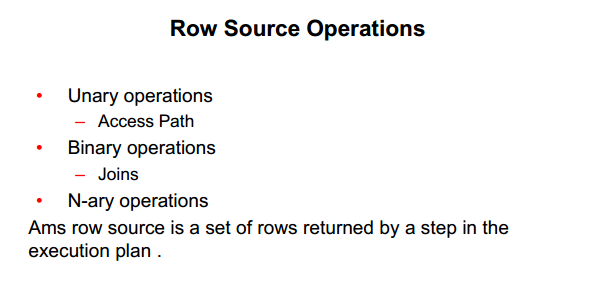
**Các phép toán tối ưu**

# Row source Operation



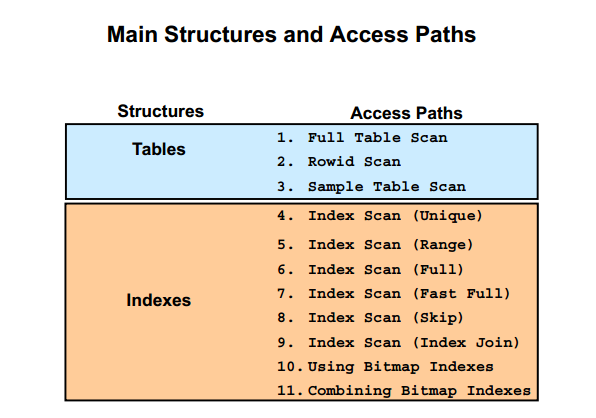
Một row source là một tập các hàng được trả về sau mỗi bước trong kế hoạch truy vấn. Row source có thể là một bảng, một phần của bảng hoặc là kết quả của phép join hay phép group.

Có thể xác định row source bằng các cách sau:

* Phép toán đơn: đầu vào chỉ có 1 tham số, ví dụ các phép truy xuất 1 bảng.
* Phép toán đôi: đầu vào có 2 tham số, ví dụ phép join.
* Phép toán đa: đầu vào có nhiều hơn 2 tham số, ví dụ phép toán quan hệ

Các cách truy nhập là cách mà dữ liệu được lấy ra từ cơ sở dữ liệu. Thông thường, các truy nhập bằng index nên được sử dụng khi muốn lấy ra một tập nhỏ các hàng trong bảng, trong khi full scan hiệu quả khi truy nhập một phần lớn của bảng.

# Main Structures and Access Paths

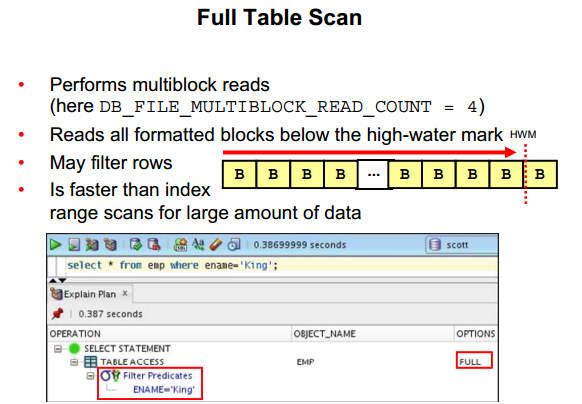


Bất cứ một hàng nào đều có thể được xác định và lấy ra bởi một trong các cách trên. Thông thường, các truy nhập bằng index nên được sử dụng khi muốn lấy ra một tập nhỏ các hàng trong bảng, trong khi full scan hiệu quả khi truy nhập một phần lớn của bảng. Để quyết định xem kế hoạch truy vấn nào được chọn, bộ tối ưu sẽ cho mỗi kế hoạch truy vấn một giá trị cost. Kế hoạch nào có cost thấp hơn sẽ được chọn.

Có các vài loại đặng biệt của truy vấn bảng là clusters, bảng index được sắp xếp và phân vùng (partition).

Clusters là một phương pháp tùy chọn của lưu trữ bảng dữ liệu. Một cluster là một nhóm các bảng mà chia sẻ các khối dữ liệu giống nhau bởi vì chúng có một số cột chung và thường được sử dụng cùng nhau. VD, 2 bảng EMP and DEPT có chung cột DEPTNO. Khi mà cluster bảng EMP và DEPT, oracle lưu tất cả các hàng của EMP và DEPT theo từng phòng vào cùng các khối dữ liệu.

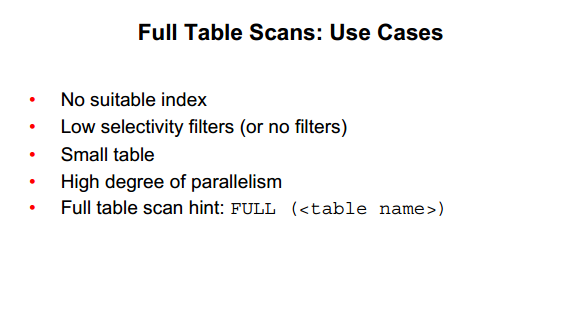
# Full Table Scan



Một full table scan (FTS) sẽ tuần tự đọc hết tất cả các hàng từ một bảng và lọc ra những giá trị phù hợp. Trong mỗi lần quét, tất cả các khối được định dạng là thấp hơn high-water mark (đã được sử dụng) đều sẽ được quét kể cả tất cả các hàng đã được xóa khỏi bảng. Mỗi block chỉ được đọc một lần. High-water mark chỉ ra block đã được sử dụng bao nhiêu hay bao nhiêu phần có thể nhận thêm dữ liệu. Mỗi hàng sẽ được kiểm tra xem có phù hợp với mệnh đề WHERE hay không.

Bởi vì một FTS đọc tất cả block trong bảng, nó đọc các block liền kề nhau. Do vậy hiệu suất đạt được phụ thuộc vào việc tận dụng các lời gọi vào ra (I/O) mà đọc nhiều block trong cùng một khoảng thời gian. Kích thước của một lời gọi đọc có thể là một block cho đến vô số block, phụ thuộc vào giá trị của tham số: DB\_FILE\_MULTIBLOCK\_READ\_COUNT.

