SQLite Index Benchmarks

Звіт

Зміст

1. Вступ

Оптимізація запитів до баз даних є важливим аспектом побудови ефективних інформаційних систем, особливо при роботі з великими обсягами просторових даних. Одним із поширених підходів до підвищення швидкодії запитів є використання індексів, проте їх ефективність може суттєво змінюватися в залежності від структури запиту, обсягу даних та характеру вибірки.

Я провів експериментальне дослідження продуктивності SQL-запитів до бази даних SQLite з різними конфігураціями індексів та типами запитів (у тому числі з використанням операторів INTERSECT). Для аналізу були використані як **агреговані метрики продуктивності** (середній/загальний час виконання та кількість результатів), так і **неагреговані результати** окремих запусків, що дозволяє більш гнучко аналізувати поведінку запитів на різних підмножинах даних.

Мета роботи

Дослідити вплив індексації та формулювання SQL-запитів на ефективність вибірки просторових даних у базі SQLite, використовуючи як агреговані, так і неагреговані метрики продуктивності.

Постановка задачі

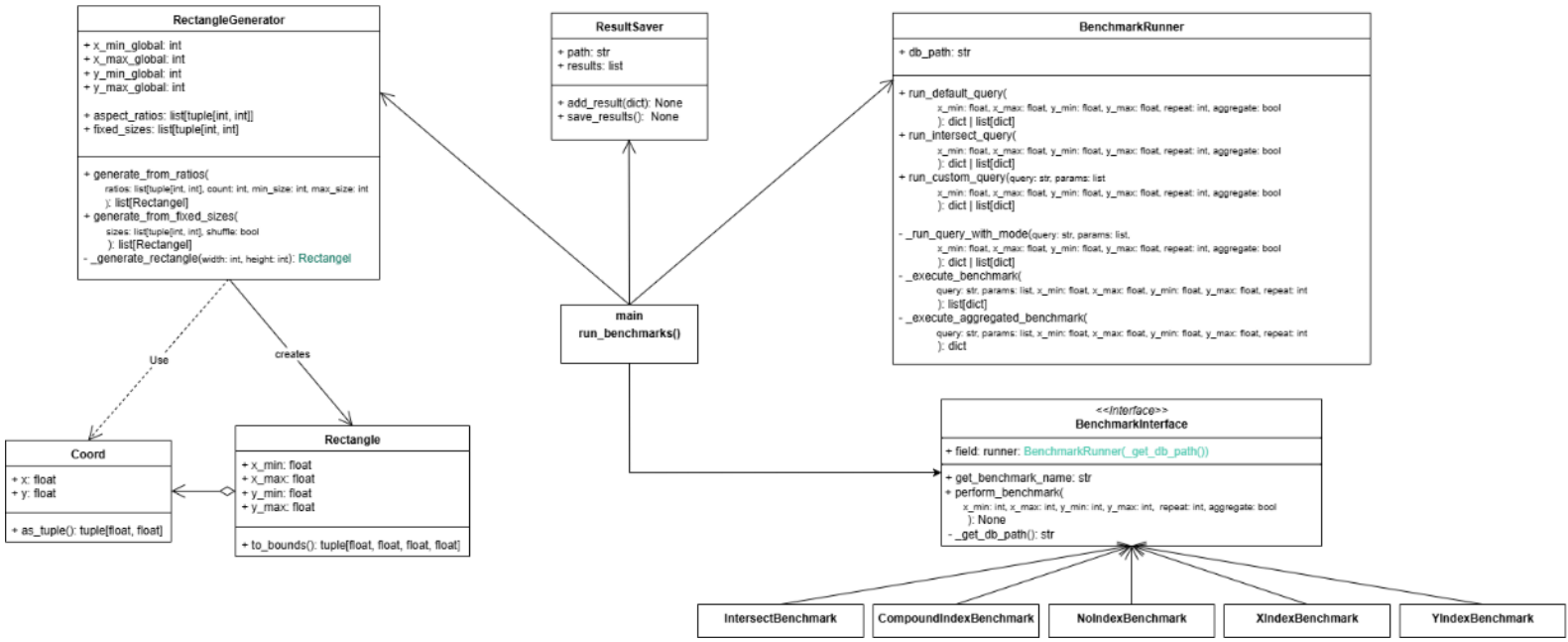
Для досягнення мети необхідно:

1. Реалізувати інструмент бенчмаркінгу SQL-запитів, що підтримує як збір агрегованих метрик (час виконання, кількість результатів), так і детальних (по кожному запуску).

