Trường đại học Bách Khoa Hà Nội

Viện Công Nghệ Thông Tin và Truyền Thông

BÁO CÁO BÀI TẬP LỚN

Học phần: Thiết kế và quản trị CSDL

Chương 6: Các phép toán tối ưu

Nhóm sinh viên thực hiện:

Tạ Công Sơn	20112083
Vũ Mạnh Kiểm	20112731
Lê Quyết Thắng	20112226
Nguyễn Khắc Nhất	20111938

Giáo viên hướng dẫn: Ts Trần Việt Trung

Lời nói đầu.

Có thể nói công nghệ thông tin là một ngành còn non trẻ so với nhiều lĩnh vực khác của đời sống xã hội loài người. Công nghệ thông tin bắt đầu phát triển mạnh mới từ năm 1996 nhưng sự phát triểu của nó là theo cấp số nhân. Thực sự sức mạnh của công nghệ thông tin là không gì có thể ngăn cản. Nó đi sâu và trợ giúp các lĩnh vực khác trong đời sống xã hội. Và theo như tác giả của cuốn sách "Thế giới phẳng" - Thomas Friedman thì công nghệ thông tin là nhân tố tạo nên thế giới phẳng 3.0. Vì vậy việc học tập, nghiên cứu và phát triển trong lĩnh vực công nghệ thông tin là một nhu cầu, một nhiệm vụ và môt hướng đi tất yếu của giới trẻ. Nó không chỉ giúp cho các bạn phát triển bản thân mà còn giúp cho đất nước tiến bộ và phát triển hơn.

Ngày nay yêu cầu về công nghệ thông tin không còn chỉ là hoạt động được và phục vụ được như giai đoạn đầu tiện nữa. Yêu cầu với mỗi hệ thống hiện nay là đáp ứng được đúng yêu cầu người dùng với thời gian tốt nhất, do đó các yêu cầu về tối ưu tổ chức hệ thống, tối ưu cách thức tổ chức cơ sở dữ liệu được đặt lên hàng đầu. Rất nhiều công trình nghiên cứu, các hướng đi mới trong cơ sở dữ liệu đã được đưa ra và áp dụng hiệu quả vào thực tế. Việc nghiên cứu về cơ sở dữ liệu vẫn đã, đang và sẽ là con đường rộng mở, hấp dẫn và cũng đầy thử thách với sinh viên cũng như những kỹ sư, chuyên gia công nghệ thông tin. Do đó việc nắm rõ các kiến thức cơ bản về cơ sở dữ liệu, hiểu, làm chủ và tùy biến được nó là vô cùng quan trọng đố với sinh viên hiện nay.

Môn học thiết kế và quản trị cơ sở dữ liệu đã cung cấp cho chúng em rất nhiều kiến thức bổ ích về việc tối ưu cách thức tổ chức lưu trữ và tối ưu đối với cơ sở dữ liệu. Để nắm vững lý thuyết cũng như hiểu sâu hơn bản chất vấn đề, chúng em đã dịch và nghiên cứu tài liệu của Hệ quản trị cơ sở dữ liệu Oracle về các phép toán tối ưu đối với việc truy vấn dữ liệu. Nội dung kiến thức thu được sau quá trình tìm hiểu này thực sự rất bổ ích với chúng em. Hy vọng những nội dung chúng em tìm hiểu và trình bày sẽ đáp ứng được sự kỳ vọng của thầy và các bạn trong lớp. Đồng thời chúng em cũng muốn trình bày để các nhóm khác nghiên cứu các đề tài khác nhau có thể trao đổi và hoàn thiện nội dung nghiên cứu, cùng nhau đóng góp và bổ sung được kiến thức.

Hà Nội, tháng 5 năm 2015

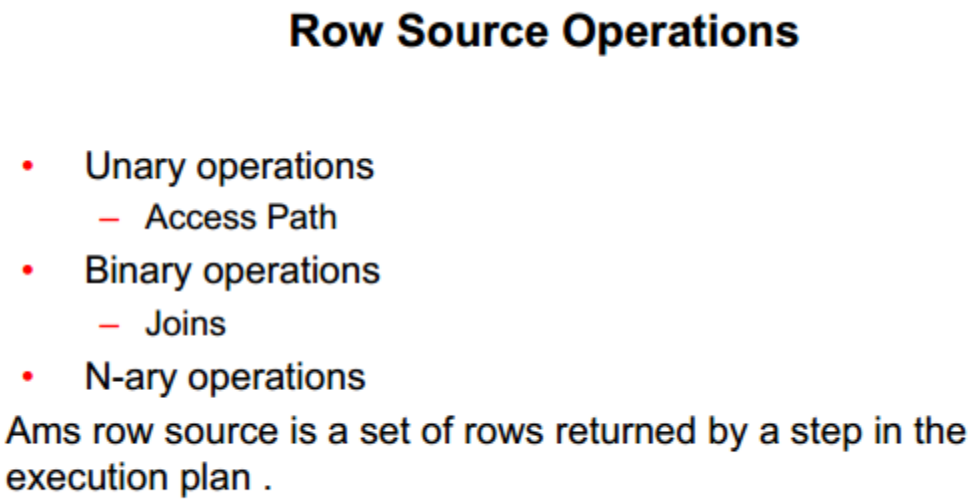
Nhóm sinh viên thực hiện đề tài.
Nhóm 3

Contents

Phân công công việc

Thành viên	Công việc
Lê Quyết Thắng	Dịch chương I →VI 1
Tạ Công Sơn	Dịch chương VI 2 → VI 6
Nguyễn Khắc Nhất	Dịch chương VI 7 → VI 11
Vũ Mạnh Kiểm	Dịch chương VI 12 → VI 16

