УДК 004.04, 004.65
DOI: 10.18523/2617-3808.2025.8.113-117

Зважій Д. В.

ОСОБЛИВОСТІ ІНДЕКСАЦІЇ У PostgreSQL

У статті наведено перелік основних типових індексів, реалізованих у PostgreSQL. Розглянуто можливості їх розширення та вдосконалення з урахуванням бізнес-завдань. Проаналізовано підхід Access Methods API. Описано методи життєвого циклу індексу в PostgreSQL. Також запропоновано інтерфейс для реалізації нового індексу в PostgreSQL на основі суфіксного дерева.

Ключові слова: бази даних, пошук рядків, PostgreSQL, суфіксні дерева, SP-GiST, GiST, хеш-таблиці, Б-дерева, Access Methods API.

Вступ

Індекси є одним із ключових підходів для оптимізації продуктивності в системах управління базами даних (СУБД). Завдяки використанню індексів можна значно пришвидшити пошук, сортування та доступ до даних, зменшуючи кількість операцій читання диска. Під час роботи з великими обсягами даних або використання складних запитів правильне використання індексів має критичне значення для забезпечення ефективної роботи програмних систем.

PostgreSQL, як одна з найбільш поширених СУБД із відкритим вихідним кодом, пропонує широкий набір стандартних типів індексів, які дають можливість вирішувати широкий спектр завдань. Проте специфічні випадки, такі як геопросторовий пошук, оброблення векторних моделей даних або оптимізація пошуку підрядків, можуть потребувати створення власних, спеціалізованих типів індексів.

Одна з визначних рис PostgreSQL полягає в можливості розширення його функціональності. Серед іншого, розробники можуть створювати власні типи індексів, інтегруючи їх у ядро системи через механізм Access Methods API [12]. Це дозволяє гнучко адаптувати базу даних під особливі вимоги застосунку без необхідності змінювати самі дані або логіку їхнього збереження.

У цій статті розглянуто:
– які стандартні індекси підтримує PostgreSQL;
– які можливості PostgreSQL надає для розширення наявних та реалізації власних індексів;
– які методи необхідні для створення власного індексу;
– запропонований набір інтерфейсних методів для індексу на базі суфіксного дерева.

Метою є надати практичне уявлення про те, як індексація в PostgreSQL може бути розширена відповідно до потреб конкретних проєктів, а саме для створення індексу на основі суфіксного дерева.

Вбудовані типи індексів у *PostgreSQL*

PostgreSQL підтримує кілька типів індексів, кожен із яких оптимізований під конкретні види запитів або структури даних. Вибір правильного типу індексу є вирішальним для досягнення високої продуктивності бази даних. Нижче наведено короткий огляд основних типів індексів, які доступні для негайного використання.

Б-дерево (B-tree) [14] є типовим індексом для PostgreSQL і використовується при створенні індексу без явного вказання типу індексу. Для PostgreSQL реалізовано збалансоване Б-дерево, що не має фіксованої кількості елементів у кожному вузлі [3]. Кількість елементів визначається динамічно залежно від того, скільки елементів можна записати у фіксований розмір сторінки PostgreSQL. За замовчуванням розмір сторінки у PostgreSQL дорівнює 8 КВ (8192 байти) [9]. Дизайн цієї структури даних дозво-

ляє підтримувати дерево збалансованим, що забезпечує ефективність операцій =, <, >, BETWEEN, ORDER BY. Цей індекс часто застосовують для індексації числових, рядкових полів, міток часу.

Хеш-таблиця є ще одним типом індексів, який підтримує PostgreSQL. Цей вид індексу підтримує лише оператор рівності. Результатом функції хешування у PostgreSQL є 32-бітове ціле число. Хеш індекс зберігає хеші у відсортованому вигляді, що дає змогу використовувати бінарний пошук для доступу до потрібного елементу [8]. У разі колізій елементи з однаковими хешами зберігаються разом із додатковими даними для їхньої подальшої ідентифікації. Цей тип індексу найкраще підходить для оптимізації операції точного збігу для великих обсягів даних.

