Tối ưu hóa index

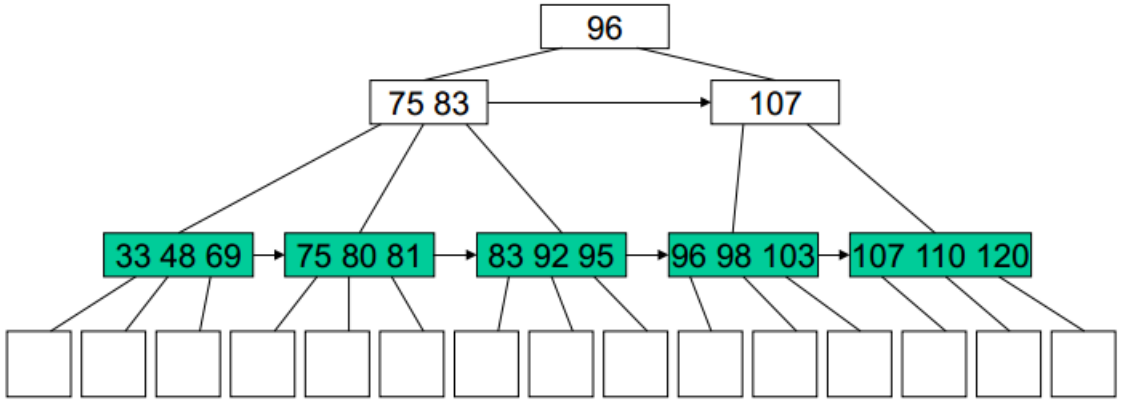
Tối ưu hóa index

- Cấu trúc dữ liệu
- Index hỗn hợp

Cấu trúc dữ liệu index

- Index có thể được khai báo với các cấu trúc dữ liệu khác nhau:
- Chúng ta chỉ thảo luận:
  - B+-tree index
  - Hash index
  - Bitmap index (tóm tắt)
- Không thảo luận:
  - Dynamic hash index: số nhóm sửa tự động
  - R-tree: index với các dữ liệu không gian (points, lines, shapes)
  - Octree: phiên bản quadtree cho dữ liệu 3 chiều.
  - Main memory index: T-tree, 2-3 tree, cây tìm kiếm nhị phân

B+-tree



- ✓Cây cân bằng với các cặp key-con trỏ
- ✓Key được sắp xếp bởi giá trị
- ✓Các node được ít nhất đầy một nửa
- ✓Truy xuất bản ghi lấy key: di chuyển từ node root đến node lá.

Key Length và Fanout

- Key length là quan trọng trong B+-tree: key ngắn là tốt
  - Fanout là số cặp con trỏ key tối đa trong 1 node
  - Key càng dài thì fanout càng nhỏ
  - Càng gần gốc fanout càng nhỏ.

VD Key Length và Fanout

- Lưu 40M cặp con trỏ key trong các leaf page(page: 4kB, pointer: 4B)
  - 6B key: fanout 400 => 3block đọc trên các truy xuất

level	nodes	key-pointer pairs
1	1	400
2	400	160,000
3	160,000	64,000,000

  - 96B key: fanout 40 => 5block đọc trên các truy xuất

level	nodes	key-pointer pairs
1	1	40
2	40	1,600
3	1,600	64,000
4	64,000	2,560,000
5	2,560,000	102,400,000

→ 6B key gần như nhanh gấp đôi 96B key!

Ước tính cấp độ của node

- Sử dụng page:
  - VD giả sử sử dụng 100%
  - Sử dụng phần lớn 69% (nếu các node đầy một nửa được sát nhập)
- Cấp độ:

$$\text{fanout} = \left\lfloor \frac{\text{node size}}{\text{key-pointer size}} \right\rfloor$$

$$\text{number of levels} = \lceil \log_{\text{fanout} \times \text{utilization}}(\text{leaf key-pointer pairs}) \rceil$$

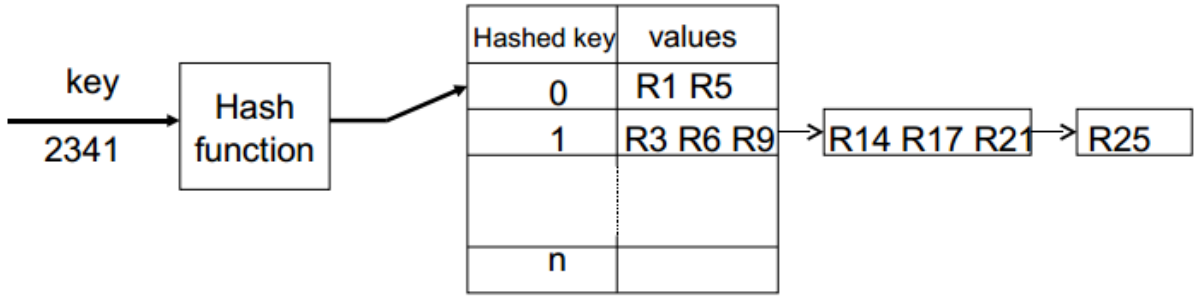
- VD trước với việc sử dụng = 69%:

