DOI: https://doi.org/10.36910/6775-2524-0560-2024-54-25
УДК 004.06
Семенюк Вадим Володимирович, старший інженер
https://orcid.org/0009-0007-8670-4023
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ, Україна

АНАЛІЗ ТА ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ БАЗ ДАНИХ У РОЗПОДІЛЕНИХ СИСТЕМАХ

Семенюк В.В. Аналіз та оптимізація продуктивності баз даних у розподілених системах. У роботі проведено аналіз та розкрито принципи оптимізації баз даних у розподілених системах. Наголошується, що в умовах швидко зростаючої бази цифрових користувачів високонавантажені програми – ті, які можуть обробляти від тисяч до мільйонів запитів за хвилину, – вимагають точного налаштування для задоволення попиту, що росте. Також розглядаються різні методи оптимізації та стратегії, необхідні підвищення продуктивності баз даних у розподілених системах. Спочатку аналізується важливість оптимізації коду від написання ефективних алгоритмів до використання правильних структур даних, підкреслюючи її вплив на зниження обчислювальної складності додатків, що входять до складу розподілених систем. Далі обговорюється актуальність оптимізації бази даних та досліджуються такі методи, як індексація, кешування та оптимізація запитів, які відіграють життєво важливу роль у збільшенні часу відгуку бази даних, а отже, і продуктивності програми. Докладно обговорюються основні покоління оптимізаторів запитів: оптимізація, керована певними фазами життєвого циклу (ODCP), і оптимізація, керована всіма фазами (ODAP). Схематично представлено фази життєвого циклу БД і процес оптимізації, що дозволяє наочно прослідкувати весь процес. Окреслено питання матеріалізованого та не матеріалізованого представлення, наведено особливості. Сформовано процес перезапису запиту в процесі оптимізації. Зазначається, що у процесі оптимізації запитів у паралельній обробці оптимальний послідовний план виконання, створений наприкінці фази оптимізації, перетворюється на паралельний план виконання після кроку розпаралелювання.
Ключові слова: база даних, розподілена система, оптимізація, аналіз, структура, продуктивність.

Semeniuk V. Analysis and optimization of database performance in distributed systemsc. The paper analyzes and discloses the principles of database optimization in distributed systems. It emphasizes that with a rapidly growing digital user base, high-load applications – those that can process thousands to millions of requests per minute – require fine-tuning to meet growing demand. Various optimization techniques and strategies needed to improve database performance in distributed systems are also discussed. First, the importance of code optimization is analyzed, from writing efficient algorithms to using the right data structures, emphasizing its impact on reducing the computational complexity of applications that are part of distributed systems. It then discusses the relevance of database optimization and explores techniques such as indexing, caching, and query optimization that play a vital role in increasing database response time and therefore application performance. The main generations of query optimizers are discussed in detail: optimization directed by specific lifecycle phases (ODCP) and optimization directed by all phases (ODAP). The phases of the database life cycle and the optimization process are schematically presented, which allows you to visually follow the entire process. The issue of materialized and non-materialized representation is outlined, features are given. The process of overwriting the request in the optimization process has been created. It is noted that in the process of query optimization in parallel processing, the optimal sequential execution plan created at the end of the optimization phase is transformed into a parallel execution plan after the parallelization step.
Key words: database, distributed system, optimization, analysis, structure, performance.

Вступ та постановка проблеми. Налаштування розширених баз даних у розподілених системах, які обробляють великі обсяги даних, таких як сховища даних [1, 2], є важливою проблемою для спільноти баз даних. Вимога до швидкого отримання результату запиту, виконаного в базі даних (БД), залишається незмінною для всіх користувачів. Адміністратор БД відповідає за реалізацію необхідних структур даних, щоб гарантувати продуктивність бази даних у розподілених системах.

Цей аспект роботи адміністратора є питанням фізичного дизайну, який називається налаштуванням бази даних. Фізичне проектування визначається як фаза життєвого циклу БД, на якій виконується впровадження структур для забезпечення ефективності БД. Метою фізичного проектування є оптимізація запитів, щоб результати запитів отримувалися максимально швидко [1].

