DOI: https://doi.org/10.36910/6775-2524-0560-2025-59-19
УДК 004.42:519.86
Кошелюк Віктор Андрійович, к.т.н., доцент
https://orcid.org/0000-0002-4136-5087
Тимчук Владислав Володимирович, магістр
Луцький національний технічний університет, м. Луцьк, Україна

МЕТОДИ ОРГАНІЗАЦІЇ ПОРЯДКУ СОРТУВАННЯ У ВПОРЯДКОВАНИХ СПИСКАХ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Кошелюк В. А., Тимчук В. В. Методи організації порядку сортування у впорядкованих списках у програмному забезпеченні. У сучасному програмному забезпеченні, де важливу роль відіграє обробка впорядкованих списків даних, особливо у веб-додатках, системах керування завданнями, контентом чи іншими структурованими сутностями, критичним аспектом стає правильна організація порядку елементів. Найпоширенішим способом зберігання таких структур є реляційні бази даних, де забезпечення порядку вимагає додаткової уваги. Основними підходами є використання окремого числового поля (індексу порядку) або побудова зв'язних структур, де кожен запис містить посилання на попередній чи наступний. Однак зі зростанням кількості елементів виникає проблема ефективного оновлення цих значень при зміні порядку, особливо при переміщенні елементів у списку. У зв'язку з цим постає необхідність ґрунтовного аналізу сучасних підходів до організації та підтримки впорядкованості списків, зокрема методів оптимізації значень сортування з урахуванням резервного простору між суміжними індексами. Такий підхід дозволяє уникнути частих операцій з перерахунку позицій елементів під час зміни їх порядку, що, своєю чергою, сприяє підвищенню продуктивності та зменшенню навантаження на базу даних. Серед альтернативних рішень заслуговує на увагу використання горизонтального розширення порядкових значень, наприклад, шляхом застосування дробових або лексикографічних форматів. Окремо розглядаються переваги та недоліки використання зв'язних списків у порівнянні з традиційним підходом до сортування за значенням порядку. Для порівняння підходів проводиться оцінка з погляду точності та швидкодії, ефективності реалізації, та можливості масштабованості списків.
Ключові слова: база даних, сортування, оптимізація, індекс порядку, впорядкований список, зв'язний список.

Kosheliuk V., Tymchuk V. Methods of organizing sorting in ordered lists in software systems. In modern software systems, particularly in web applications, task management systems, content management platforms, and other structured data environments, the correct organization of item order in ordered lists plays a critical role. The most common method for storing such structures involves relational databases, where maintaining item order requires additional attention. The primary approaches include the use of a separate numeric field (order index) or the construction of linked structures, in which each record contains a reference to the previous or next item. However, as the number of elements grows, updating these values efficiently becomes a challenge, especially when items are repositioned within the list. This creates the need for a thorough analysis of modern approaches to organizing and maintaining list order, particularly methods for optimizing sorting values by incorporating reserved space between adjacent indices. Such approach helps to avoid frequent re-indexing operations when positions change, which in turn improves performance and reduces database load. Among alternative solutions, horizontal expansion of ordering values, such as the use of fractional or lexicographic formats, deserves particular attention. Additionally, the advantages and disadvantages of using linked lists compared to traditional sorting by order values are examined. To compare the approaches, an evaluation is conducted based on accuracy and performance, implementation efficiency, and a possibility of scaling lists.
Keywords: database, sorting, optimization, sorting index, sorted list, linked list.

Постановка проблеми. У багатьох програмних системах постає завдання зберігання елементів у довільному порядку, заданому користувачем, із можливістю динамічного переміщення цих елементів. Наприклад, у таких програмних продуктах, як: системи управління завданнями, аудіо та відео програвачі, додатки для нотаток, платформи для навчання та у багатьох інших видах програмного забезпечення, користувачі мають бажання оформити та користуватися довільними списками, у яких елементи розташовані у порядку, заданому самим користувачем. Так як постійне зберігання даних у пам'яті сервера чи комп'ютера практично неможливо через її обмежений об'єм, використовуються бази даних. Хоч порядок елементів, теоретично, може визначатися їх фізичним розташуванням у базі даних, проте це є поганою практикою, оскільки у такому випадку, при перенесенні записів потрібно перезаписувати велику кількість даних, що є поганою практикою з боку затраченого часу та інших ресурсів. Тому у практику розробників програмного забезпечення входить створення додаткового поля даних, яке використовується для впорядкування. Зазвичай це поле задає сортувальний ключ або структуру посилань між записами.