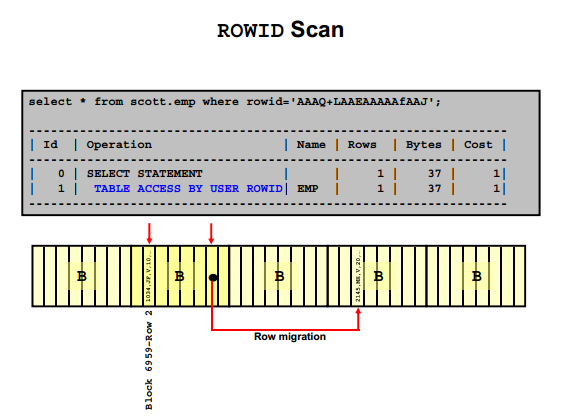
⇒ Chú ý: trong Oracle 6, FTS có thể gây tràn bộ nhớ cache, nhưng kể từ oracle 7 FTS chỉ được chiếm một phần nhỏ của cache. Ngày nay trong hầu hết các trường hợp, FTS được đọc thẳng vào PGA thông qua cache.



Bộ tối ưu hóa sử dụng FTS trong các trường hợp sau:

* Thiếu Index: nếu câu truy vấn không thể sử dụng bất cứ index nào, nó sẽ sử dụng FTS (trừ khi một bộ lọc ROWID hoặc một cách truy nhập cluster khả dụng). VD: nếu một hàm sử dụng bảng index trong câu truy vấn, bộ tối ưu không thể sử dụng index và thay bằng FTS.
* Khối lượng lớn dữ liệu (tỷ lệ chọn thấp): nếu bộ tối ưu nghĩ là câu truy vấn truy nhập đủ các block trong bảng, nó có thể sử dụng FTS mặc dù có thể sử dụng index.
* Bảng nhỏ: nếu một bảng chứa ít hơn DB\_FILE\_MULTIBLOCK\_READ\_COUNT block, dưới chỉ số high-water mark, FTS sẽ được đánh giá cao hơn index range scan, bất kể bảng đó có index hay không.
* Độ tương đồng cao: một bảng có độ tương đồng cao sẽ làm bộ tối ưu nghiêng về cách dùng FTS hơn là range scan. Kiểm tra cột DEGREE trong bảng ALL\_TABLES để xác định độ tương đồng của các bảng.
* Gợi ý sử dụng FTS: sử dụng FULL(table alias) để gợi ý bộ tối ưu chọn FTS.

# ROWID Scan



Rowid của một hàng xác định file dữ liệu và block dữ liệu chứa trong hàng và vị trí của hàng trong block đó. Xác định vị trí một hàng bằng cách xác định rowid của nó là cách nhanh nhất để lấy ra một hàng bởi vì vị trí chính xác của hàng trong CSDL đã được xác định.

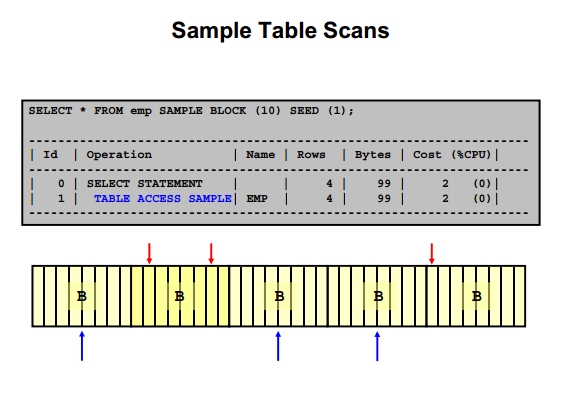
Để truy nhập bảng bằng rowid, hệ thống đầu tiên lấy rowid của các hàng được chọn hoặc từ mệnh đề WHERE hay thông qua index scan một hay nhiều index của bảng. Hệ thống sẽ các định vị trí các hàng được chọn trong bảng dựa theo rowid của chúng.

Bộ tối ưu phần lớn sử dụng rowid sau khi lấy chúng từ một index. Một truy nhập bảng bởi rowid không cần thiết phải theo tất cả index scan. Nếu index chứa tất cả cột cần thiết cho câu truy vấn, truy nhập bảng bởi rowid có thể không xảy ra.

Rowid là đại diện cho vị trí lưu trữ của dữ liệu, Truy nhập dữ liệu dựa trên vị trí không được khuyến khích bởi vì các hàng có thể di chuyển sau khi chuyển vị trí, xâu chuỗi, hoặc export hay import.

Chú ý: bỏi vì sự di chuyển, một rowis có thể đôi lúc trỏ tói một địa chỉ khác với vị trí thực tế của hàng, kết qua là hơn một block sẽ bị truy nhập. VD: update hàng có thể làm hàng đó bị chuyển sang block khác mà rowid vẫn trỏ tới block cũ.

# Sample Table Scans



Một sample table scan (STS) lấy một mẫu dữ liệu bất kì từ một bảng hoặc một câu SELECT phức tạp, như bao gồm join và view. Cách truy nhập này được sử dụng khi mệnh đề FROM bao gồm mệnh đề SAMPLE hoặc SAMPLE BLOCK. Để thực hiện một STS khi đang thử bởi hàng với mệnh đề SAMPLE, hệ thống đọc một số nhất định các hàng của bảng.

* Sample option: thực hiện STS khi thử bới các hàng, hệ thống sẽ đọc dữ liệu đó để xem có phù hợp với mệnh đề WHERE không.
* Sample block option: thực hiện STS khi sample bởi các block, hệ thống đọc một phần block của bảng và kiểm tra mỗi hàng trong các block sampled (đã được lấy mẫu) xem có phù hợp với mệnh đề WHERE không.

Tỷ lệ mẫu là một số xác định tỷ lệ của tất cả các hàng hoặc block . Mẫu giá trị từ [0,000001 . 99,99999). Tỷ số đó chỉ ra xác suất của mỗi hàng hoặc mỗi cluster của hàn trong trường hợp block được chọn như một phần của sample. Điều đó không có nghĩa là CSDL lấy chính xác sample\_percent của hàng trong bảng.