GIN (Generalized Inverted Index) — узагальнений інвертований індекс — структура даних, що добре зарекомендувала себе в інформаційному пошуку. Цей індекс дає змогу ефективно працювати з композитними значеннями, які складаються з багатьох елементів. Зазвичай це текстові документи або масиви. Інвертований індекс дозволяє ефективно відповідати на запитання, чи містить документ той чи інший елемент. Узагальненість цієї структури полягає в тому, що її реалізація дозволяє працювати із різними типами даних, орієнтуючись на типові стратегії [6]. Також реалізація цього індексу передбачає розширення власними методами доступу, залежно від бізнес-завдань.

BRIN (Block Range Index) — індекс, що спроєктований для роботи з таблицями великого розміру, значення колонок яких прямо корелюють із їхнім розташуванням у таблиці [7], наприклад відсортовані колонки з датами. Такий індекс має невеликий розмір і не потребує значних зусиль для підтримки. Цей індекс належить до таких, що можуть повертати результати, які не відповідають умові запиту. У такому разі додаткова відповідальність покладається на рівень виконавця запиту: він мусить повторно перевіряти кортежі. Така концепція може бути виправданою, якщо індекс дає змогу уникнути повного перебору величезних таблиць.

GiST (Generalized Search Tree) — узагальнене дерево пошуку — це збалансована деревовидна структура, що слугує взірцем для створення власних індексних стратегій [4]. Вихідний код PostgreSQL також містить кілька готових реалізацій на основі цього індексу. Запропонована реалізація GiST дозволяє імплементувати Б-дерево, R-дерево та багато інших залежно від бізнес-завдання і заданого типу даних. Ядро цього індексу перебирає на себе управління внутрішньою комунікацією з базою даних. Сюди також відносяться вирішення задач багатопоточності, керування блокуваннями, логування та інші. Це дає можливість розробнику зосередитися на розробленні методів доступу, пов'язаних із семантикою запропонованого типу даних.

SP-GiST (Space-partitioned generalized search tree) [5; 10] — узагальнене дерево пошуку із просторовим розбиттям. Ідея цього індексу дуже схожа на GiST, проте основний акцент зроблено на підтримці незбалансованих дерев. До таких структур належать дерева квадрантів [15], k-вимірні дерева [11], префіксні дерева та їхні варіації і багато інших. Одна з важливих характеристик, що об'єднує ці структури даних, це те, що такі структури даних можуть мати різну глибину заглиблення для різних вузлів. SP-GiST також дозволяє створювати нові розширення залежно від завдань та обраного типу даних. Головна ідея цього підходу полягає в тому, що ядро індексу бере на себе взаємодію із PostgreSQL, підтримує виділення пам'яті, реалізовує реагування на блокування. Розробник власного розширення має можливість реалізувати певні функції зворотного виклику, які будуть викликані під час життєвого циклу індексу. Таким чином можна впливати на процес побудови дерева та його обходу для пошуку потрібних елементів у структурі.

Одним з обмежень у дизайні та реалізації SP-GIST індексу є те, що він надає доступ лише до певної частини дерева при вставці елементів. Тому розробник не має можливості додавати елементи, які пов'язані з індексованим кортежем, до інших частини дерева. Така можливість є дуже важливою при побудові суфіксного дерева [16]. Підхід до побудови суфіксного дерева значною мірою залежить від обраного алгоритму та початкових умов, проте жоден із можливих алгоритмів не дозволяє отримати необхідний результат, користуючись рушієм SP-GiST. Така реалізація потребуватиме великої кількості додаткового коду, який багато в чому дублюватиме наявний SP-GiST рушій.

У наступному розділі цієї статті буде запропоновано кроки для реалізації індексу на основі суфіксного дерева в межах можливостей, які доступні у PostgreSQL.

Інтерфейс взаємодії індексу на базі суфіксного дерева з PostgreSQL

Як уже було зазначено у попередньому розділі, архітектура PostgreSQL передбачає можливість для створення власної реалізації для індексування. Рушій бази даних не має знання про алгоритм

індексації чи використані структури даних. Натомість взаємодія відбувається через методи інтерфейсу доступу до індексу [13]. Такий підхід забезпечує незалежність між кодом, що реалізує логіку організації та зберігання сирих даних, і алгоритмом створення та підтримки індексу. На практиці це дає змогу розробнику зосередитися на деталях реалізації нового методу індексації, уникаючи певних особливостей роботи рушія з даними.

