**CÁC PHƯƠNG PHÁP CHỈ MỤC CƠ BẢN**

Cuối những năm 1970, index của các cơ sở dữ liệu quan hệ, phân cấp và CODASYL ra đời với nhiều loại: tuần tự, tuần tự có chỉ mục, hash, cây tìm kiếm nhị phân, cây B, cấu trúc TRIE, file đa danh sách, file đảo và cây đôi kết nối (doubly chained tree). Có nhiều lựa chọn cho lập trình viên, vì vậy mà chọn lựa một loại cây thích hợp là một quyết định khó khăn. Đối với các loại cơ sở dữ liệu – hướng đối tượng, không gian, thời gian, …, danh sách các chiến lược chỉ mục tiềm năng sẽ còn tiếp diễn. May mắn thay, đối với cơ sở dữ liệu quan hệ, cây B+ trở thành phương pháp chỉ mục thực tế cho hầu hết cơ sở dữ liệu quan hệ ngày nay.

# Chỉ mục cây B+

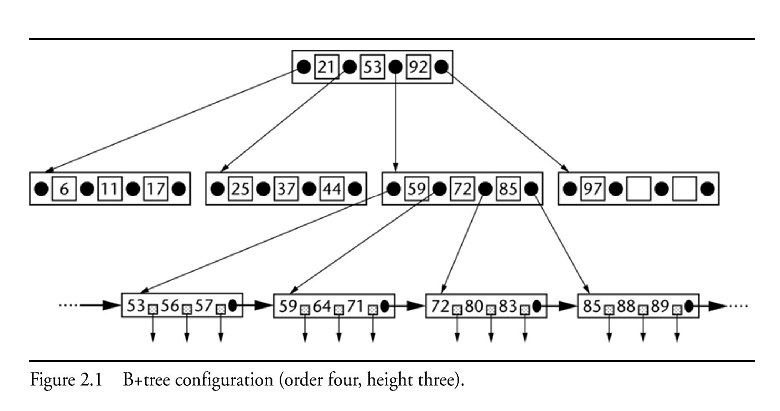
Cây B+ là một phương pháp chỉ mục cơ bản được hỗ trợ trong DB2, Oracle, SQL server. Nó không chỉ đề cao truy vấn nhanh mà còn bảo trì nghĩa là có thể loại bỏ các vấn đề tràn xảy ra trên các phương pháp chỉ mục tuần tự và hash cũ. Hãy xem xét một cây B+ điển hình trên hình 2.1. Mỗi nút chỉ mục không lá, có thể bao gồm một dãy kết hợp các khóa và con trỏ dữ liệu trỏ đến mỗi dòng. Nút chỉ mục lá được liên kết logic bằng con trỏ block sao cho mỗi dòng theo thứ tự có thể được tìm kiếm nhanh nhất. Biến p đại diện cho thứ tự của cây B+, sự phân nhánh của các con trỏ từ một node đến node thấp hơn trong cây.

***Độ cao của cây B+ là bậc của nút lá, nút gốc là bậc 1.***

Nút ở giữa giúp cơ sở dữ liệu tìm kiếm nút lá mong muốn với ít I/O. Tuy nhiên, những gì lưu trong nút lá mới quan trọng. nút lá sẽ lưu 3 thứ rất quan trọng: ***khóa, một mã record (record identifier – RID) và một con trỏ đến nút lá kế tiếp***. Ví dụ, nếu một index được định nghĩa trên cột CITY, khóa của index có thể là NEW YORK, BOSTON, TORONTO. Với mỗi thành phố, chỉ mục sẽ bao gồm một định đanh của một nơi có thể tìm thấy mỗi record trong bảng trùng khớp với khóa. Nếu bảng có 400 entries cho NEW YORK, nút lá sẽ chứa một khóa entry của NEW YORK và định danh cho nơi có thể tìm thấy mỗi 400 record của bảng gốc. Những định danh này được gọi là định danh bản ghi, hay RID. Khóa và RID là hai phần thiết yếu của nội dung lá.