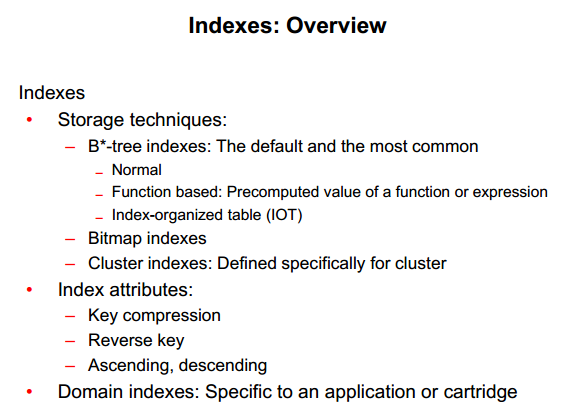
* Seed\_value: xác định mệnh đề này có chỉ thị CSDL thực thi cùng một sample theo thứ tự các câu lệnh hay không. Seed\_value phải là số nguyên giữa 0 và 4294967295. Nếu bỏ qua mệnh đề này, kết quả của sample sẽ thay đổi.

Trong hàng sampling, hơn một block cần được truy nhập, số block đó là kích thước mẫu nhưng các kết quả thường chính xác hơn. Block mẫu có giá trị cost thấp nhưng với mẫu kích thước nhỏ giá trị đó có thể cao hơn.

Chú ý: block sampling là có thể chỉ trong FTS hay index fast full scan. Nếu một các thực thi khác tốt hơn tồn tại, CSDL Oracle sẽ không thực hiện block sampling. Nếu muốn đảm bảo block sampling được dùng cho những bảng hay index đặc biệt, sử dụng FULL hay INDEX\_FFS làm gợi ý cho bộ tối ưu.

# Indexes

## Overview



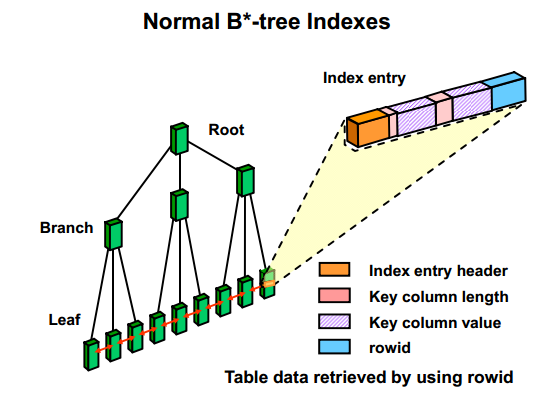
Một index là một đối tượng không bắt buộc của CSDL mà không phụ thuộc cả về mặt logic hay vật lý đối với dữ liệu trong bảng. Trong các cấu trúc không phụ thuộc, index cần khoảng bộ nhớ để lưu trữ. Giống như index của một quyển sách giúp bạn tìm vị trí thông tin nhanh, một index của CSDL Oracle chung cấp cách truy nhập dữ liệu trong bảng nhanh hơn. CSDL Oracle có thể sử dụng index để truy nhập dữ liệu được yêu cầu bởi câu SQL, hoặc sử dụng các index để thực thi các ràng buộc toàn vẹn. Hệ thống sẽ sự động duy trì các index khi dữ liệu liên quan thay đổi. Có thể tạo hay bỏ index bất cứ khi nào. Nếu bỏ một index, tất cả các ứng dụng vẫn làm việc tuy nhiên các truy xuất trước đó sử dụng index sẽ bị chậm hươn. Các index có thể là độc nhất hoặc giống nhau.

Một index hỗn hợp, hay còn gọi là index nối, là một index mà bạn tạo trong nhiều cột trong một bảng. Các cột trong index hỗn hợp có thể xuất hiện theo bất kỳ thứ tự nào và không cần nằm kề bảng.

Với các index tiêu chuẩn, CSDL sử dụng cây B\*-tree index cân bằng để làm cân bằng các lần truy xuất. Cây B\*-tree index có thể là cây bình thường, đảo ngược khóa, giảm dần hay dựa trên chức năng.

* B\*-tree index: là các index thông thường nhất. Giống như cấu trúc của cây nhị phân, B\*-tree index cung cấp cách truy nhập nhanh, bằng khóa, cho một hàng nhất định hay một dãy hàng, thông thường một vài lượt đọc để tìm ra hàng cần tìm. “B” trong “B\*-tree” là “balanced”(cân bằng) chứ không phải “binary”(nhị phân).
* Descending index: Descending index cho phép dữ liệu được sắp xếp từ lớn đến nhỏ(giảm dần - descending) thay cho từ nhỏ đến lớn (tăng dần - ascending) trong cấu trúc index thông thường.
* Reverse key index (khóa index đảo ngược): trong cây B\*-tree, các khóa bị đảo ngược. khóa index đảo có thể được dùng để đạt được nhiều hơn bản phân phối của các mục index trong suốt một index mà được phổ biến với giá trị tăng. VD: nếu sử dụng chuỗi để tạo khóa chính, chuỗi sẽ tạo ra các giá trị như: 987500, 987501, 987502,... Với khóa index đảo, CSDL sẽ đánh chỉ số 005789, 105789, 205789,... thay cho 987500, 987501, 987502. Bởi vì những khóa đảo đó bây giờ gần như được đặt ở những vị trí khác, nên có thể giảm sự tranh chấp cho các block. Tuy nhiên chỉ các vị từ tương đương là có thể hưởng lợi từ loại index này.
* Index key compression: khái niệm cơ bản đằng sau một khóa nén index là tất cả chỉ mục bị hỏng trong hai thành phần: tiền tố và hậu tố. Tiền tố được tạo trên các cột đầu tiên của index nối và có nhiều giá trị lặp đi lặp lại. Hậu tố được tại trên các cột cuối cùng trong khóa index và là thành phần độc nhất cỉa chỉ mục index trong tiền tố. Đây không phải là nén bằng cùng một cách như các file ZIP được nén, hơn nữa, đây là một các nén không bắt buộc mà có thể loại bỏ dư thừa trong index nối.
* Function-based index: có B\*-tree hoặc bitmap index mà có lưu trữ các kết quả sau tính toán của một hàm trên hàng hoặc cột và không phải chỉ là cột dữ liệu. Có thể xem chúng như index trong cột ảo. Mặt khác, nó là cột mà không chỉ được lưu trong bảng về mặt vật lý. Có thể kết hợp thống kê trên cột ảo này.
* Index-organized table (IOT): có các bảng lưu trong một B\*-tree. Trong khi các hàng dữ liệu trong một bảng được sắp xếp, lưu trong heap, được lưu trong một cách bất kì (bất cứ đâu mà bộ nhớ trống), dữ liệu trong một IOT được lưu và sắp xếp bởi một khóa chính. Các IOT hoạt động cũng giống như các bảng thông thường.
* Bitmap index: là một cây B\*-tree thông thường, không có quan hệ một-một giữa một chỉ mục index và một hàng, mà là một chỉ mục index trỏ tới một hàng. Với bitmap index, một đơn chỉ mục index sử dụng một bitmap để trỏ tới nhiều hàng đồng thời. Chúng thích hợp với dữ liệu lặp (dữ liệu với một vài giá trị khác nhau, liên quan đến tổng số hàng của bảng) mà gần như là chỉ đọc(read-only). Bitmap index nên không bao giờ được xem xét trong CSDL OLTP cho các vấn đề liên quan đồng thời.
* Bitmap join index: một bitmap join index là một bitmap index cho join của 2 hay nhiều hơn 2 bảng. Một bitmap join index có thể được sử dụng để tránh join thực sự vào bảng hay để giảm khối lượng dữ liệu cần join, bằng cách sử dụng các hạn chế trước. Các truy vấn sử dụng bitmap join index có thể được tăng tốc bằng cách sử dụng bit-wise.
* Application domain index: là index mà được tạo với package và bộ nh, cả trong CSDL hay thậm chí ngoài CSDL. Cần cho bộ tối ưu biết mức độ chọn của index là như thế nào và chi phí nếu index đó được chạy, và bộ tối ưu sẽ quyết định có sử dụng index đó hay không.