I. Row source Operation



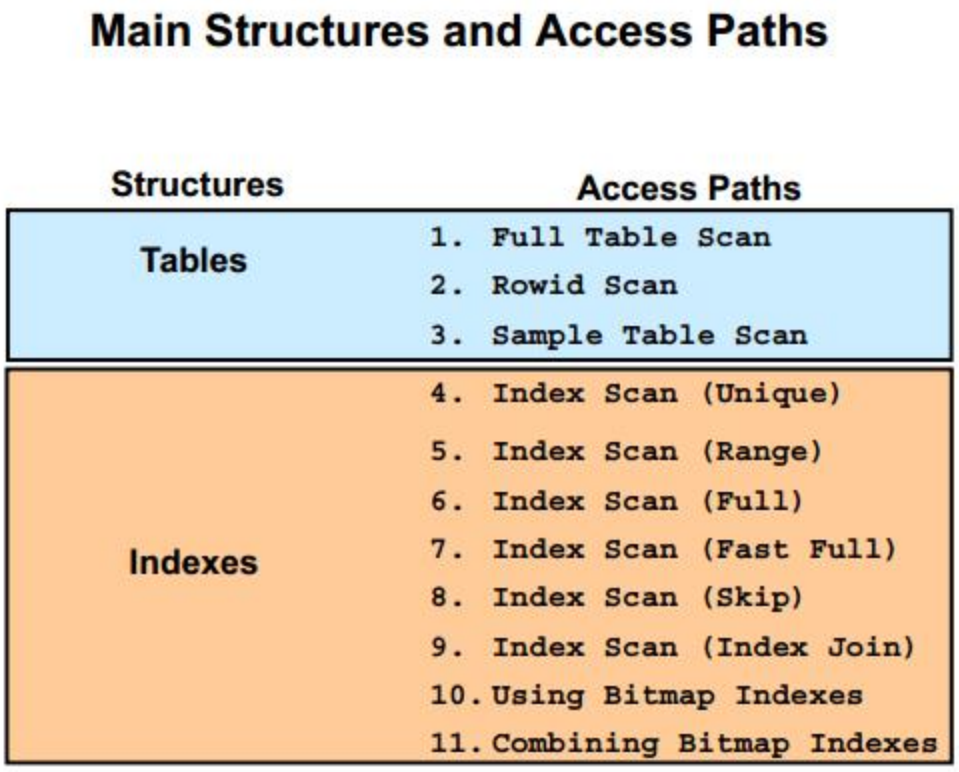
Một row source là một tập các hàng được trả về sau mỗi bước trong kế hoạch truy vấn. Row source có thể là một bảng, một phần của bảng hoặc là kết quả của phép join hay phép group.

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.

Cách truy nhập (Access Path): là cách mà dữ liệu được lấy ra từ cơ sở dữ liệu. Thông thường, cách 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 các phần tử của bảng.

II. Main Structures and Access Paths

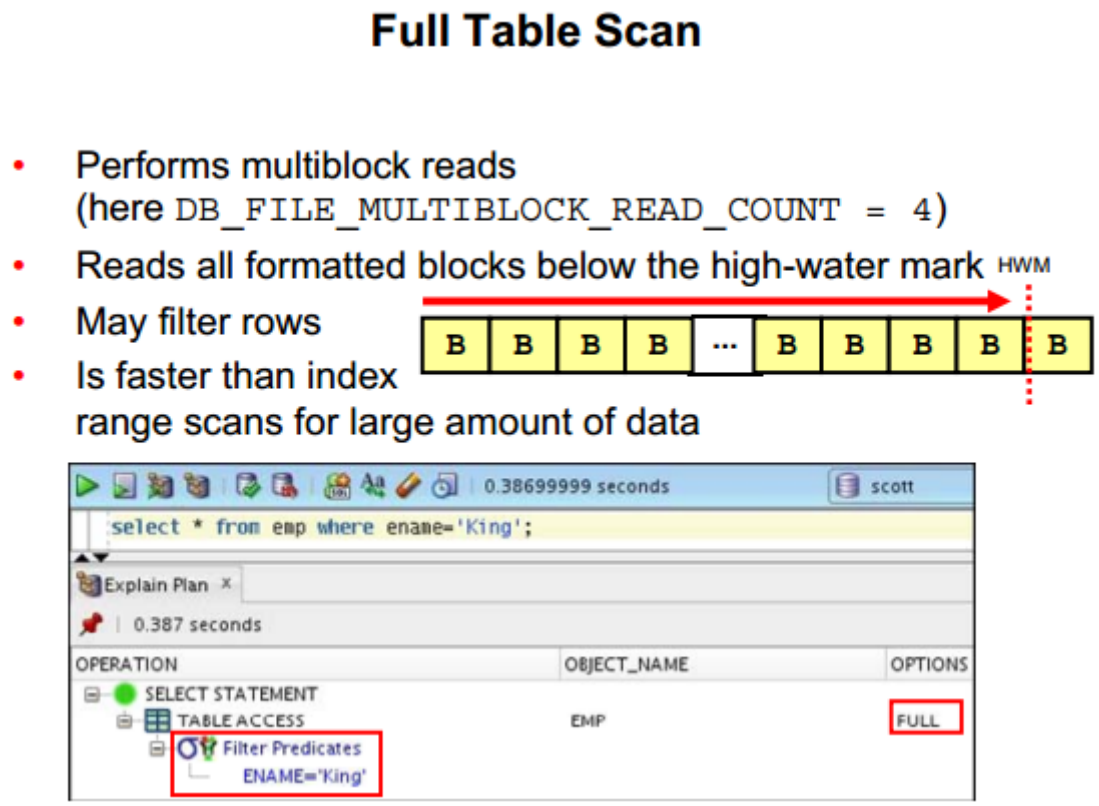


Bất cứ một hàng nào đều có thể được xác định và lấy ra bởi một trong các cách trên. Thông thường, các truy nhập bằng index nên được sử dụng khi muốn lấy ra một tập nhỏ các hàng trong bảng, trong khi full scan hiệu quả khi truy nhập một phần lớn của bảng. Để quyết định xem kế hoạch truy vấn nào được chọn,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

Có một số loại truy vấn đặc biệt đối với 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.

III. Full Table Scan



Khi thực hiện full table scan (FTS), hệ thống 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 thấp hơn high-water mark (HWM) 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.

