on partitioning, although they speed up OLAP queries by cutting off unnecessary data blocks, are often not flexible enough for fast access to individual rows, which is typical for OLTP operations. To overcome these shortcomings, a combined method has been developed that combines two-level data partitioning with an adaptive indexing strategy. The first level of partitioning is performed according to a time criterion, which allows logically separating "hot" (current) data from "cold" (historical) data. The second level, applied within each time partition, is based on a categorical criterion (e.g., region, product type), which additionally localizes data. The key feature of the method is adaptive indexing: for active, frequently changed partitions, lightweight non-clustered indexes are used, which minimize the overhead of write operations (INSERT, UPDATE, DELETE). At the same time, for historical partitions, which are mainly used for reading, indexes are automatically transformed into column (column-store), which is ideal for OLAP queries, as they provide a high level of compression and minimize the amount of reading from disk. To automate the data life cycle, an algorithm has been developed that dynamically manages the creation of new partitions and changes in index types. The algorithm works on the basis of specified threshold values, such as the volume of data in the partition and the intensity of accesses, automatically transferring partitions from the "active" to "historical" state. The effectiveness of the developed approach was confirmed by experimental analysis on a synthetic data model that simulated the activities of a large retail chain. The results showed that the combined method allowed to achieve several times the acceleration of complex OLAP queries compared to systems that use only indexes or only partitioning, while maintaining high throughput for transaction operations at the level of specialized OLTP systems. Thus, the developed method provides an optimal balance between speed and scalability, eliminating the conflict between the two types of loads, and can be recommended for implementation in modern high-performance hybrid systems, where real-time analytics is a business requirement.

Keywords: *OLTP, OLAP, improving the efficiency of SQL queries, data partitioning, databases, indexes.*

Стаття надійшла до редакції / Received 03.10.2025
Прийнята до друку / Accepted 15.11.2025

Вступ

Сучасні бізнес-процеси вимагають від інформаційних систем здатності не тільки швидко опрацьовувати потокові транзакції (OLTP)[1,2], а й надавати актуальні дані для прийняття рішень через складні аналітичні запити (OLAP)[3,4]. Традиційний підхід, що передбачає розділення цих навантажень на окремі системи (наприклад, OLTP-база для операцій та data warehouse для аналітики), стає все менш ефективним[5,6,7]. Він призводить до значних затримок у доступі до даних, додаткових накладних витрат на ETL-процеси та створює ризики, пов'язані з несумісністю даних. Зростаюча потреба в аналізі даних у режимі реального часу стимулює розвиток гібридних систем, які поєднують обидві можливості в межах однієї платформи[8,9]. У зв’язку з цим виникає потреба у комбінованому підході, що поєднує переваги партиціювання та індексування[10]. Метою цього дослідження є розробити метод, який дає змогу автоматично підтримувати оптимальну структуру даних у гібридних системах шляхом контролю розміру партицій та створення індексів на великих сегментах.

У статті також буде проведено порівняльний аналіз продуктивності трьох стратегій: лише партиціювання, лише індексування та комбінований метод, з акцентом на баланс між ефективністю OLTP-операцій та швидкістю OLAP-запитів та розроблено підхід реалізований на основі спеціальної збереженої процедури, яка динамічно створює індекси та виконує розділ партицій у разі перевищення встановлених порогів.

Аналіз останніх досліджень і публікацій

Існуючі підходи до підвищення ефективності продуктивності баз даних, як правило, націлені на вирішення лише однієї частини проблеми. Якщо фокус зміщується на виключно індексування, то для забезпечення високої швидкості аналітичних запитів створюють численні індекси[11,12]. Хоча це значно прискорює операції читання, воно чинить значний негативний вплив на продуктивність OLTP-операцій[13,14,15]. Кожна операція вставки, оновлення або видалення даних вимагає внесення відповідних змін до всіх існуючих індексів, що створює значне навантаження на систему, уповільнює обробку транзакцій та збільшує обсяг сховища[16,17]. З іншого боку, підхід, заснований виключно на партиціонуванні, ефективний для прискорення вставки нових даних шляхом додавання їх до останньої партиції[18,19]. Це також спрощує управління великими обсягами інформації, даючи змогу швидко архівувати або видаляти старі дані. Однак, такий підхід не забезпечує достатньої оптимізації для складних аналітичних запитів, які потребують сканування або агрегації даних з кількох партицій[20,21]. У цих сценаріях продуктивність залишається низькою, оскільки відсутність відповідних індексів змушує систему обробляти великі обсяги даних[22,23]. Метою даного дослідження є розробка та аналіз адаптивного методу управління базами даних, який динамічно поєднує партиціонування та індексування для оптимізації навантажень. Інтеграція цих двох механізмів дає змогу досягти синергії[24, 25], де швидкість вставки даних зберігається на високому рівні завдяки партиціонуванню, тоді як продуктивність аналітичних запитів значно зростає завдяки селективному застосуванню індексів. Центральним елементом розробленого підходу є автоматизована процедура, що ідентифікує "холодні" (старіші, нечасто оновлювані) партиції і створює на них індекси, зменшуючи при цьому накладні витрати на OLTP-операції, які відбуваються в основному на "гарячій" (новій) партиції. У статті буде представлено обґрунтування цього методу, а також змодельовано його вплив на загальну продуктивність системи.

Виклад основного матеріалу дослідження

Дослідження проводилось на моделі даних, яка імітує систему зберігання інформації щодо продажів. Обрана модель даних може бути використана як для зберігання трансакційних даних так і для проведення аналітичних операцій, що відповідає цілі дослідження. Нижче буде наведено ER-діаграма структури, що використовувалась (рисунок 1).

