Бази даних

Лекція 8

Тематика лекції

- Індекси

Індекс

Індекси це інструмент для покращення швидкодії SELECT запитів.

Індекс - окрема структура даних, що зберігає копії проіндексованих даних у організованій формі та містить посилання на відповідні повні рядки оригінальної таблиці.

Типи індексів

- **B-Tree index - найбільш розповсюджений тип індексів. Використовується для порівняння ідентичності (=), порівняльних запитів (>, <, BETWEEN), сортування, пошуку за префіксом.**
- **Hash index - використовується лише для порівняння ідентичності (=), проте швидший ніж B-Tree.**
- **Вузькоспеціалізовані типи (GiST, GIN, etc).**

Створення індекса

```
CREATE INDEX idx_active_courses_by_name
ON course(name)
WHERE is_active = true;
```

Часткові індекси

- **Часткові індекси індексують лише частину рядочків таблиці - лише ті рядочки, які задовольняють предикат у WHERE частині індексу.**
- **Часткові індекси споживають менше пам’яті.**
- **Трохи більш ефективні ніж повні індекси по швидкодії.**

Часткові індекси

```
CREATE INDEX idx_active_courses_by_name
ON course(name)
WHERE is_active = true;
```

```
SELECT * FROM course WHERE name = ‘Бази даних’;  -- ← запит НЕ використає індекс
```

```
SELECT * FROM course WHERE name = ‘Бази даних’ AND is_active = true; -- ← використає
```

Композитні індекси

CREATE INDEX idx_some_name ON some_table(col1, col2, col3);

Даний індекс ефективний для запитів:

SELECT * FROM some_table WHERE col1 = ?
SELECT * FROM some_table WHERE col1 = ? AND col2 = ?
SELECT * FROM some_table WHERE col1 = ? AND col2 = ? AND col3 = ?

І НЕ ефективний для запитів:

SELECT * FROM some_table WHERE col2 = ?
SELECT * FROM some_table WHERE col3 = ?

Композитні індекси

Оптимальний порядок колонок в композитному індексі:

- **Колонки, по яким йде пошук за ідентичністю попереду.**
- **Колонки, по яким йде пошук за порівняннями останні.**
- **Колонки з більшою кардинальністю попереду.**

Переваги індексів

Підвищення швидкодії запитів для таблиць, що містять багато даних.

Недоліки індексів

- **Споживають доволі багато пом’яті.**
- **Сповільнюють записи нових рядків в таблицю.**
- **В певних випадках сповільнюють оновлення та видалення рядків з таблиці.**

Коли індекси шкідливі

- **Коли відбувається багато записів в табличку (wright-heavy workload).**
- **Коли багато оновлень проіндексованих колонок.**
- **Коли кардинальність проіндексованої колонки низька - зазвичай планувальник PostgreSQL сам вирішує не використовувати індекс в запитах у такому випадку.**

Write-Heavy Workload

При записі відбувається:

1. **Запис даних в табличку**
2. **Оновлення індексу первинного ключа (PRIMARY KEY index)**
3. **Оновлення всіх інших індексів**

Таким чином, кожен INSERT перетворюється у 3+ записи на диск, якщо на табличці є хоч один індекс (окрім PK).

Оновлення проіндексованої колонки

При оновленні проіндексованої колонки в PostgreSQL відбувається:

1. **Створюється нова версія рядочку (механізм MVCC).**
2. **В індексі попередня версія рядочку помічається мертвою (dead tuple)**
3. **В індекс вставляється нова версія**

Таким чином, кожен UPDATE перетворюється у 3+ записи на диск.
Також відбувається роздування індексу (index bloat) через те, що стара версія значення оновленої колонки з індексу не видаляється, а лише помічається видаленою. З index bloat Postgres справляється у фоновому режимі за домогогою AUTOVACUUM процесу.