Задача полягає в ефективній підтримці цього порядку при додаванні, переміщенні чи видаленні елементів, а також уникати необхідності повного пересортування всіх даних при кожному внесенні змін. Окрім того, важливо розглянути оптимізацію та нормалізацію значень порядку, коли між двома суміжними значеннями закінчується вільний простір для нових ключів.

Таким чином, проблемою є реалізація методів організації та коректного оновлення поля порядку у впорядкованому списку з погляду продуктивності, точності та зручності реалізації.

Аналіз досліджень та публікацій Проблема підтримки впорядкованого списку (order-maintenance problem) є відомою в теорії структур даних. Вона формалізується як задача підтримки лінійного порядку з операціями вставки, видалення та запитами порівняння порядку.

Наприклад, у статті [1] запропоновано ефективні структури підтримки порядку даних в динамічних структурах XML, які можна використовувати, працюючи із записами у базах даних, адже у БД важливо мінімізувати кількість операцій при виконанні операцій над елементами списку. Авторами було описано використання спеціалізованих міток, що дозволяють уникнути частої оптимізації значень порядку елементів списку навіть при великій кількості операцій. Також, у статті [2] представлені спрощені алгоритми для підтримки порядку у списках, які досягають тих самих теоретичних меж, що й інші популярні рішення, але зі спрощеною реалізацією. Автори також надають експериментальні докази ефективності своїх підходів.

Метою роботи є систематизувати існуючі підходи до реалізації впорядкованих списків та провести їх порівняльний аналіз за ключовими критеріями. Зокрема, розглянути варіанти зберігання порядку (поле в записі проти структурованих зв'язків), поведінку при додаванні, переміщенні та видаленні елементів, вплив обраного способу на продуктивність та споживання пам'яті, особливості точності представлення порядку (наприклад, проблеми даних формату IEEE-754) та необхідність нормалізації значень порядку.

Виклад основного матеріалу. Існує низка можливих підходів до реалізації впорядкованих структур, кожен з яких має свої переваги та недоліки залежно від контексту використання. На практиці застосовують як структури на основі посилань між елементами (включаючи прямі або двобічні зв'язки), так і підходи із використанням числових значень порядку сортування. Вибір того чи іншого методу часто залежить від наявних ресурсів системи, частоти потреби у нормалізації значень, та впливає на точність збереження порядку, а також стійкість до масштабування. Нижче наведено детальний порівняльний аналіз цих методів у типових сценаріях змін структури списку.

До популярних підходів реалізації впорядкованих списків відносяться використання поля порядку сортування або зв'язних списків.

Використання поля порядку є найпростішим і найпоширенішим підходом збереження впорядкованості елементів у реляційних базах даних. Даний підхід передбачає додавання в таблицю колонки, що визначає позицію кожного запису при сортуванні. Зазвичай такій колонці надають ім'я «*sort_index*» або «*order_index*». Значенням поля індексу сортування рядка таблиці також називають ключем сортування, індексом упорядкування тощо.

Значення даної колонки (поля) залежить від вимог до програмного забезпечення, потужностей системи, на якій розгорнуте ПЗ, та від персональних уподобань розробників. Зазвичай для значення індексу сортування використовують цілочисельні, дробові або рядкові (текстові) ключі.

У випадку використання підходу з цілочисельними значеннями – у колонці сортування елементів використовуються послідовні цілі числа (наприклад, 1, 2, 3, …). До переваг такого підходу відносяться легкість в реалізації, точність, інтуїтивність та очевидність, оскільки такі значення легко зрозуміти в процесі дебагінгу (під час розробки програмного забезпечення) або ручного виставлення ключів. До суттєвих недоліків даного підходу належить відсутність гнучкості для вставки елементів – кожна вставка нового елементу не в кінці списку потребує зсуву великої кількості індексів сортування інших елементів, за рахунок чого дуже страждає швидкодія системи. Прикладом використання даного методу є статичні списки, або списки, де операція вставки не є частим випадком, наприклад – таблиці лідерів.

Використання підходу розрідженого цілочисельного підходу використовується для того, щоб полегшити вставку нових елементів, часто залишають пусті простори**.** Наприклад, записам можуть бути присвоєні значення «100», «200», «300» і так далі. Це дозволяє вставляти нові елементи між існуючими, присвоюючи їм значення середнього арифметичного ключа нових «сусідів»: «150», «250» тощо, за рахунок чого цей підхід являється набагато продуктивнішим за підхід послідовних цілочисельних індексів. Також, дане рішення являється безпечнішим у випадку спроби вставки елементів кількома користувачами одночасно. До недоліків можна віднести значення індексів сортування менш читабельними та все ще наявну (хоч і нерегулярну) необхідність нормалізації ключів сортування, оскільки все ще можлива відсутність вільних цілочисельних значень між двома індексами елементів. *Нормалізація* – це процес переприсвоєння індексів (що потребує оновлення

багатьох рядків). У списку систем, де використовується даний підхід є такі популярний додаток для продуктивності Trello [3].

