DOI 10.70286/ISU-04.02.2026.009

ВПЛИВ ІНДЕКСАЦІЇ НА ПРОДУКТИВНІСТЬ ЗАПИТІВ У ВЕЛИКИХ БАЗАХ ДАНИХ

Голубінка Віталій Петрович
аспірант
Кафедра інформаційних систем та мереж
НУ «Львівська політехніка», Україна

Оптимізатор запитів обирає план виконання, порівнюючи оцінки вартості альтернатив: повний перегляд таблиці, індексний пошук (index seek), індексне сканування (index scan), поєднання індексу з додатковими зверненнями до таблиці (lookup) тощо. Ключовим чинником є кардинальність - оцінка кількості рядків, що пройдуть фільтрацію.

Нехай N - кількість рядків у таблиці, s - селективність предикату (частка рядків, що задовольняють умову). Тоді очікувана кількість відібраних рядків дорівнює:

$$R=N \cdot s \quad (1)$$

Якщо R мала (висока селективність), індексний доступ зазвичай вигідніший. Якщо ж R велика, індекс може спричинити велику кількість випадкових читань, і оптимізатор віддасть перевагу послідовному скануванню.

Окремою проблемою є неточні статистики: при зміні розподілу даних або після масових вставок/оновлень оцінки s можуть бути хибними, що призводить до вибору невдалого плану.

B-tree (B+tree) індекси є базовими в більшості реляційних СУБД. Вони ефективні для:

- пошуку за рівністю (=),
- діапазонних умов (BETWEEN, >=, <=),
- запитів із ORDER BY за ключем індексу,
- приєднань (JOIN) за індексованим ключем.

Для великих БД важливо, що B-tree підтримує впорядкований доступ, тобто «перегляд діапазону» виконується послідовно всередині структури.

Hash-індекси (за наявності та підтримки в СУБД) можуть бути корисні для точного пошуку за рівністю. Їх типове обмеження - відсутність ефективної підтримки діапазонів і сортування, тому вони рідше є універсальним рішенням.

Композитні індекси ефективні для запитів, які фільтрують за кількома колонками, на- приклад:

WHERE status = ? AND created_at >= ? AND created_at < ?.

Порядок полів у ключі критичний. Узагальнено:

- першими доцільно ставити колонки, які найчастіше входять у фільтрацію та/або мають високу селективність;
- колонки для сортування (ORDER BY) варто враховувати, якщо це дозволить уникнути додаткового сортування;

• правила залежать від конкретної СУБД і того, як оптимізатор застосовує «ліва-префікс» (left-prefix) для композитних ключів.

Покривний індекс дозволяє виконати запит без звернення до таблиці (index-only). Це особливо важливо для великих таблиць, де lookup породжує багато випадкових читань. Практично покривні індекси застосовують для «гарячих» запитів з фіксованим набором колонок у SELECT.

У великих БД поширене патриціювання(наприклад, за датою). Воно не замінює індекси, але зменшує обсяг даних, які потрібно сканувати, за рахунок відсікання партицій (partition pruning). У поєднанні з індексами це часто дає найкращий ефект для часових діапазонів.

Кожен додатковий індекс:

• збільшує час INSERT/UPDATE/DELETE (потрібно підтримувати структуру індексу),

• збільшує обсяг журналювання та реплікації,

• ускладнює обслуговування (перебудова, дефрагментація, оновлення статистик),

• може погіршувати конкуренцію (конвенція на сторінках індексу) у пікових навантаженнях.

Тому доцільно контролювати «вартість індексу» через метрики запису та реальну частоту його використання в планах виконання.

Для кількісного підтвердження ефектів індексації пропонується порівнювати сценарії індексації на одному й тому самому наборі даних.

Рекомендовано використати таблицю подій/транзакцій із кількістю рядків $N \geq 10^7$ та полями: первинний ключ, часовий атрибут, статус/категорія, зовнішній ключ для JOIN, числові поля для агрегацій.

Типові запити для вимірювань:

• Q1: точковий пошук за ключем (WHERE id = ?);

• Q2: діапазон за часом з сортуванням (WHERE created_at BETWEEN ... ORDER BY created_at);

• Q3: селективний фільтр + діапазон (status + created_at);

• Q4: JOIN великої таблиці з довідником за ключем;

• Q5: агрегація (GROUP BY) на великому проміжку.

• S0: лише первинний ключ (baseline);

• S1: індекс на created_at (діапазонні запити);

• S2: композитний індекс (status, created_at);

• S3: покривний індекс під Q3 (додати колонки вибірки до індексу);

• S4: надлишкова індексація (для демонстрації росту вартості запису).

Доцільно збирати:

• час відповіді (середній та p95),

• кількість логічних читань (із буфера) та фізичних (диск),

• кількість звернень до таблиці (lookup),

• стабільність плану виконання (чи змінюється план між прогонами),

• накладні витрати на записи (час пакетного INSERT/UPDATE).