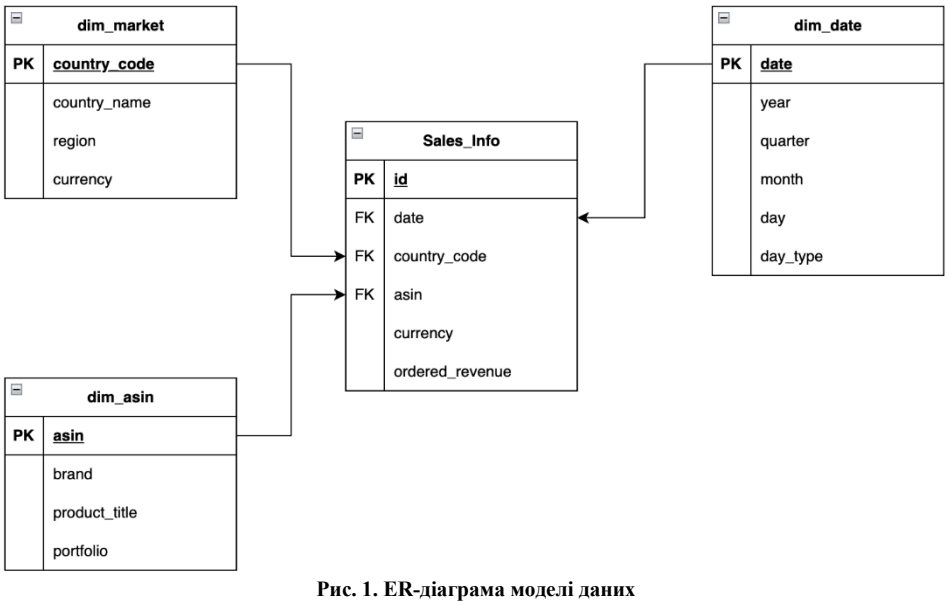

Рис. 1. ER-діаграма моделі даних

Представлена схема демонструє зіркоподібну модель даних для аналізу продажів. В її основі - центральна таблиця Sales_Info, яка фіксує інформацію про дохід та пов'язана з допоміжними таблицями вимірів (dim_market, dim_asin, dim_date). Ця структура даних спрощує багатовимірний аналіз продажів, даючи змогу розглядати дані в розрізі ринку, продукту та часу. Модель легко інтегрується з аналітичними системами, а її гнучкість дає змогу без проблем адаптувати схему для опрацювання нових даних.

Для розуміння переваг розробленого адаптивного методу, необхідно провести детальний аналіз традиційних підходів, що використовуються для підвищення ефективності баз даних. Тож розглянуто сильні та слабкі сторони стратегій, які базуються виключно на індексуванні або лише на партиціонуванні. Цей порівняльний аналіз продемонструє, чому кожен із цих підходів не є ідеальним для гібридних систем, і слугуватиме обґрунтуванням для розроблення комбінованого рішення.

Перевірку ефективності методів тестування проведено на двох типах запитів, один з яких напрямлений більше на аналітичні активності, другий - на транзакційні.

Проведемо дослідження з використанням підходу де використовується лише індексування. Для OLTP важлива швидкість точкового доступу та мінімальний вплив на транзакції. Тому переважно використовують кластеризовані індекси на первинних ключах та обмежену кількість некластеризованих індексів на часто запитуваних колонках. Для OLAP важлива швидка обробка великих обсягів даних та ефективне агрегування, тому часто використовують стовпцеві індекси. Для підтвердження цього, проведено експерименти для цих двох типів з вище описаними запитами на різних об'ємах даних.

Таблиця 1

Результати отримання даних при методах індексування

Тип індексу	**100 тис (розмір ~50 MB)**		**1 млн (розмір ~500 MB)**		**10 млн (розмір ~5 GB)**	
	OLAP (s)	OLTP (ms)	OLAP (s)	OLTP (ms)	OLAP (s)	OLTP (ms)
Некластеризований	2.1	12	15	45	310	480
Стовпцевий	0.8	20	3.5	70	32	720

Спостерігається пряма залежність часу виконання аналітичного та транзакційного запитів від обсягу даних. Для кількісної оцінки цієї залежності та визначення ефективності опрацювання даних введено коефіцієнт K, який відповідає середньому часу, що витрачається на обробку одного запису. Він розраховується за наступною формулою:

$$K = \frac{t}{N},$$

де, t – час виконання аналітичного запиту, N – кількість рядків, на яких проводиться експеримент

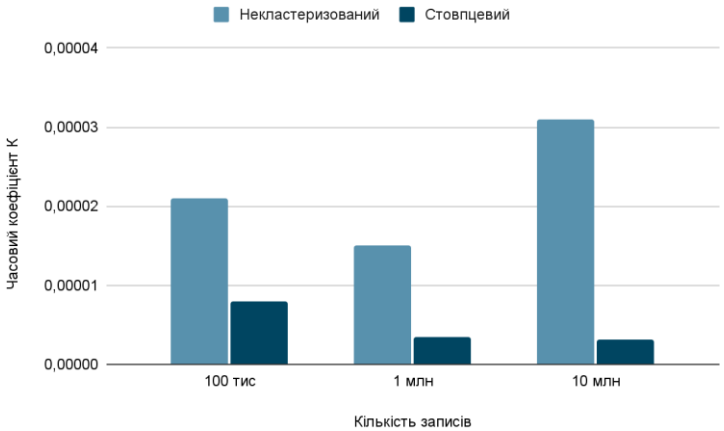

Рис. 2. Графік часу виконання OLAP запиту при різних стратегіях індексування

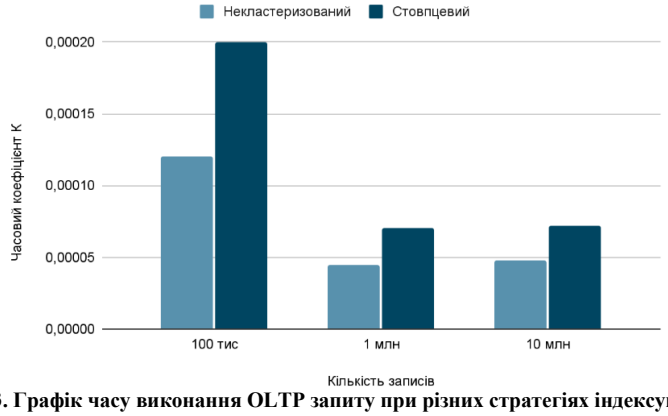

Рис. 3. Графік часу виконання OLTP запиту при різних стратегіях індексування

У ході експериментального дослідження ефективності різних підходів до індексування отримано результати, наведені в таблиці 1. Вимірювання проводилися для транзакційного та аналітичного запитів при різних обсягах даних, використовуючи некластеризовані та стовпцеві індекси. Отримані дані свідчать, що для транзакційних сценаріїв некластеризований індекс демонструє кращі результати, оскільки забезпечує прямий доступ до потрібних рядків без додаткового сканування. Натомість у випадку аналітичних запитів, орієнтованих на агрегування та вибірку великих масивів, спостерігається значна перевага стовпцевого індексу, що пояснюється його здатністю виконувати високоефективне стиснення та зчитування лише необхідних стовпців. Зі збільшенням обсягу даних різниця у швидкодії між двома підходами лише посилюється: накладні витрати стовпцевого індексу на точкові операції стають відчутними, тоді як його перевага в аналітичних сценаріях зростає майже на порядок. Таким чином, отримані результати підтверджують доцільність використання некластеризованих індексів у високонавантажених OLTP-системах, тоді як у випадку OLAP-запитів доцільно застосовувати стовпцеве індексування.

Далі, проведемо дослідження з використанням підходу, де використовується лише партиціювання. Як розглянуто раніше, жоден зі стандартних методів партиціювання не дає ту ж ефективність при універсальному застосуванні та не може повною мірою задовольнити одночасні вимоги OLAP і OLTP систем. Тому розроблено комбінований підхід, що поєднує переваги діапазонного та спискового методів.