Khóa được lưu trong nút lá theo thứ tự sắp xếp. Kết quả là khi chỉ mục cây B+ có thể được dùng trong DBMS để tìm dòng hoặc một tập hợp các dòng trùng khớp với khóa đơn, thì chúng còn có thể được dùng để tìm một miền khóa (ví dụ một miền số học hoặc alphabet) một cách dễ dàng cũng như hoàn toàn trả về một record theo thứ tự sắp xếp.

**Ví dụ: *Xác định bậc và độ cao của cây B+***

******

***Giả sử có một cơ sở dữ liệu có 500,000 dòng – 200byte, khóa tìm kiếm là 15 byte, và con trỏ dữ liệu 5byte, nút index (và kích thước block dữ liệu) là 1024byte.***

*Gọi p là số con trỏ dữ liệu*

*Kích thước nút index không lá = 1024 = p x 5 + (p -1) x 15byte*

* *p = floor ((1024 + 15)/20) = floor(51.95) = 51*

Trong đó hàm floor là hàm lấy phần nguyên. Vì vậy chúng ta có thể có đến ***p -1*** hoặc 50 khóa trong mỗi node index không lá. Trong nút index lá, có 15byte cho giá trị khóa tìm kiếm và 5 byte cho con trỏ dữ liệu. Mỗi nút lá có một con trỏ đơn đến nút lá kế tiếp để có thể quét các dòng mà không cần đi qua các nút mức index. Trong ví dụ này, số lượng giá trị khóa trong nút lá là floor ((1024 – 5 )/ (5 + 15)) = 50, cùng số lượng với giá trị khóa trong một nút.

Độ cao của cây B+ là số mức index, bao gồm cả nút lá. Chú ý là nút index gốc(mức thứ i) có p con trỏ, bậc 2 của mức i có con trỏ, bậc 3 có con trỏ và tương tự. Ở mức lá, số entry khóa và con trỏ là p -1 trên mỗi nút index, nhưng để dễ tính toán có thể nói một nút index lá có p con trỏ, và p giá trị khóa. Tổng số của con trỏ trên tất cả các node ở mức h có thể lớn hơn hoặc bằng số dòng trong một bảng. Vì vậy chúng ta có

Trong trường hợp này n = 500,000 dòng và p = 50 con trỏ/nút, độ cao của cây B+ là h > 3,35 hoặc h = 4

Một truy vấn đến một dòng cụ thể trong cây B+ là thời gian đòi hỏi để truy xuất tất cả mức h của chỉ mục cây cộng với truy xuất dòng dữ liệu (trong block hoặc page). Tất cả truy vấn đến các mức khác nhau của truy vấn và dữ liệu được cho là ngẫu nhiên

***Đọc một dòng trong bảng (sử dụng cây B+) = h + 1 truy vấn block***

Cập nhật dòng trong cây B+ có thể được thực hiện với một truy vấn đơn giản và ghi lại trừ khi cập nhật liên quan đến việc chèn làm tràn một nút dữ liệu hoặc một nút index hoặc một lệnh xóa nút dữ liệu hoặc nút index. Một thao tác ghi lại dòng vừa mới đọc là một truy vấn block ngẫu nhiên trong môi trường chia sẻ ổ đĩa. Ví dụ cập nhật dữ liệu cho dòng, giả định mỗi nút index được cài đặt như một block

***Chi phí cập nhật cho một dòng đơn (cây B+)***

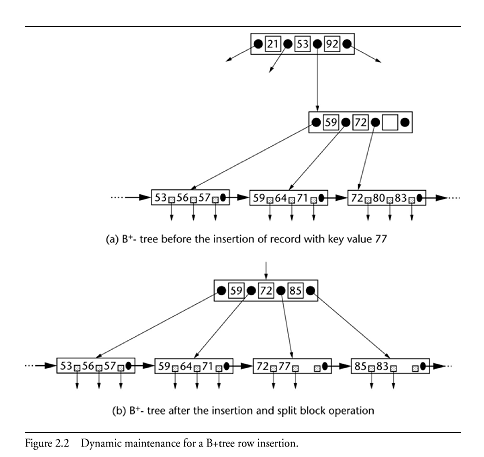
***= chi phí tìm kiếm + ghi lại block dữ liệu***