Колонки низької кардинальності

Якщо використовується індекс для колонки з низькою кардинальністю:

1. **В індексі знаходяться всі посилання на дані за фільтром.**
2. **Для кожного посилання відбувається читання даних з таблиці (random I/O).**

Якщо НЕ використовується індекс для колонки з низькою кардинальністю:

1. **Відбувається послідовне читання всіх даних з таблички (sequential I/O).**
2. **Фільтрація відбувається підчас читання.**

Sequential I/O завжди швидше ніж random I/O і якщо індекс зменшує об’єм даних, які читаються з таблички менш ніж в ~8-10 разів - Seq Scan таблички ефективніший.

Як зрозуміти чому запит повільний

EXPLAIN SELECT * FROM course WHERE name = ‘Бази даних’;

EXPLAIN ANALYZE SELECT * FROM course WHERE name = ‘Бази даних’;

План запиту

```
6  EXPLAIN ANALYZE
7  select c.name, pc.name as prerequisite from course c
8  left join course_prerequisite p on c.course_id = p.course_id
9  left join course pc ON p.prerequisite_course_id = pc.course_id;
10
```

Data Output | Messages | Notifications

SQL

	QUERY PLAN text
1	Hash Left Join (cost=2.54..47.25 rows=2260 width=62) (actual time=1.253..1.272 rows=13 loops=1)
2	Hash Cond: (p.prerequisite_course_id = pc.course_id)
3	-> Hash Right Join (cost=1.27..39.92 rows=2260 width=35) (actual time=0.993..1.010 rows=13 loops=1)
4	Hash Cond: (p.course_id = c.course_id)
5	-> Seq Scan on course_prerequisite p (cost=0.00..32.60 rows=2260 width=8) (actual time=0.560..0.568 rows=7 loo...
6	-> Hash (cost=1.12..1.12 rows=12 width=35) (actual time=0.066..0.067 rows=12 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on course c (cost=0.00..1.12 rows=12 width=35) (actual time=0.036..0.038 rows=12 loops=1)
9	-> Hash (cost=1.12..1.12 rows=12 width=35) (actual time=0.020..0.020 rows=12 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on course pc (cost=0.00..1.12 rows=12 width=35) (actual time=0.007..0.008 rows=12 loops=1)
12	Planning Time: 2.949 ms
13	Execution Time: 1.727 ms

Як зрозуміти план запиту

- **Seq Scan - читання всіх даних в табличці. Це швидко для невеличких таблиць і дуже повільно для великих.**
- **Index Scan - пошук по індексу та читання рядків з основної таблички, на які вказує індекс.**
- **Index Only Scan - читання лише з індексу. Це найкращий варіант для великих таблиць. Використовується лише тоді, коли всі дані для запиту знаходяться в індексі.**
- **Bitmap Index Scan - читання з кількох індексів.**

Практичні поради використання індексів

- **Треба створювати індекси коли швидкодія SELECT запитів є критичною.**
- **Варто додавати індекси до FOREIGN KEY.**
- **Варто додавати індекси до колонок, що активно використовуються в WHERE, JOIN, ORDER BY.**
- **Варто використовувати часткові індекси для предикатів, які часто використовуються в запитах (наприклад “is_active = true” для soft deletion).**

Практичні поради використання індексів

- **Треба відноситись до індексів з розумом - не можна індексувати всі колонки “про всяк випадок”.**
- **Не варто створювати зайві індекси. До прикладу якщо є idx(a, b, c), то індекс idx(a, b) чи idx(a) є надлишковими та зайвими.**
- **Варто обережно відноситись до індексування колонок, які часто змінюються (багато UPDATE).**
- **Треба не забувати про моніторинг індексів та запитів.**

