Однак хеш-таблиці також мають свої недоліки. Колізії можуть уповільнити операції і потребують додаткових механізмів вирішення. Хеш-таблиці можуть споживати багато пам’яті, особливо коли прагнуть мінімізувати колізії. Також у хеш-таблицях неможливо зберігати порядок елементів, що може бути критичним у деяких застосуваннях.

Хеш-таблиці широко використовуються в різних галузях. У базах даних вони слугують для індексації та швидкого пошуку. У комп’ютерних мережах вони використовуються для кешування та маршрутизації. У мовах програмування вони є основою для реалізації словників, карт і асоціативних масивів.

Отже, хеш-таблиці є важливим інструментом у структурі даних. Їх використання дає змогу забезпечити високу ефективність, особливо під час роботи з великими обсягами інформації, де швидкість доступу є критичною. Водночас необхідно враховувати можливі недоліки і застосовувати відповідні стратегії для вирішення проблем, пов’язаних із колізіями та використанням пам’яті.

Список використаних джерел

1. Михалько В. Г. Адаптивна індексна структура бази даних для точкового пошуку за допомогою підходів машинного навчання: магістерська дисертація. *Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».* 2018. URL: https://ela.kpi.ua/server/api/core/bitstreams/af870de7-5956-4265-9b4b-0b91d588e666/content

2. Hash Table Data Structure. *geeksforgeeks.org.* URL: https://www.geeksforgeeks.org/hash-table-data-structure/ (дата звернення: 20.05.2024).

3. Розподілені хеш-таблиці. *Medium.* Oct. 30, 2020. URL: http://surl.li/suowl (дата звернення: 20.05.24).

УДК 004.06

Яценко В. В., *здобувач 2 курсу спеціальності 122 Комп’ютерні науки,*
Вєтров О. С., *старший викладач кафедри інформаційних технологій*

ОГЛЯД АЛГОРИТМІВ ЧИТАННЯ ТА ЗАПИСУ В РЕЛЯЦІЙНИХ БАЗАХ ДАНИХ ТА СПОСОБИ ЇХ ОПТИМІЗАЦІЇ

Донецький національний університет імені Василя Стуса, м. Вінниця

Методи аналізу і обробки певних даних, упорядкованих і неупорядкованих, для добування знань називається наукою про дані [1]. Упорядкована інформація, об’єднана між собою за певними критеріями, називається базою даних [2]. Правила представлення, організації даних і їх зв’язків називаються моделлю даних [3]. Існують реляційна, об’єктно-орієнтована, графова, мережева, ієрархічна моделі даних. Найпопулярніша модель – реляційна – представляється у вигляді двовимірних таблиць. Для роботи з реляційними базами даних використовується SQL [4]. SQL – це мова структурованих запитів, що дає змогу користувачу взаємодіяти з даними у реляційних базах даних.

Ресурси knowledgehut, gigaspaces досліджують метод аналізу даних у реальному часу [5–6]. Згідно з їх дослідженням, сучасні компанії все частіше викорис-

товують метод аналізу даних у реальному часі, оскільки зі звичайними методами збору і аналізу інформації організації не зможуть швидко реагувати на зміни і відповідно миттєво приймати рішення, залежно від ситуації. Швидка обробка даних критично важлива, оскільки дає змогу виявити загрози, проблеми та недоліки у різних галузях на початковому етапі і швидко зреагувати на подібні події. Відтак швидкість читання і запису даних критично важливі для швидкого опрацювання інформації та реагування на неї.

Метою дослідження стане аналіз алгоритмів читання і запису та методики їх оптимізації. Необхідно дослідити, як у реляційних базах даних відбувається читання і запис даних, дослідити роботу запитів, надати можливі методи підвищення продуктивності цих алгоритмів і підсумувати інформацію.