Важливість фізичного дизайну зросла, оскільки оптимізатори запитів стають складнішими для роботи із запитами підтримки прийняття рішень [3]. Це посилення пов'язано з певними характеристиками, пов'язаними з розширеними базами даних у розподілених системах:

– складність схеми бази даних, де таблиці не мають однакових характеристик;
– складність запитів, які все частіше вимагають складних операцій, таких як об'єднання та агрегація;
– обсяг даних: максимізація даних у БД стає важливою, особливо з додатками БД, пов'язаними з Інтернетом і соціальними мережами;

- вимоги до часу відповіді на запити;
- різноманітність оптимізаційних структур.

Деякі принципи оптимізації застосовуються під час створення бази даних, такі як горизонтальна фрагментація, а інші під час експлуатації бази даних, такі як матеріалізовані перегляди. Це ускладнює процес вибору та використання.

Аналіз останніх досліджень і публікацій. Формулювання наукової думки в окресі оптимізації баз даних у розподілених системах є різнорідною та масштабною. У сучасній науковій площині з'являються роботи присвячені дослідженням механізмів оптимізації баз даних для підвищення продуктивності систем.

С. В. Суліма та О. Д. Єрмолаєв [4] розробилио метод оптимізації SQL-запитів спеціально для ситуацій, коли швидкість вибірки даних погіршується з часом. Цей метод включає заміну оператора IN на тимчасову таблицю та використання не кластеризованого індексу. Таким чином, він прискорює процес вибірки даних, зменшуючи логічні звернення.

У [5] проаналізовано недоліки сучасних методів для розподілу завдань у розподілених системах оброблення даних; створено оптимізований метод планування завдань на основі метаданих у РСОД, який ґрунтується на методології пошуку найближчих сусідів із використанням методу локалізованого хешування та алгебри скінченних предикатів; деталізовано процеси в модифікованому алгоритмі пошуку найближчих сусідів; розроблено архітектуру програмного рішення, що інтегрує оптимізований метод планування завдань на основі метаданих та алокації ресурсів; за допомогою практичного сценарію здійснено валідацію програмного рішення – використання створеного алгоритму в задачі планування для декодування відеоінформації.

Із зарубіжних авторів варто відмітити роботи таких науковців як: Лі Цзі, Чжан Цзячі, Чжоу Веньчао, Лю Юйхан, Чжан Шуай, Сюе Чжуомін, Сюй Дін, Фань Хуа, Чжоу Фангюань, Лі Фейфей [6], Аннесер Крістоф, Татбул Незіме, Коен Девід, Сюй Чженган, Пандіан Прітх , Маркус Райан [7], Ден Акін [8], Хоссейнзаде Шима, Парвареш Амірхоссейн, Фей Дітмар [9], Ткіндт Вінсент [10], Хуан Шіюе, Цінь Яньчжао, Чжан Сіньї, Ту Яофен, Лі Чжунлян, Цуй Бінь], Каррас Арістейдіс, Каррас Христос, Перванас Антоніос, Сіутас Спірос, Зарольягіс Христос [12], Бхардвадж Анкур [13] та інших.

Однак, незважаючи на масштабність наукових досліджень, питання актуальності даної роботи не викликає сумнівів.

Постановка завдання. Метою роботи є аналіз та оптимізація баз даних у розподілених системах.

Викладення основного матеріалу дослідження. Оптимізація запитів у контексті баз даних у розподілених системах займає важливе місце серед різних поколінь баз даних: традиційних баз даних, баз даних XML, баз даних рішень, статистичних і наукових, і семантичних баз даних. Дійсно, програми баз даних завжди шукають більш ефективний час обробки запитів. Таким чином, оптимізація запитів полягає в переписуванні дерева виконання запиту для вибору найбільш ефективного плану виконання [3, 4]. Перед виконанням запиту виконується розбір для перевірки синтаксису та перекладу в алгебраїчні операції. Цей аналіз створює дерево операцій, які потрібно виконати. Однак можна трансформувати це дерево, щоб отримати інші еквіваленти, які пропонують різні засоби для отримання того самого результату. Ці дерева називаються планами виконання [5]. Роль механізму оптимізації (його також називають оптимізатором) полягає у створенні різних планів виконання та виборі найкращого.

Фізичний дизайн розвинувся та поступово охопив увесь життєвий цикл БД. Дійсно, процес оптимізації запитів поступово враховував параметри з фаз життєвого циклу проектування бази даних (концептуальний дизайн, логічний дизайн, фізичний дизайн і розгортання (рис. 1)) і мову запитів, що використовується цільовою системою керування базами даних (СУБД). Існує два основних покоління оптимізаторів запитів: оптимізація, керована певними фазами життєвого циклу (ODCP), і оптимізація, керована всіма фазами (ODAP).