## Normal B\*-tree Indexes



Mỗi cây B\*-tree index đều có root block như là điểm bắt đầu. Tùy thuộc vào số chỉ mục sẽ có nhiều block nhánh(branch block) mà những nhánh đó lại có nhiều block lá(leaf block). Block lá chứa toàn bộ giá trị của index cộng với các ROWID mà trỏ tới các hàng trong segment tương ứng.

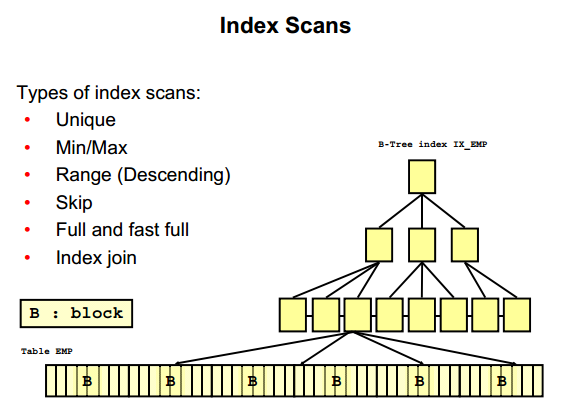
Con trỏ block trước và sau kết nối các block lá để chúng có thể dịch chuyển từ trái qua phải hoặc ngược lại.

Index luôn được cân bằng và chúng phát triển từ trên xuống. Trong các trường hợp cụ thể, thuật toán cân bằng có thể làm chiều cao B\*-tree tăng không cần thiết. Có thể tái tổ chức index bằng cách sử dụng câu lệnh: ALTER INDEX … REBUILD|COALESCE.

Cấu trúc bên trong của một cây B\*-tree index cho phép lặp lại truy nhập đến các giá trị index. Hệ thống có thể trực tiếp truy nhập các hàng sau khi lấy được địa chỉ (ROWID) từ index trong block lá.

Chú ý: kích thước tối đa của một chỉ mục index là khoảng một nửa kích thước block.

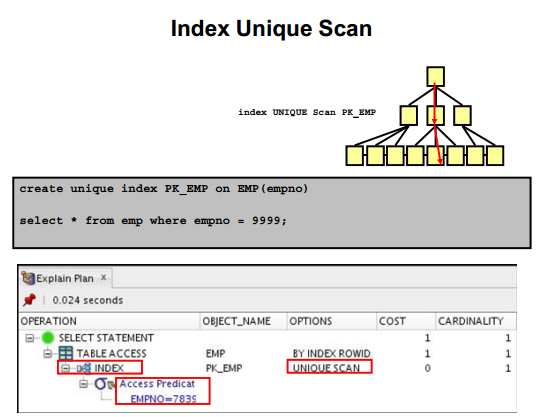
## Index Scans



Một index scan có thể có những kiểu sau:  
Một hàng được lấy ra nhờ đi qua các index, sử dụng các giá trị cột được index được xác định bằng mệnh đề WHERE. Một index scan lấy dữ liệu từ một index dựa trên giá trị của một hoặc nhiều bảng trong index. Để thực hiện một index scan, hệ thống tìm index cho các giá trị cột được index được truy nhập bởi câu truy vấn. Nếu truy vấn chỉ đến các cột của index, hệ thống sẽ hay đọc giá trị cột được index trực tiếp từ index hơn là từ bảng.

Index chứa không chỉ các giá trị được index mà còn các rowid của các hàng trong bảng. Nên nếu truy vấn đến các cột khác ngoài các cột được index, hệ thống có thể tìm các hàng trong bảng bằng cách sử dụng rowid hay một cluster scan.   
Chú ý: hình minh họa chỉ ra một trường hợp mà 4 hàng được lấy từ bảng sử dụng rowid của chúng có được từ index scan.

## Index Unique Scan



Trong hầu hết trường hợp, một index unique scan trả vềmột ROWID. Hệ thống thực hiện một unique scan nếu câu truy vấn chứa một UNIQUE hay một PRIMARY KEY, để đảm bảo chỉ một hàng được truy nhập. Cách truy nhập này được sử dụng khi tất cả các cột của một unique (B\*-tree) index được xác định với các điều kiện đồng đều.