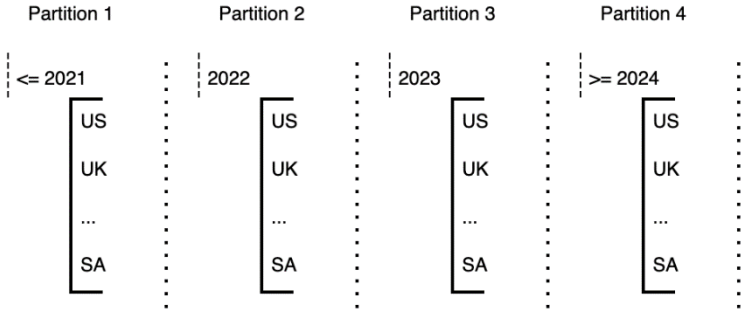

Рис 4. Розроблений комбінований метод партиціювання даних

Для обробки транзакційних навантажень використовується партиціювання за часовим атрибутом (date), тоді як для оптимізації аналітичних запитів додатково застосовується розділення за категоріальною ознакою (country_code). У результаті кожна діапазонна партиція поділяється на підмножини за категоріями, що забезпечує підвищену ефективність і прозору логіку доступу до даних. Для реалізації комбінованого методу доцільно створити службовий стовпець, у якому зберігатимуться значення, що поєднують часовий діапазон і категоріальну ознаку, що спрощує роботу СУБД із партиціями. Цей метод є комбінацією партиціювання за діапазоном (Range Partitioning) і партиціювання за списком (List Partitioning), що створює дворівневу ієрархічну структуру.

Нехай є таблиця *T*, яка є скінченною множиною записів (рядків). Кожен запис $r \in T$ має набір атрибутів, серед яких є часовий атрибут *d* (date) і категоріальний атрибут *c* (country_code).

- *D* — множина всіх можливих значень дати.
- $C=\{c_1,c_2,\ldots,c_k\}$ — скінченна множина всіх можливих значень для *country_code*

Загальна множина записів T розбивається на підмножини (партиції) S_{ij}, які не перетинаються і в сукупності дають вихідну таблицю:

$$T = \bigcup_{i=1}^{n} \bigcup_{j=1}^{k} S_{ij}$$

$$\forall (i,j) \neq (p,q) \qquad S_{ij} \cap S_{pg}$$

де S_{ij} — це кінцева партиція (субпартиція).

На першому рівні ми ділимо всю множину *T* на *n* основних партицій $P_1,P_2,\ldots,P_n$ за часовим атрибутом *d*.

Визначаємо *n* часових діапазонів, що не перетинаються:

$$\Delta t_1 = [t_0, t_1), \Delta t_2 = [t_1, t_2), \ldots, \Delta t_n = [t_n - 1, t_n)$$

Тоді кожна основна партиція P_i визначається як множина записів, часова мітка яких *r.d* потрапляє у відповідний діапазон Δt_i:

$$P_i = \{r \in T \mid r.d \in \Delta t_i\}$$

Це гарантує, що:

$$T = \cup_{i=1}^{n} P_i,\ P_i \cap P_j = \emptyset \text{ для } i = j$$

На другому рівні кожна основна партиція P_i далі ділиться на *k* субпартицій за значенням атрибута country_code *(r.c)*.

Для кожної партиції P_i створюються субпартиції $S_{i1},S_{i2},\ldots,S_{ik}$, де кожна субпартиція S_{ij} містить записи з P_i, у яких значення *country_code* дорівнює c_j:

$$S_{ij} = \{r \in P_i \mid r.c = c_j\}$$

Поєднуючи обидва рівні, ми можемо дати повне визначення для будь-якої кінцевої субпартиції S_{ij}:

$$S_{ij} = \{r \in T \mid (r.d \in \Delta t_i) \wedge (r.c = c_j)\}$$

Створення службового стовпця – це спосіб спростити реалізацію дворівневої логіки для СУБД, звівши її до однорівневого партиціювання.

Вводиться нова функція (або згенерований стовпець) *H(r)*, яка відображає комбінацію часового діапазону та категорії в одне унікальне значення.

Нехай є функція *f(d)*, яка повертає індекс часового діапазону:

$$f(r.d) = i \iff r.d \in \Delta t_i$$

Тоді функцію для службового стовпця можна визначити як конкатенацію або іншу комбінацію:

$$H(r) = concat(f(r.d), r.c)$$

Наприклад, якщо запис має дату *2024-10-15* (що потрапляє в діапазон *i=4*, який відповідає 2024 року) і *country_code = 'UA'*, то значення у службовому стовпці може бути *'4_10_UA'* або *'202410_UA'*.

Тепер СУБД може виконувати просте партиціювання за списком на основі унікальних значень, згенерованих функцією *H(r)*. Кожна партиція буде точно відповідати субпартиції S_{ij} з дворівневої моделі.

Переваги можна описати через скорочення простору пошуку:

- Транзакційний запит: *WHERE date BETWEEN t_start AND t_end*

Система визначає множину індексів:

$$I = \{i \mid \Delta t_i \cap [t_{start}, t_{end}] = \emptyset\}$$

Пошук виконується лише в об'єднанні партицій $U_{i \in I} P_i$ а не по всій таблиці *T*.

- Аналітичний запит: *WHERE country_code = c_j*

Замість сканування всієї таблиці T, система сканує лише відповідні субпартиції в кожному часовому діапазоні:

$$\cup_{i=1}^{n} \quad S_{ij}$$

Це значно ефективніше, оскільки система пропускає всі субпартиції S_{ij} , де $i \neq j$

- Комбінований запит: *WHERE date BETWEEN t_start AND t_end AND country_code = c_j*

Це найефективніший випадок. Система обчислює множину індексів *I* (як у першому випадку) і виконує пошук лише в цільових субпартиціях: $U_{i \in I} S_{ij}$. Обсяг даних для сканування мінімальний.

Для валідації та оцінки ефективності розробленого методу партиціювання проведено порівняльний аналіз з двома поширеними базовий підходами, що традиційно застосовуються для систем обробки транзакцій та аналітичної обробки.

Для симуляції навантаження, характерного для OLTP-систем, як базовий метод порівняння обрано циклічне секціонування (Round-Robin Partitioning). Цей вибір обгрунтований його здатністю забезпечувати максимально рівномірний розподіл даних між усіма секціями. Для OLAP-систем, де типовими є складні аналітичні запити до великих обсягів історичних даних, як базовий метод обрано секціонування за діапазоном (Range Partitioning).