2. Згенерувати прямокутні області вибірки різного розміру та з різними співвідношеннями сторін для імітації просторових запитів.
3. Реалізувати і протестувати запити за наступними сценаріями:
   - без індексації;
   - з індексом лише по x;
   - з індексом лише по y;
   - з комбінованим індексом x, y;
   - запит окремо по x, окремо по y, та INTERSECT цих запитів.
4. Зібрати результати виконання запитів на різних наборах прямокутників із заданим числом повторень (для стабілізації метрик).
5. Зберегти результати в структурованому вигляді у CSV-форматі.
6. Провести візуалізацію та порівняння результатів за ключовими метриками (час, кількість результатів, їх залежності від розміру області).
7. Зробити висновки щодо доцільності використання індексів та різних підходів до формулювання запитів для конкретного типу даних.

2. Програмне забезпечення

У межах дослідження було розроблено програмне забезпечення для генерації тестових прямокутників, запуску SQL-бенчмарків, збереження результатів та їх подальшої обробки. Архітектура системи реалізована в об'єктно-орієнтованому стилі з чітким розділенням відповідальностей між компонентами. На рисунку нижче представлено UML-діаграму класів, яка відображає взаємозв’язки основних модулів системи.



Ключові компоненти:

- RectangleGenerator — відповідає за генерацію прямокутників відповідно до заданих параметрів (фіксовані розміри або пропорції). Повертає об'єкти класу Rectangle.
- Rectangle — описує прямокутну область у вигляді координат x_min, x_max, y_min, y_max. Має зручний метод to_bounds() для передачі координат до запиту.
- Coord — допоміжний клас, що інкапсулює окрему координату (x, y) і використовується у генераторі для формування меж.
- BenchmarkRunner — клас для виконання SQL-запитів з вимірюванням часу. Підтримує запуск:
    1. стандартного запиту (run_default_query);
    2. запиту з INTERSECT (run_intersect_query);
    3. кастомного запиту (run_custom_query);

    Повертає як агреговану, так і неагреговану статистику (залежно від параметра aggregate).

- BenchmarkInterface — абстрактний інтерфейс для конкретних типів бенчмарків. Містить метод perform_benchmark() та обов’язкову реалізацію get_benchmark_name() і _get_db_path().
- Реалізації BenchmarkInterface:
    1. NoIndexBenchmark
    2. XIndexBenchmark
    3. YIndexBenchmark
    4. CompoundIndexBenchmark
    5. IntersectBenchmark
       Кожен з них вказує відповідний шлях до своєї бази даних з різними індексаційними стратегіями.
- ResultSaver — модуль для накопичення результатів тестів у пам’яті та подальшого експорту у формат CSV.
- main / run_benchmarks() — головний сценарій виконання, який координує генерацію областей, запуск бенчмарків і збереження результатів.

3. Підготовка до тестування

Перед запуском серії експериментів було виконано попередню підготовку, яка включала:

1. Створення варіантів бази даних

На основі оригінальної бази даних KievRegion.db було створено кілька її копій із різними індексаційними стратегіями. Це дало змогу протестувати вплив різних типів індексів на ефективність виконання запитів:

- **no_index.db** — база без індексів (референсна версія).
- **single_index_x.db** — індекс тільки по координаті x.
- **single_index_y.db** — індекс тільки по координаті y.
- **double_index.db** — два окремі індекси по x та y.
- **compound_index.db** — комбінований (складений) індекс по парі x, y.

Кожна база створюється копіюванням оригінальної БД та застосуванням відповідних SQL-інструкцій для створення індексів. При цьому видаляються попередні індекси, якщо вони існували (DROP INDEX ...).

```
shutil.copy(ORIGINAL_DB, target_path)
cursor.execute("CREATE INDEX IF NOT EXISTS idx_x ON SGMSTR(x);")
```

2. Визначення меж координат

Для коректного генератору тестових прямокутників була реалізована функція get_coordinate_bounds, яка зчитує мінімальні та максимальні значення координат x та y з таблиці sgmstr:

```
SELECT MIN(x), MAX(x), MIN(y), MAX(y) FROM sgmstr
```

Отримані координатні межі використовуються як глобальні границі при генерації випадкових прямокутних областей запитів. Це дозволяє гарантувати, що всі прямокутники будуть знаходитися в межах реальних географічних даних.