***= (h +1 ) +1***

***= h + 2 truy vấn block***

*Trong đó một truy vấn block được xem là một truy vấn tới một block (hoặc trang) độc lập trong môi trường chia sẻ đĩa*

Nếu thao tác mong muốn là một phép chèn và phép chèn gây ra tràn dữ liệu trên nút index hoặc dữ liệu, cần những truy xuất mở rộng để chia nút bão hòa thành 2 nút (sử dụng thuật toán phân chia), mỗi nút lưu một nửa cộng với nhu cầu ghi lại nút index cao hơn kế tiếp với một con trỏ đến nút kế tiếp (Hình 2.2). Nhu cầu phân chia được nhận biết sau khi thực hiện bước tìm kiếm dòng lúc đầu. Một sự phân chia nút index lá đòi hỏi thao tác ghi lại của một nút index lá bão hòa, ghi một nửa dành cho dữ liệu, cộng với thao tác ghi một nút index lá mới cũng lưu một nửa, cộng thao tác ghi một nút index không lá với một giá trị con trỏ mới đến nút index lá.



***Chi phí chèn một dòng (cây B+)***

***= chi phí tìm kiếm + ghi lại block dữ liệu + ghi lại block index***

***= (h +1) + 1 + 1***

***= h +3 truy vấn block***

*Cộng với những thao tác ghi nếu cần thiết đối với phân chia*

Đôi khi, thao tác chia một nút index lá đòi hỏi sự phân chia của nút index cao hơn, và trong trường hợp xấu nhất thao tác phân chia có thể tác động nối tiếp tới toàn bộ đường đi lên nút gốc index

Thao tác xóa có thể dẫn đến xóa một block dữ liệu hoặc nút index, dẫn đến sự gom nhóm 2 nút thành một. Điều này có thể đòi hỏi ghi lại nút index lá để reset lại con trỏ của nó. Nút bị xóa có thể được tách riêng rã hoặc ghi đè giá trị null, tùy thuộc cài đặt. Chúng ta có thể giả định nút bị xóa không cần ghi lại. Ngẫu nhiên nút lá hoặc không lá trở thành rỗng và cần gom nhóm lại. Vì vậy ta có chi phí xóa:

***Chi phí xóa một dòng (cây B+)***

***= chi phí tìm kiếm + ghi lại block dữ liệu + ghi lại block index***

***= (h +1 ) +1 + 1***

***= h +3 truy vấn block***

*Cộng với các thao tác ghi mở rộng nếu cần cho gom nhóm*

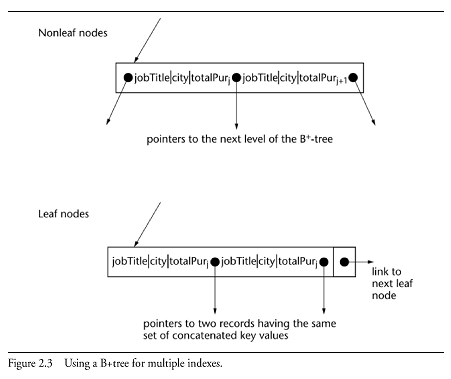
# Tìm kiếm index kết hợp (Composite index search)

Một index đơn có thể được định nghĩa trên nhiều cột. Một khóa index có thể là hỗn hợp của nhiều giá trị. Ví dụ index trên (CITY, STATE).