Таблиця 2

Результати отримання даних при комбінованому методі партиціювання

Тип індексу	**100 тис (розмір ~50 MB)**		**1 млн (розмір ~500 MB)**		**10 млн (розмір ~5 GB)**	
	OLAP (s)	OLTP (ms)	OLAP (s)	OLTP (ms)	OLAP (s)	OLTP (ms)
Комбінований метод	1.8	15	8.7	55	65	510
Базовий метод	2.1	10	8.9	35	75	350

Розроблена комбінована стратегія забезпечила найвищу адаптивність до різних типів навантажень, зберігаючи баланс між продуктивністю аналітичних запитів та ефективністю транзакційних операцій. При виконанні аналіничних запитів стратегія демонструє стабільно нижчий час виконання, при цьому не втрачаючи у ефективності при виконанні транзакційних операцій. Таким чином, найкращим підходом для уніфікованих моделей, що повинні працювати як з OLTP-, так і з OLAP-навантаженням, є використання комбінованих стратегій партиціювання, адаптованих до характеру даних. Це дає змогу створити масштабовану та гнучку архітектуру, придатну до перевикористання для широкого спектра задач.

Розроблений метод підвищення ефективності поєднує підхід динамічного партиціювання з адаптивним застосуванням індексів залежно від характеру навантаження на різні частини даних. Основна ідея полягає в ідентифікації так званих "холодних" партицій - тобто таких, що відповідають старішим діапазонам даних (наприклад, попереднім датам і країнам), які рідко зазнають змін і використовуються переважно для аналітичних вибірок. Для цих партицій автоматично створюється стовпцевий індекс, що забезпечує суттєве прискорення агрегувальних і багатовимірних OLAP-запитів, водночас не створюючи накладних витрат на часті транзакційні операції.

Натомість "гарячі" партиції, що містять найновіші дані та активно використовуються для OLTP-операцій (вставок, оновлень, видалень), залишаються без стовпцевих індексів, аби уникнути надмірних витрат

на їхнє постійне оновлення. Для підвищення ефективності точкових запитів на цих партиціях накладаються лише легковагові некластеризовані індекси, які оптимізують швидкий доступ до актуальних рядків. Додатково метод передбачає механізм контролю розміру гарячих партицій: при досягненні заздалегідь визначеного граничного значення (наприклад, 2 млн рядків) виконується їх автоматичне розбиття на дрібніші частини. Це дає змогу уникати концентрації надмірного навантаження на одну партицію та підтримувати баланс між продуктивністю транзакцій і аналітичних обчислень. Таким чином, аналітичні OLAP-запити працюють із високою швидкістю завдяки стовпцевим індексам на "холодних" даних, тоді як OLTP-операції залишаються максимально ефективними завдяки некластеризованим індексам на "гарячих" партиціях і контрольованому механізму їх динамічного розподілу. Це дає змогу мінімізувати компроміс між двома типами навантажень та робить метод доцільним для використання в гібридних системах з одночасними OLTP та OLAP-запитами. Нижче зобразимо алгоритм роботи методу

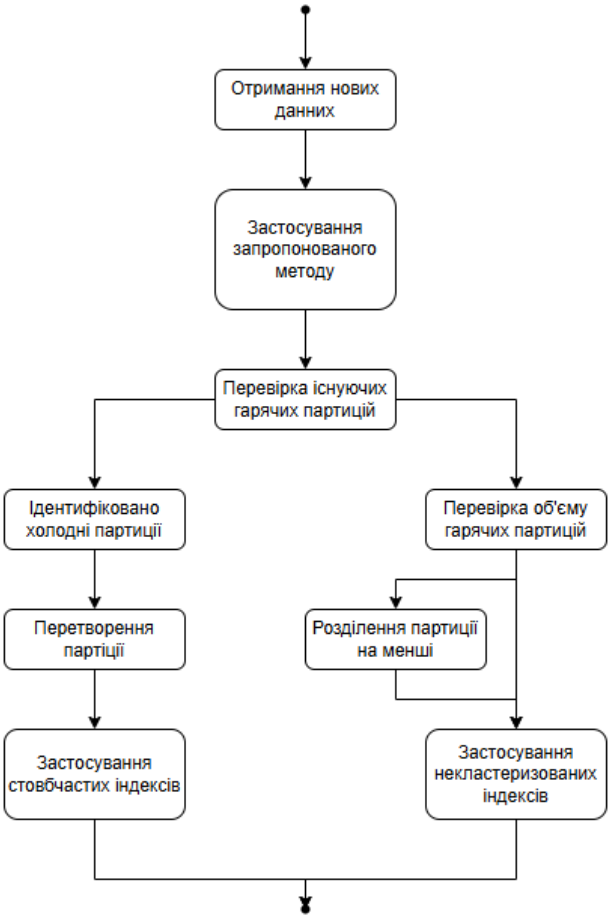

Рис. 5 Послідовність роботи розробленого методу

Цей метод є надбудовою над системою партиціювання і формалізує керування індексами та розміром партицій на основі частоти доступу та модифікації даних.

Будемо виходити з того, що таблиця T вже розділена на основні партиції $P_1,P_2,\ldots,P_n$ за часовим критерієм, де кожна партиція P_i відповідає часовому інтервалу $\Delta t_i=[t_i-1,t_i)$.

- $P_i=\{r \in T \mid r.d \in \Delta t_i\}$ — множина записів у і-й партиції.
- $|P_i|$ — кардинальність (кількість записів) партиції P_i.
- I_{col} — стовпцевий (columnstore) індекс.
- I_{ncl} — множина "легковагових" некластеризованих індексів.

Вводиться часова межа t_{hot}, яка динамічно визначає актуальність даних. Ця межа може бути фіксованою або передаватись через параметр. Визначається функція класифікації температури партиції $Temp(P_i)$:

$$Temp(P_i) = \{\text{'hot' if } t_i > t_{hot}\text{; 'cold' if } t_i \le t_{hot}\}$$

де t_i — це верхня межа часового діапазону Δt_i для партиції P_i

Таким чином, уся множина партицій $\{P_i\}$ ділиться на дві підмножини, що не перетинаються:

- Множина "гарячих" партицій: $P_{hot}=\{P_i \mid Temp(P_i)=\text{'hot'}\}$;
- Множина "холодних" партицій: $P_{cold}=\{P_i \mid Temp(P_i)=cold\}$;

Політика індексації $I(P_i)$ визначає набір індексів, що застосовуються до партиції P_i, і залежить від частоти доступу та модифікації даних.

$$I(P_i) = \{I_{col} \text{ if } P_i \in P_{cold}\text{; } I_{ncl} \text{ if } P_i \in P_{hot}\}$$

Мета цієї політики - мінімізувати сукупну вартість операцій. Нехай $C_{write}(I)$ - вартість операції запису (INSERT, UPDATE, DELETE) за наявності індексу I, а $C_{read}(I)$ - вартість операції читання.

- Для "гарячих" партицій, де операції запису часті, вибір I_{ncl} мінімізує C_{write}, оскільки $C_{write}(I_{ncl}) < C_{write}(I_{col})$.
- Для "холодних" партицій, де домінують аналітичні читання, вибір I_{col} мінімізує C_{read}, оскільки для агрегатних запитів $C_{read}(I_{col}) < C_{read}(I_{ncl})$.