4. Результати тестування

У процесі тестування виконувались SQL-запити до баз даних з різними індексаційними стратегіями. Результати були збережені у двох групах:

1. Агреговані результати (aggregated)

Ці файли містять **зведену статистику** для кожного типу SQL-запитів, включаючи:

- середню кількість знайдених записів (row_count_avg);
- середній час виконання запиту (execution_time_avg_ms);
- загальну кількість запусків (repeat);
- межі координатного прямокутника, в межах якого здійснювався запит (x_min, x_max, y_min, y_max);
- сумарну кількість записів, отриманих за всі повторення (row_count_total);
- загальний час виконання всіх запитів у мілісекундах (execution_time_total_ms).

Файли:

- comp_index_result.csv
- intersect_result.csv
- no_index_result.csv
- x_index_result.csv
- y_index_result.csv

Ці результати використовуються для порівняльного аналізу ефективності індексації.

2. Індивідуальні результати (individual)

Ці файли зберігають **неагреговані** (поодинокі) значення часу виконання для кожного запиту окремо, що дозволяє:

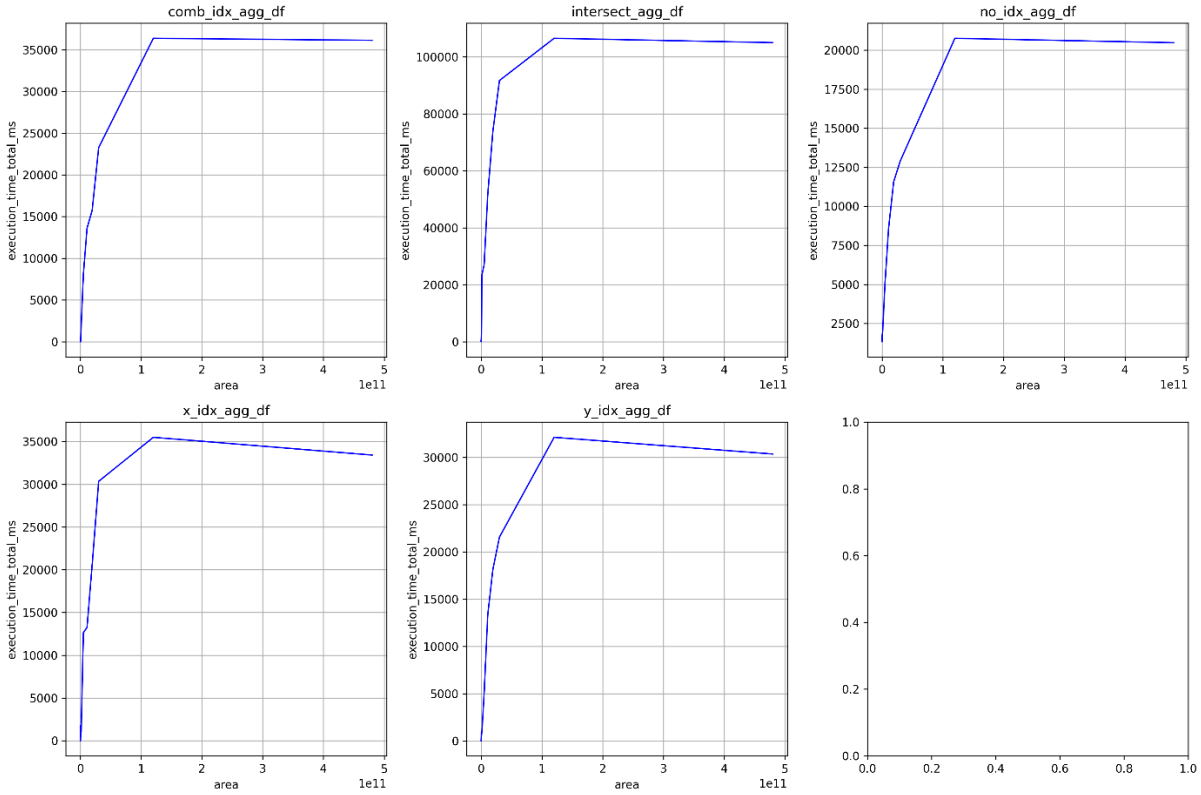
- будувати boxplot-графіки;
- аналізувати розподіл значень;
- виявляти аномалії чи варіації між виконаннями.

Формат запису містить координати меж запиту (x_min, x_max, y_min, y_max), час виконання (execution_time_ms) та кількість знайдених записів.