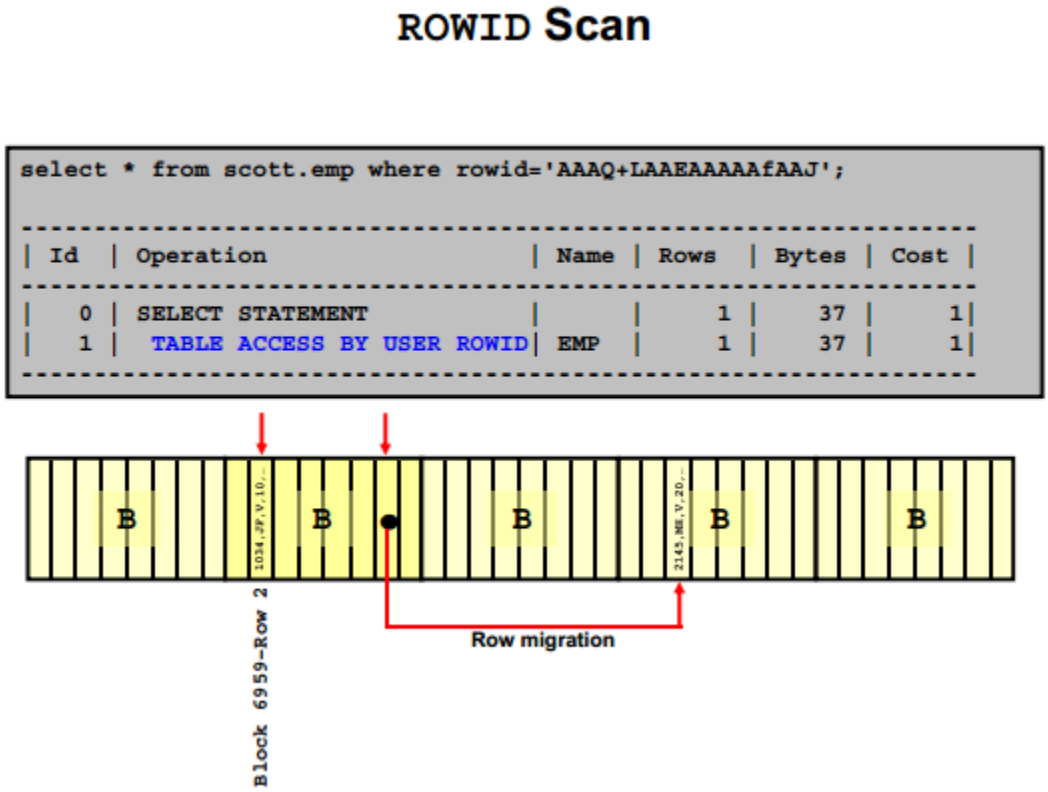
- Chú ý: trong Oracle 6, FTS có thể gây tràn bộ nhớ cache, nhưng kể từ oracle 7 FTS chỉ được chiếm một phần nhỏ của cache. Ngày nay trong hầu hết các trường hợp, FTS được đọc thẳng vào PGA (program global data) thông qua cache.
- High-water mark: là 1 điểm cao nhất mà data đã từng lưu tới trong mỗi segment. HWM không bị thay đổi khi delete data, nhưng sẽ tăng lên khi insert thêm data.

- DB_FILE_MULTIBLOCK_READ_COUNT: Số Block tối đa có thể đọc trong một lần truy suất dữ liệu.

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 có một bộ lọc ROWID hoặc một cách truy nhập cluster khả dụng).
- 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.

IV. 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.

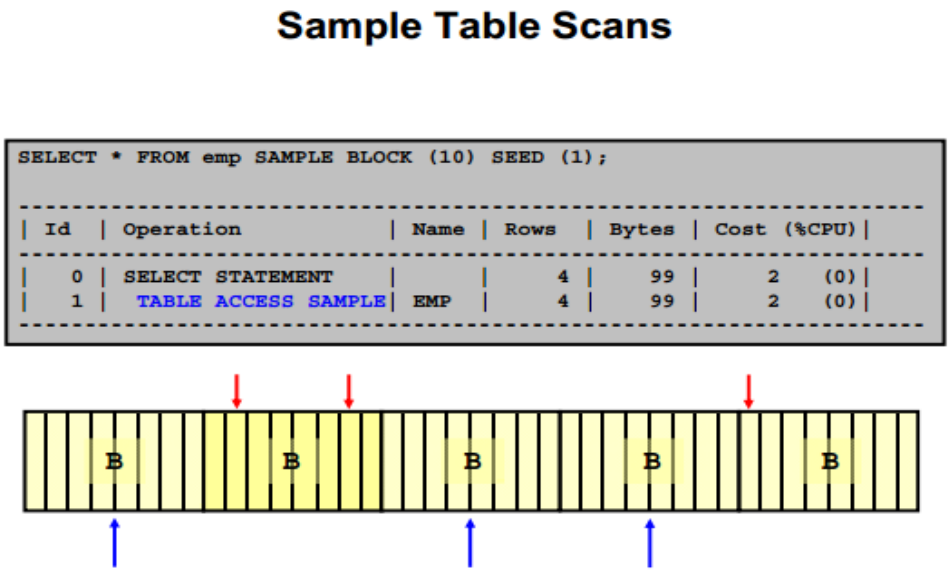
Để truy nhập bảng bằng rowid, hệ thống đầu tiên lấy rowid của các hàng được chọn hoặc từ mệnh đề WHERE hay thông qua index scan một hay nhiều index của bảng. Hệ thống sẽ các định vị trí các hàng được chọn trong bảng dựa theo rowid của chúng.

Bộ tối ưu phần lớn sử dụng rowid sau khi lấy chúng từ một index. 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: bởi vì sự di chuyển, một rowis có thể đôi lúc trỏ tới một địa chỉ khác với vị trí thực tế của hàng, kết quả 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.

V. 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 bởi 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 hàng 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.

VI. Indexes

1. Overview

Indexes: Overview

Indexes

- Storage techniques:
  - B*-tree indexes: The default and the most common
    - Normal
    - Function based: Precomputed value of a function or expression
    - Index-organized table (IOT)
  - Bitmap indexes
  - Cluster indexes: Defined specifically for cluster
- Index attributes:
  - Key compression
  - Reverse key
  - Ascending, descending
- Domain indexes: Specific to an application or cartridge

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 cung 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 hơn. Các index có thể là độc nhất hoặc giống nhau.