Слід зазначити, що у більшості систем явний порядковий номер не є необхідним, а у випадках, де він потрібен – його можна неявно сформувати прямо в інтерфейсі, то недолік неінтуїтивного представлення ключа не є суттєвим.

Підхід з використанням значень з плаваючою комою використовують значення типу «*float*» або «*double*» для більш точної вставки. Наприклад, якщо є елемент з індексом «1,0» і елемент з індексом «2,0», можна вставити новий елемент з індексом «1,5». Це дозволяє легше керувати порядком, зменшити або взагалі позбутися процесу нормалізації ключів сортування, хоча теж не без обмежень: використання сучасними мовами програмування даних з плаваючою комою за стандартом «64-bit IEEE-754 float» характеризується частими проблемами з точністю та арифметичними операціями [4], зокрема з операціями порівняння та ділення, які є ключовими у процесі маніпуляції ключами порядку елементів довільних списків. Даний тип ключа упорядкування використовується у сучасних системах, у яких є типовим випадок спільного доступу до редагування довільних списків.

Використання лексикографічного підходу з текстовими ключами упорядкування набуло популярності за рахунок можливості умовно-безкінечної генерації нових ключів між існуючими (обмеженням стає лише фізична пам'ять хоста бази даних, де вона розгорнута, рідше – обмеження розміру текстового поля даних). При використанні даного підходу не виникають проблеми паралельної вставки та немає недоліків у точності, на відміну від числових значень. Проте, реалізація даного підходу є складнішою, у порівнянні з іншими підходами, також, порівняння рядків є більш ресурсозатратним процесом. Даний метод часто використовується різним програмним забезпеченням для реалізації операції функціоналу відміни або повтору.

У таблиці 1 наведено порівняльну характеристику усіх згаданих підходів з використанням індексу упорядкування за основними критеріями.

Таблиця 1. Порівняльна характеристика підходів з використанням індексу сортування

Тип індексу порядку	Швидкодія операції вставки	Точність	Чи потребує нормалізації з часом
Послідовні цілочисельні значення	Повільна	Висока	Так
Розріджені цілочисельні значення	Середня	Висока	Так (рідко)
Значення з плаваючою комою	Швидка	Середня	Так (у випадку дрейфу плаваючої коми)
Текстові значення	Швидка	Висока	Ні

Інша методологія зберігання порядку передбачає використання посилань з одного елементу на сусідні. Наприклад, кожен запис може мати поле «prev_id» (та, в разі двозв'язного списку, ще «next_id»), що вказує на сусідні елементи. У такій реалізації елементи фізично зв'язані, й зміна порядку здійснюється переназначенням цих посилань. Наприклад, у однозв'язному списку при вставці нового елемента на певну позицію достатньо змінити «next_id» попереднього вузла. Підхід дозволяє легко додавати і видаляти окремі елементи (достатньо кількох операцій оновлення полів) без зміни інших. У випадку використання реляційних баз даних, надійність порядку може бути забезпеченою створенням прямих зв'язків між елементами. Проте для одержання всього списку в правильному порядку потрібне або багаторядкове з'єднання *JOIN* або використання рекурсивного запиту (*Common Table Expression)* до таблиці. У випадку використання СКБД PostgreSQL можна використати функціонал спеціалізованого поля індексу рівня [5]. Існує й другий недолік: якщо елемент одночасно повинен належати до декількох впорядкованих представлень (наприклад, завдання належать до різних категорій або проектів), потрібно мати окремі зв'язки для кожної з цих «лістів». У такому випадку зазвичай використовують допоміжні таблиці для зв'язку «список–елемент» з полями порядку в цьому списку. Прикладом може бути таблиця «category_member» (із колонками «category_id», «task_id», «order_index»), де «order_index» задає позицію завдання у категорії. Найсуттєвіший недолік даного підходу виражається тоді, коли є потреба створення функціоналу фільтрування елементів списку – у цьому випадку використання даного методу не є доцільним, адже не можливо передбачити усі можливі комбінації результатів пошуку з фільтрами.