5. Аналіз результатів

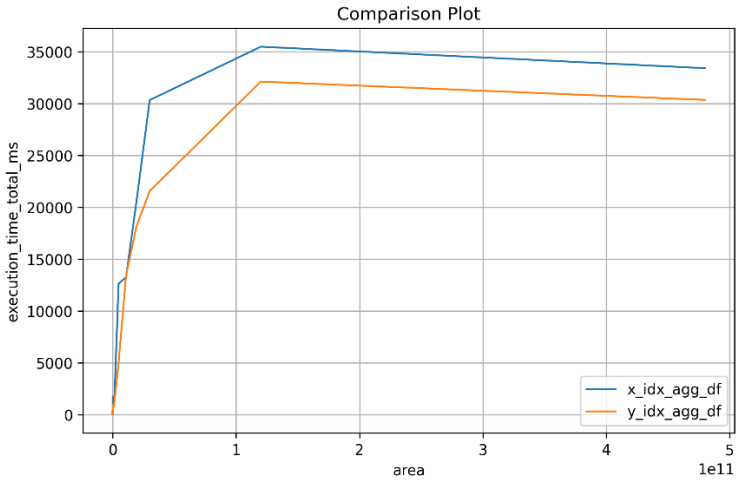
Аналіз часу виконання запитів

1. Залежність загального часу виконання від площі області пошуку для кожної індексаційної стратегії окремо.



Графіки показують, що при малій площі області, по якій ми дістаємо координати з БД час виконання запитів також не велика, проте зі збільшенням площі вона різко зростає, а вже після ста мільярдів час запиту стає максимальним та виходить на плато.

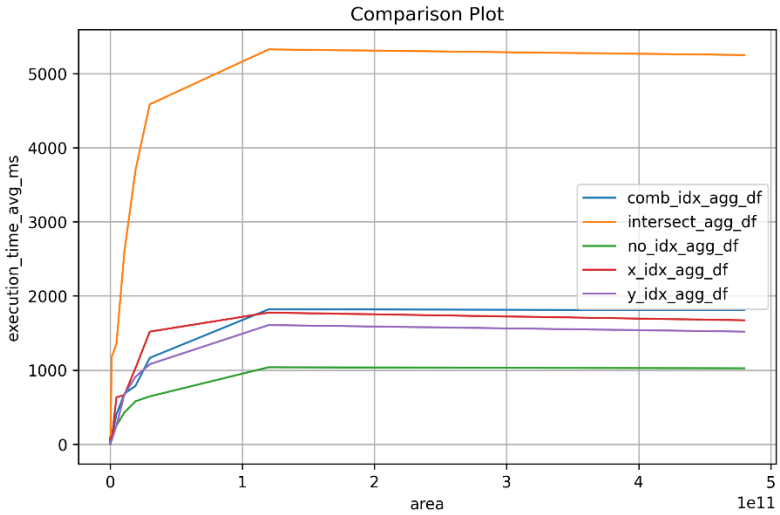
2. Порівняння X та Y індексів демонструє пряме порівняння ефективності одиночних індексів по координатам X та Y, що дозволяє оцінити вплив селективності по різних координатах.



Графік показує, що загалом індекс по Y є більш ефективним, у порівнянні з індексом по X. Можливо це пов’язано з тим, що Y має більший діапазон значень.

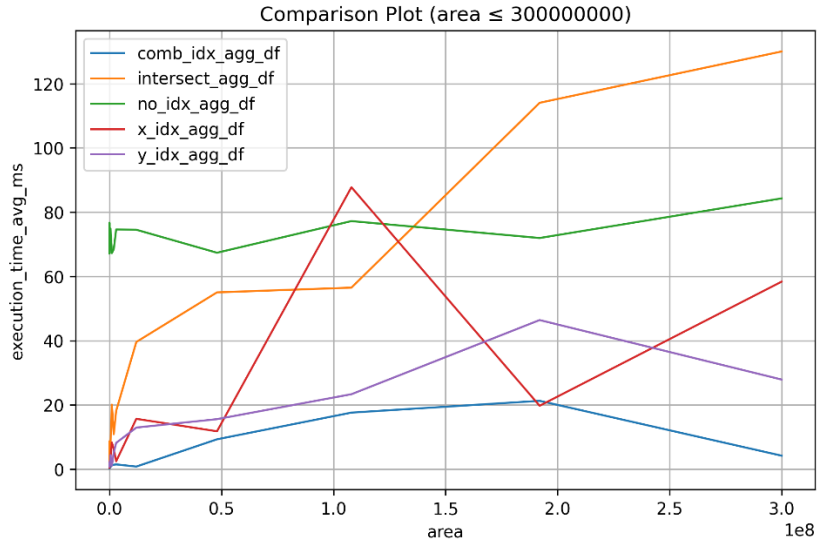
3. Порівняння за областями різних розмірів