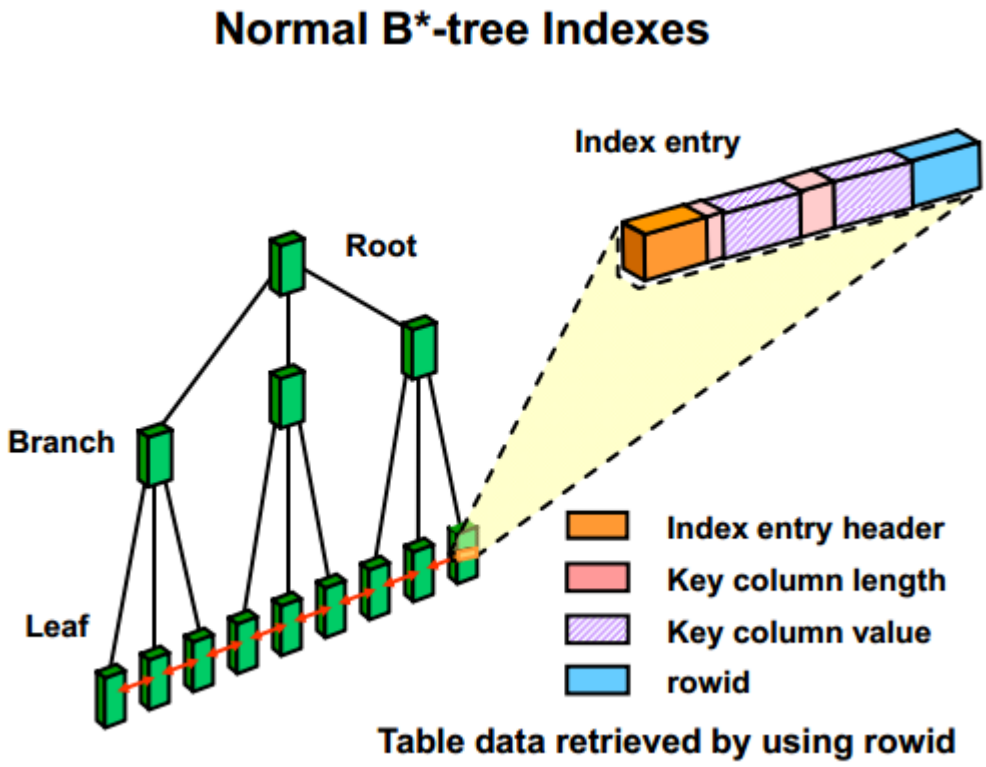
Một index hỗn hợp, hay còn gọi là index nối, là một index mà bạn tạo trong nhiều cột trong một bảng. Các cột trong index hỗn hợp có thể xuất hiện theo bất kỳ thứ tự nào và không cần nằm kề bảng.

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 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 tạo trên các cột cuối cùng trong khóa index và là thành phần độc nhất của 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 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 nhớ, 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.

2. 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.

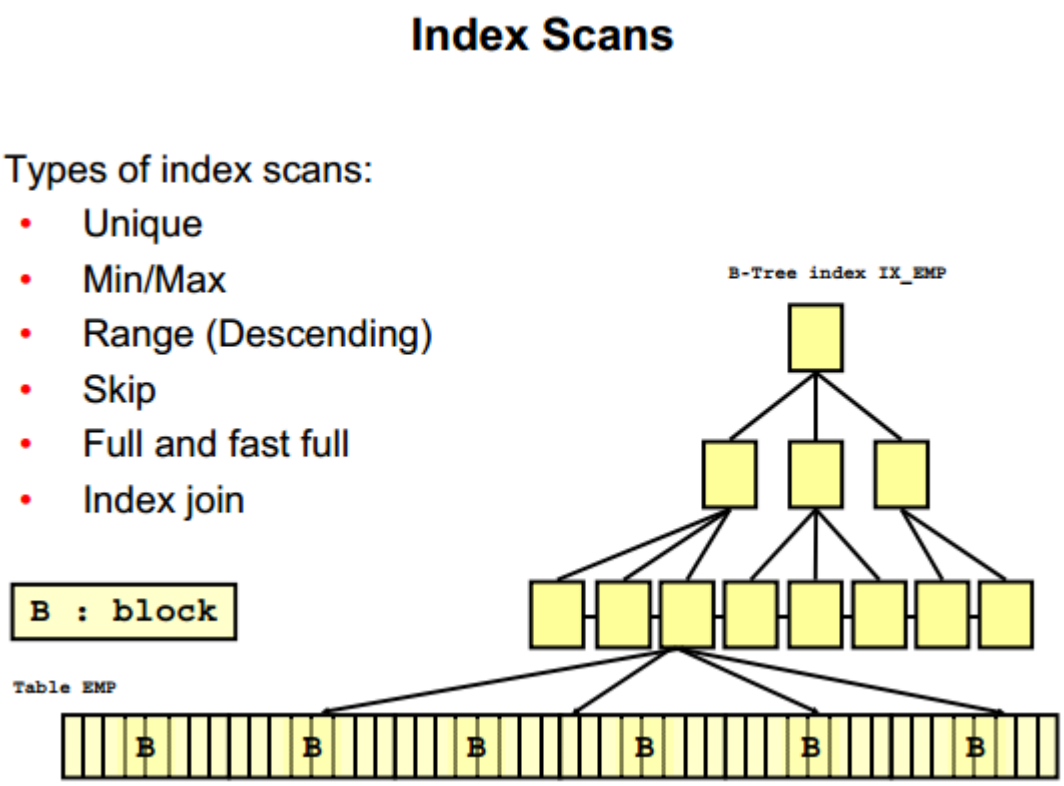
Con trỏ block trước và sau kết nối các block lá để chúng có thể dịch chuyển từ trái qua phải hoặc ngược lại.

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.

3. Index Scans



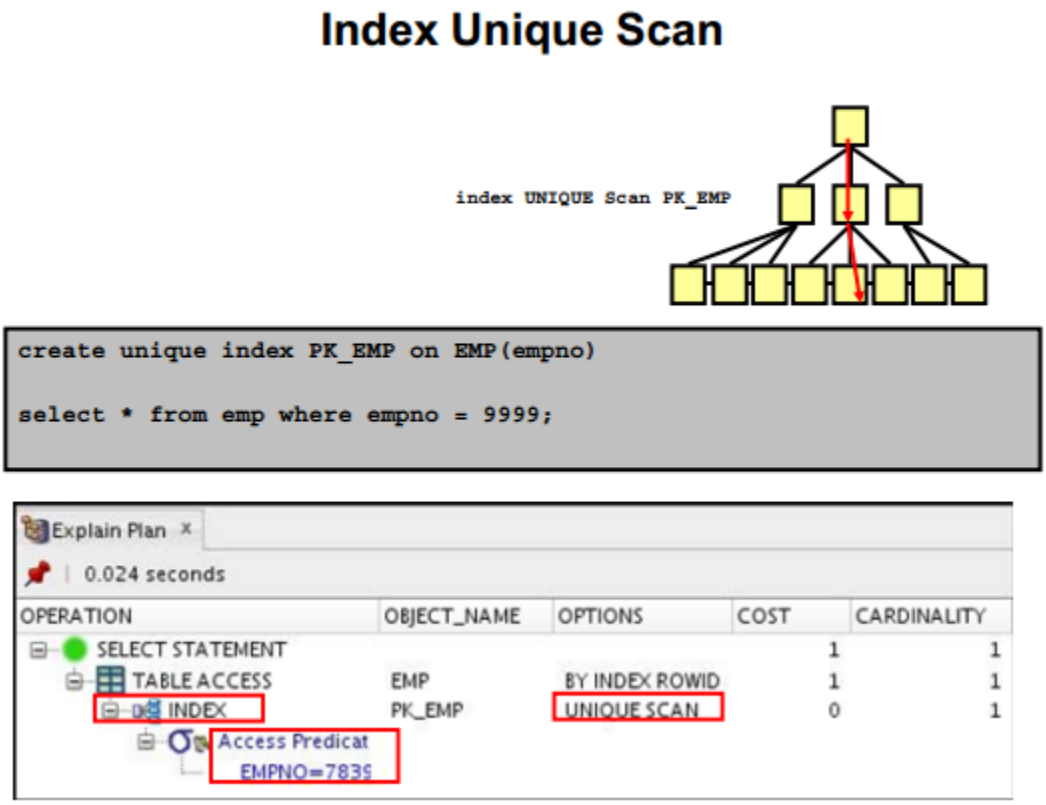
Một index scan có thể có những kiểu sau:

Một hàng được lấy ra nhờ đi qua các index, sử dụng các giá trị (của 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 và 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.