  - $6B$ key: $\text{fanout} = 400$, $\text{levels} = \lceil 3.11 \rceil = 4$
  - $96B$ key: $\text{fanout} = 40$, $\text{levels} = \lceil 5.28 \rceil = 6$

Nén Key

- Nén key: giảm kích thước của key
  - Giảm cấp độ
  - Thêm CPU cost (vào khoảng 30% trên một truy xuất)
- Nén key là hiệu quả nếu:
  - Key là dài, vd: string key
  - Dữ liệu là static (update ít)
  - CPU time không phải là vấn đề
- Nén prefix: rất phổ biến
  - Node không phải lá chỉ lưu prefix của key
  - Prefix là đủ dài để phân biệt neighbor
  - VD: Cagliari, Casoria, Catanzaro → Cag, Cas, Cat

Hash Index



- Hàm băm:
  - Dẫn key tới các khoảng giá trị nguyên [0..n] (giá trị băm)
  - Pseudo-randomizing: gần hết tất cả các key được phân bố đều trên khoảng
  - Các key giống nhau thường có các giá trị băm rất khác nhau
  - CSDL sẽ tự động chọn hàm băm tốt nhất
- Hash index
  - Hàm băm là "root node" của cây index
  - Giá trị băm là một nhóm giá trị, bao gồm cả bản ghi cho search key hoặc con trỏ tới chuỗi tràn của bản ghi

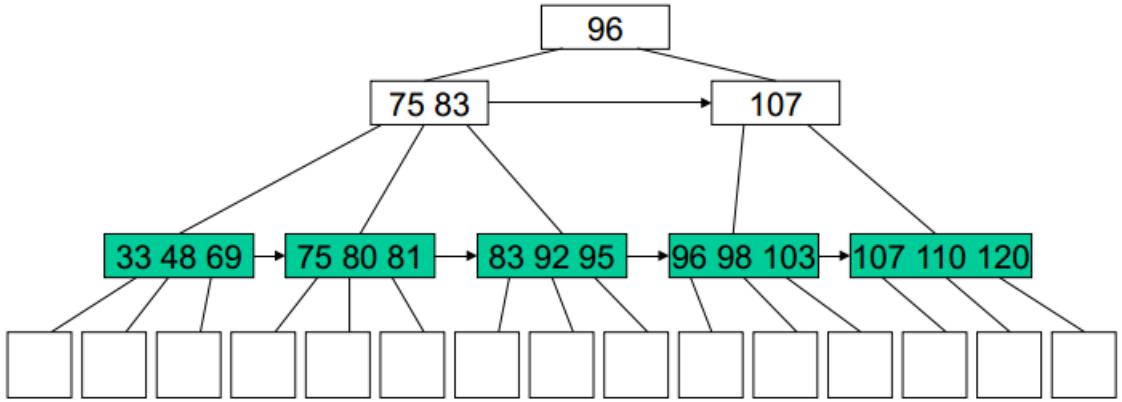
Chuỗi tràn

- Hash index không có overflow: truy nhập bộ nhớ đơn
- Nếu nhóm trang đầy: tràn chuỗi
  - Mỗi trang tràn yêu cầu một phần bộ nhớ thêm
  - Tận dụng không gian băm để tránh dây truyền
  - Thực tế, giá trị sử dụng: 50%
- Hash index với nhiều trang tràn: tái tổ chức
  - Sử dụng hàm tái tổ chức đặc biệt
  - Hoặc đơn giản là xóa và thêm index.