Для "гарячих" партицій вводиться граничний розмір N_{max} (параметр @HotPartitionRowCountThreshold). Цей механізм є умовою, що запускає процес ре-партиціювання. Алгоритм можна описати так:

Для кожної $P_i \in P_{hot}$:

Якщо $|P_i| > N_{max}$

Виконати $Split(P_i)$

Операція $Split(P_i)$ для партиції P_i, що відповідає інтервалу $[t_i - 1, t_i)$, виконує наступні дії:

1. Знаходить точку розбиття t_{mid}, середину інтервалу: $t_{i-1} + \frac{t_i - t_{i-1}}{2}$
2. Замінює одну партицію P_i на дві нові:

- P_i' для нового інтервалу $\Delta t_i' = [t_{i-1}, t_{mid})$.
- P_i'' для нового інтервалу $\Delta t_i'' = [t_{mid}, t_i)$.

3. Перерозподіляє записи з P_i у P_i' та P_i'':

- $P_i' = \{r \in P_i \mid r.d < t_{mid}\}$
- $P_i'' = \{r \in P_i \mid r.d \geq t_{mid}\}$

Цей процес гарантує, що розмір найактивніших партицій залишається керованим, що запобігає деградації продуктивності OLTP-операцій.

Ефективність системи зростає завдяки тому, що розроблений метод стратегічно зменшує обсяг "роботи", яку база даних виконує для кожного типу запитів, застосовуючи доречний інструмент до відповідного завдання.

Загальну роботу системи (W_{total}) можна представити як суму робіт, виконаних над усіма партиціями:

$$W_{total} = \sum_{i=1}^{n} W(P_i)$$

де $W(P_i)$ — це сукупне навантаження на партицію P_i.

Це навантаження складається з двох компонентів:

$$W(P_i) = |\text{робота з транзакціями}| + |\text{робота з аналітикою}|$$

$$W(P_i) = \lambda_{OLTP}^{(i)} \cdot C_{OLTP}(P_i) + \lambda_{OLAP}^{(i)} \cdot C_{OLAP}(P_i)$$

$\lambda_{OLTP}^{(i)}$ та $\lambda_{OLAP}^{(i)}$ - це частота транзакційних та аналітичних запитів до партиції P_i

$C_{OLTP}(P_i)$ та $C_{OLAP}(P_i)$ - це необхідні ресурси виконання одного запиту

Як саме метод підвищує ефективність:

1. Для “гарячих” партицій $P_i \in P_{hot}$:

- Частота транзакцій $\lambda_{OLTP}^{(i)}$ висока;
- Метод застосовує легкі індекси (I_{ncl}), що робить вартість транзакції $C_{OLTP}(P_i)$ нижчою.
- Добуток $\lambda_{OLTP}^{(i)} \cdot C_{OLTP}(P_i)$ стає нижчим ⇒ система не "гальмує" на частих операціях

2. Для “холодних” партицій $P_i \in P_{cold}$:

- Частота аналітичних запитів $\lambda_{OLAP}^{(i)}$ висока
- Метод застосовує стовпцеві індекси (I_{col}), що робить вартість аналітики $C_{OLAP}(P_i)$ нижчою
- Добуток $\lambda_{OLAP}^{(i)} \cdot C_{OLAP}(P_i)$ стає нижчим ⇒ звіти та аналітика працюють значно швидше.

Таким чином, замість пошуку одного компромісного рішення для всіх даних, цей підхід динамічно знижує вартість саме тих операцій, які є найчастішими для конкретного блоку даних. Це веде до суттєвого зменшення сумарного навантаження W_{total} і, як наслідок, до підвищення загальної продуктивності та ефективності системи.

Для реалізації розробленого методу необхідно дотриматися низки пререквізитів. Передусім у системі має бути визначена функція партиціювання та відповідна схема (наприклад, pf_SalesByOrderDate та ps_SalesByOrderDate), що задають логіку розподілу даних між секціями. Цільова таблиця повинна бути створена безпосередньо на основі цієї схеми, що гарантує коректне фізичне розташування рядків у межах визначених партицій. Обов’язковою умовою є наявність у таблиці стовпця типу date/datetime, який використовується як ключ для розподілу записів. Крім того, передбачається, що схема партиціювання використовує механізм NEXT USED filegroup для автоматичного призначення нових секцій у процесі динамічного розширення структури даних. Це забезпечує масштабованість та стабільність роботи системи при збільшенні обсягів інформації.

У межах дослідження розроблено та імплементовано метод підвищення ефективності баз даних, що поєднує динамічне партиціонування та адаптивну індексацію для ефективного управління змішаними OLTP/OLAP-навантаженнями. Для реалізації цього підходу була створена процедура *dbo.usp_ManageHybridPartitions*, яка оперує набором ключових параметрів:

- @SchemaName;

- @TableName;
- @PartitioningColumn;
- @HotPartitionRowCountThreshold;
- @ColdPartitionAgeInDays;
- @HotPartitionNCIndexColumns;

Нижче буде наведено приклад виклику даної процедури:

```
EXEC dbo.usp_ManageHybridPartitions
    @SchemaName = 'dbo',
    @TableName = 'Sales_Info',
    @PartitioningColumn = 'date',
    @HotPartitionRowCountThreshold = 2000000,
    @ColdPartitionAgeInDays = 90,
    @HotPartitionNCIndexColumns = 'Country_Code,asin'
```

Рис. 6 Приклад виклику даної процедури

Параметр *@SchemaName* та *@TableName* ідентифікують цільову таблицю, в даному випадку dbo.FactSales. Параметр *@PartitioningColumn* вказує на стовпець OrderDate, що є основою для розподілу даних за часовим критерієм. Крім того, були визначені два ключові пороги: *@HotPartitionRowCountThreshold* зі значенням 2000000 та *@ColdPartitionAgeInDays* зі значенням 90. Ці параметри дають змогу системі автоматично ідентифікувати «гарячі» партиції, що містять актуальні дані з високою транзакційною активністю, та «холодні» партиції, що містять історичні дані, які переважно використовуються для аналітичних запитів. На гарячих партиціях для прискорення точкових запитів застосовується легковаговий некластеризований індекс, який створюється на основі стовпців, вказаних у параметрі *@HotPartitionNCIndexColumns(Country_Code, asin)*. Такий підхід мінімізує накладні витрати, пов'язані з оновленням індексів під час інтенсивних операцій запису, та забезпечує високу продуктивність для обох типів навантажень. Приклад результату роботи даної процедури:

```
Messages
Processing table: [dbo].[Sales_Info]
Partition Function: pf_SalesByDate
Partition Scheme: ps_SalesByDate
Found 25 partitions to analyze.
Partition 2 identified as COLD. Applying Columnstore Index.
... Dropping existing indexes on partition 2
... Rebuilding table with Clustered Columnstore Index. NOTE: This will affect all partitions.
... [SIMULATED] CREATE CLUSTERED COLUMNSTORE INDEX [CCI_Sales_Info_Partition_2] ON [dbo].[Sales_Info] ON [ps_SalesByDate](date);
... NOTE: True per-partition index type requires partition switching to a staging table with the desired index structure.
Partition 3 identified as COLD. Applying Columnstore Index.
... Dropping existing indexes on partition 3
... Rebuilding table with Clustered Columnstore Index. NOTE: This will affect all partitions.
... [SIMULATED] CREATE CLUSTERED COLUMNSTORE INDEX [CCI_Sales_Info_Partition_3] ON [dbo].[Sales_Info] ON [ps_SalesByDate](date);
... NOTE: True per-partition index type requires partition switching to a staging table with the desired index structure.
Partition 4 identified as COLD. Applying Columnstore Index.
... Dropping existing indexes on partition 4
... Rebuilding table with Clustered Columnstore Index. NOTE: This will affect all partitions.
... [SIMULATED] CREATE CLUSTERED COLUMNSTORE INDEX [CCI_Sales_Info_Partition_4] ON [dbo].[Sales_Info] ON [ps_SalesByDate](date);
... NOTE: True per-partition index type requires partition switching to a staging table with the desired index structure.
```