Розглянемо, як виконуються запити у базах даних. Для дії над даними необхідно здійснити запит мовою SQL. Процес здійснення запиту можна поділити на декілька етапів. На першому етапі здійснюється розбір тексту запиту. Відбувається синтаксичний аналіз і перевірка запиту на помилки. Цей процес називається парсингом запиту. Якщо на цьому етапі буде знайдена синтаксична помилка – обробка запиту припиниться і виведеться помилка. Якщо етап синтаксичного аналізу буде пройдено вдало, він буде розділений на частини і буде створено дерево послідовності виконання запиту. Дерево являє собою кроки, які необхідно здійснити для виконання запиту. На наступному етапі дерево парсингу передається до алгебризатора. Алгебризатор перевіряє назви таблиць, атрибутів, полів, вказаних у запиті. Перевіряються і розпізнаються всі посилання, вказані у запиті. Якщо хоча б одне з посилань – некоректне, робота над запитом буде зупинена і виведеться відповідна помилка. Процес, що проводиться алгебризатором, називається прив’язкою запиту. Згодом, якщо всі посилання пройшли перевірку, відбувається перевірка типів даних посилань і визначаються агрегатори – відбувається прив’язка агрегаторів. Фактично на цьому етапі відбувається перевірка, чи дійсно імена, вказані у запиті, існують у вказаних таблицях, чи правильно вказані назви полів. Якщо всі перевірки були успішно пройдені, створюється дерево процесора запитів. Це двійковий файл, що містить дані про запит у вигляді кодованого значення – хеш-значення. За допомогою хеш-значення визначається, чи має цей запит актуальний план виконання. План вважається доцільним, якщо не було здійснено змін у таблицю. Якщо у запиту існує дійсний план виконання, процес обробки запиту припиняється і використовується план виконання запиту для оптимального виконання операцій, заданих користувачем. Якщо ж такого плану не існує, виконання запиту переходить на наступний етап. Відбувається планування виконання запиту – пошуку найоптимальнішого виконання цього запиту. Оптимізатор запитів здійснює пошук і оцінку всіх шляхів виконання запиту. Оцінюється швидкість виконання і ресурсозатратність кожного методу виконання запиту. Залежно від складності запит може пройти повну оптимізацію і отримати план на основі повної переоцінки всіх можливих шляхів виконання та може отримати тривіальний план. Тривіальні плани отримують запити, настільки легкі у виконанні, що запуск повної оптимізації не є раціональним рішенням.

Зчитування даних із бази даних здійснюється за допомогою запиту SELECT FROM. За допомогою цього запиту можна вибирати певні поля. Для вибірки пев-

ного поля з певної таблиці необхідно вказати назву поля і назву таблиці, якій це поле належить, наприклад, SELECT name FROM employee, де name – це назва поля, дані з якого потрібно зчитати, employee – назва таблиці, де знаходиться це поле. Також можна обрати всі поля з певної таблиці, використовуючи синтаксис SELECT*. Дані вибірки можуть бути відсортовані за допомогою ORDER BY. Також можна обмежити вибірку за допомогою LIMIT.

Для запису даних у базу даних використовується запит INSERT INTO VALUES. За допомогою цього запиту можна записувати значення в певні або в усі стовпці. За допомогою запиту UPDATE SET WHERE можна оновити дані у певних таблицях.

Першим методом оптимізації стане нормалізація бази даних. Нормалізація бази даних дасть змогу зменшити надмірність даних, ймовірність появи аномалій, пришвидшити запис даних у базу даних. До того ж нормалізовані бази даних мають більш ефективні індекси, що також позитивно вплине на продуктивність читання і запису. Але з іншого боку, це збільшить кількість з'єднань, через які запиту потрібно пройти для пошуку необхідної інформації, що відповідно сповільнить процес читання даних таблиць.