Index kết hợp có thể được dùng để tìm kiếm hiệu quả một bảng dữ liệu, cho phép tìm kiếm trên các điều kiện phức tạp hơn

Chúng ta có thể dễ dàng mở rộng phương thức index cây B+ để giải quyết các truy vấn phức tạp hơn của loại này. Trong một index unique, nút không lá có thể chứa cặp con trỏ nơi khóa có thể là khóa chính của bảng. Trong index không unique, vùng khóa là một sự móc nối tất cả các thuộc tính bạn muốn cài đặt để truy vấn đến tập hợp dòng mong muốn. Chúng ta có thể hình thành một khóa index không unique bằng cách móc nối các khóa thứ cấp cho job title, city, và total amount of purchase được thể hiện trong hình 2.3

Nút lá của index unique chứa toàn bộ tập các giá trị khóa chính, với mỗi khóa sẽ đi cặp với một con trỏ đến block chứa dòng có giá trị khóa chính cho trước. Đối với nút lá không unique, chúng ta còn có cặp con trỏ khóa, nhưng các giá trị khóa là tập hợp toàn bộ các tổ hợp của khóa phức hợp xảy ra trong dòng. Nếu một giá trị khóa phức xảy ra ở dòng k, có cặp con trỏ khóa k được dùng để trỏ đến các dòng với giá trị khóa cho trước được lặp lại k lần.

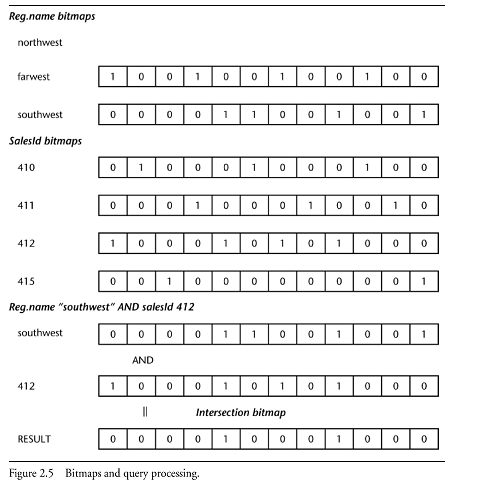


# Bitmap Index

Một trong những qui tắc theo kinh nghiệm cơ bản đối với thiết kế index trong data warehouse (kho dữ liệu) là tạo index trên mỗi cột của mỗi chiều của bảng và tất cả khóa ngoại tương ứng trên fact table – bảng thực tế (trong data warehouse, fact table bao gồm các thước đo,metric hoặc các sự thật của một qui trình doanh nghiệp) . Điều này có ích cho bất kì phép kết nào đòi hỏi giữa bảng thực tế và bảng kích thước tương ứng, bao gồm nhiều phép kết. Index kết tương ứng một giá trị thuộc tính của một bảng kích thước với một hay nhiều dòng của bảng thực tế chỉ là các biến trong tìm kiếm index phức. Vì vậy, index kết thực ra là một phép kết tính toán nhị phân và index kết phức là các phép kết tính toán n – chiều. Những kĩ thuật này có thể cải thiện đáng kể hiệu suất của một data warehouse

Bitmap index đặc biệt có ích đối với các câu truy vấn ít sự chọn lựa, ví dụ những tìm kiếm trên ít giá trị thay thế ví dụ male / female, nhân viên toàn thời gian hay tính theo giờ. Bitmap index tăng tốc các thao tác index đặc biệt như phép giao hoặc hợp. Trong một cây B+, index thứ cấp, chẳng hạn nút lá có mảng con trỏ của địa chỉ dòng vật lý của tất cả dòng thỏa điều kiện được xác định bằng index

Trong bitmap, tuy nhiên, cấu trúc lại khác. Mỗi dòng một bảng được đại diện bởi một bit trong chuỗi bit, và mỗi dòng đáp ứng điều kiện sẽ có bit được set như trong chuỗi bit trong hình 2.5. Ở đây chugs ta thấy mỗi thuộc tính có dãy bit riêng lẻ. Những dòng thỏa tất cả điều kiện được tìm bằng cách sử dụng phép AND các chuỗi bit thể hiện trong hình 2.5