Під час порівняння швидкості запитів залежно від площі виникла проблема, через швидке збільшення площі області при збільшенні розміру сторін, через що неможливо було порівняти швидкість запитів при невеликих областях, бо графік масштабувався на великі розміри:



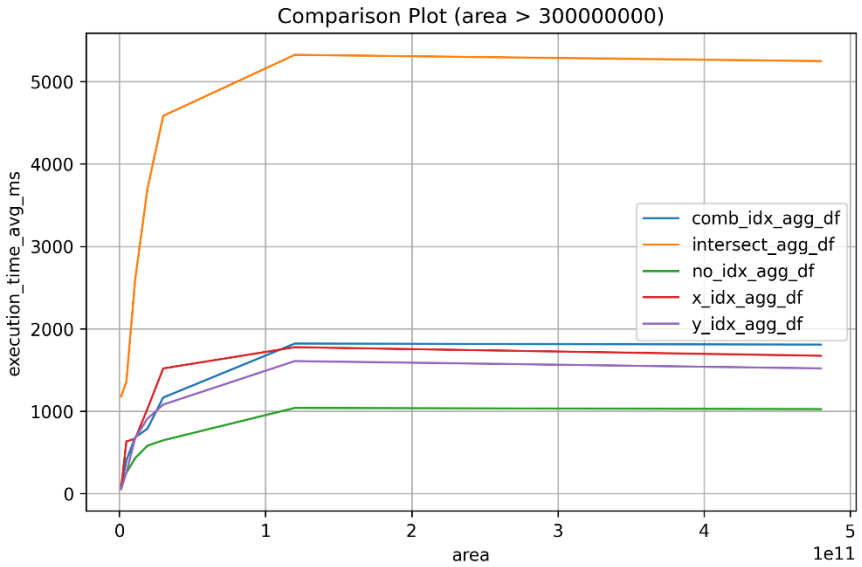
На перший погляд результати – невтішні, бо виходить так, що без індексів швидкість запитів найменша. Але чи дійсно це так? Щоб це перевірити було знайдено значення, при якому швидкість запитів без індексів починає перевищувати інші варіанти. Я обрав значення – 300 000 000

Запити для малих областей демонструють поведінку комбінованого індексу, INTERSECT-запитів та запитів без індексації на невеликих просторових запитах (площа < 300,000,000), де важливими є швидкість відповіді та мінімальні накладні витрати.



На цьому графіку ми бачимо, що комбінований індекс найкраще працює з-поміж інших і показує дуже хороші результати. Проте запити за допомогою методу INTERSECT, на який було покладено надії хоч і був швидшим ніж без індексів, але тільки коли площа області була меншаю за 130 000 000.

Запити для великих областей показують, як ті ж стратегії справляються з масштабними запитами (площа ≥ 300,000,000), де критичною стає ефективність обробки великих обсягів даних.

