ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №5

Тема: «Індекси в SQL»

Виконав:

Ст. *Лук'янчук Денис*

Група ПМІ-23

2025

Тема: «Індекси в SQL».

Мета роботи: Вивчення створення і використання запитів мови SQL.

Завдання лабораторної роботи:

1. Опрацювати теоретичний матеріал.

2. Створити вказані індекси до вибраних таблиць, попередньо додавши, по можливості, якомога більшу кількість кортежів – індекс для одного стовпчика, складений індекс, унікальний індекс, індекс за виразами, частковий індекс.

3. Підібрати запити для демонстрації використання індексів, застосовуючи контроль використання індексів і продемонструвати це відповідними знімками екрану.

Хід роботи

1. Опрацювання теоретичного матеріалу

Ознайомлено з теоретичним матеріалом щодо індексів у SQL. Вивчено їх типи (B-дерева, унікальні, часткові), принципи створення та використання для прискорення запитів (SELECT, UPDATE, DELETE, JOIN). Розглянуто витрати на оновлення індексів, а також команди EXPLAIN і EXPLAIN ANALYZE для аналізу планів виконання запитів. Додатково опрацьовано вплив індексів на сортування (ORDER BY), об'єднання таблиць і фільтрацію.

2. Створення індексів до таблиць

Для виконання завдання використано базу даних PartsTradeDB із таблицями Mechanism і Part. Щоб збільшити обсяг даних, додано додаткові записи до таблиці Part (загалом 20 запчастин), залишивши таблицю Mechanism із 5 механізмами.

3. Створення індексів:

- **Індекс для одного стовпця: Для прискорення фільтрації за ціною.**
- **Складений (багатостовпчиковий) індекс**: Для запитів, які фільтрують за виробником і кількістю на складі.
- **Унікальний індекс**: Уже створений у таблиці Part для Serial_Number (UNIQUE обмеження).
- **Індекс за виразом**: Для пошуку за назвою запчастини у нижньому регістрі.
- **Частковий індекс**: Для запчастин із ціною понад 100.

```sql
-- Index for a single column (Price)
CREATE INDEX idx_part_price ON Part (Price);

-- Composite index (ID_Manufacturer, Stock_Quantity)
CREATE INDEX idx_part_manufacturer_stock ON Part (ID_Manufacturer, Stock_Quantity);

-- Unique index (Serial_Number, already enforced by UNIQUE constraint, but explicitly created)
CREATE UNIQUE INDEX idx_part_serial_number ON Part (Serial_Number);

-- Index on expression (LOWER(Name))
CREATE INDEX idx_part_name_lower ON Part (LOWER(Name));

-- Partial index (Price > 100)
CREATE INDEX idx_part_high_price ON Part (Price) WHERE Price > 100;
```

4. Підбір запитів для демонстрації використання індексів:

```sql
EXPLAIN ANALYZE
SELECT
    p.Name AS Назва_Запчастини,
    p.Serial_Number AS Серійний_Номер,
    p.Price AS Ціна,
    m.Name AS Назва_Механізму
FROM Part p
JOIN Mechanism m ON m.ID_Mechanism = ANY(p.Mechanism_IDs)
WHERE m.ID_Mechanism = 1
ORDER BY p.Price DESC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=2.43..2.44 rows=1 width=128) (actual time=0.149..0.151 rows=2 loops=1)
2	Sort Key: p.price DESC
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=0.00..2.42 rows=1 width=128) (actual time=0.100..0.113 rows=2 loops=1)
5	Join Filter: (m.id_mechanism = ANY (p.mechanism_ids))
6	Rows Removed by Join Filter: 9
7	-> Seq Scan on mechanism m (cost=0.00..1.06 rows=1 width=36) (actual time=0.061..0.067 rows=1 l...
8	Filter: (id_mechanism = 1)
9	Rows Removed by Filter: 32
10	-> Seq Scan on part p (cost=0.00..1.11 rows=11 width=128) (actual time=0.022..0.026 rows=11 loops...
11	Planning Time: 0.292 ms
12	Execution Time: 0.223 ms