Giá trị key và các ROWID có được từ index và các hàng của bảng có được nhờ sử dụng ROWID.

Có thể tìm các điều kiện truy nhập trong phần “thông tin vị từ” của kế hoạch truy vấn.

Trong hình minh họa hệ thống chỉ truy nhập hàng nơi EMPNO=9999.

Chú ý: bộ lọc điều kiện lọc các hàng sau khi bắt được output và hoạt động của các hàng được lọc.

## Index Range Scan

(4 slide của index range scan)

## Index Full Scan

(2slide)

* Index fast full scan

## Index Skip Scan

(2 slide)

## Index Join Scan

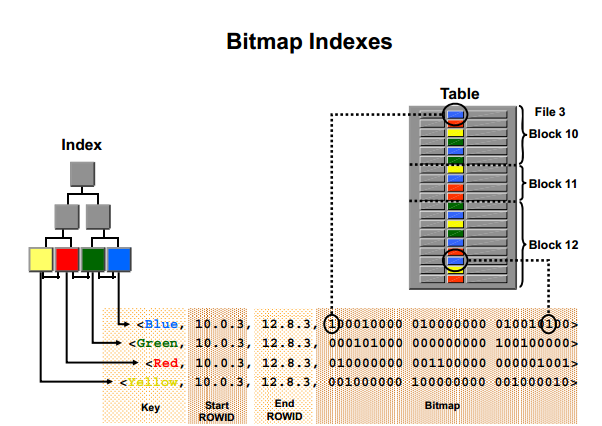
## B\*-tree Indexes and Nulls

## Using Indexes: Considering Nullable Columns

## Index-Organized Tables

(2slide)

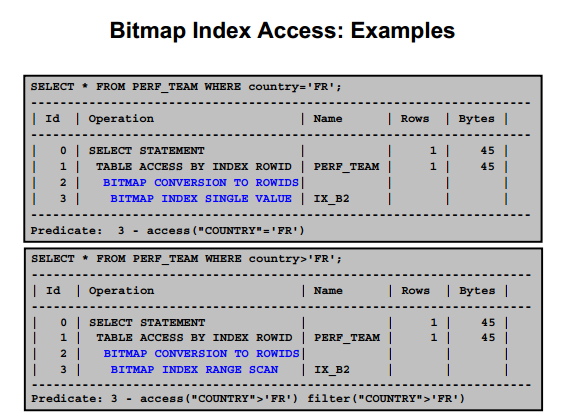
## Bitmap Indexes



Trong B tree, có một mối quan hệ 1-1 giữa index và hàng, mỗi index trỏ tới một hàng. Bitmap index cũng có trật tự giống như B tree nhưng với bitmap index, mỗi index đơn lẻ sẽ sử dụng một điểm bimap để trỏ tới đồng thời nhiều hàng hơn. Nếu một bitmap index bao gồm nhiều hơn một cột, sẽ có một bitmap cho mỗi tổ hợp. Mỗi bitmap header lưu trữ những ROWID bắt đầu và kết thúc. Nó là điều có thể được bởi vì hệ thống biết số tượng tối đa số hàng có thể lưu trữ trong một block hệ thống. Mỗi vị trí trong một bitmap vạch ra một hàng tiềm tàng trong bảng nếu hàng trừ khi hàng đó không tồn tại. Nội dung của vị trí trong bitmap cho một giá trị đặc biệt cho biết giá trị của của hàng trong các bitmap cột. Giá trị được lưu trữ là 1 nếu giá trị của hàng hợp với điều kiện bitmap, trong trường hợp khác nó là 0 Bitmap indexs được sử dụng rộng trong môi trường kho dữ liệu.

Đó là môi trường điển hình có phạm vi dữ liệu lớn với nhiều loại câu truy vấn nhưng không sử dụng đồng thời nhiều câu truy vấn bởi vì khi ta đang khóa một bitmap, ta cũng đang khóa nhiều hàng trong bảng cùng lúc. Với những ứng dụng như vậy, bitmap indexs cung cấp các câu trả lời bị chậm ở một số lớn các lớp câu truy vấn, giảm đi các yêu cầu lưu trữ khi so sánh với các kỹ thuật index khác, đặc biệt là về hiệu năng trên một nền tảng phần cứng với số lượng cpu nhỏ, bộ nhớ nhỏ, và hiệu quả trong việc duy trì việc thực hiện đồng thời truy vấn và load dữ liệu.

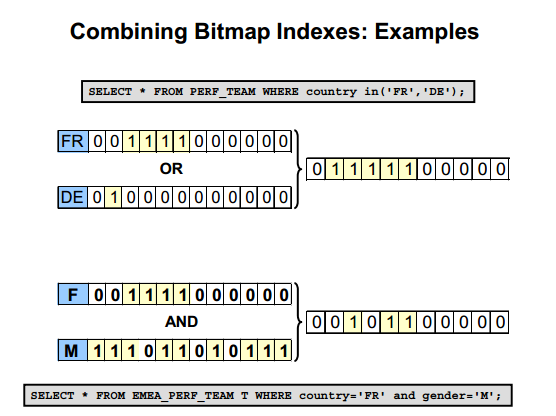
Ghi chú: Không giống hầu hết các loại index khác, bitmap indexs bao gồm cả các hàng có giá trị NULL. Việc đánh chỉ mục cả các giá trị NULL có thể hữu ích trong một số loại truy vấn SQL, ví dụ câu truy vấn với sự tích hợp hàm COUNT. Vị ngữ IS NOT NULL cũng có thể có giá trị từ bitmap indexs. Mặc dù bitmaps đã bị nén bên trong, nó cũng bị chia ra nhiều lá nếu số hàng tăng lên.



Bitmap Index Access: Examples

Ví dụ trong slide minh họa 2 cách tác động có thể cho bitmap indexs - BITMAP INDEX SINGLE VALUE và BITMAP INDEX RANGE SCAN - phụ thuộc và vị ngữ bạn dùng trong câu tru vấn. Câu truy vấn đầu tiên quét bitmap cho COUNTRY là FR với các vị trí có giá trị 1. Các vị trí có giá trị 1 được convert sang ROWID và có hàng tương ứng của chúng trả về cho câu truy vấn.