Оптимізатори ODCP в основному базуються на вивченні алгебраїчних властивостей мови запитів, визначених на логічній моделі бази даних [6]. Одним із прикладів таких оптимізаторів є підхід на основі правил (RBA), який використовує інтуїтивно зрозумілий набір правил для оптимізації запитів.

Дійсно, деякі властивості дозволяють змінювати порядок алгебраїчних операторів, щоб отримати приріст продуктивності.

Оптимізатори ODAP з'явилися для подолання недоліків оптимізаторів ODCP. Вони враховують концептуальні, логічні, фізичні параметри та параметри розгортання. Як приклад – Cost-Based Approach (CBA). Цей підхід полягає в тому, що спочатку обчислюють вартість різних можливих стратегій, що відповідають планам виконання запиту відповідно до характеристик файлів, на яких встановлені зв'язки, а потім вибирають план із мінімальними витратами. Одна з дій оптимізаторів ODAP полягає в тому, щоб упорядкувати з'єднання (і встановити операції) для обробки якомога меншої кількості кортежів, якомога більше мобілізувати існуючі індекси та використовувати найефективніші алгоритми.

Для цього механізму оптимізації потрібні параметри з усіх фаз життєвого циклу (рис. 1):

– концептуальний рівень: довжина кожного атрибута кожної таблиці;

– логічний рівень: розмір (у термінах екземплярів) кожної таблиці;

– фізичний рівень: модель зберігання, яка використовується для зберігання таблиць (сховище рядків, сховище стовпців) і методів доступу до даних;

– рівень розгортання: характеристики диска, такі як розмір сторінки диска.

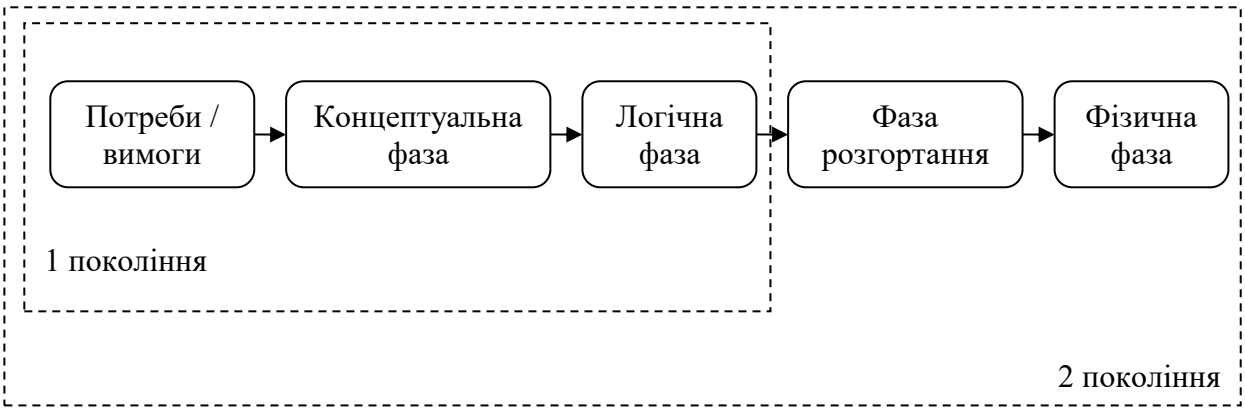

Рис. 1 – Фази життєвого циклу БД і процес оптимізації

Таким чином, оптимізація запитів ODAP чутлива до всіх фаз життєвого циклу розробки бази даних. Припустімо, змінюється мова запитів, тип бази даних, модель зберігання або платформа розгортання; потім слід переглянути процес оптимізації. Усі оптимізації часто розглядаються на фізичній фазі життєвого циклу.

Необ'єктно-орієнтовані бази даних беруть лише один параметр із концептуальної фази: словник даних. Концептуальна модель зрозуміло виражає потреби програми та знання предметної області для наступного користувача. Однак, використовується логічна модель, що є результатом нормалізації (у випадку реляційних баз даних) та адаптації до системи підтримки, яка, як правило, сильно відрізняється від концептуальної моделі. В об'єктно-орієнтованих базах даних оптимізатори враховують характеристики об'єктної моделі під час оптимізації: вирази шляхів, групування об'єктів, сканування вказівників, індекси шляхів та методи користувача [2].