Отже, зв'язні списки (або таблиці зв'язків) зручно використовувати, якщо потрібна часта локальна реорганізація (переміщення блоків елементів) і можна виконати рекурсивний обхід за цими посиланнями, але їх використання є обмеженим в залежності від функціоналу ПЗ. Водночас, використання даного підходу унеможливлює швидке виведення упорядкованих даних без додаткових обчислень.

Наведені методології можна детально порівняти за критеріями маніпуляцій даних, збереження даних у БД, витрати фізичної пам'яті на збереження даних та точність. Нижче наведено розгорнуте порівняння за даними критеріями.

Додавання і переміщення елементів: при використанні ключа сортування, для вставки на початок чи кінець списку досить присвоїти новому елементу відповідно мінімальне або максимальне значення (наприклад, нуль чи дуже велике число), не змінюючи інших рядків. Для вставки всередину потрібно обрати значення між двома сусідніми ключами. Якщо це можливо (наприклад, маємо ключі «100» і «200», тоді беремо новий ключ «150»), то оновлення торкнеться лише нового рядка. При переміщенні окремих елементів достатньо оновити значення «sort_index» (іноді – декілька, якщо змінюється відносний порядок сусідів). У випадку зв'язних списків переміщення елемента (або навіть групи суміжних елементів) виконується зміненням кількох покажчиків. Наприклад, при переміщенні одного елемента в іншу позицію у однозв'язному списку оновлюються поля «next_id» попередника нового місця та попередника старого місця. Такий підхід, хоч і вимагає кількох операцій оновлення даних, проте, у загальному є простішим, ніж оновлення великої кількості полів sort_index. Однак при використанні зв'язних списків слід бути обережним з пустими (пропущеними) посиланнями і забезпечувати цілісність зв'язків (фіксуючи транзакцією оновлення двох суміжних записів або використовуючи обмеження зовнішніх ключів). З огляду на продуктивність, реляційні СКБД більш оптимізовані для маніпуляцій даними за числовим полем, ніж під довгі ланцюжки з'єднань, тому при великій кількості елементів запит на всі записи у правильному порядку може бути помітно повільнішим при використанні методології зв'язків без допоміжного сортувального поля.

Видалення елементів: у підході з полем сортування видалення будь-якого елемента не вимагає зміни інших – достатньо виконати операцію видалення. Навіть якщо між значеннями утворюються пропуски (наприклад, після видалення елементу з порядком «2» - лишаються елементи з ключами «1» і «3»), це не впливає на результат, оскільки сортування за полем «sort_index» все одно поверне список із правильним порядком. В окремих реалізаціях може застосовуватися періодична *нормалізація* – перезапис індексів спочатку (1, 2, 3…) для зручності користувача, але це радше опція для читабельності. У зв'язних списках видалення також доволі легке: потрібно перезаписати вказівники сусідніх вузлів так, щоб видалений елемент був вилучений з ланцюга. Зазвичай це потребує оновлення не більше двох записів («next_id» попереднього та, у двозв'язному списку, «prev_id» наступного елемента). Фактично в обох підходах видалення виконується швидко і не вимагає корекцій усього набору даних.

Фізичне збереження даних у базі та витрати пам'яті: при зберіганні поля порядку як атрибута таблиці кожен рядок БД містить одне числове (чи рядкове) значення. Цей метод є компактним: додаткова пам'ять витрачається лише на одне поле числового або текстового на запис у таблиці. На відміну від цього, зв'язний список передбачає зберігання посилань (принаймні одного «next_id», іноді – ще й «prev_id»), і якщо ці посилання реалізовані як цілі числа (зовнішній ключ), то для кожного елемента додається 4–8 байт на зберігання ідентифікаторів суміжних елементів. Таким чином, зв'язний список майже вдвічі «важчий» за таблицю з одним полем індексу. За критерієм швидкості вибірки впорядкованих даних використання цифрового індексу порядку зазвичай є ефективнішим підходом: СКБД може застосовувати індекси на основі бінарного дерева або зовнішнє сортування на цьому полі. У випадку використання зв'язків доводиться застосовувати рекурсію або декілька поєднань, що при великій глибині списку може призводити до повільної вибірки.