Рис. 7 Приклад результату роботи процедури

Розглянемо вплив застосування розробленого методу на продуктивність виконання аналітичних (OLAP) та транзакційних (OLTP) запитів. Основною метою використання даного підходу є збереження переваг партиціювання та індексування, характерних для OLTP-систем, без втрати ефективності застосування стовпцевих індексів у контексті OLAP-навантажень. Для порівняння візьмемо результати при застосуванні тільки індексування, тільки партиціювання та результат використання розробленого комбінованого методу

Проведемо експерименти для OLTP запиту та зобразимо результати у таблиці нижче

Таблиця 3

Результати отримання даних виконання OLTP запиту при різних сценаріях

Тип запиту	**100 тис (розмір ~50 MB)**	**1 млн (розмір ~500 MB)**	**10 млн (розмір ~5 GB)**
Індексування	12	45	480
Партиціювання	15	55	510
Комбінований метод	10	30	150

Проведений аналіз демонструє, що підходи, засновані як на виключно індексуванні, так і на партиціонуванні, забезпечують високу швидкість виконання OLTP-запитів. При використанні тільки індексування стандартний некластеризований індекс на ключових полях забезпечує ефективний точковий пошук, продуктивність якого не залежить від загального розміру таблиці. Незначне зниження швидкодії може спостерігатися лише при дуже великих обсягах даних через збільшення глибини дерева індексу. При використанні комбінованого продуктивність OLTP-операцій залишається відмінною, оскільки нові дані завжди вставляються в активну партицію, що мінімізує накладні витрати на обслуговування індексів. Це підтверджує, що для транзакційних систем ключовим фактором є швидка вставка та доступ до найновіших даних, що ефективно забезпечується партиціонуванням.

Розрахуємо часовий коефіцієнт К для кожного з розглянутих сценаріїв та зобразимо на графіку нижче.

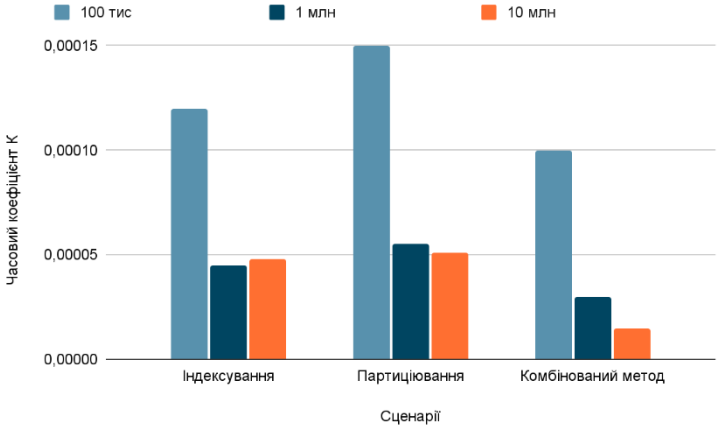

Рис. 8 Порівняння результатів виконання OLTP запиту при різних сценаріях

При використанні тільки індексування часовий коефіцієнт значно знижується зі зростанням обсягу даних від 100 тис. до 1 млн рядків, що свідчить про більш ефективний пошук на великих таблицях, де індекс стає більш повним. Однак, при 10 млн рядків коефіцієнт зростає, що вказує на зростання накладних витрат. При використанні тільки партиціонування коефіцієнт при 100 тис. рядків є високим, оскільки відсутність індексу ускладнює пошук, але при 1 млн і 10 млн рядків ефективність значно зростає завдяки підвищенню ефективності запитів для роботи лише з конкретною партицією. Комбінований метод демонструє найкращі результати, особливо на великих масштабах. Часовий коефіцієнт значно знижується від 100 тис. до 10 млн рядків, що підкреслює переваги гібридного підходу, який забезпечує високу швидкість доступу до даних, незалежно від їхнього обсягу, і є найбільш масштабним для транзакційних робочих навантажень.

Проведемо експерименти для OLAP запиту та зобразимо результати у таблиці нижче (значення у с)

Таблиця 4

Результати отримання даних виконання OLAP запиту при різних сценаріях

Тип запиту	**100 тис (розмір ~50 MB)**	**1 млн (розмір ~500 MB)**	**10 млн (розмір ~5 GB)**
Індексування	0.8	3.5	32
Партиціювання	1.8	8.7	65
Комбінований метод	0.3	3.2	10

Розрахуємо часовий коефіцієнт К для кожного з розглянутих сценаріїв та зобразимо на графіку нижче.

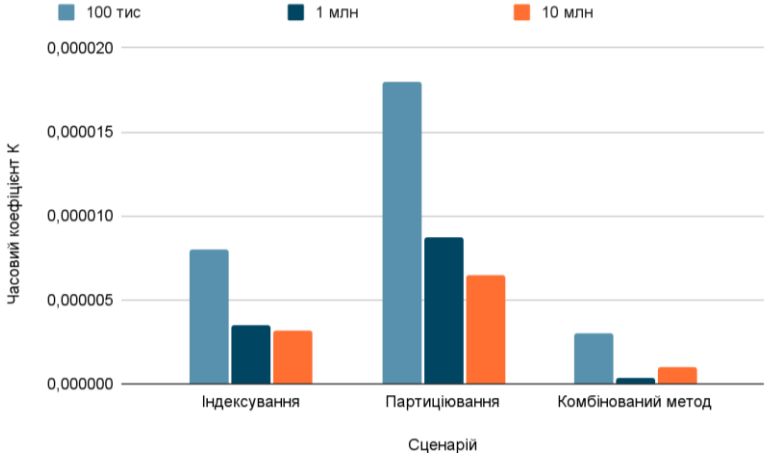

Рис. 9 Порівняння результатів виконання OLAP запиту при різних сценаріях

Результати тестування OLAP-запитів демонструють значні відмінності між підходами. При використанні тільки індексування час виконання запитів лінійно зростає зі збільшенням обсягу даних, що призводить до значних затримок на великих таблицях. Це обумовлено тим, що для виконання агрегації запит змушений сканувати великий діапазон row-store індексу. Натомість, при використанні тільки партиціонування виключення партицій значно скорочує обсяг даних для сканування, що забезпечує значне зростання продуктивності. Однак, найбільшу ефективність демонструє сценарій комбінований метод. Завдяки динамічному створенню columnstore індексів на "холодних" партиціях, аналітичні запити виконуються в пакетному режимі з екстремальним стисненням даних, що робить їх на порядки швидшими за інші підходи. З