Інструменти фізичного проектування також називають оптимізаційними структурами. Деякі з цих структур втручаються дуже рано в життєвий цикл БД; це випадок фрагментації, паралельної обробки та кластеризації, які розглядаються з концептуальної фази до створення на фазі розгортання. Інші – ближче до кінця свого життєвого циклу; це стосується індексів і матеріалізованих представлень, створених на етапі експлуатації.

Техніка індексування є дуже важливою опцією для налаштування бази даних для традиційних і розширених баз даних у розподілених системах. Індекси використовуються для покращення часу доступу до даних. Визначення індексів часто виконується під час роботи бази даних. Значна кількість індексів запропонована та підтримується комерційними та академічними СУБД. Перевагою індексів є прискорення пошуку інформації. Дійсно, індекс представляє таблицю, відсортовану за даним полем. Але в індексів є й недоліки: щоразу, коли створюється індекс, навантаження на СУБД збільшується. Таким чином, операції введення та обслуговування даних сповільнюються через наявність індексів, оскільки вони повинні бути негайно оновлені. Вибір набору індексів для оптимізації заданого навантаження запиту є складною проблемою.

Щоб відповісти на цю проблему, були запропоновані рішення за підходом ODCP, де використовуються певні правила, такі як використання атрибутів-кандидатів для індексування та частота запитів [3]. Цих правил недостатньо для успішного вибору. Щоб подолати цей тип вибору, пропонується формалізувати проблему вибору індексу (ISP) як задачу оптимізації відповідно до підходу ODAP. Оскільки дано:

- завантаження запитів Q, де кожен запит має частоту доступу;
- база даних або діаграма сховища даних, розгорнута на певній платформі;
- простір зберігання індексу S.

Проблема вибору індексу полягає в тому, щоб знайти кращу конфігурацію індексу, CI, що дозволяє оптимізувати Q, тобто зменшити загальну вартість виконання запитів, дотримуючись обмеження на зберігання. Розроблено кілька підходів до вирішення цієї проблеми [4]. Вони починаються з переліку всіх атрибутів кандидатів для індексації. Ці атрибути вибираються з атрибутів, присутніх у реченнях WHERE, GROUP BY і ORDER BY запитів. Запропоновано алгоритми відбору атрибутів, які можна індексувати; вони керуються математичною моделлю витрат, яка кількісно визначає якість отриманого рішення. До них належать генетичні алгоритми, симуляція відпалу або повне лінійне програмування. Один із цих підходів використовується в модулі What-if СУБД SQL Server [3].

Матеріалізоване представлення (MV) – це іменований запит, результат якого постійно зберігається в базі даних. Матеріалізовані представлення покращують виконання запитів шляхом попереднього обчислення найдорожчих операцій, таких як об'єднання та збереження їх результатів у базі даних. Таким чином, деякі запити вимагають лише доступу до матеріалізованих представлень і, отже, виконуються швидше [6]. Матеріалізовані представлення використовуються для досягнення кількох цілей, таких як покращення продуктивності запитів або надання дублікатів даних (наприклад, кеш-пам'ять у проксі-серверах). Використання MV має три основні проблеми, а саме: проблема їх вибору, проблема їх підтримки та переписування запитів на основі представлень.

Проблему вибору матеріалізованих поглядів можна формалізувати наступним чином:

- авантаження запитів Q, де кожен запит має частоту доступу;
- схема БД або сховища даних, розгорнута на даній платформі;
- набір обмежень C (простір для зберігання, час обслуговування тощо).

Проблема вибору матеріалізованого представлення (MVSP) полягає в пошуку набору представлень, що зменшує загальну вартість виконання запитів Q і враховує обмеження C.

Таблиці бази даних змінюються та розвиваються зі швидкістю оновлення. Однак, якщо ці зміни не перенести на матеріалізовані об'єкти, їхній зміст застаріє, а їхні об'єкти більше не представлятимуть реальність. Обслуговування матеріалізованих представлень складається з перенесення модифікацій у таблицях бази даних на рівень матеріалізованих представлень. Це можна зробити за допомогою трьох підходів: періодичного, негайного та відстроченого. У періодичному підході [3] перегляди оновлюються постійно в певний час. При негайному підході [8] перегляди оновлюються негайно в кінці кожної транзакції. В останньому підході зміни поширюються у відкладений спосіб [9].