Під час ініціалізації індексу PostgreSQL очікує, що функція —обробник індексу у відповідь на аргумент типу internal повертатиме структуру типу IndexAmRoutine, що містить всю необхідну інформацію для роботи з індексом [1]. Найважливіша інформація про поля структури IndexAmRoutine міститься у табл. 1. Крім того, для індексу потрібно задати набір операторів, які він здатний підтримувати. Ця інформація є важливою для планувальника запитів під час формування стратегії виконання запиту.

Таблиця 1. **Основні поля структури IndexAmRoutine у PostgreSQL**

Поле	Короткий опис
ambuild	указник на метод для створення індексу
ambuildempty	указник на метод для побудови порожнього індексу
aminsert	указник на метод для вставки одного кортежу
ambeginscan	указник на метод ініціалізації сканування індексу
amgettuple	метод повертає один підхожий кортеж
amgetbitmap	указник на метод для повернення всіх кортежів, які відповідають критерію пошуку
amrescan	указник на метод для повторного сканування індексу
amendscan	указник на метод для завершення сканування
ambulkdelete / amvacuumcleanup	указник на метод для операції видалення елементів з індексу

У такому разі для підтримки індексу на базі суфіксного дерева обов'язково потрібно забезпечити реалізацію інтерфейсу функцій підтримки доступу. Під час аналізу вихідного коду PostgreSQL було помічено, що найчастіше для префікса методів інтерфейсу використовується ключове слово, яке згодом дає змогу ідентифікувати підхід, що застосовується. Відповідно, реалізації методів індексу на основі суфіксного дерева матимуть префікс "stree".

Варто почати з функції Datum streehandler(PG_FUNCTION_ARGS). Як уже зазначено вище, цей метод є необхідним для взаємодії з основним кодом PostgreSQL. Метод streehandler повертає структуру типу IndexAmRoutine, яка містить усю необхідну інформацію для життєвого циклу індексу.

Однією з обов'язкових частин цієї структури є метод IndexBuildResult * streebuild(Relation heapRelation, Relation indexRelation, IndexInfo *indexInfo). Цей метод відповідає за побудову всіх структур, необхідних для роботи індексу. Йому на вхід приходить два аргументи типу Relation. Аргумент з ім'ям heapRelation — це представлення таблиці, для якої створюється індекс, аргумент indexRelation — дає доступ до об'єкта, доступного для запису, де будуть зберігатися дані індексу на базі суфіксного дерева. Аргумент indexInfo містить необхідну інформацію про сам індекс. Його структура значною мірою перегукується зі структурою вищезгаданого об'єкту IndexAmRoutine. Результатом виконання функції є структура, яка містить статистику щодо кількості вже просканованих кортежів у таблиці, а також кількості кортежів, доданих до індексу.

Також необхідно визначити метод void streebuildempty(Relation indexRelation). Цей метод призначений для ініціалізації порожнього індексу на диску без сканування та вставки кортежів. Він використовується як частина процесу ініціалізації бази даних — ще до того, як будь-які дані були додані. Також цей метод може застосовуватися під час клонування бази даних.

Одним із наступних методів інтерфейсу є bool streeinsert(Relation indexRelation, Datum *values, bool *isnull, ItemPointer ht_ctid, Relation heapRel, IndexUniqueCheck checkUnique, bool indexUnchanged, IndexInfo *indexInfo). Цей метод відповідає за вставку кортежу в індекс. Усередині нього відбувається робота з пам'яттю PostgreSQL, обробка виняткових ситуацій, пов'язаних із блокуваннями, а також виклик основної логіки підтримки суфіксного дерева. На вхід метод приймає вже знайомий об'єкт для запису indexRelation. Аргумент values містить вказівник на рядок із даними. Аргумент isnull — це масив, розмірність якого дорівнює кількості елементів у values; він вказує, чи містить відповідна колонка значення NULL у рядку. Аргумент ht_ctid містить вказівник на ідентифікатор кортежу. На практиці такий підхід дає змогу PostgreSQL підтримувати концепцію MVCC (Multi-Version Concurrency Control), відповідно до якої PostgreSQL не модифікує наявні значення, а створює нові