Другий метод оптимізації читання і запису – оптимізація запитів. По-перше, використання оператора OR не є раціональним. SQL не може обробити оператор в одній операції. У разі використання оператора UNION і задання подвійної умови в якості двох умов SQL зможе використати індекси, і запит відбудеться швидше й ефективніше. По-друге, часте використання JOIN і підзапитів у запиті сповільнює процес аналізу запиту. SQL не зможе якісно побудувати план виконання і оптимізувати запит. До того ж не гарантується, що побудований план буде максимально ефективний. Розділення запиту на декілька менших запитів зменшить кількість JOIN і підзапитів, і так прискорить аналіз та виконання. По-третє, варто уникати SELECT DISTINCT. Такі запити відбуваються значно повільніше. До того ж використання SELECT* у всіх запитах не є раціональним рішенням. Потрібно вказувати лише потрібні поля, і якщо це можливо, виставляти ліміт вибірки. Це значно прискорить швидкість роботи запиту і значно зменшить використання ресурсів ним.

Третім методом оптимізації стане індексація даних. Індексування значно прискорює пошук інформації у базі даних. Існує декілька видів індексів: кластеризовані і некластеризовані індекси. Кластеризовані індекси – це індекси, які сортують дані в таблиці за ключовим значенням. Кластеризований індекс визначає порядок розміщення даних у таблиці. На кожну таблицю може бути створений лише один кластеризований індекс. Кластеризовані індекси значно пришвидшують послідовне читання інформації. Такі індекси також значно прискорять вибірки великої кількості полів. До того ж більша продуктивність буде помітна і під час роботи з діапазонами. Однак кластеризований індекс ускладнює запис даних у таблиці, особливо якщо відбувається багато непослідовних вставок. У такому разі продуктивність запису даних буде знижена. Некластеризований індекс – індекс, який зберігає посилання на певні дані, і, на відміну від кластеризованого, не впорядковує їх фізично. Некластеризований індекс дає змогу швидко отримати доступ до певного поля, але якщо буде вибірка з багатьох полів, то цей метод

індексації не буде раціональним у використанні. Перевагою некластеризованого індексу також є пришвидшення роботи INSERT- і UPDATE-запитів, оскільки для здійснення таких операцій не потрібне сортування даних. Використання індексів дійсно пришвидшує роботу з даними, але занадто велика кількість індексів може сповільнити швидкість запису даних і зайняти багато дискового простору.

Четвертий метод оптимізації – кешування. Це процес зберігання даних, що часто використовуються в оперативній пам'яті або на жорсткому диску користувача. Існує декілька методик кешування: Cache-Aside, Read-through Cache, Write-back Cache, Write-through Cache. Cache-aside працює так, що якщо даних немає в пам'яті, відбувається звернення до бази даних. Кеш оновлюється новими даними, і тоді повертається результат запиту. Перевагою цієї методики є вирішення проблеми частих зчитувань. До того ж дані зберігаються в кеші, лише якщо вони потрібні, що заощаджує пам'ять системи. Read-through Cache працює аналогічно, за винятком того, що користувач завжди звертається до кешу. Переваги і недоліки аналогічні першому методу. Write-back Cache – дані постійно записуються в кеш, але не зберігаються в базі даних доти, доки не відбудеться запит на збереження внесених змін. Отже, в пам'яті завжди буде актуальна інформація, а запис даних у базу даних буде значно ефективніший. Недоліком способу є можлива втрата даних у разі зупинки системи. Write-through Cache записує дані в кеш і одразу переносить їх у базу даних. Перевагою методу є те, що у пам'яті завжди буде актуальна інформація, але необхідно робити два записи одночасно – в кеш і в базу даних, що може бути ресурсозатратно. Отже, методики кешування пришвидшують доступ до даних, що регулярно використовуються, шляхом зменшення кількості звернень до бази даних. Це відповідно приводить до зменшення навантаження на базу даних і значного скорочення часу обробки запиту. Вибір методики кешування залежить від потреб користувача.

П'ятий метод оптимізації – використання масових операцій, або об'єднання малих запитів у один. Методика масових операцій – виконання багатьох однотипних операцій одночасно. Це дає змогу значно прискорити обробку даних. Також можна власноруч об'єднати запити в один, що значно пришвидшить запис і оновлення даних у таблиці.