Обслуговування представлень може виконуватися шляхом перерахунку цих представлень із таблиць бази даних. Однак такий підхід дуже дорогий. Належне обслуговування представлень виконується, коли зміни (вставки, видалення, модифікації), зроблені у вихідних таблицях, можна поширювати на представлення без повного перерахунку їх вмісту. Для вирішення цієї проблеми запропоновано три типи обслуговування: поступове, автономне та пакетне [8].

Поступове обслуговування складається з ідентифікації нового набору кортежів, які потрібно додати до подання у разі вставки, або підмножини кортежів, які потрібно видалити з подання у разі видалення, без повної повторної оцінки подання. Пакетне обслуговування виконується за допомогою оновлених транзакцій.

Після вибору та створення представлень усі запити необхідно переписати на основі представлень. Але пошук найкращого перепису для запиту є ґрунтовним завданням [4]. Процес переписування запитів використовувався як техніка оптимізації для зменшення вартості оцінки запиту [4]. На рисунку 2 показано процес перезапису в процесі оптимізації. Система оцінює різні плани виконання і вибирає найкращий оптимальний.

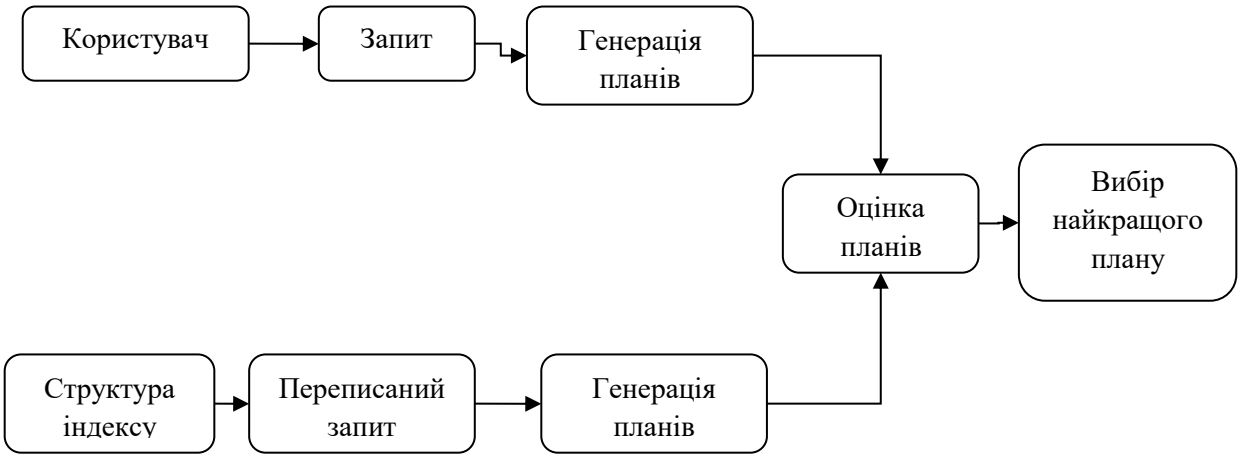

Рис. 2 – Процес перезапису запиту в процесі оптимізації

Горизонтальна фрагментація складається з розділення таблиці відповідно до її кортежів, щоб зменшити кількість доступів, які не потрібні для обробки запитів. Існує два типи горизонтальної фрагментації: первинна фрагментація [4] і похідна фрагментація [9]. Основна горизонтальна фрагментація таблиці базується на атрибутах, визначених у цій таблиці. Похідна горизонтальна фрагментація – це поширення фрагментації з однієї таблиці на іншу.

На відміну від інших оптимізаційних структур, таких як індекси та матеріалізовані представлення, де вибір схеми оптимізації зазвичай здійснюється, коли база даних працює (або створюється), вибір схеми фрагментації бази даних (або сховища даних) повинен бути прийнятий до її створення. Ця ситуація робить його вибір більш чутливим, ніж інші структури. Крім того, його також можна комбінувати з іншими оптимізаційними структурами, такими як індекси [4], матеріалізовані представлення [8] і паралельна обробка [5].