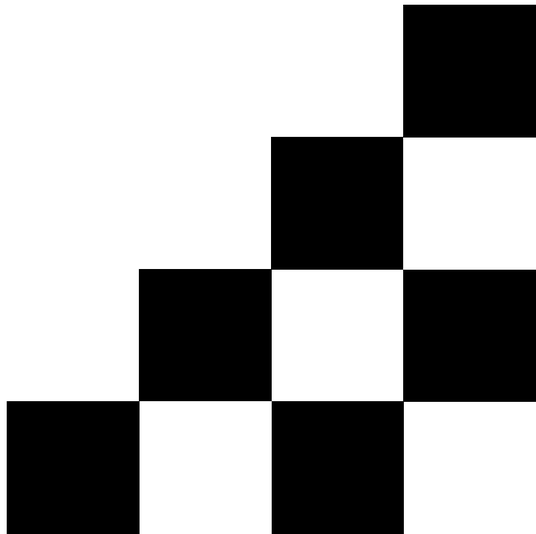
Практичні поради використання індексів

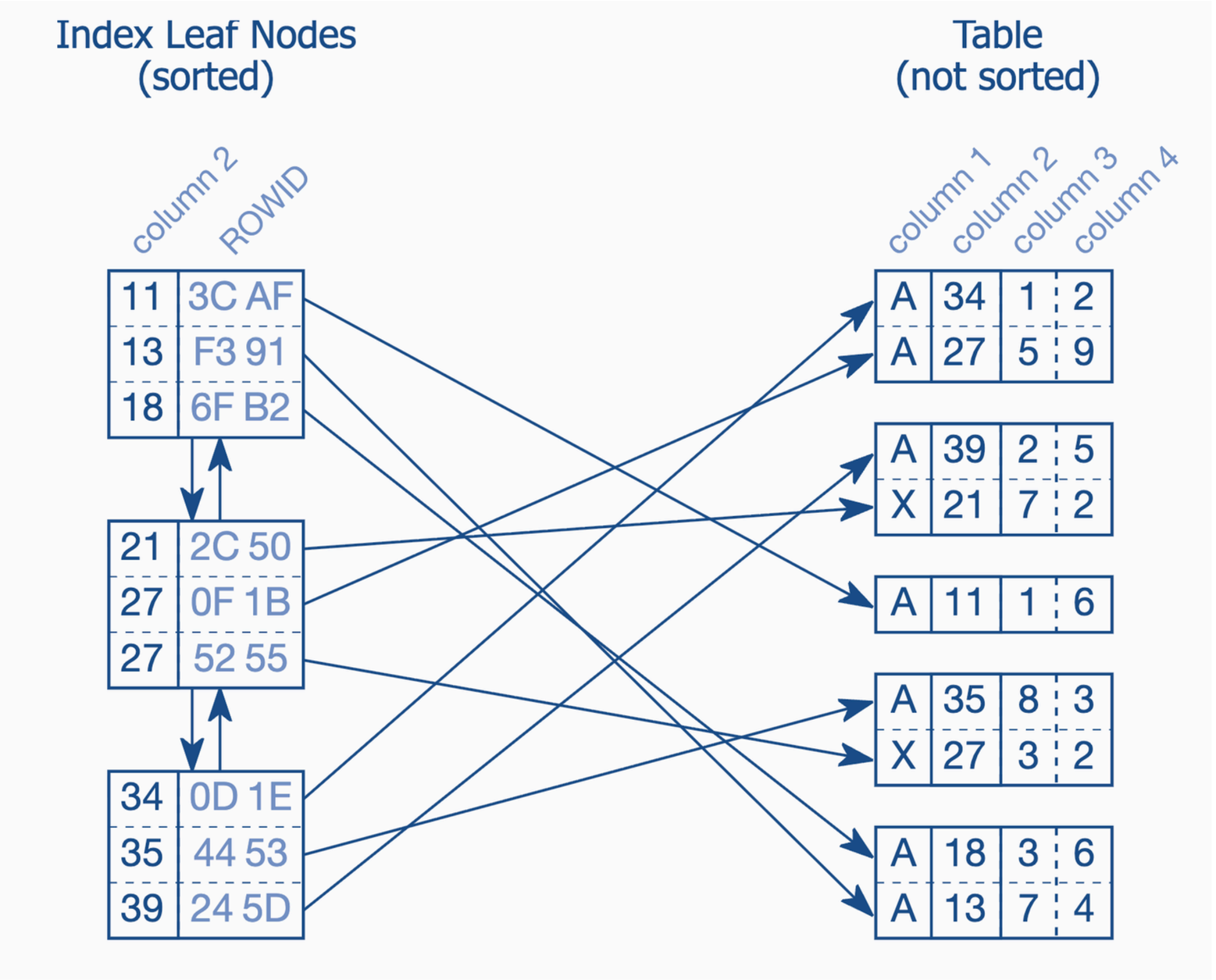
- **Не варто індексувати колонки з низькою варіативністю запитів (наприклад idx on course(is_active) - це погана ідея).**
- **Не варто створювати індекси для запитів, що повертають велику частину даних таблиці (> 15-20% рядків).**
- **Не варто використовувати “SELECT *”, адже це унеможливлює використання планувальником Index Only Scan.**