Trong một số trường hợp (ví dụ truy vấn đếm số hàng với COUNTRY FR), câu truy vấn có thể đơn giản sử dụng bitmap của nó và đếm số giá trị 1 (không cần tác động tới các hàng) .



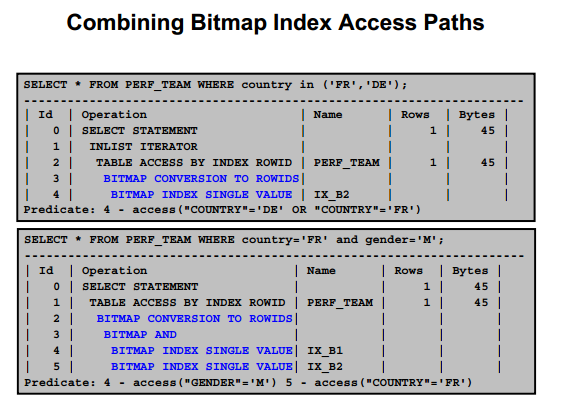
Kết hợp bimap indexs: ví dụ

Bitmap indexs có hiệu quả tốt nhất với các truy vấn bao gồm nhiều điều kiện trong mệnh đề WHERE. Các hàng mà làm thỏa mãn một số lượng (không phải tất cả) các điều kiện mà đã lọc ra trước khi tác động tới bảng. Nó cải thiện thời gian phản hồi.

Giống như một bimap từ các bimap index có thể được được phối hợp nhanh chóng, nó thường là cách tốt nhất để sử dụng bitmap index của cột đơn.

Bitmap index hiệu quả khi dử dụng:

* IN (value list)
* Vị ngữ được tổ hợp với And hoặc Or

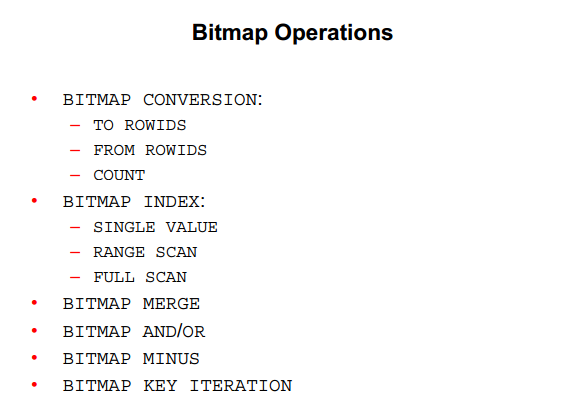


Bitmap index có thể được sử dụng hiệu quả khi câu truy vấn kết hợp một số giá trị có thể được cho một cột hoặc khi 2 cột đã được đánh index không trùng nhau được dùng.

Trong một số trường hợp, mệnh đề WHERE có thể tham chiếu một vài cột đã đánh index không trùng nhau giống như ví dụ trong slide.

Nếu cả 2 cột COUNTRY và GENDER đều có bitmap index, các toán tử trên bit trong 2 bitmap nhanh chóng xác định được hàng mà chúng mong muốn.

Khi mệnh đề WHERE trở nên phức tạp và rắc rối hơn, có nhiều giá trị mà bạn có thể tìm thấy từ bimap index.

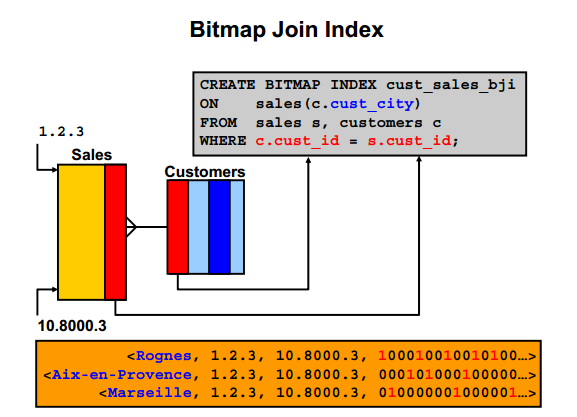


Bitmap Operations

Silde trên tổng hợp tất cả các toán tử bitmap có thể.

Các toán tử dưới đây đã không được giảng giải nhiều:

* BITMAP CONVERSION FROM ROWID: B-tree index được convert bởi bộ tối ưu sang bimap(chi phí thấp hơn các phương thức khác) để tận dụng hiệu quả của bitmap khi so sánh với các toán tử phù hợp khác. Sau khi bitmap được thực hiện xong, kết quả của nó được convert lại ROWID (BITMAP CONVERSION TO ROWID) để thực hiện việc tra cứu dữ liệu.
* BITMAP MERGE: kết hợp một số kết quả bitmap từ range scan vào một bitmap.
* BITMAP MINUS: là một toán tử kép, nó lấy toán tử bimap thứ 2 rồi phủ định nó (1 thành 0 và 0 thành 1).Toán tử bimap minus sau đó thực hiện giống như toán tử BITMAP AND sử dụng bitmap phủ định vừa tạo ra.
* BITMAP KEY ITERATION: đem mỗi hàng từ table row source và tìm bimap tương ứng từ bimap index. Tập hợp các bimap nàu sao đó được trộn vào trong toán tử BITMAP MERGE.



Bitmap Join Index

Thêm một điều về bitmap index trên một bảng đơn, bạn có thể tạo một bitmap join index. Một bitmap join index là một bimap index của sự kết nối của 2 hay nhiều hơn các bảng. Bitmap join index là một cách hiệu quả hơn trong việc giản kích thước của data mà phải được nối bởi hiệu năng của việc kết nối nâng cao.

Ghi chú: Bitmap join index có nhiều hiều quả hơn trong lưu trữ so với việc kết nối cụ thể.

Cùng xem Case study: Start Transformation (chương 9)

Đây, bạn tạo một bitmap join index mới với tên: cust\_sales\_bji trong bảng SALES.

Khóa của index này là cột CUST\_CITY của bảng CUSTOMERS. Ví dụ này cho rằng khóa chính của CUSTOMERS được lưu trữ trong bimap index tồ tại thực sự trong dữ liệu của bảng. Cột CUST\_ID là khóa chính của bảng CUSTOMERS nhưng cũng là khóa ngoài của bảng SALES, cho dù nó không đòi hỏi.