записи — це дає можливість уникати блокування бази даних і гарантує актуальність інформації. Результатом виконання функції є булеве значення, яке є важливим у разі певних значень аргументу checkUnique. Позитивний результат означає, що вхідне значення завідомо унікальне, а негативний — що значення, передане на вхід, можливо, не є унікальним і потребує перевірки на унікальність. Зазвичай, якщо аргумент checkUnique не має значення, документація PostgreSQL рекомендує повертати негативний результат [2].

Ще одним важливим методом, який обов'язково потрібно реалізувати, є void streecostestimate (PlannerInfo *root, IndexPath *path, double loop_count, Cost *indexStartupCost, Cost *indexTotalCost, Selectivity *indexSelectivity, double *indexCorrelation, double *indexPages). Цей метод відіграє важливу роль для планувальника під час генерації стратегій виконання запитів. Завдання цього методу — надати оцінки, які відображають вартість виконання умови WHERE при використанні індексу. Оцінка враховує різні чинники: наскільки витратно розпочати використання індексу; які будуть загальні витрати; наскільки добре порядок, що задається індексом, співвідноситься з очікуваним порядком результату; а також скільки сторінок індексу, ймовірно, буде прочитано.

Що стосується сканування індексу, то виникає потреба реалізувати підтримку таких методів інтерфейсу:

– IndexScanDesc streebeginscan (Relation indexRelation, int nkeys, int norderbys);
– void streerescan (IndexScanDesc scan, ScanKey keys, int nkeys, ScanKey orderbys, int norderbys);
– bool streegettuple (IndexScanDesc scan, ScanDirection direction);
– int64 streegetbitmap (IndexScanDesc scan, TIDBitmap *tbm);
– void streeendscan (IndexScanDesc scan).

Усі ці методи беруть участь у процесі сканування індексу та пошуку результатів. На першому етапі викликається streebeginscan, який ініціалізує процес сканування та виділяє необхідні ресурси для підтримки його стану. Далі результат streebeginscan — об'єкт, що описує стан сканування, — передається в метод streerescan. У методі streerescan запускається або перезапускається процес сканування, а також задаються нові умови пошуку, наприклад: WHERE name LIKE '%query'. Після цього уже виконується streegettuple або streegetbitmap, де streegettuple повертає true, якщо кортеж був знайдений, та false, якщо ні. Метод streegetbitmap повертає число усіх результатів, що задовольняють умову, і також наповнює ідентифікаторами структуру типу TIDBitmap. Метод streeendscan завершує процес сканування та вивільняє усі використані для сканування ресурси.

Висновок

У статті було описано основні можливості індексації в PostgreSQL. Розглянуто готові індекси, які надає PostgreSQL, можливості їх розширення та ймовірні сценарії використання.

Основну увагу зосереджено на можливостях і інструментах для створення власних індексів, зокрема на Index Access Method Interface [13]. Було проаналізовано ключові методи інтерфейсу, які необхідно реалізувати для створення власного індексу. Також подано аналіз і запропоновано визначення основних методів інтерфейсу доступу для реалізації індексу на основі суфіксного дерева.

Список літератури

1. 62.1. Basic API Structure for Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/index-api.html.
2. 62.2. Index Access Method Functions. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/index-functions.html.
3. 64.1. B-Tree Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/btree.html.
4. 64.2. GiST Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/gist.html.
5. 64.3. SP-GiST Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/spgist.html.
6. 64.4. GIN Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/gin.html.
7. 64.5. BRIN Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/brin.html.
8. 64.6. Hash Indexes. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/hash-index.html.
9. 65.2. TOAST. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/storage-toast.html.
10. Aref W. G. SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees / W. G. Aref, I. F. Ilyas. — 2001. —https://doi.org/10.1023/A:1012809914301.
11. Bentley J. L. Multidimensional binary search trees used for associative searching / J. L. Bentley // Commun. ACM. — 1975. — Vol. 18 (9). — Pp. 509–517. — https://doi.org/10.1145/361002.361007.