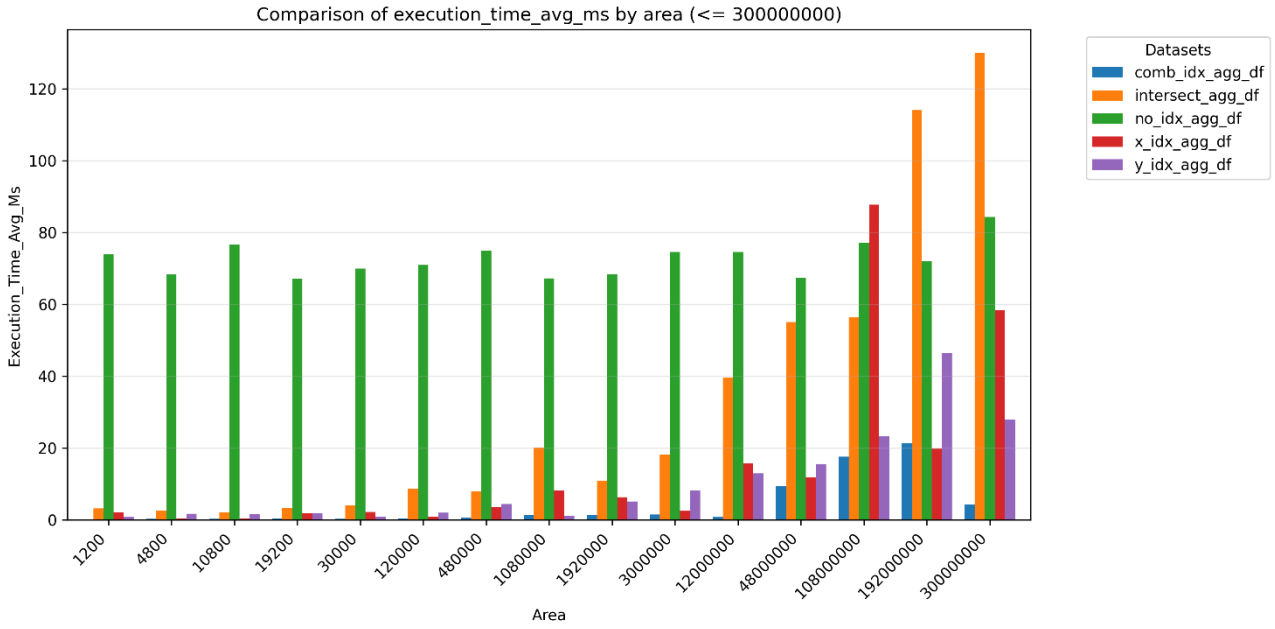


У випадку великих областей пошуку (коли площа більша за 300 000 000) будь-які індекси не дають нам перевагу і звичайний запит без індексації показує кращі результати.

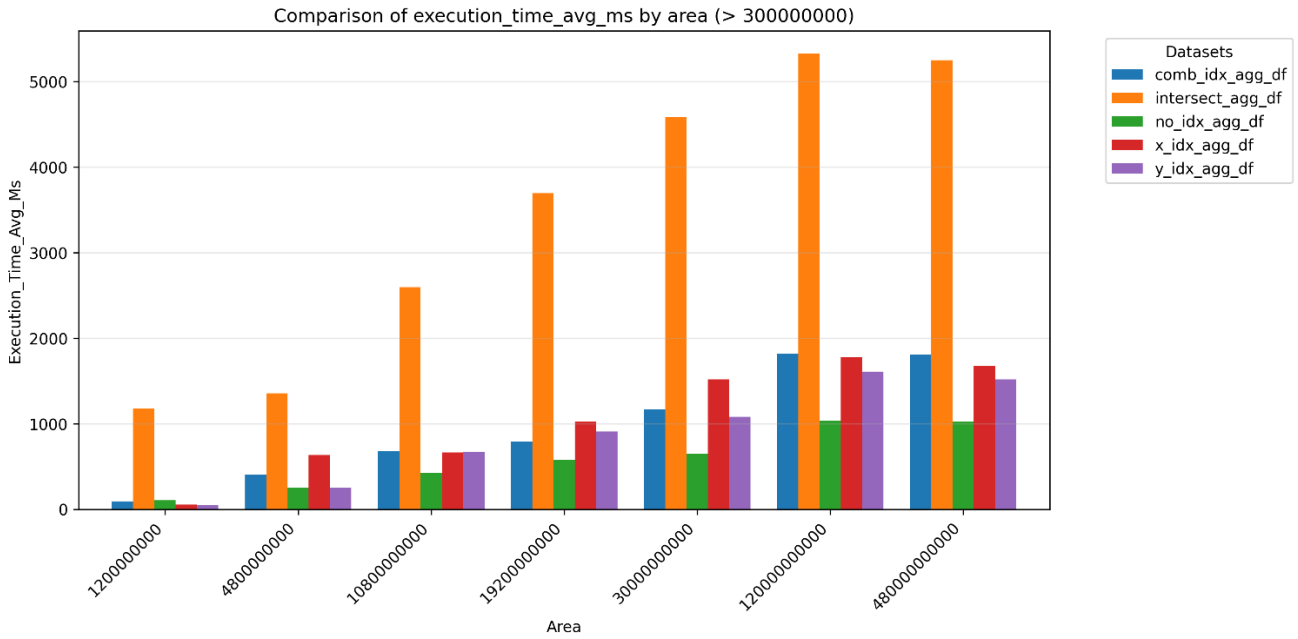
4. Стовпчасті діаграми порівняння

Порівняння для малих областей та великих областей у вигляді стовпчастих діаграм надають наочне порівняння середнього часу виконання для всіх індексаційних стратегій, згруповане за розміром області.