графіку видно, що при 1 млн рядків часовий коефіцієнт нижчий, ніж при 10 млн. Це пояснюється тим, що при 1 млн рядків таблиця, містить меншу кількість "холодних" партицій, і Columnstore індекси, які забезпечують максимальну ефективність, ще не були створені на всіх релевантних частинах даних. Натомість, коли таблиця досягає 10 млн рядків, більша частина даних стає "холодною" і покривається індексами Columnstore, що максимально прискорює аналітику. Цей метод дає змогу отримати результати за декілька секунд навіть при роботі з великими даними, підтверджуючи його перевагу для гібридних систем.

Висновок

У роботі розроблено та обґрунтовано метод підвищення ефективності для гібридних систем баз даних, що працюють в умовах змішаних OLTP та OLAP-навантажень. В основі методу лежить концепція динамічного партиціювання з розділенням даних на "гарячі" та "холодні" сегменти та застосуванням диференційованої стратегії індексування.

Продемонстровано, що такий підхід має ключові переваги:

- Прискорення аналітики: Створення стовпцевих індексів на "холодних" (переважно статичних) партиціях забезпечує значне підвищення продуктивності складних OLAP-запитів.
- Ефективність транзакцій: Відмова від громіздких індексів на "гарячих" партиціях на користь легковагових некластеризованих індексів дає змогу уникнути накладних витрат під час виконання частих операцій вставки, оновлення та видалення (OLTP).
- Балансування навантаження: Механізм автоматичного розбиття переповнених "гарячих" партицій слугує ефективним інструментом для розподілу навантаження та підтримки стабільно високої продуктивності.

У ході дослідження доведено, що використання лише індексування або тільки партиціювання не здатне повною мірою задовольнити потреби гібридних OLTP/OLAP-систем. Розроблений комбінований метод поєднує дворівневе партиціювання даних (за діапазоном часу та списком категорій) із динамічною індексацією, що дає змогу ефективно розподіляти навантаження та підвищувати продуктивність системи. З одного боку, партиціювання оптимізує роботу з великими обсягами даних і мінімізує накладні витрати при інтенсивних транзакціях; з іншого - адаптивне застосування індексів суттєво прискорює аналітичні запити. Експериментальні результати підтвердили перевагу комбінованого підходу, який забезпечує найкращий баланс між OLTP та OLAP-навантаженнями. Реалізація у вигляді збереженої процедури автоматизує управління партиціями та індексами, що робить метод практично придатним для промислових гібридних систем.

Таким чином, розроблений метод дає змогу ефективно поєднати переваги двох типів індексування, мінімізуючи їхні недоліки. Він є перспективним рішенням для сучасних систем, де потрібна одночасна підтримка оперативних транзакцій та аналітики в реальному часі без необхідності утримувати окремі сховища даних.

Література

1. Ажмуратов, М. О. Методи та моделі оптимізації запитів в реляційних базах даних / М. О. Ажмуратов // Управління розвитком складних систем. – 2019. – № 38. – С. 132–137.
2. Грицунов, О. В. Технології сховищ даних : навч. посіб. / О. В. Грицунов. – Харків : Вид-во "Форт", 2017. – 320 с.
3. Коваленко, І. І. Архітектурні підходи до побудови гібридних систем обробки транзакцій та аналітики / І. І. Коваленко, Ю. П. Зайченко // Системні дослідження та інформаційні технології. – 2020. – № 4. – С. 45–58.
4. Лисенко, С. М. Методи партиціонування таблиць для підвищення продуктивності аналітичних запитів / С. М. Лисенко, О. В. Павленко // Вісник Національного технічного університету "ХПІ". Серія: Системний аналіз, управління та інформаційні технології. – 2021. – № 2 (8). – С. 34–41.
5. Пасічник, В. В. Сховища даних : навч. посіб. / В. В. Пасічник, Н. Б. Шаховська. – Львів : Магнолія, 2014. – 496 с.
6. Петренко, В. В. Адаптивне індексування в гібридних сховищах даних / В. В. Петренко // Інформаційні технології та комп'ютерна інженерія. – 2022. – № 3 (54). – С. 25–33.
7. Широчин, В. П. Архітектура, алгоритми та моделі систем обробки великих даних / В. П. Широчин, С. В. Іщенко // Проблеми програмування. – 2018. – № 2. – С. 78–91.
8. Abadi, D. J. The Beckman report on database research / D. J. Abadi, S. Agrawal, A. Ailamaki // Communications of the ACM. – 2016. – Vol. 59, № 2. – P. 92–99.
9. Al-Jarrah, O. Y. A survey of machine learning-based query optimization / O. Y. Al-Jarrah, A. Al-Herz, P. Yoo // The Knowledge Engineering Review. – 2021. – Vol. 36. – P. 1–25.
10. Armbrust, M. Spark SQL: Relational Data Processing in Spark / M. Armbrust, R. S. Xin, M. J. Franklin // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. – 2015. – P. 1383–1394.
11. Dageville, B. The Snowflake Elastic Data Warehouse / B. Dageville, T. Cruanes, M. Zukowski // Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data. – 2016. – P. 215–226.
12. Dageville, B. CockroachDB: The Resilient Geo-Distributed SQL Database / B. Dageville, P. Mattis, B. Sumville // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. – 2020. – P. 1513–1530.
13. Farhadi, N. A decade of HTAP: A survey and future outlook / N. Farhadi, T. Rabl, M. Poess // Proceedings of the VLDB Endowment. – 2022. – Vol. 15, № 12. – P. 3601–3614.

14. Garcia-Molina, H. Database Systems: The Complete Book / H. Garcia-Molina, J. D. Ullman, J. Widom. – 2nd ed. – Pearson, 2014. – 1248 p.

15. Huang, D. TiDB: A Raft-based HTAP Database / D. Huang, Q. Wang, E. Zhang // Proceedings of the VLDB Endowment. – 2020. – Vol. 13, № 12. – P. 3052–3065.

16. Idreos, S. The Data Calculator: Data-driven query optimization / S. Idreos, K. Kossmann, G. Karypis // Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR '15). – 2015. – P. 233–244.