Масова паралельна обробка – це використання великої кількості процесорів або комп'ютерів для виконання набору узгоджених обчислень паралельно (тобто одночасно). У контексті баз даних, головним чином у великих даних, паралельна обробка вважається технікою для оптимізації запитів [9]. Для впровадження масової паралельної обробки використовувалися різні підходи, зокрема:

– грід-обчислення: у цьому підході обчислювальна потужність великої кількості розподілених комп'ютерів використовується за потреби, коли комп'ютер доступний для обробки запиту;

– комп'ютерний кластер: цей підхід використовує велику кількість процесорів, розташованих близько один до одного, для обробки запиту.

У такій централізованій системі дуже важливі швидкість і гнучкість взаємозв'язку процесорів.

У процесі оптимізації запитів у паралельній обробці оптимальний послідовний план виконання, створений наприкінці фази оптимізації, перетворюється на паралельний план виконання після кроку розпаралелювання. Для паралельного виконання запитів існує два рівні паралелізму: паралелізм між запитами та паралелізм усередині запитів.

Паралелізм між запитами полягає у використанні кількох процесорів в архітектурі для виконання кількох запитів одночасно;

Внутрішньо-запитовий паралелізм полягає у використанні кількох процесорів в архітектурі для виконання певного запиту. Існує дві форми внутрішнього паралелізму запитів: міжопераційний паралелізм і внутрішньо-операційний паралелізм. Взаємо-операційний паралелізм – це коли виконується кілька операцій одного запиту одночасно. Ці операції бувають незалежними або послідовними. Внутрішньо-операційний паралелізм розбиває дану операцію на набір завдань, кожна з яких розміщується на одному процесорі та виконує операцію алгоритму над частиною даних [7]. Деякі оператори, наприклад вибір і проектування, можна легко розбити на паралельні завдання. Інші, такі як об'єднання або декомпозиція, є більш складними.

Щоб полегшити паралельну обробку великих наборів даних, було запропоновано MapReduce, потужну алгоритмічну модель, яка використовує дві функції (map і reduce) [9]. Запити вказані як функції Map і Reduce. Інструменти були розроблені для представлення плану

паралельного виконання набором залежних завдань MapReduce. Завдання MapReduce містить фазу Map і Reduce. Кожна фаза створюється набором паралельних завдань типу Map або Reduce. Однак ця модель має проблеми систематичного читання та запису в розподілену файлову систему між двома завданнями; це сповільнює час виконання запиту.

Інша модель була запропонована та інтегрована в деякі інструменти, такі як Hive [5] і SparkSQL [10]. Читання з розподіленої файлової системи виконується лише на початку виконання запиту. Запис виконується тільки в кінці виконання запиту.

Оператори відношення згруповані в етапи, набір паралельних завдань створює кожен етап.

Групування об'єктів – це техніка, яка використовується для оптимізації об'єктно-орієнтованих баз даних. Вона полягає у зберіганні на одній сторінці об'єктів, пов'язаних асоціацією, щоб пришвидшити доступ до цих об'єктів шляхом навігації або приєднання [4]. Методи групування дозволяють помістити об'єкти з різних колекцій в одну групу. Групування дозволяє автономне використання пов'язаних об'єктів, які можуть існувати без пов'язаних об'єктів. У випадку об'єктів без головного або об'єктів, які спільно використовуються кількома головними, вибір сторінки зберігання неочевидний. Можна визначити групи за допомогою предикатів вибору або асоціації, щоб охопити широкий клас стратегій групування. Предикат вибору дає змогу, наприклад, визначити групу відповідно до предикату вибору. Предикат асоціації дає змогу, наприклад, визначити групу для кожного типу об'єкта.

Таким чином, індекси є основним інструментом фізичного проектування, і вони мають привілей розглядатися в усіх типах баз даних. Вони використовуються в перших базах даних (ієрархічні та мережеві бази даних), реляційних, об'єктно-орієнтованих, XML базах даних, сховищах і нових типах баз даних (бази даних NoSQL і Big Data). Матеріалізовані представлення використовуються та є дієвими в ситуаціях, коли вони мають справу з множинними запитами і потребують попереднього часткового або повного розрахунку. Вони з'явилися разом зі сховищами даних, але їх дизайн використовується не тільки в БД, наприклад, в проксі-серверах і веб-кешуванні. Індекси та матеріалізовані представлення можна використовувати разом для оптимізації сховищ даних [6]. Фрагментація – це рішення, яке полягає в наближенні даних користувача, які він часто запитує. У розподіленому середовищі, що стосується паралельної обробки, то їх використання зросло з новими типами баз даних, зокрема базами даних NoSQL і програмуванням типу MapReduce.