Щоб уникнути викривлень результатів, важливо окремо оцінювати «холодний» (cold cache) та «теплий» (warm cache) сценарії, а також фіксувати однакові параметри СУБД і однаковий розмір вибірки.

Таблиця 1. Порівняння сценаріїв індексації (приклад)

Запит	Сценарій	Середній час, мс	p95, мс	Логічні читання	Фізичні читання	Примітка
Q2 (діапазон часу)	S0	…	…	…	…	full scan
Q2 (діапазон часу)	S1	…	…	…	…	range scan
Q3 (status+date)	S2	…	…	…	…	composite
Q3 (status+date)	S3	…	…	…	…	covering

Очікується, що S1 суттєво зменшить час Q2 у випадках, коли діапазон вибірки невеликий відносно всієї таблиці (висока селективність по часу). Сценарій S2, як правило, дає помітний виграш для Q3, якщо status є достатньо селективним або в запиті фігурує стабільна комбінація status+date. Покривний індекс (S3) може додатково зменшити латентність за рахунок усунення lookup до таблиці.

Водночас S4 демонструє зворотний бік індексації: при великій кількості індексів зростає час пакетних вставок/оновлень, а також обсяг журналювання, що може стати критичним у високонавантажених системах.

1. Індексувати під реальні, найчастіші запити (топ- N за частотою та/або за часом виконання).
2. Для композитних індексів підбирати порядок полів під типові предикати та враховувати селективність і сортування.
3. Застосовувати покривні індекси для найбільш критичних read-запитів, щоб зменшити lookup і I/O.
4. Контролювати накладні витрати на запис: порівнювати час пакетних INSERT/UPDATE до/після додавання індексів.
5. Регулярно оновлювати статистики та відстежувати зміну планів виконання при зміні розподілу даних.
6. Для часових даних розглядати партиціювання як доповнення до індексів, особливо коли запити обмежені «вікном» часу.
7. Видаляти/консолідувати невикористовувані або дублюючі індекси, керуючись фактичним використанням у планах.

Індексація здатна суттєво скоротити час виконання запитів у великих БД завдяки зменшенню обсягу читань і кращому використанню кешу. Найбільший ефект зазвичай дають коректно підібрані B-tree індекси для рівностей/діапазонів та композитні індекси під найбільш типові комбінації предикатів. Покривні індекси додатково підвищують продуктивність, зменшуючи кількість звернень до таблиці, але мають вищу «ціну» при модифікаціях даних.

Отже, ефективна індексація вимагає системного підходу: аналізу профілю запитів, експериментального підтвердження на метриках I/O, а також постійного контролю накладних витрат на запис і експлуатаційних ризиків (фрагментація, застарілі статистики, нестабільні плани).

Список використаних джерел

1. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts (7th ed.). McGraw-Hill.
2. Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
3. Petrov, A. (2019). Database Internals: A Deep Dive into How Distributed Data Systems Work. O'Reilly Media.
4. Botros, S., & Tinley, J. (2021). High Performance MySQL: Proven Strategies for Operating at Scale (4th ed.). O'Reilly Media.
5. Pavlo, A., Angulo, G., Arulraj, J., Lin, H., Lin, J., Ma, L., Menon, P., Mowry, T. C., Perron, M., Quah, I., Santurkar, S., Tomasic, A., Toor, S., Van Aken, D., Wang, Z., Wu, Y., Xian, R., & Zhang, T. (2017). Self-driving database management systems. CIDR 2017. https://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf
6. Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018). The case for learned index structures. Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). https://doi.org/10.1145/3183713.3196909
7. PostgreSQL Global Development Group. (2023). Index types. PostgreSQL 16 Documentation. https://www.postgresql.org/docs/16/indexes-types.html

ВИКЛИКИ ТА ПЕРСПЕКТИВИ ВПРОВАДЖЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ У ВИЩІЙ ОСВІТІ

Сараєва Ольга Віталіївна
канд. іст. наук, доцент
Кафедра української філології та міжкультурної комунікації
Харківський національний університет міського
господарства імені О.М. Бекетова, Україна

Станом на 2025 рік генеративний штучний інтелект (GenAI) інтегрувався в архітектоніку навчального процесу закладів вищої освіти (ЗВО) України, ставши невід'ємним інструментом академічної діяльності. Однак, соціологічні дані (зокрема, дослідження КМІС) фіксують феномен «подвійної реальності»: попри широке фактичне використання технології, лише 26% респондентів відкрито декларують застосування ШІ. Така диспропорція свідчить про стигматизацію інструментарію та домінування страху перед адміністративним тиском. В умовах відсутності уніфікованого державного регулювання актуалізується потреба у дослідженні інституційних стратегій адаптації університетів до нових технологічних викликів.

Для дослідження регуляторного ландшафту було проведено моніторинг політик академічної доброчесності репрезентативної вибірки ЗВО, що охоплює заклади класичного, технічного, гуманітарного та спеціалізованого (безпекового) профілів.

Результати компаративного аналізу систематизовано в Таблиці 1.