4. Index Unique Scan



Trong hầu hết trường hợp, một index unique scan trả về một 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.

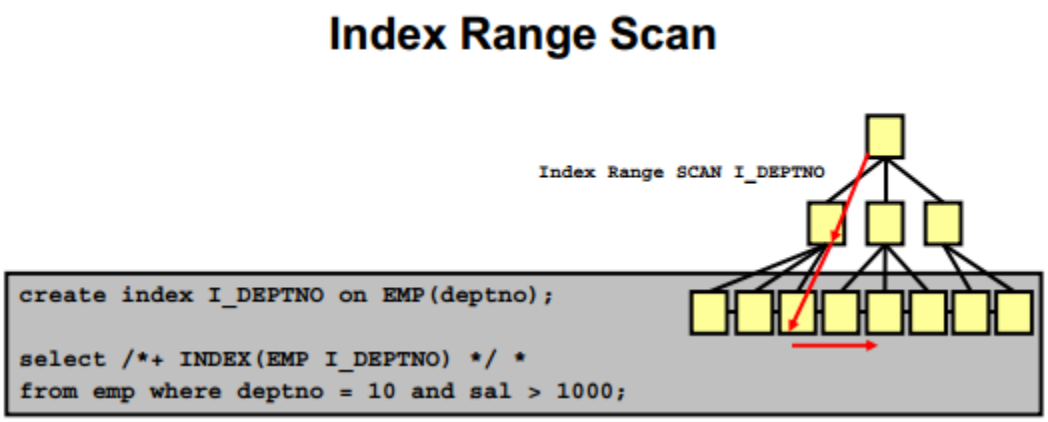
Giá trị key và các ROWID có được từ index và các hàng của bảng có được nhờ sử dụng ROWID.

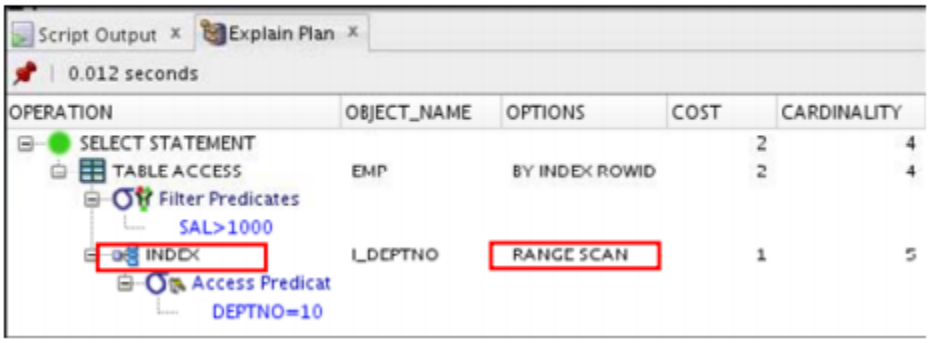
Có thể tìm các điều kiện truy nhập trong phần "thông tin vị từ" của kế hoạch truy vấn.

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

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

5. Index Range Scan





Index Range Scan là thao tác phổ biến nhất trong việc truy cập dữ liệu có chọn lọc. Dữ liệu trả về sẽ được xắp xếp tăng dần theo giá trị của cột được đánh chỉ mục. Khi có nhiều hàng có cùng giá trị, kết quả sẽ được sắp xếp theo thứ tự tăng dần của ROWID

Bộ tối ưu sử dụng Index Range Scan khi nó tìm thấy một hoặc nhiều cột đầu tiên (leading column) trong cấu trúc chỉ mục thỏa mãn điều kiện xuất hiện trong điều kiện của mệnh đề WHERE như col1 =: b1, col1 <: b1, col1 >: b1 và kết hợp với bất kì điều kiện preceding conditions. (Nếu có cấu trúc chỉ mục được định nghĩa trên 3 cột (a,b,c) thì a là "leading column", (a,b) là leading columns)

Không nên tìm kiếm với ký tự đại diện ở vị trí đầu tiên như col1 like '%ASD' (tìm kiếm với ký tự kết thúc bằng "ASD") vì nó không phải là kết quả từ một phép range scan.

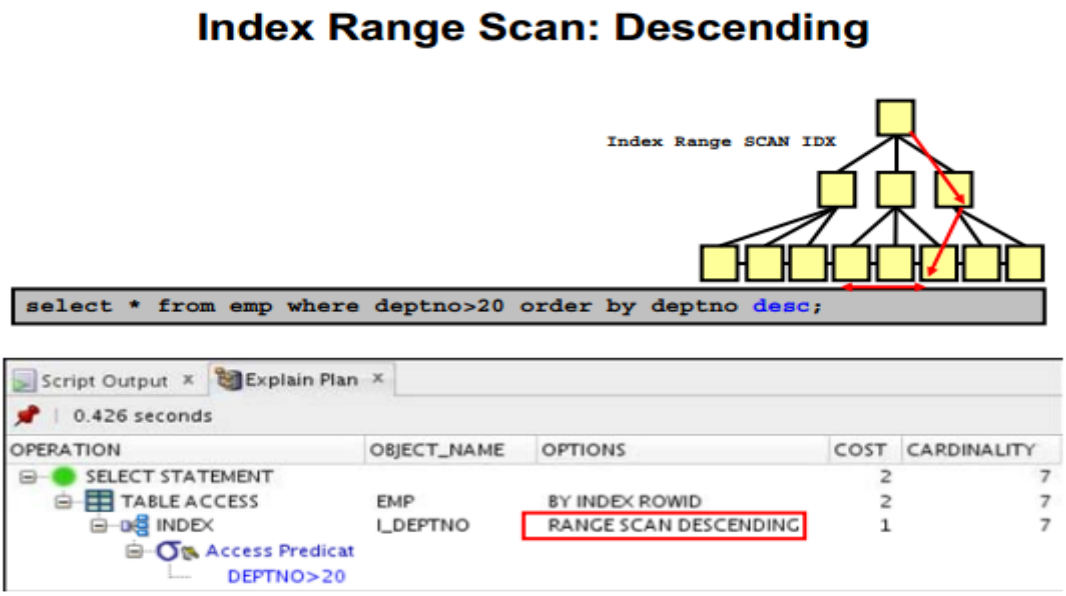
Range scan có thể sử dụng cấu trúc chỉ mục unique or nonunique indexes. Range scan có thể tránh được việc xắp xếp khi sử dụng truy vấn với mệnh đề ORDER BY/GROUP BY trên các cột được đánh chỉ mục và cột được đánh chỉ mục phải là NOT NULL nếu không sẽ được bỏ qua.