Малі розміри області:



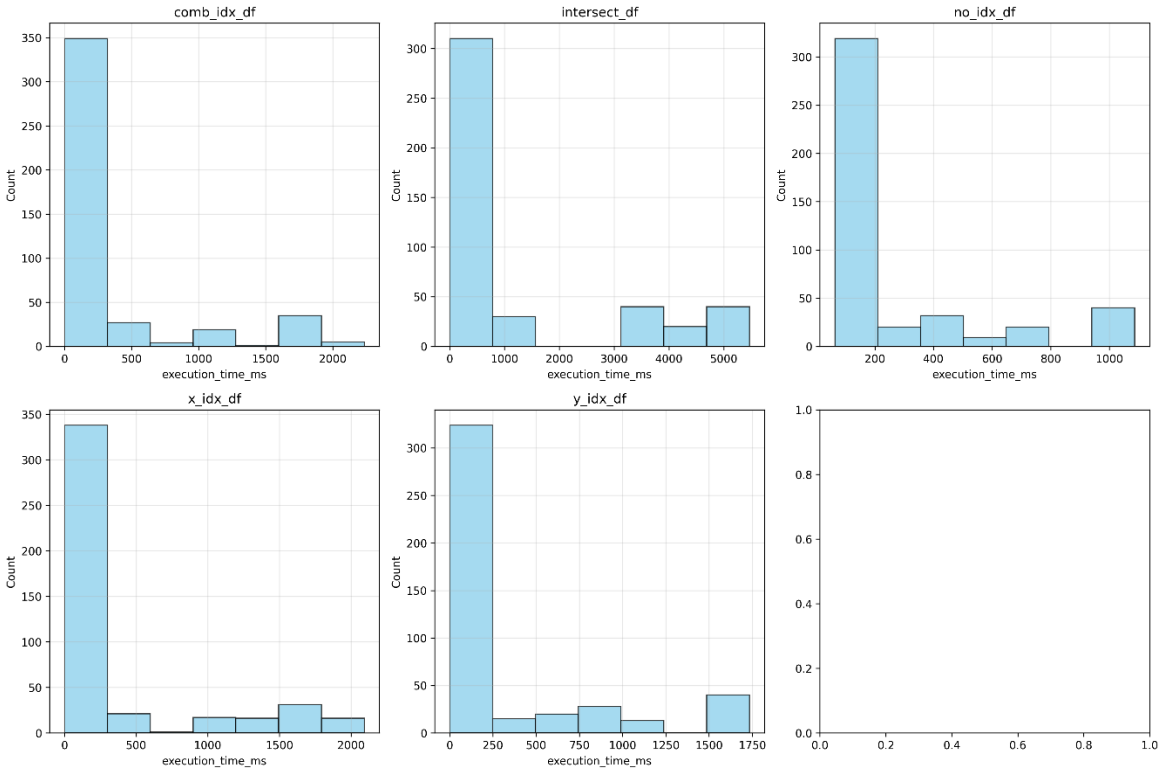
Великі розміри області:



Ці графіки тільки додатково підтверджують минулі спостереження. Видно, що при великих областях запити з INTERSECT взагалі не справляються.

Статистичний аналіз. Розподіл часових характеристик

Гістограми розподілу часу показують розподіл часу виконання.