Практичні поради використання індексів

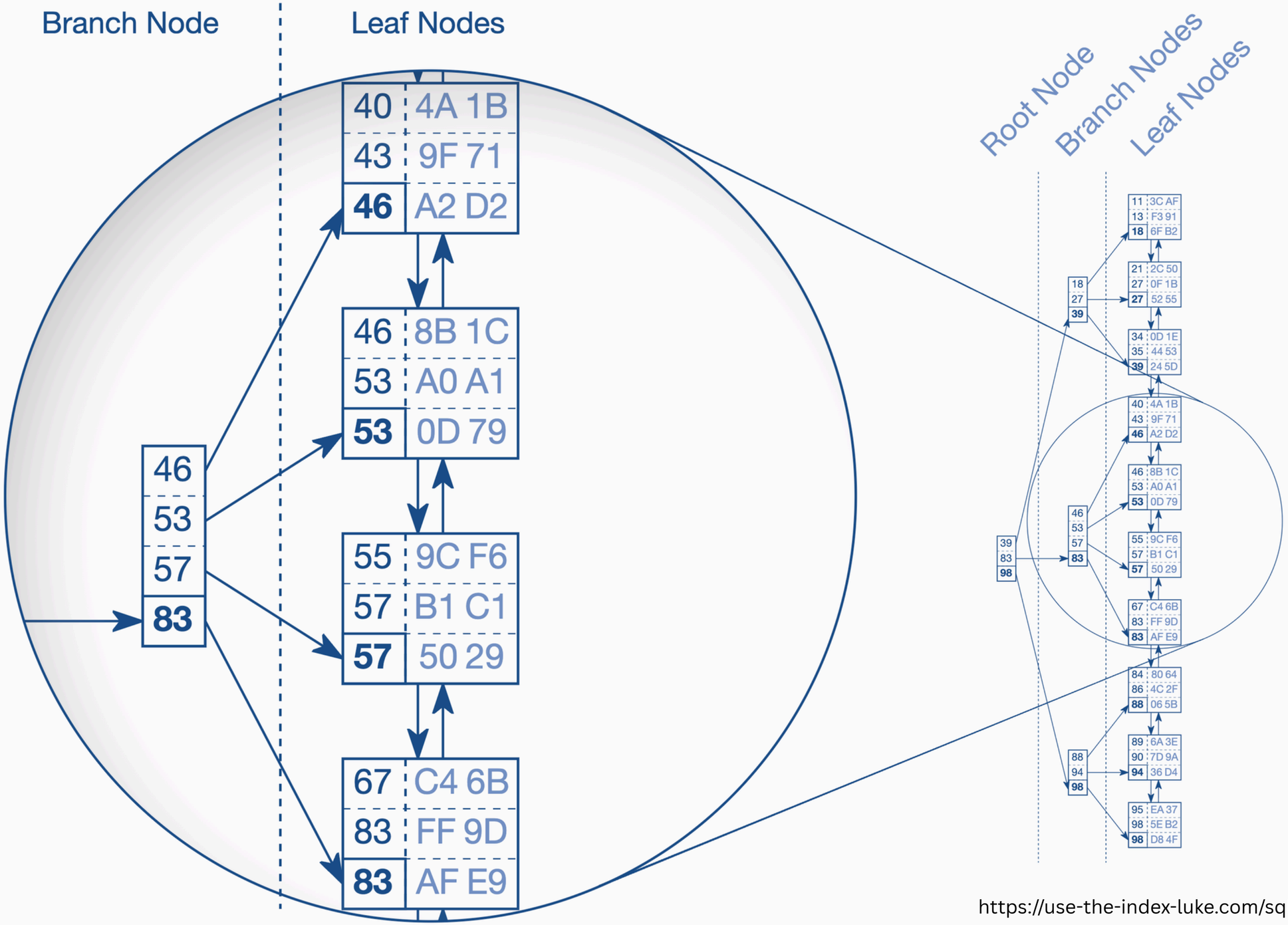
Потрібно перевіряти запит за допомогою EXPLAIN ANALYZE до та після додавання індексів. Якщо індекс не використовується або не робить запит швидшим - його варто прибрати.