Index range scan theo thứ tự giảm dần giống với index range scan khi dữ liệu trả về được yêu cầu xắp xếp theo thứ tự giảm dần. Bộ tối ưu sẽ sử dụng cấu trúc chỉ mục index range scan descending khi có một mệnh đề ORDER BY theo thứ tự giảm dần của trường đã được đánh chỉ mục.

Trong ví dụ trong slide, sử dụng chỉ mục I_DEPTNO, hệ thống truy cập vào hàng có EMP.DEPTNO=10. Hệ thống sẽ lấy ROWIDs của những cột đó và lấy giá trị của những cột khác trong bảng EMP, sau đó thực hiện điều kiện EMP.SAL >1000 với các cột tìm được sau đó trả lại kết quả cho người dùng.

*VD : SELECT **
FROM employees
WHERE department_id = 20
AND salary > 1000;

Index Range Scan: Descending

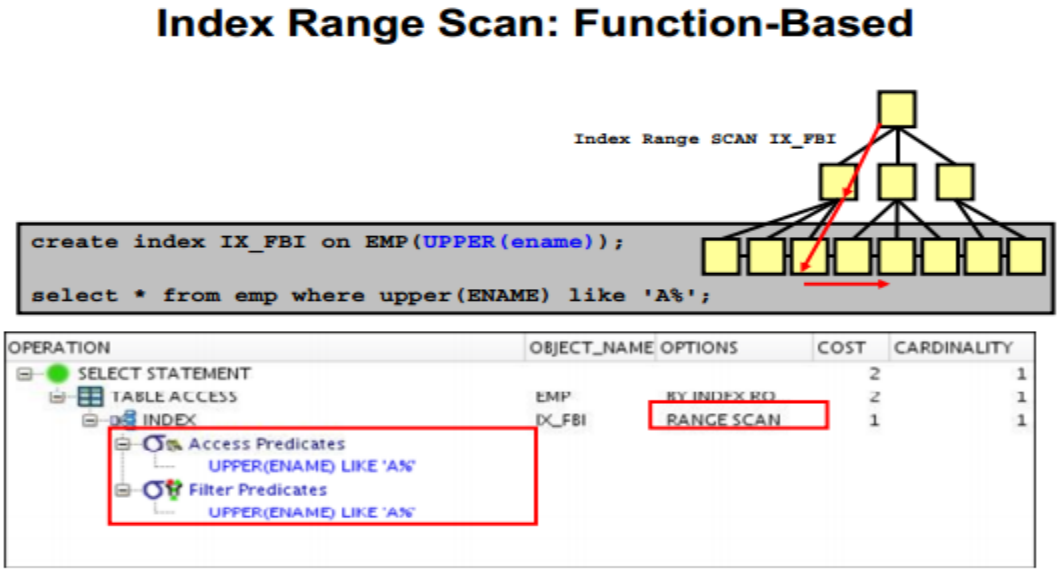


- Mặc định index range scans là thực hiện xắp xếp theo thứ tự tăng dần.
- Index range scan xắp xếp kết quả theo thứ tự giảm dần giống với index range scan khi dữ liệu trả về được xắp xếp theo thứ tự giảm dần. Descending indexes cho phép dữ liệu được xắp xếp từ lớn tới bé (descending) thay vì từ nhỏ tới lớn (ascending).
- Bộ tối ưu sẽ sử dụng index range scan descending khi câu truy vấn có sử dụng mệnh đề ORDER BY nhằm xắp xếp kết quả theo thứ tự giảm dần của trường đã được đánh chỉ mục.
- NOTE : hệ thống coi chỉ mục giảm dần như là cấu trúc chỉ mục "function-based" indexes. Những cột được đánh dấu DESC được lưu trữ theo một thứ

tự giảm dần đặc biệt bằng cách nghịch đảo lại cấu trúc chỉ mục bằng cách sử dụng hàm SYS_OP_UNDESCEND.

*VD : SELECT **
FROM employees
WHERE department_id < 20
ORDER BY department_id DESC;