```sql
EXPLAIN ANALYZE
SELECT
    m.Name AS Назва_Механізму,
    COUNT(p.ID_Part) AS Кількість_Запчастин
FROM Mechanism m
LEFT JOIN Part p ON m.ID_Mechanism = ANY(p.Mechanism_IDs)
WHERE p.ID_Manufacturer = 1
GROUP BY m.Name
ORDER BY Кількість_Запчастин DESC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=2.34..2.34 rows=1 width=40) (actual time=0.147..0.148 rows=2 loops=1)
2	Sort Key: (count(p.id_part)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> GroupAggregate (cost=2.31..2.33 rows=1 width=40) (actual time=0.136..0.140 rows=2 loops=1)
5	Group Key: m.name
6	-> Sort (cost=2.31..2.31 rows=1 width=36) (actual time=0.126..0.128 rows=2 loops=1)
7	Sort Key: m.name
8	Sort Method: quicksort Memory: 25kB
9	-> Nested Loop (cost=0.00..2.30 rows=1 width=36) (actual time=0.064..0.105 rows=2 loops=1)
10	Join Filter: (m.id_mechanism = ANY (p.mechanism_ids))
11	Rows Removed by Join Filter: 64
12	-> Seq Scan on part p (cost=0.00..1.14 rows=1 width=36) (actual time=0.043..0.046 rows=2 loop...
13	Filter: (id_manufacturer = 1)
14	Rows Removed by Filter: 9
15	-> Seq Scan on mechanism m (cost=0.00..1.05 rows=5 width=36) (actual time=0.008..0.013 row...
16	Planning Time: 0.348 ms
17	Execution Time: 0.226 ms

```sql
EXPLAIN ANALYZE
WITH MechanismCounts AS (
    SELECT
        m.Name AS Назва_Механізму,
        COUNT(p.ID_Part) AS Кількість_Запчастин,
        RANK() OVER (ORDER BY COUNT(p.ID_Part) DESC) AS Ранг
    FROM Mechanism m
    LEFT JOIN Part p ON m.ID_Mechanism = ANY(p.Mechanism_IDs)
    WHERE p.Price > 100
    GROUP BY m.Name
)
SELECT
    Назва_Механізму,
    Кількість_Запчастин
FROM MechanismCounts
WHERE Ранг = 1
ORDER BY Назва_Механізму;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=2.78..2.78 rows=1 width=40) (actual time=0.308..0.311 rows=3 loops=1)
2	Sort Key: mechanismcounts."Назва_Механізму"
3	Sort Method: quicksort Memory: 25kB
4	-> Subquery Scan on mechanismcounts (cost=2.74..2.77 rows=1 width=40) (actual time=0.281.....
5	Filter: (mechanismcounts."Ранг" = 1)
6	-> WindowAgg (cost=2.74..2.76 rows=1 width=48) (actual time=0.279..0.290 rows=3 loops=1)
7	Run Condition: (rank() OVER (?) <= 1)
8	-> Sort (cost=2.74..2.74 rows=1 width=40) (actual time=0.270..0.272 rows=4 loops=1)
9	Sort Key: (count(p.id_part)) DESC
10	Sort Method: quicksort Memory: 25kB
11	-> GroupAggregate (cost=2.71..2.73 rows=1 width=40) (actual time=0.245..0.255 rows...

Висновок: Лабораторна робота дозволила ознайомитися з індексами в SQL на прикладі бази даних PartsTradeDB. Створено індекси різних типів (одностовпчиковий, складений, унікальний, за виразом, частковий) для таблиці Part. За допомогою EXPLAIN ANALYZE підтверджено, що індекси прискорюють виконання запитів: idx_part_price оптимізував сортування, idx_part_manufacturer_stock — фільтрацію за кількома стовпцями, а idx_part_high_price — часткову фільтрацію за ціною. Результати та аналіз додано до звіту.

Відповіді на контрольні питання

1. Яка перевага використання індексів?

Індекси значно прискорюють виконання запитів (`SELECT`, `UPDATE`, `DELETE`, `JOIN`), зменшуючи кількість даних, які потрібно сканувати. Вони дозволяють швидко знайти потрібні рядки, уникаючи повного сканування таблиці, що особливо ефективно для великих таблиць із вибірковими умовами.

2. Типи індексів?

-B-дерево (B-tree): Найпоширеніший тип, ефективний для порівнянь (`=`, `<`, `>`).
- Хеш (Hash): Оптимальний для точних порівнянь (`=`), але не підтримує діапазони.
- GiST (Generalized Search Tree): Для складних типів даних (геометричних, текстових).
- GIN (Generalized Inverted Index): Для масивів, JSON, повнотекстового пошуку.
- BRIN (Block Range Index): Для великих таблиць із упорядкованими даними (наприклад, за датою).
- Складені (Composite): На кількох стовпцях.
- Унікальні (Unique): Забезпечують унікальність значень.
- Часткові (Partial): Для підмножини даних (з умовою).
- Індекси за виразами: Для обчислених значень (наприклад, `LOWER(column)`).

3. Які індекси використовуються для sequential file?

Для послідовних файлів (sequential files) зазвичай використовуються **B-дерева** або **хеш-індекси**, якщо структура даних підтримує швидкий доступ за ключем. Однак послідовні файли часто не індексуються, оскільки вони призначені для послідовного читання, а індекси більше підходять для прямого доступу. Якщо індексація потрібна, B-дерево є стандартним вибором через його універсальність.

4. У якому випадку можна вважати, що індекс вдалий?

Індекс вважається вдалим, якщо:

- Він значно прискорює виконання запитів (перевірка через `EXPLAIN ANALYZE` показує використання індексу замість повного сканування).
- Запити, для яких створено індекс, виконуються часто.
- Витрати на оновлення індексу (при `INSERT`, `UPDATE`, `DELETE`) не перевищують вигоду від прискорення запитів.
- Індекс відповідає умовам фільтрації, сортування чи об’єднання в запитах.

5. Багаторівневі індекси?

Багаторівневі індекси (multilevel indexes) — це структури, які використовують кілька рівнів для організації даних, наприклад, B-дерева або B+-дерева. Вони складаються з:

- Листових вузлів (з посиланнями на дані).
- Проміжних вузлів (для навігації).
- Кореня (вхідна точка).

Така структура дозволяє ефективно шукати дані, зменшуючи кількість операцій доступу до диска, що особливо корисно для великих таблиць.

6. Які структури індексів застосовують до багатовимірних даних?

Для багатовимірних даних (наприклад, геометричних, просторових) використовують:

- R-дерева (R-tree): Для просторових даних (координати, прямокутники).
- GiST (Generalized Search Tree): У PostgreSQL для багатовимірних даних, таких як геометрія чи повнотекстовий пошук.
- KD-дерева (K-dimensional tree): Для багатовимірного пошуку (хоча в PostgreSQL не підтримується напряму).
- Quad-дерева (Quad-tree): Для двовимірних даних (наприклад, карти).

У PostgreSQL найчастіше застосовують GiST або R-дерева через їх підтримку в ядрі СУБД.