Bitmap Index

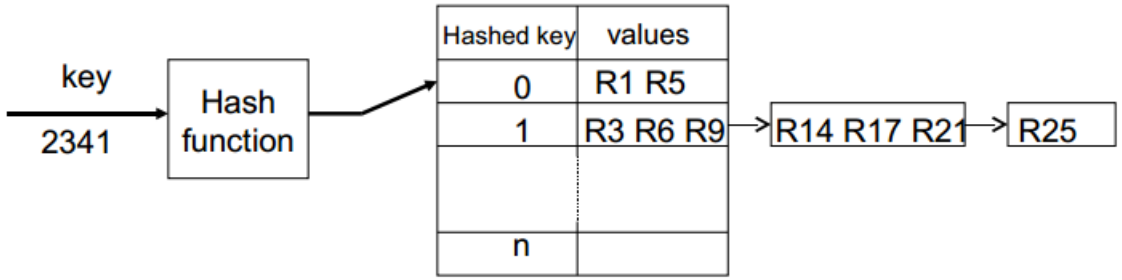
- Index cho các kho dữ liệu
- Một bit vector trên một giá trị thuộc tính
  - Độ dài của mỗi bit vector là số bản ghi
  - Nhưng i cho vector "male" được thiết lập nếu giá trị key trong hàng i là "male".
- Bitmap index hiệu quả khi:
  - Các vị từ truy vấn trên nhiều thuộc tính
  - Các vị từ đơn có khả năng chọn cao (vd: male/female)
  - Tất cả các vị từ cùng nhau lại có khả năng chọn thấp (vd: trả về vài bộ dữ liệu)
- VD: "Find females who have brown hair, blue eyes, wear glasses,
- are between 50 and 60, work in computer industry, and live in
- Bolzano"

Các truy vấn nào đang được hỗ trợ



- B+-tree index hỗ trợ:
  - point: duyệt cây một lần để tìm page
  - multi-point: duyệt cây một lần để tìm page(s)
  - range: duyệt cây một lần để tìm một interval enpoint và follow pointers giữa các node index
  - prefix: duyệt cây một lần để tìm prefix và follow pointer giữa các node index
  - extremal: duyệt cây luôn luôn từ trái sang phairc(min→max)
  - ordering: key được sắp xếp bởi giá trị của chúng
  - grouping: thứ tự key lưu giữ sự sắp xếp.

Truy vấn nào được hỗ trợ



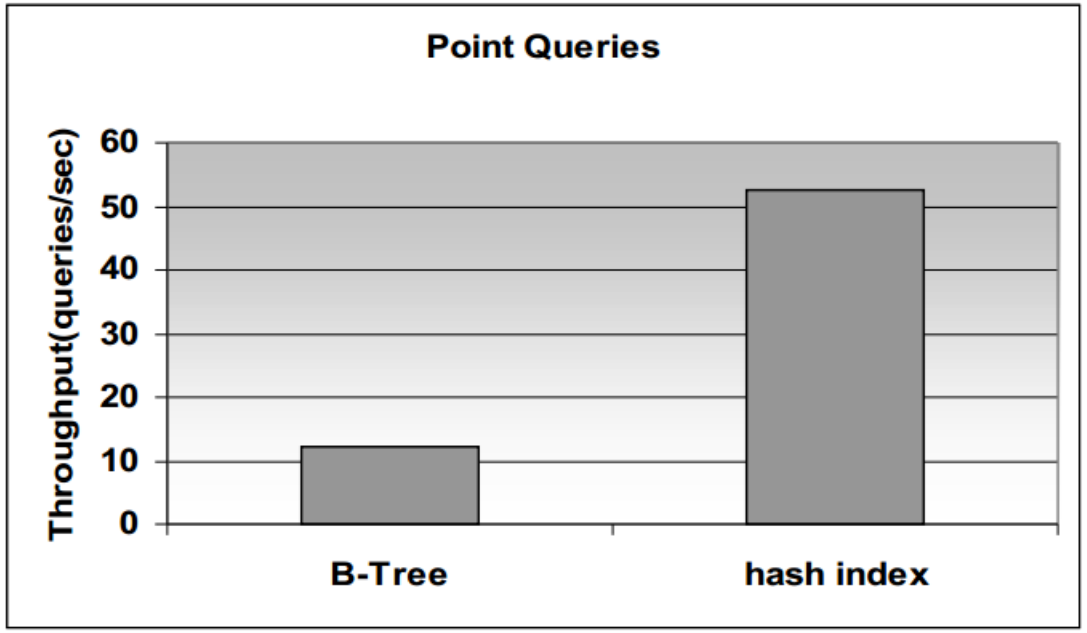
- Hash index hỗ trợ:
  - point: đơn truy nhập bộ nhớ.
  - multi-point: đơn truy nhập bộ nhớ để tìm ra bản ghi đầu tiên
  - grouping: nhóm bản ghi có cùng giá trị băm
- Hash index không hiệu quả cho
  - range, prefix, extremal, ordering
  - các key có giá trị giống nhau nhưng giá trị băm khác nhau