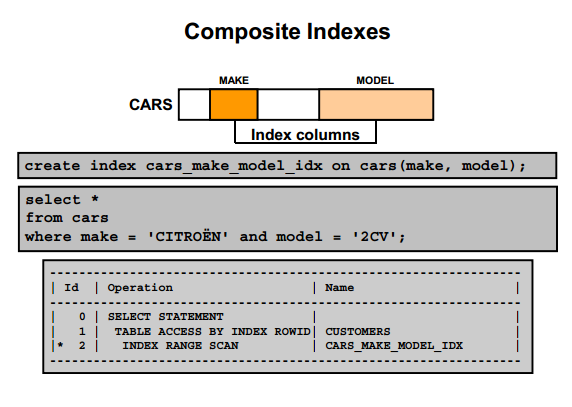
Mệnh đề From và WHERE trong CREATE cho phép hệ thống kết nối giữa 2 bảng. Chúng tượng trưng cho điều kiện nối giữa 2 bảng.

Phần giữa của bức ảnh cho bạn thấy sự thể hiện về lý thuyết của sự kết nối bitmap index. Mỗi mục hoặc khóa trong index tiêu biểu cho một thành phố có thể tìm thấy trong bảng CUSTOMERS.

Một bimap sau đó liên kết tới một key đặc biệt. Mỗi bitmap tương ứng với một hàng trong bảng SALES. Trong khóa đầu tiên trong Slide, bạn thấy hàng đầu tiên trong bảng SALES tương ứng tới một sản phẩm tới Khách hàng Rognes, trong khi bit thứ 2 không là một sản phẩm tới Khách hàng Rognes. Bằng cách lưu trữ kết quả của kết nối, kết nối có thể bị hủy bỏ hoàn toàn toàn trong

câu lệnh SQL sử dụng bimap join index.

## Composite Indexes



Composite indexs

Index kết hợp cũng được qui vào giống như index móc nối bởi vì nó móc nối giá trị cột này với cộ khác để tạo thành giá trị khóa index.

Trong slide minh họa, cột MAKE và MODEL được móc nối với nhau để tạo index. Nó là không cần thiết khi mà các cột trong index sát nhau. Bạn có thể bao gồm tới 32 cột trong index, trừ khi nó là index kết hợp, nó giới hạn là 30 cột.

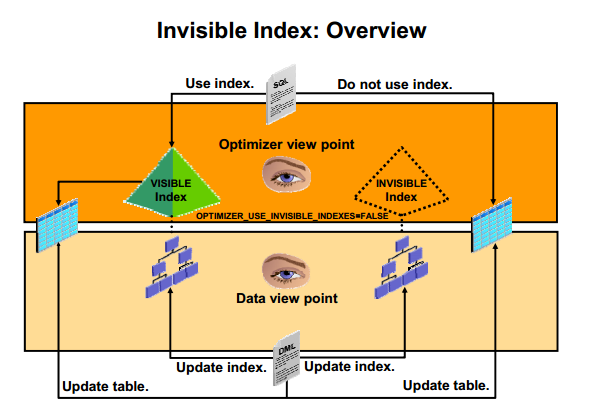
Index kết hợp có thể cung cấp những tinh năng nâng cao hơn index một cột.

* Cải thiện tính chọn lọc: Đôi khi hai hay nhiều cột hoặc biểu thức, mỗi cái với tính chọn lọc nghèo nàn có thể được tổ hợp thành index kết hợp với tính chọn lọc cao hơn.
* Giảm quá trình vào ra: nếu tất cả các cột được chọn bởi truy vấn đều nằm trong index kết hợp, hệ thống có thể trả về giá trị đó từ index mà không cần tác động tới bảng.

Index kết hợp có tác dụng chính khi bạn thường xuyên có mệnh đề WHERE tham chiếu tới tất cả hay một số các cột trong index. Nếu một số khóa được sử dụng trong mệnh đề WHERE thường xuyên, và bạn quyết định tạo index kết hợp, chắc rằng việc tạo index kết hợp này chứa các khóa thường được chọn và các câu truy vấn này chỉ cần sử dụng index là đủ.

Ghi chú: Bộ tối ưu có thể chọn các index móc vào nhau này một cách hợp lý trừ khi truy vấn của bạn không tham chiều tới một phần của index. Nó có thể dừng scan index và scan nhanh toàn bộ có thể được thi hành.

## Invisible Index

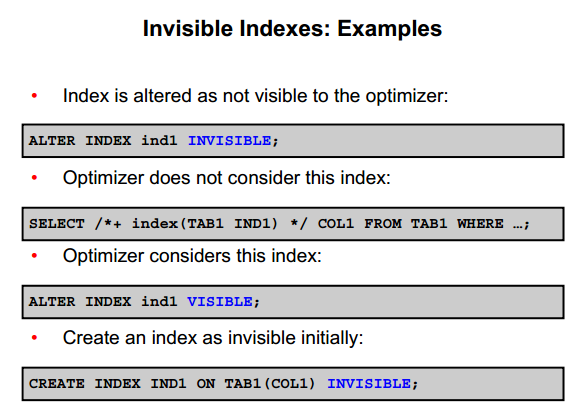


Invisible Index: Overview

Một index vô hình là một index bị lờ đi bởi trình tối ưu trừ khi bạn thiết lập tường minh OPTIMIZED\_USE\_INVISIBLE\_INDEXES khởi tạo tham số TRUE tại phiên làm việc hoặc mức hệ thống. Giá trị mặc định là FALSE.

Tạo một index vô hình là thay đổi để làm nó không phù hợp hoặc xóa nó. Sử dụng index vộ hình bạn có thể thực hiện các hành động:

* Kiểm tra dự dời đi của index trước khi xóa nó.
* Sử dụng cấu trúc index tạm thời cho một toán tử chắc chắc hay module của ứng dụng mà không làm ảnh hưởng tới toàn bộ ứng dụng.

Không giống như index không dùng được, index vô hình được bảo vệ trong câu lện DML.