Підсумовуючи дослідження, зазначимо, що швидка обробка інформації критично важлива. Велика швидкість аналізу інформації дає змогу ефективно виявляти проблеми на перших етапах і відповідно негайно реагувати та приймати миттєві рішення. Особливо важливою стає швидкість обробки запитів читання і запису в базу даних, оскільки від цього буде залежати вчасність прийняття рішення. SQL володіє вбудованими методами оптимізації запитів і виконує їх за найбільш ефективним планом, але методи оптимізації все ж відіграють важливу роль у продуктивності системи. Ефективна структура бази даних, оптимізовані запити, правильно налаштовані індекси, а також використання різних методик кешування і масових операцій можуть значно збільшити швидкість і ефективність системи.

Список використаних джерел

1. Avijeet B. What is data science. *SimpliLearn.* 2024. URL: https://www.simplilearn.com/tutorials/data-science-tutorial/what-is-data-science (дата звернення: 15.05.2024).

2. What is a database. *amazon.com.* URL: https://aws.amazon.com/what-is/database/ (дата звернення: 15.05.2024).
3. What is data model. *Center for Data, Analytics and Reporting (CeDAR).* URL: https://cedar.princeton.edu/understanding-data/what-data-model (дата звернення: 15.05.2024).
4. What is SQL. *geeksforgeeks.org.* 2024. URL: https://www.geeksforgeeks.org/what-is-sql/ (дата звернення: 15.05.2024).
5. Gulati A. Real-time data analytics. *upGrad. Knowledgehut.* 24 Apr, 2024. URL: https://www.knowledgehut.com/blog/data-science/real-time-data-analytics (дата звернення: 15.05.2024).
6. Real-Time Data Processing. *Gigaspaces.* URL: https://www.gigaspaces.com/data-terms/real-time-data-processing (дата звернення: 15.05.2024).

УДК 004.9

Грибаніна А. О., *здобувачка 2 курсу спеціальності 122 Комп'ютерні науки,*
Поремський Ю. В., *канд. техн. наук, старший викладач кафедри інформаційних технологій*

ОСОБЛИВОСТІ ОРГАНІЗАЦІЇ ЧЕРГИ

Донецький національний університет імені Василя Стуса, м. Вінниця

Черга – це змінюваний упорядкований (чи ні) набір елементів. Додавання елементу в чергу проводиться з одного кінця (хвоста черги, REAR), а вибірка – з іншого кінця (голови черги, FRONT) відповідно до правила «Першим прийшов – першим пішов» (FIFO: First Input – First Output). Така черга є простою чергою без пріоритетів (рис. 1). Часто використовуються черги з пріоритетами, в них більш пріоритетні елементи включаються ближче до голови черги, вибірка здійснюється зазвичай з голови черги.

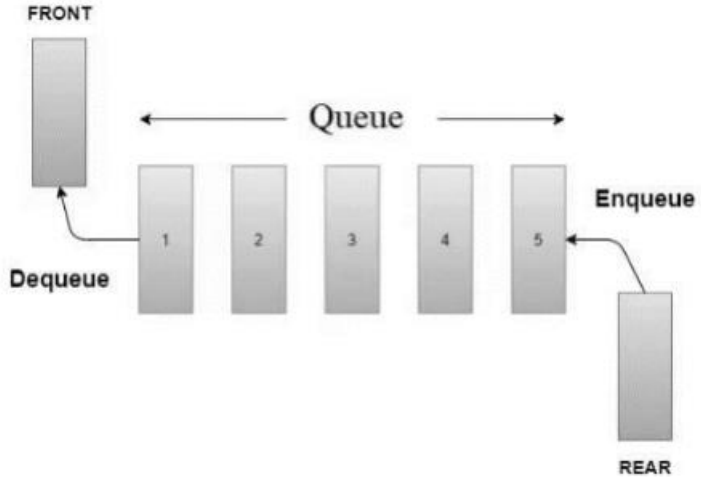

Рис. 1. Схема простої черги

Основні операції над чергою – ті ж, що і над стеком – включення, виключення, визначення розміру. Класична реалізація черги вимагає реалізувати структуру даних із такими операціями:

1. Створення порожньої черги.