Index Range Scan: Function-Based

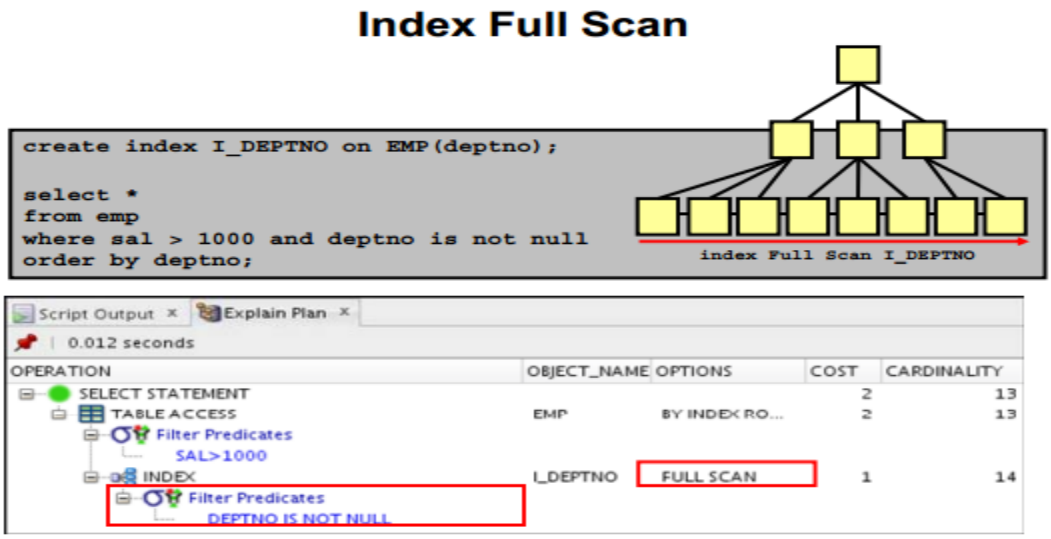


- Một chỉ mục Function-Based có thể được lưu trữ như một cấu trúc B*-tree hay bitmap. Những chỉ mục này bao gồm các cột được biến đổi bởi một hàm (như hàm UPPER) hoặc bao gồm các biểu thức (như col1 + col2). Với một chỉ mục Function-Based ta có thể lưu trữ biểu thức computation-intensive trong cấu trúc chỉ mục.
- Định nghĩa một cấu trúc chỉ mục Function-Based biến đổi một cột hoặc biểu thức là cho phép dữ liệu đươc trả về khi sử dụng cấu trúc chỉ mục khi hàm hoặc biểu thức được sử dụng trong mệnh đề WHERE hoặc mệnh đề GROUP BY. Điều này cho phép hệ thống bỏ qua việc tính toán của biểu thức khi xử lý câu truy vấn SELECT hay DELETED. Do đó, chỉ mục Function-Based mang lại nhiều lợi ích khi các câu truy vấn SQL bao gồm các cột đã được biến đổi hay biểu thức trong cột trong mệnh đề WHERE hoặc ORDER BY được sử dụng thường xuyên.

  Ví dụ : SELECT * FROM Student, Employees WHERE sid + eid = 5

- Khi sử dụng Function-Based index sẽ lưu trữ như B-tree thông thường, các node của cây là giá trị của "sid + edi", các node lá sẽ lưu con trỏ trỏ đến các bản ghi tương ứng trên disk của 2 table Student và Employees.

6. Index Full Scan



Một thao tác Full scan sẵn sàng được sử dụng nếu một thuộc tính trong câu truy vấn tham chiếu đến một cột được đánh chỉ mục tuy nhiên cột đó không phải điều kiện chính trong điều kiện của mệnh đề WHERE hoặc truy vấn bao gồm mệnh đề ORDER BY trên cột được đánh chỉ mục với điều kiện cột đó là NOT NULL. Thao tác full scan cũng có thể được thực hiện nếu tất cả các điều kiện sau đều thỏa mãn:

- Tất cả các cột trong bảng được tham chiếu trong câu truy vấn đều được đánh chỉ mục.
- Có ít nhất một cột được đánh chỉ mục là NOT NULL.

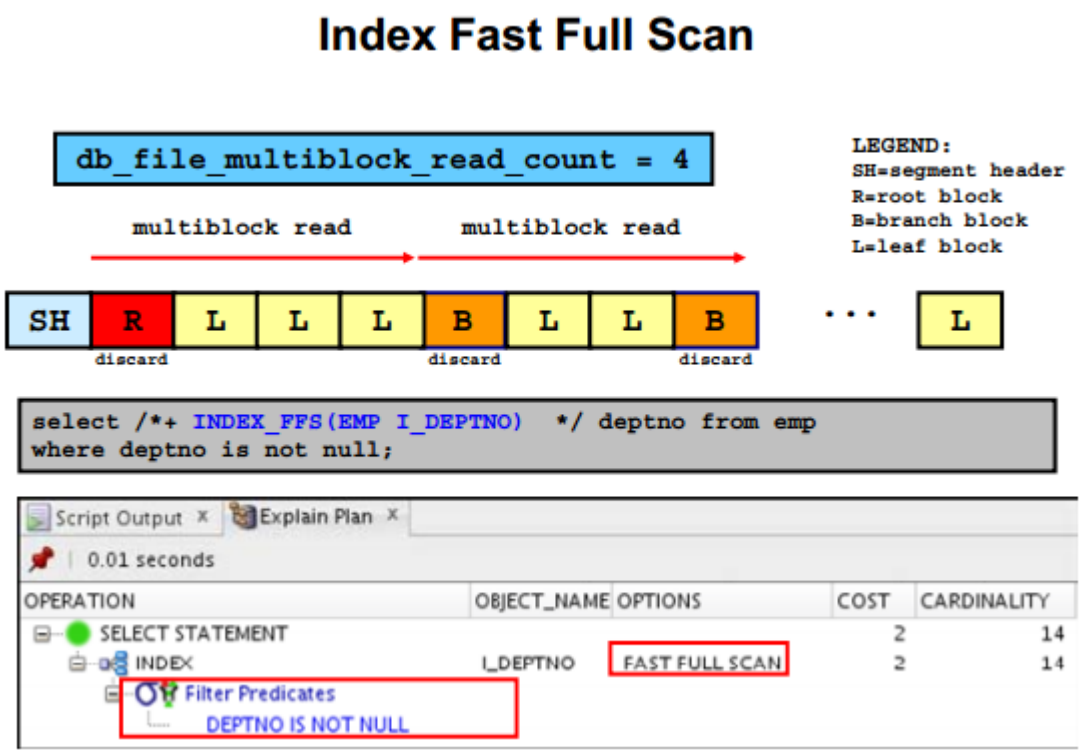
Một thao tác full scan có thể được sử dụng để loại bỏ thao tác xắp xếp bởi vì dữ liệu được xắp xếp bởi giá trị khóa của index.

VD1 : SELECT department_id, department_name
FROM departments
ORDER BY department_id;

*VD2 : SELECT **

FROM emp
WHERE sal > 1000 AND deptno IS NOT NULL
ORDER deptno;

Index Fast Full Scan



Index Fast Full Scan là sự lựa chọn thay cho full table scans trong trường hợp cấu trúc chỉ mục chứa tất cả các cột cần cho câu truy vấn và ít nhất một cột trong cấu trúc chỉ mục có giá trị khóa với ràng buộc là NOT NULL. Một Fast full scan sẽ truy cập dữ liệu trực tiếp trong cấu trúc chỉ mục mà không cần truy cập đến cấu trúc bảng.