Invisible Indexes: Examples

Khi index là vô hình, bộ tối ưu lựa chọn kế hoạch không dùng index. Nếu không thấy rõ nó ảnh hưởng tới hiệu năng, bạn có thể xóa index. Bạn cũng có thể tạo index và khởi tạo là vô hình, thực hiện test, và sau đó quyết định có tạo index hay không.

Bạn có thể truy vấn cột VISIBILITY của từ điển dữ liệu \*\_INDEXES để xác định index là VISIBLE hay INVISIBLE

Ghi chú: với tất cả các câu đưa ra trong slide, nó coi như OPTIMIZED\_USE\_INVISIBLE\_INDEXES được thiết lập là FALSE.

## Guidelines for Managing Indexes

Các nguyên tắc quản trị Indexes:

* Tạo index sau khi chèn dữ liệu vào bảng: Dữ liệu thường xuyên được chèn hay tải vào trong bảng. Sẽ là hiệu quả hơn khi tạo index cho bảng sau khi chèn hay load dữ liệu.
* Index the correct tables and colums: Sử dụng các hướng dẫn sau để xác định qhi tạo index:
* Tạo index nếu bạn thường xuyên muốn nhận ít hợp 15% số hàng trong một bảng lớn.
* Để cải thiện hiệu năng kho nối nhiều bảng, index cột dùng để nối.
* Bảng nhỏ không cần dùng index.
* colums suitable for indexing: một số cột là ứng cử viên mạnh mẽ để đánh index:
* Giá trị tương đối duy nhất trong cột.
* Có miền giá trị rộng (tốt cho regular index)
* Có miền giá trịnh nhỏ (tốt cho bitmap index)
* Cột có nhiều giá trị NULL, nhưng truy vấn thường chọn các hàng có giá trị.
* colums not suitable for indexing:
* Có nhiều giá trị NULL và bạn không tìm các giá trị không null.
* Cột LONG và LONG RAW không thể đánh index.
* Cột ảo: bạn có thể tạo index duy nhất hay không duy nhật cho cột ảo.
* Order index colums for performance: Thứ tự của cột trong câu lệnh CREATE INDEX có thể ảnh hưởng tới hiệu năng của câu truy vấn. Thông thường cột sử dụng nhiều nhất là cột đánh index đầu tiên.
* Limit the number of indexes for each table: một bảng có thể có nhiều index. Tuy nhiên, có nhiều index thì sẽ có nhiều chi phí xử lý khi mà bảng có sự thay đổi. Vì vậy cần có sự cân bằng giữa tốc độ phản hồi dữ liệu và tốc độ cập nhật lại bảng.

Xóa index là điều không được khuyến khích.

* Specify the tablespace for each index: Nếu bạn sử dụng cùng một không gian cho bảng và index của nó, nó co thể thuận lợi cho việc duy trì cơ sở dữ liệu, ví dụ như sao lưu cơ sở dữ liệu.
* Consider parallelizing index creation: bạn có thể thực hiện song song việc tạo index với tạo bảng. Điều đó sẽ tăng tốc độ tạo index. Tuy nhiên, mộn index được tạo với giá trị khởi tạo là 5M và nếu có 12 cột thì sẽ là 60MB dùng chỉ dùng để lưu trữ index trong quá trình khởi tạo.
* Consider creating indexs with NOLOGGING: bạn có thể tạo một index và sinh ra một đoạn Log với việc chỉ rõ NOLOGGING trong câu lện CREATE INDEX. Bởi vì index tạo ra sử dụng NOLOGGING sẽ không được lưu trữ. Nó sẽ được tạo sau khi bạn tạo xong bảng. NOLOGGING là giá trị mặc định trong cơ sở dữ liệu NOARCHIVELOG.
* Consider costs and benefits of coalescing or rebuilding indexes: Kích thước hay sự phát triển không phù hợp các index có thể làm chúng bị phân mảnh. Bạn có thể phải quan tâm tới chí để xóa và tạo index có hay không có ràng buộc UNIQUE hoặc PRIMARY KEY. Nếu bạn tạo liên kết với ràng buộc UNIQUE hay PRIMARY KEY đã quá lớn, bạn sẽ phải dành thời gian để kích hoạt các ràng buộc này hơn là xóa vào tạo lại một index lớn. Bạn phải lựa chọn rõ ràng khi mà bạn muốn xóa hay giữ hay vô hiệu hóa ràng buộc UNIQUE hay PRIMARY KEY.

## Investigating Index Usage

Bạn có thể thường xuyên thực thi câu lệnh SQL và mong rằng các index sẽ được sử dụng, và nó không được dùng. Điều đó có thể do bộ tối ưu không biết một số thông tin hoặc nó không nên sử dùng index.

* Functions : Hàm

Nếu bạn áp dụng một hàm vào cột đã được đánh index trong mệnh đề WHERE, index sẽ không được sử dụng. Index dựa trên cơ sở cột không được áp dụng cho hàm. Ví dụ truy vấn sau:

SELECT \* FROM employees WHERE 1.10\*salary > 10000

Nếu bạn muốn dùng index trong trường hợp này, bạn phải tạo một index cho hàm này.

* Data type mismatch: Kiểu dữ liệu không tương ứng.

Có nhiều sự không tương xứng với kiểu dữ liệu được định nghĩa của cột và dữ liệu khi truy vấn. Khi đó index sẽ không được sử dụng.

Có một sự chuyển đổi ngầm kiểu dữ liệu dẽ được thực hiện và khi đó giống như ta đã dùng hàm.

Ví dụ: cột SSN có kiểu dữ liệu là VARCHAR2, ta có truy vấn:

SELECT \* FROM person WHERE SSN = 123456789.

* Old Statistics: Thống kê cũ

Thống kê cũ là trường hợp bộ tối ưu quyết định sử dụng index, tuy nhiên index này đã lỗi thời so với CSDL hiện tại do đó nó sẽ ảnh hưởng tới quyết định của bộ tối ưu khi sử dụng index.

* NULL colums:

Nếu một cột có thể có giá trị NULL, nó có thể ảnh hưởng tới index trên hàng đó.

* Slower index:

Thỉnh thoảng, việc sử dụng index lại không có hiệu quả.