Точність представлення: якщо для поля порядку використовується тип з плаваючою комою («*float*» або «*double*»), слід враховувати обмеження точності стандарту IEEE-754. Наприклад, між числами 1,0 і 2,0 існує скінченне число представлень «*double*», якщо послідовно вставляти елементи між сусідніми позиціями, може настати момент, коли жодне нове число не впишеться між двома існуючими через втрату розрядності. Як показує практика, для вставок (де нове значення встановлюється у процесі вставки) «*double*» дозволяє виконати близько 50–100 послідовних вставок, перш ніж дробову частину доведеться нормалізувати. Альтернативою є використання

числового типу «*DECIMAL*» із достатньою точністю або власних алгоритмів з мітками, як-от дробові мітки на *дереві Stern-Brocot* [6]. У будь-якому разі, обраний метод має передбачати періодичну нормалізацію коли між значеннями ключа «sort_index» вже неможливо вставити нове, доводиться перераховувати всі позиції в базі (наприклад, зробити всім записам порядкові номери 1…N, з кроком 1). При збереженні поля цілого типу така нормалізація зазвичай проводиться тоді, коли кроки між сусідніми значеннями стали однаковими та замалими для подальших вставок. При зв'язному підході нормалізація не застосовується: адже зв'язки не мають фіксованого кроку, завжди можна під'єднати новий вузол, змінивши вказівники.

Висновки та перспектива подальших досліджень. Таким чином, У статті проаналізовано дві основні методології зберігання та оновлення впорядкованих списків: поле сортування з числовим або рядковим індексом, зв'язні структури (список або таблиця зв'язків). При використанні методології з полем сортування можливий вибір між типами ключів упорядкування. Кожний метод має свої плюси і мінуси. Використання числових типів є простим і зручним для вибірки через звичайний SQL-запит, але потребує періодичної нормалізації та врахування обмежень точності. Використання рядкового (текстового) ключа виключає часту потребу у нормалізації, але, у порівнянні, є дещо повільнішим, за рахунок особливостей порівняння значень у системах керування базами даних. Зв'язні списки дозволяють значно полегшити виконання операцій вставки й переміщення у простіших системах, проте ускладнюють пряму вибірку впорядкованих даних і додають накладні витрати на зберігання покажчиків. Практичний висновок полягає в тому, що для більшості завдань (списки завдань, прості черги) достатньо поля сортування з періодичною нормалізацією або розширеними кроками.

Перспективним напрямом подальшого дослідження є розгляд конкурентних сценаріїв роботи системи, коли декілька користувачів одночасно змінюють порядок. Тут можливі конфлікти оновлення (наприклад, двоє користувачів намагаються вставити елементи між одними й тими самими сусідніми). Потенційним рішенням можуть бути механізми оптимістичної локалізації оновлень (наприклад, системи контролю версій) або застосування CRDT/OT-підходів для списків (як це робиться у спільних редакторах). Також можна дослідити автоматизовані стратегії нормалізації, які запускаються при певних умовах (наприклад, коли сусідні ключі стають однаковими або дуже близькими).

Список бібліографічного опису

1. A. Silberstein et al. Efficient Maintenance of Order-Based Labeling for Dynamic XML Data. *International Conference on Data Engineering*. 2005. P. 285–296
2. Bender, M. A., Cole, R., Demaine, E. D., Farach-Colton, M., & Zito, J. Two Simplified Algorithms for Maintaining Order in a List. *European Symposium on Algorithms*. 2002. Vol. 2461. P. 152–164.
3. The Trello tech stack. *Atlassian*. URL: https://surl.li/ylnalm (дата звернення: 10.04.2025).
4. Floating-Point Arithmetic: Issues and Limitations. URL: https://surl.li/kfnkuw (дата звернення: 12.04.2025).
5. PostgreSQL: CREATE INDEX. URL: https://surl.li/ksseer (дата звернення: 02.05.2025).
6. The Stern-Brocot Tree and Farey Sequences. URL: https://surli.cc/sgkeil (дата звернення: 03.05.2025).

References

1. A. Silberstein et al. Efficient Maintenance of Order-Based Labeling for Dynamic XML Data. *International Conference on Data Engineering*. 2005. P. 285–296
2. Bender, M. A., Cole, R., Demaine, E. D., Farach-Colton, M., & Zito, J. Two Simplified Algorithms for Maintaining Order in a List. *European Symposium on Algorithms*. 2002. Vol. 2461. P. 152–164.
3. The Trello tech stack. *Atlassian*. URL: https://surl.li/ylnalm (date of access: 10.04.2025).
4. Floating-Point Arithmetic: Issues and Limitations. URL: https://surl.li/kfnkuw (date of access: 12.04.2025).
5. PostgreSQL: CREATE INDEX. URL: https://surl.li/ksseer (date of access: 02.05.2025).
6. The Stern-Brocot Tree and Farey Sequences. URL: https://surli.cc/sgkeil (date of access: 03.05.2025).