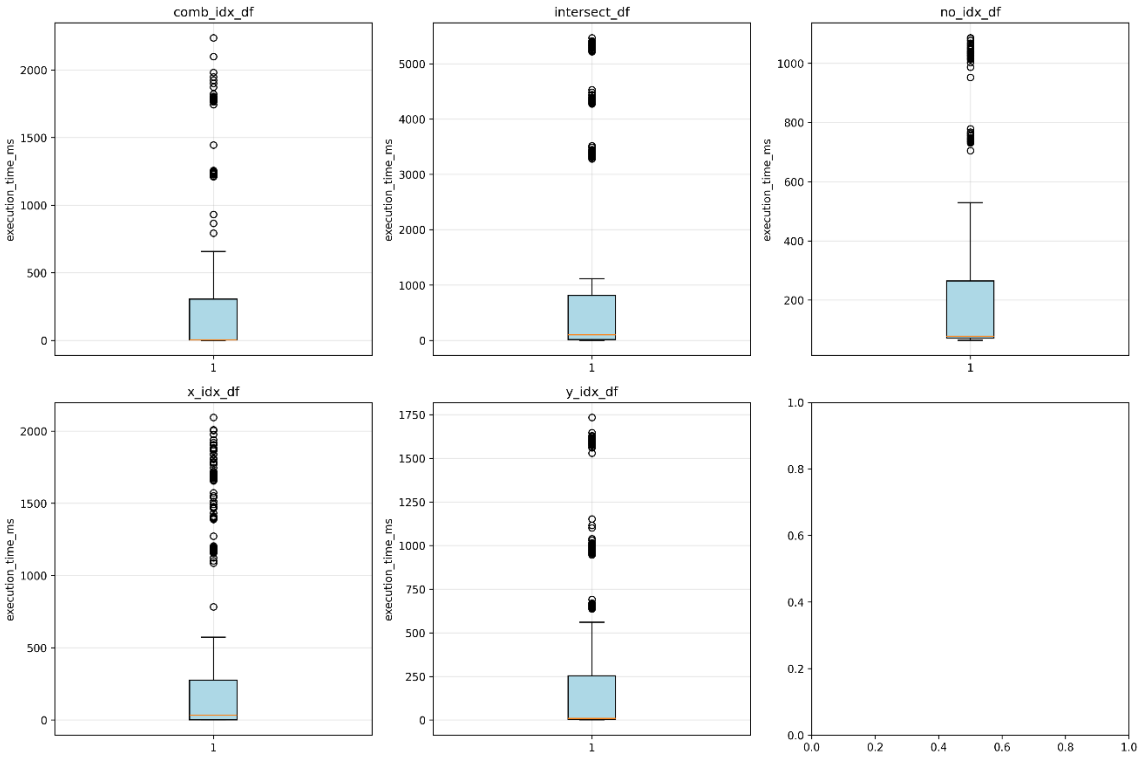
Представлені гістограми демонструють типовий експоненціальний розподіл часових характеристик виконання SQL-запитів, що є нормальним явищем для таких бенчмарків.

Основні спостереження:

- Переважна більшість запитів виконується швидко (0-500 мс)
- Невелика кількість запитів має значно більший час виконання
- Розподіл має характерну форму з "довгим хвостом"

Пояснення аномалій: Поодинокі викиди з високим часом виконання (1000+ мс) пов'язані зі значним збільшенням площі найбільших прямокутних областей порівняно з меншими. Коли область вибірки охоплює набагато більше записів, час обробки запиту природно зростає пропорційно кількості даних, що обробляються.

Box-plot аналіз надає статистичну інформацію про медіанні значення, квартилі та викиди для кожної індексаційної стратегії, що дозволяє оцінити їх надійність та передбачуваність роботи.



Box-plot діаграми підтверджують попередні спостереження щодо розподілу часових характеристик та аномалій:

Викиди (outliers): Численні точки вище верхньої межі ящика легко пояснюються, вони представляють окремі розміри областей - найбільші прямокутні області вибірки, які природно вимагають більше часу для обробки через збільшену кількість записів.

6. Висновки

За результатами проведеного експериментального дослідження продуктивності SQL-запитів до бази даних SQLite з різними конфігураціями індексів було отримано наступні висновки:

1. Вплив розміру області пошуку

Критична точка ефективності індексації знаходиться приблизно на рівні площі області 300,000,000 одиниць. Це значення є межею, після якої поведінка індексів кардинально змінюється:

- Для малих областей (< 300 000 000): індекси забезпечують значне покращення продуктивності
- Для великих областей (>= 300 000 000): індекси втрачають ефективність, а запити без індексації показують кращі результати

2. Ефективність різних типів індексів

Для малих областей:

1. **Комбінований індекс (x, y)** - найкращий результат за всіма показниками
2. **Індекс по Y** - більш ефективний, ніж індекс по X (можливо, через більший діапазон значень Y)
3. **Індекс по X** - менш ефективний, ніж по Y
4. **INTERSECT-запити** - показують помірну ефективність тільки при площі < 130,000,000
5. **Без індексації** - найгірший результат

Для великих областей:

Відсутність індексації виявилася найефективнішою стратегією, оскільки накладні витрати на обслуговування індексів перевищують їх переваги при обробці великих обсягів даних.

INTERSECT-запити виявились надзвичайно неефективними при великих областях пошуку.

3. Підсумки

Результати дослідження підтверджують, що **не існує універсальної стратегії індексації**, яка була б оптимальною для всіх типів просторових запитів. Ефективність індексації критично залежить від розміру області пошуку, і для досягнення максимальної продуктивності необхідно використовувати адаптивний підхід, який враховує специфіку конкретних запитів системи.