17. Kemper, A. Data Warehousing and Data Mining: Concepts and Principles / A. Kemper, A. Eickler. – Morgan Kaufmann, 2015. – 450 p.

18. Kleppmann, M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – O'Reilly Media, 2017. – 616 p.

19. Larson, P. A. The Microsoft SQL Server Hekaton in-memory OLTP engine / P. A. Larson, C. Diaconu, D. DeWitt // Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. – 2015. – P. 1239–1248.

20. Marcus, R. Neo: A Learned Query Optimizer / R. Marcus, P. Papaemmanouil, O. K. Tuncel // Proceedings of the VLDB Endowment. – 2019. – Vol. 12, № 11. – P. 1705–1718.

21. Özcan, F. What's Next in Analytics and HTAP? / F. Özcan, M. T. Özsu // Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). – 2018. – P. 1–6.

22. Pavlo, A. What's Really New with NewSQL? / A. Pavlo, M. Aslett // SIGMOD Record. – 2016. – Vol. 45, № 2. – P. 45–55.

23. Raasveldt, M. The design and implementation of the DuckDB in-process analytical database system / M. Raasveldt, H. Mühleisen // Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data. – 2019. – P. 1989–1992.

24. Shute, J. F1: A Distributed SQL Database That Scales / J. Shute, R. Vingralek, B. Samwel // Proceedings of the VLDB Endowment. – 2013. – Vol. 6, № 11. – P. 1068-1079.

25. Using Clustered Columnstore Indexes [Електронний ресурс] // Microsoft SQL Server Docs. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview. – (Дата звернення: 22.09.2025).

References

1. Azhmuratov, M. O. Metody ta modeli optymizatsii zapytiv v reliatsiinykh bazakh danykh / M. O. Azhmuratov // Upravlinnia rozvytkom skladnykh system. – 2019. – № 38. – S. 132–137.

2. Hrytsunov, O. V. Tekhnolohii skhovyshch danykh : navch. posib. / O. V. Hrytsunov. – Kharkiv : Vyd-vo "Fort", 2017. – 320 s.

3. Kovalenko, I. I. Arkhitekturni pidkhody do pobudovy hibrydnykh system obrobky tranzaktsii ta analityky / I. I. Kovalenko, Yu. P. Zaichenko // Systemni doslidzhennia ta informatsiini tekhnolohii. – 2020. – № 4. – S. 45–58.

4. Lysenko, S. M. Metody partytsionuvannia tablyts dlia pidvyshchennia produktyvnosti analitychnykh zapytiv / S. M. Lysenko, O. V. Pavlenko // Visnyk Natsionalnoho tekhnichnoho universytetu "KhPI". Seriia: Systemnyi analiz, upravlinnia ta informatsiini tekhnolohii. – 2021. – № 2 (8). – S. 34–41.

5. Pasichnyk, V. V. Skhovyshcha danykh : navch. posib. / V. V. Pasichnyk, N. B. Shakhovska. – Lviv : Mahnoliia, 2014. – 496 s.

6. Petrenko, V. V. Adaptyvne indeksuvannia v hibrydnykh skhovyshchakh danykh / V. V. Petrenko // Informatsiini tekhnolohii ta kompiuterna inzheneriia. – 2022. – № 3 (54). – S. 25–33.

7. Shyrochyn, V. P. Arkhitektura, alhorytmy ta modeli system obrobky velykykh danykh / V. P. Shyrochyn, S. V. Ishchenko // Problemy prohramuvannia. – 2018. – № 2. – S. 78–91.

8. Abadi, D. J. The Beckman report on database research / D. J. Abadi, S. Agrawal, A. Ailamaki // Communications of the ACM. – 2016. – Vol. 59, № 2. – P. 92–99.

9. Al-Jarrah, O. Y. A survey of machine learning-based query optimization / O. Y. Al-Jarrah, A. Al-Herz, P. Yoo // The Knowledge Engineering Review. – 2021. – Vol. 36. – P. 1–25.

10. Armbrust, M. Spark SQL: Relational Data Processing in Spark / M. Armbrust, R. S. Xin, M. J. Franklin // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. – 2015. – P. 1383–1394.

11. Dageville, B. The Snowflake Elastic Data Warehouse / B. Dageville, T. Cruanes, M. Zukowski // Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data. – 2016. – P. 215–226.

12. Dageville, B. CockroachDB: The Resilient Geo-Distributed SQL Database / B. Dageville, P. Mattis, B. Sumville // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. – 2020. – P. 1513–1530.

13. Farhadi, N. A decade of HTAP: A survey and future outlook / N. Farhadi, T. Rabl, M. Poess // Proceedings of the VLDB Endowment. – 2022. – Vol. 15, № 12. – P. 3601–3614.

14. Garcia-Molina, H. Database Systems: The Complete Book / H. Garcia-Molina, J. D. Ullman, J. Widom. – 2nd ed. – Pearson, 2014. – 1248 p.

15. Huang, D. TiDB: A Raft-based HTAP Database / D. Huang, Q. Wang, E. Zhang // Proceedings of the VLDB Endowment. – 2020. – Vol. 13, № 12. – P. 3052–3065.

16. Idreos, S. The Data Calculator: Data-driven query optimization / S. Idreos, K. Kossmann, G. Karypis // Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR '15). – 2015. – P. 233–244.

17. Kemper, A. Data Warehousing and Data Mining: Concepts and Principles / A. Kemper, A. Eickler. – Morgan Kaufmann, 2015. – 450 p.

18. Kleppmann, M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – O'Reilly Media, 2017. – 616 p.

19. Larson, P. A. The Microsoft SQL Server Hekaton in-memory OLTP engine / P. A. Larson, C. Diaconu, D. DeWitt // Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. – 2015. – P. 1239–1248.

20. Marcus, R. Neo: A Learned Query Optimizer / R. Marcus, P. Papaemmanouil, O. K. Tuncel // Proceedings of the VLDB Endowment. – 2019. – Vol. 12, № 11. – P. 1705–1718.

21. Özcan, F. What's Next in Analytics and HTAP? / F. Özcan, M. T. Özsu // Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). – 2018. – P. 1–6.

22. Pavlo, A. What's Really New with NewSQL? / A. Pavlo, M. Aslett // SIGMOD Record. – 2016. – Vol. 45, № 2. – P. 45–55.

23. Raasveldt, M. The design and implementation of the DuckDB in-process analytical database system / M. Raasveldt, H. Mühleisen // Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data. – 2019. – P. 1989–1992.

24. Shute, J. F1: A Distributed SQL Database That Scales / J. Shute, R. Vingralek, B. Samwel // Proceedings of the VLDB Endowment. – 2013. – Vol. 6, № 11. – P. 1068-1079.

25. Using Clustered Columnstore Indexes [Elektronnyi resurs] // Microsoft SQL Server Docs. – 2025. – Rezhym dostupu: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview. – (Data zvernennia: 22.09.2025).