12. Chapter 61. Table Access Method Interface Definition. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/tableam.html.
13. Chapter 62. Index Access Method Interface Definition. PostgreSQL Documentation [Electronic resource]. — Mode of access: https://www.postgresql.org/docs/17/indexam.html.
14. Comer D. Ubiquitous B-Tree / D. Comer // ACM Comput. Surv. — 1979. — Vol. 11 (2). — Pp. 121–137. — https://doi.org/10.1145/356770.356776.
15. Finkel R. A. Quad trees a data structure for retrieval on composite keys / R. A. Finkel, J. L. Bentley // Acta Informatica. — 1974. — Vol. 4 (1). — Pp. 1–9. — https://doi.org/10.1007/BF00288933.
16. Weiner P. Linear pattern matching algorithms / P. Weiner // 14th Annual Symposium on Switching and Automata Theory (Swat 1973). — 1973. — Pp. 1–11. — https://doi.org/10.1109/SWAT.1973.13.

References

62.1. Basic API Structure for Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/index-api.html.
62.2. Index Access Method Functions. PostgreSQL Documentation. https://www.postgresql.org/docs/17/index-functions.html.
64.1. B-Tree Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/btree.html.
64.2. GiST Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/gist.html.
64.3. SP-GiST Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/spgist.html.
64.4. GIN Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/gin.html.
64.5. BRIN Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/brin.html.
64.6. Hash Indexes. PostgreSQL Documentation. https://www.postgresql.org/docs/17/hash-index.html.
65.2. TOAST. PostgreSQL Documentation. https://www.postgresql.org/docs/17/storage-toast.html.
Aref, W. G., & Ilyas, I. F. (2001). *SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees*. https://doi.org/10.1023/A:1012809914301.
Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, *18* (9), 509–517. https://doi.org/10.1145/361002.361007.
Chapter 61. Table Access Method Interface Definition. (2025, February 20). PostgreSQL Documentation. https://www.postgresql.org/docs/17/tableam.html.
Chapter 62. Index Access Method Interface Definition. PostgreSQL Documentation. https://www.postgresql.org/docs/17/indexam.html.
Comer, D. (1979). Ubiquitous B-Tree. *ACM Comput. Surv.*, *11* (2), 121–137. https://doi.org/10.1145/356770.356776.
Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, *4* (1), 1–9. https://doi.org/10.1007/BF00288933.
Weiner, P. (1973). Linear pattern matching algorithms. *14th Annual Symposium on Switching and Automata Theory (Swat 1973)*, 1–11. https://doi.org/10.1109/SWAT.1973.13.

D. Zvazhii

INDEXING FEATURES IN PostgreSQL

This article provides an overview of the main standard index types implemented in PostgreSQL, emphasizing their internal structures, use cases, and efficiency characteristics. The study discusses the possibilities for extending and adapting these index types to meet specific business needs. Special attention is given to PostgreSQL's extensible architecture and the Access Methods API, which enables the creation of custom indexing solutions. The article analyzes the core functions that define the lifecycle of an index in PostgreSQL, including index creation, scan operations, maintenance, and cost estimation procedures. These functions not only structure the interaction between the planner and the index access methods but also open up opportunities for experimental or domain-specific extensions. As a case study of such extensibility, the article proposes an interface for a suffix tree index that utilizes the Access Methods API. The proposed suffix tree index serves as a demonstration of the flexibility provided by PostgreSQL's Access Methods API. The article details how the API's modular architecture allows for the integration of custom data structures, such as suffix trees, by implementing a well-defined set of callback functions governing index creation, scanning, insertion, and cost estimation. This case illustrates how the Access Methods API can be leveraged to expand PostgreSQL's indexing capabilities beyond traditional use cases, making it a powerful framework for experimental development and research-driven optimization of complex query patterns.

Keywords: databases, string search, PostgreSQL, suffix tree, SP-GiST, GiST, hash index, B-tree, Access Methods API.

Матеріал надійшов 30.05.2025