  → nên các key giống nhau nằm trong các trang nhớ khác nhau

Cài đặt thử nghiệm

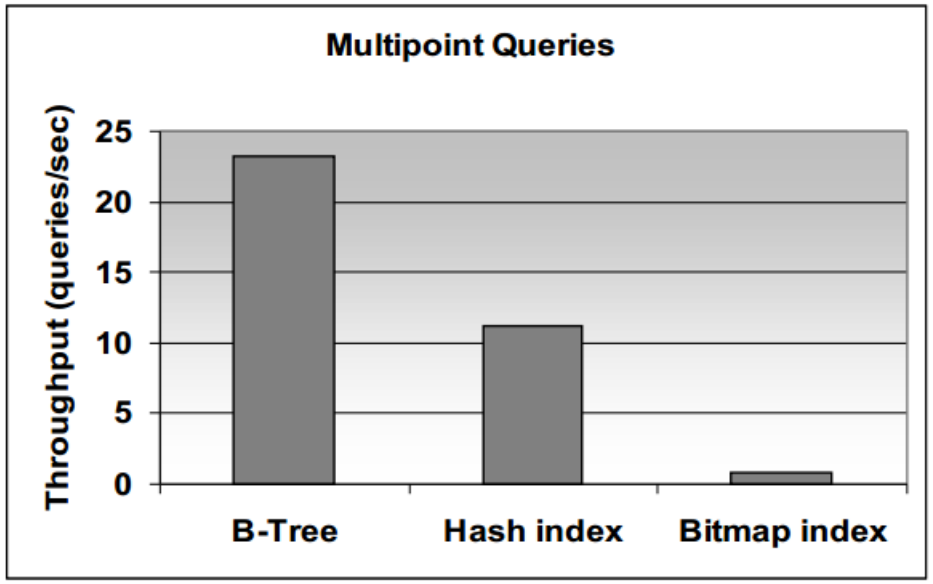
- Employee(ssnum, name, hundreds …)
- 1,000,000 bản ghi
- ssnum là một key (point query)
- Hundreds có các giá trị giống nhau cho 100 employee (multipoint query)
- Point query: index trên ssnum
- Multipoint và range query: index trên hundreds
- B+-tree và các hash index được nhóm
- Bitmap index không bao giờ được nhóm

Thử nghiệm: Point Query



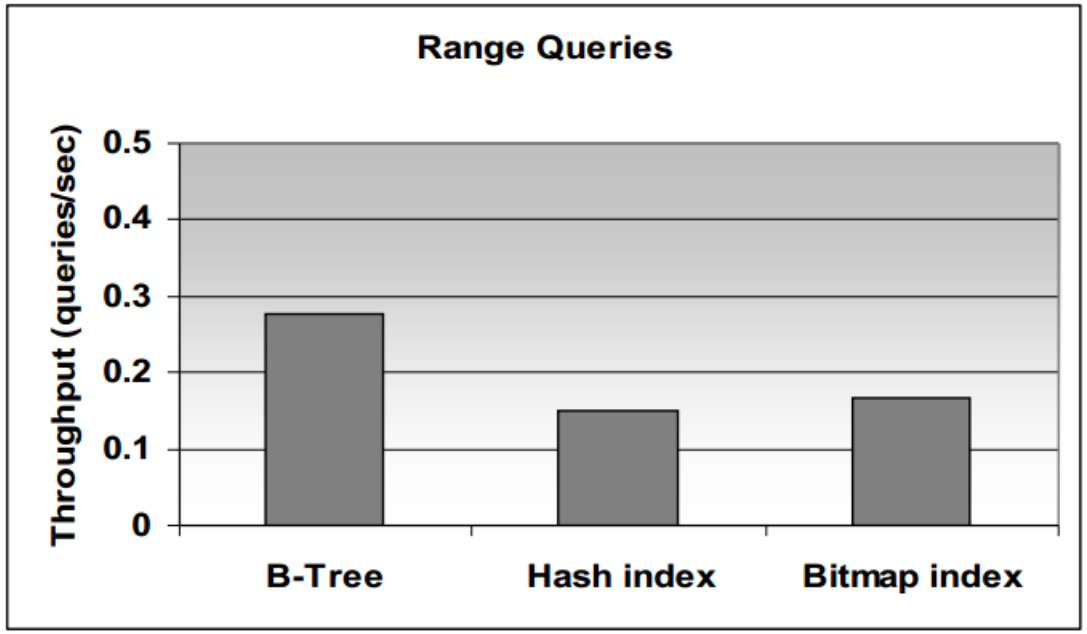
Oracle 8i Enterprise Edition on Windows 2000.