Tuy nhiên, dù hiệu quả trong xác định nhanh chóng các record, bitmap index không dễ mở rộng đối với khóa với như chỉ mục cây B+. Chúng cũng không lưu dữ liệu theo một thứ tự sắp xếp như cây B+. Tóm lại chúng thích hợp với dữ liệu được truy xuất bằng giá trị điểm và giá trị dữ liệu mới ít được thêm vào cơ sở dữ liệu

# Record identifer (mã record – định danh record)

Index luôn lưu trữ một tham chiếu đến nơi có thể tìm thấy giá trị khóa trong bảng cơ bản. Những định danh này được gọi là định danh record – RID. RID cần thiết vì index bản thân nó không chứa tất cả dữ liệu của cột cần thiết, để trả lời truy vấn đòi hỏi cơ sở dữ liệu phải truy vấn bảng cơ bản đối với thông tin mở rộng. Chính vì vậy mà RID sẽ cho cơ sở dữ liệu biết nơi có thể tìm thấy toàn bộ record của bảng

Index có thể trả về RID đơn hoặc một danh sách các RID tùy thuộc vào bao nhiêu khóa đạt yêu cầu tìm kiếm và bao nhiêu RID liên quan tới mỗi giá trị khóa

Định dạng cơ bản của RID sử dụng mô hình 4 byte. Dữ liệu trong bảng được lưu trong các page lưu trữ (điển hình là 4KB hoặc 8KB). Mỗi page được đánh số 1,2,3,… Các record trong page cũng được đánh số tương tự. Mô hình 4 byte đối với RID sử dụng 3 byte để biểu diễn số page, 1 byte biểu diễn số record. Cấu trúc hạn chế kích thước của bảng trong 3 byte địa chỉ (xFFFFFF – 16777215 page) . Tương tự đối với record (xFF – 256 record). Thiết kế RID 4 byte hiệu quả trong việc xác định một record trong page. Tuy nhiên những hạn chế trở nên nhỏ lại theo thời gian và nhà cung cấp cơ sở dữ liệu đòi hỏi thiết kế không hạn chế trong 3 byte địa chỉ hoăc 256 record/ page. Các thiết kế RID mới trong suốt thập kỷ qua sử dụng thiết kế 6 và 8 byte. Định dạng chi tiết của RID biến thiên giữa các cơ sở dữ liệu, nhưng các RID thấp vẫn còn trong các thiết bị thiết yếu trong cơ sở dữ liệu cho việc xác định chính xác một record nằm ở đâu.

# THỦ THUẬT CHO CHUYÊN GIA CƠ SỞ DỮ LIỆU

1. Cây B+ là lựa chọn phương pháp index của hầu hết các DBMS. Index nên được sử dụng để truy xuất đến số lượng nhỏ các dòng trong truy vấn
2. Phân tích cân bằng index có thể thực hiên với các đánh giá hiệu suất cơ sở dữ liệu cơ bản được đề cập bên trên. Điều này giúp chúng ta xác định các lựa chọn index. Khi ta đã hiểu mô hình hiệu suất cơ bản, sẽ dễ dàng để áp dụng cơ chế index hơn vì thế quyết định về index có thể dựa trên thời gian thực tế xử lý một tập các truy vấn
3. Hãy nhớ rằng index cũng tốn chi phí. Mỗi index được tạo cần thiết phải có không gian đĩa để lưu trữ khóa, RID, con trỏ và các node giữa. Do đó index cũng tốn bộ nhớ. Thứ hai là index cần được duy trì khi dữ liệu được thêm, xóa hoặc sửa từ bảng. Một thay đổi đơn giản trong bảng có thể yêu cầu một khóa index không chỉ xóa, mà còn chuyển từ một node lá sang một node khác, do đó tốn thêm thời gian I/O và CPU xử lý. Kết quả là trong khi index có lợi ích to lớn trong hiệu suất truy vấn, nó cũng mang lại hậu quả đáng ngại khi thực hiện các thao tác như INSERT, UPDATE, DELETE, IMPORT và LOAD