Висновки. У роботі проведено аналіз та розкрито принципи оптимізації баз даних у розподілених системах. Налаштування бази даних є дуже важливою фазою для забезпечення продуктивності БД, оскільки воно дає змогу реалізувати інструменти для швидкого отримання результатів запитів. Процес оптимізації, є центром налаштування бази даних, він наближений до життєвого циклу БД. Представлено огляд інструментів фізичного проектування та формально висвітлено проблеми їх вибору. Фізичне проектування не є застійним процесом, тобто воно не зупиняється після того, як структури вибрані та створені. Це вимагає регулярного моніторингу та переоцінки, щоб адаптувати структури до змін даних і параметрів платформи. Фізичний дизайн збагачується під час вертикальної еволюції бази даних з появою нових фаз і під час горизонтальної еволюції БД з розробкою нових інструментів і алгоритмів. Адміністратори даних повинні враховувати ці інструменти відповідно до типу бази даних під час налаштування у розподілених системах.

Список бібліографічного опису:

1. Saiapina Inna. Relational Database Work Optimization of the Transport Information System. Транспортні системи і технології. 2022. № 1. P. 261-266. DOI: 10.32703/2617-9040-2022-40-23.
2. Kheradkar Vivek, Vivek India, Shirgave S. Query Optimization in Uncertain and Probabilistic Databases. 2023. DOI: 10.21203/rs.3.rs-3268445/v1.
3. Selvaraj Sivaraj. Advanced Database Techniques and Optimization. 2023. DOI: 10.1007/978-1-4842-9789-6_17.
4. Суліма С. В., Єрмолаєв О. Д. Метод оптимізації SQL запитів системи управління базами даних. Системи управління, навігації та зв'язку. Збірник наукових праць. 2023. № 2. С. 151-157. DOI: 10.26906/SUNZ.2023.2.151.
5. Козирєв А. Д., Шубін І. Ю. Метод планування завдань оброблення даних у розподілених системах з обмеженою інформацією про доступні ресурси. Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 3 (25). С. 27-39. DOI: 10.30837/ITSSI.2023.25.027

References

6. Li Ji, Zhang Jiachi, Zhou Wenchao, Liu Yuhang, Zhang Shuai, Xue Zhuoming, Xu Ding, Fan Hua, Zhou Fangyuan, Li Feifei. Eigen: End-to-End Resource Optimization for Large-Scale Databases on the Cloud. Proceedings of the VLDB Endowment. 2023. № 16. P. 3795-3807. DOI: 10.14778/3611540.3611565.
7. Anneser Christoph, Tatbul Nesime, Cohen David, Xu Zhenggang, Pandian Prithviraj, Laptev Nikolay, Marcus Ryan. AutoSteer: Learned Query Optimization for Any SQL Database. Proceedings of the VLDB Endowment. 2023. № 16. P. 3515-3527. DOI: 10.14778/3611540.3611544.
8. Deng Aqin. Database task processing optimization based on performance evaluation and machine learning algorithm. Soft Computing. 2023. № 27. P. 1-11. DOI: 10.1007/s00500-023-08111-1.
9. Hosseinzadeh Shima, Parvaresh Amirhossein, Fey Dietmar. Optimization of OLAP In-Memory Database Management Systems with Processing-In-Memory Architecture. 2023. DOI: 10.1007/978-3-031-42785-5_18.
10. T'kindt Vincent. When Operations Research Meets Databases. 2023. DOI: 10.1007/978-3-031-42914-9_3.
11. Huang Shiyue, Qin Yanzhao, Zhang Xinyi, Tu Yaofeng, Li Zhongliang, Cui Bin. Survey on performance optimization for database systems. Science China Information Sciences. 2023. № 66. DOI: 10.1007/s11432-021-3578-6.
12. Karras Aristeidis, Karras Christos, Pervanas Antonios, Sioutas Spyros, Zaroliagis Christos. SQL Query Optimization in Distributed NoSQL Databases for Cloud-Based Applications. 2023. DOI: 10.1007/978-3-031-33437-5_2.
13. Bhardwaj Ankur. Exploring the Synergy between Query Optimization and Parallel Processing for Efficient Database Management. International Journal of Computer Applications & Information Technology. 2023 № 12. P. 453-460.