B-Tree

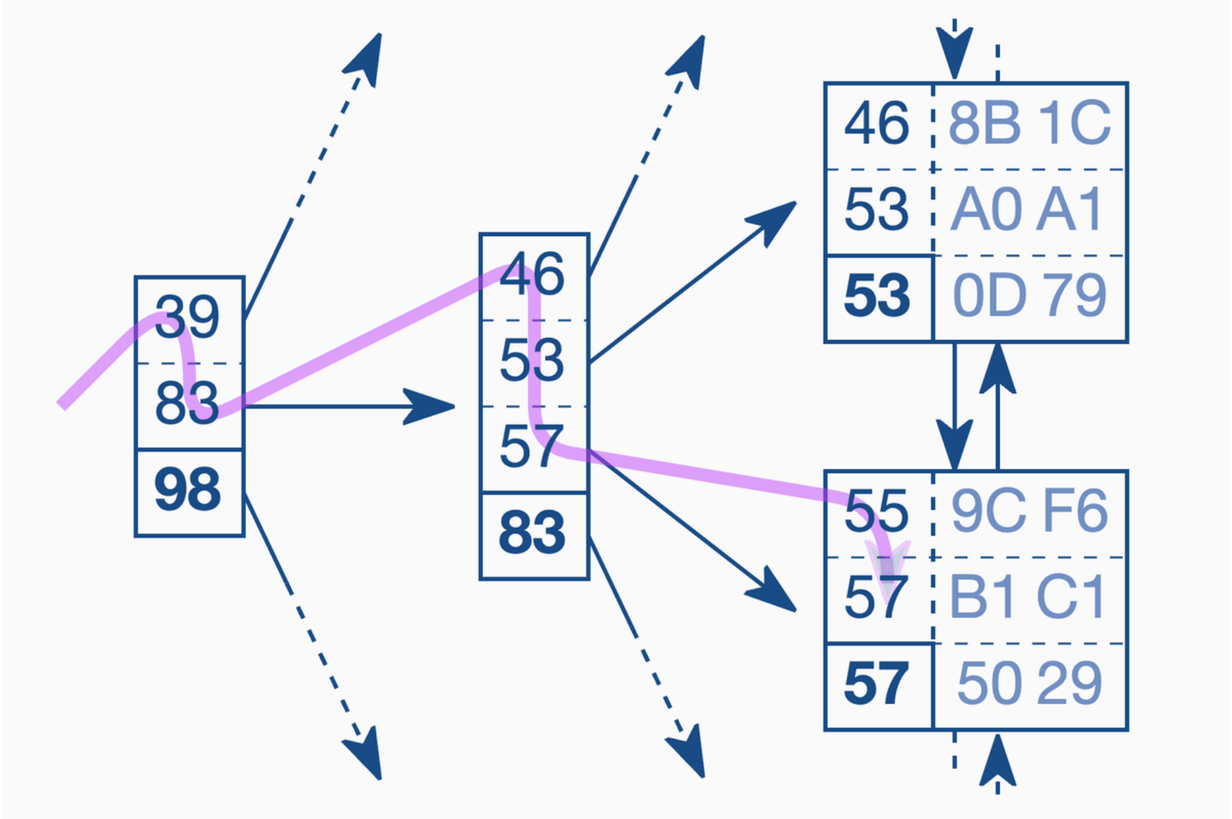




https://use-the-index-luke.com/sql/anatomy/the-leaf-nodes



https://use-the-index-luke.com/sql/anatomy/the-tree



https://use-the-index-luke.com/sql/anatomy/the-tree

Запитання