Thử nghiệm: Multipoint Query



- Setup: 100 bản ghi trả về sau mỗi truy vấn
- B+-tree: hiệu quả từ khi các bản ghi nằm trên các trang liên tiếp
- Hash index: key dẫn đến trang đơn và dẫn đến một chuỗi tràn
- Bitmap index: duyệt cả bitmap để lấy được vài bản ghi

Thử nghiệm: Range Query



- B+-tree: hiệu quả từ khi các bản ghi nằm trên các trang liên tiếp
- Hash index, bitmap index không hiệu quả

Index hỗn hợp

- Index trên nhiều hơn một thuộc tính (còn gọi là "index nối").
- VD: Person(ssnum, lastname, firstname, agem address,…)
  - index hỗn hợp trên (lastname, firstname)
  → danh bạ điện thoại được tổ chức như vậy!
- Index có thể dense hoặc sparse
- Dense index trên (A, B, C)
  - Mỗi con trỏ được lưu trên mỗi bản ghi
  - Tất cả con trỏ đến các bản ghi với giá trị A giống nhau được lưu cùng nhau
  - Trong một giá trị A, các con trỏ tới giá trị B giống nhau được lưu cùng nhau.
  - Trong một giá trị A và B, con trỏ tới giá trị C giống nhau được lưu cùng nhau

Index hỗn hợp – Hiệu quả cho Prefix Query

- VD: index hỗn hợp trên (lastname, firstname)
  SELECT * FROM Person
  WHERE lastname=‘Gates’ and firstname LIKE ‘Ge%’
- Index hỗn hợp hiệu quả hơn 2 index thuộc tính đơn:
  - Nhiều bản ghi có thể thỏa mãn firstname LIKE ‘Ge%’
  - Điều kiện trên lastname và firstname cùng nhau có tỷ lệ chọn thấp hơn
  - Giải pháp two-index: kết quả cho các index trên lastname và firstname phải được giao nhau
- Dense index hỗn hợp có thể cover prefix query.

Index hỗn hợp – Skip Scan in Oracle

- Thông thường, index hỗn hợp trên (lastname, firstname) không tiện dùng cho:

  SELECT lastname FROM Person
  WHERE firstname = ‘George’

- Problem: index cover query, nhưng điều kiện không phải là một prefix
- Giải pháp: index bỏ qua scan (được khai báo trong Oracle)
  - Index hỗn hợp trên (A, B)
  - Duyệt mỗi giá trị A cho đến khi bạn nhận được yêu cầu giá trị B
  - Sau đó nhảy đến điểm bắt đầu của giá trị A tiếp theo
  - Quét một phần thay vì quét cả bảng
  - Đặc biệt hữu dụng nếu A có thể mang vài giá trị (vd: male/female)

Index hỗn hợp – Multicolumn Uniqueness

- VD: Order(supplier, part, quantity)
  - supplier là không duy nhất
  - part là không duy nhất
  - nhưng (supplier, part) là duy nhất
- Cách hiệu quả để đảm bảo tính duy nhất
  - Tạo index hỗn hợp, duy nhất trên (supplier, part)
  - CREATE UNIQUE INDEX s_p ON Order(supplier, part)

Index hỗn hợp – Attribute Order Matters

- Đặt thuộc tính với nhiều rang buộc trước.
  VD: Geographical Query
  table: City(name, longitude, latitude, population)
  SELECT name FROM city
  WHERE population >= 10000 AND latitude = 22
  AND longitude >=5 AND longitude<=15
- Hiệu quả: nhóm index hỗn hợp trên (latitude, longitude)
  - các con trỏ trỏ đến tất cả bản ghi kết quả được nhóm với nhau
- Truy vấn geographical nói chung nên sử dụng index đa chiều (vd: một R-tree)

Nhược điểm của index hỗn hợp

- Kích thước key lớn
  - B+-tree sẽ có nhiều lớp
  - Nén key có thể giúp
  - Hash index: key lớn không phải vấn đề nhưng không hỗ trợ range và prefix query.
- Cập nhật tốn chi phí
  - Thông thường, index phải được cập nhật khi key thuộc tính được cập nhật
  - Index hỗn hợp có rất nhiều key thuộc tính
  - Cập nhật được yêu cầu nếu có bất cứ thuộc tính nào được cập nhật