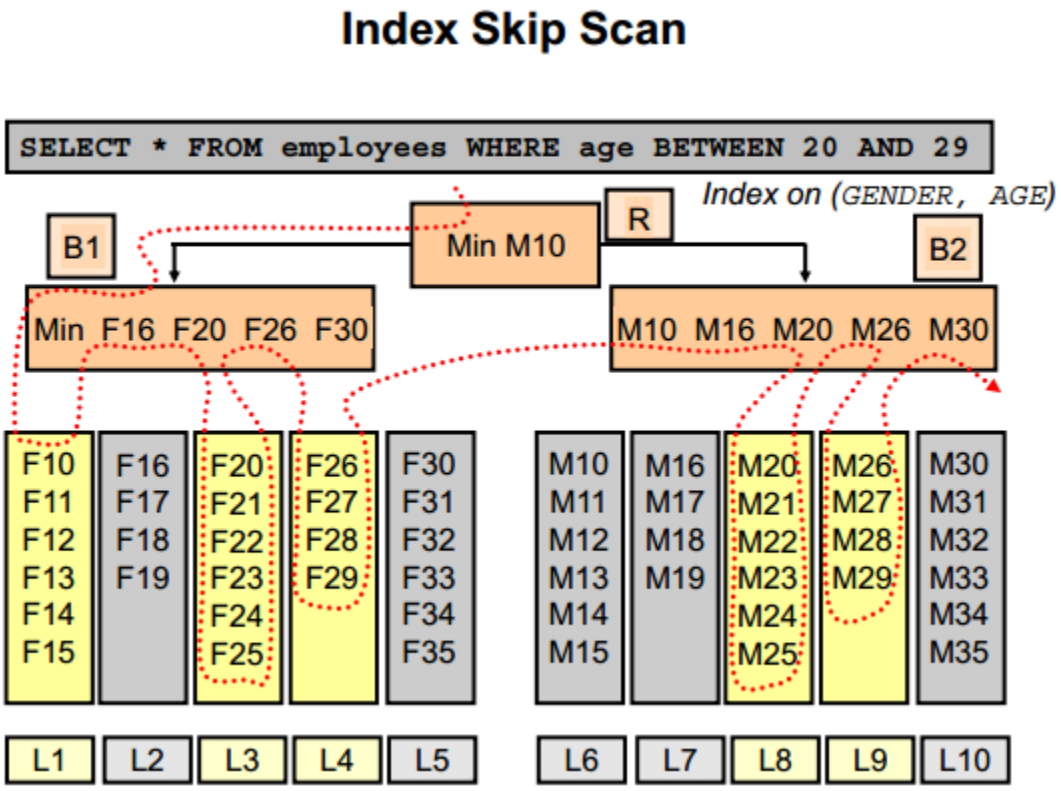
Index Fast Full Scan không thể dùng để loại bỏ thao tác xắp xếp bởi vì dữ liệu không đươc xắp xếp theo giá trị khóa của chỉ mục. Nó có thể được sử dụng cho các hàm min/avg/sum. Trong trường hợp này, bộ tối ưu cần biết rằng tất cả các hàng của bảng được đại diện bởi cấu trúc chỉ mục.

VD : SELECT /+ INDEX_FFS(departments dept_id_pk) */ COUNT(*)*
FROM departments;

➔ Ở đây bảng departments được đánh chỉ mục trên trường thuộc tính dept_id. Việc đếm số lượng phần tử của bảng departments chỉ cần đếm các nút của cấu trúc chỉ mục.

➔/*+ INDEX_FFS(departments dept_id_pk) */ : hint này bắt bộ tối ưu phải lựa chọn thao tác Index fast full scan để thực hiện câu truy vấn.

7. Index Skip Scan



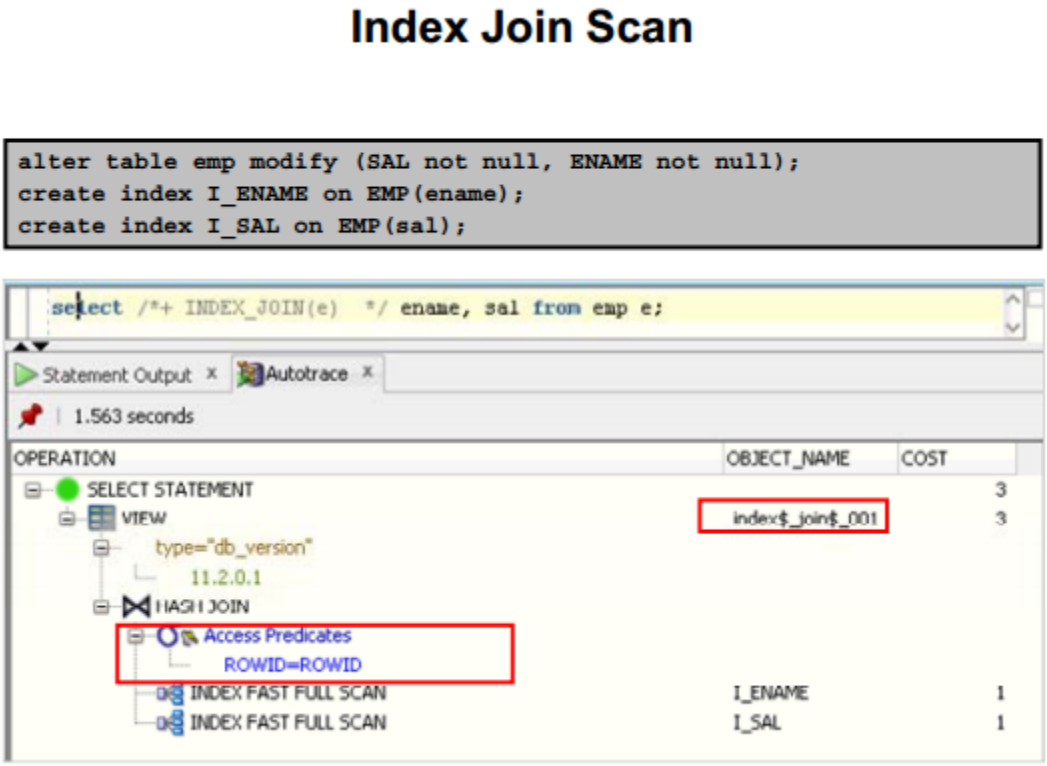
Index skip scan giúp cải thiện tốc độ khi sử dụng cơ chế index scan bằng cách bỏ qua các khối chứa các giá trị khóa index không thỏa mãn với điều kiện của câu truy vấn. Scan index theo khối thường nhanh hơn so với thao tác scan theo khối trên bảng dữ liệu.

Giả sử có một chỉ mục ghép được tạo từ 2 trường GENDER và AGE của bảng EMPLOYEES. Ví dụ trong hình mô phỏng xử lý thao tác skip scanning khi xử lý câu truy vấn. (Fx : nữ, x tuổi ; My : nam, y tuổi)

- Hệ thống bắt đầu scan tại nút gốc R và đi theo các nhánh theo chiều từ trái qua phải. Nút tiếp theo được xét là B1. Từ đây hệ thống xét giá trị đầu tiên của nút con nút MIN , đọc tiếp giá trị nút tiếp theo là F16 và giá trị đầu tiên trong khối bắt đầu bởi nút MIN là F10, và thấy rằng khối bắt đầu bởi nút F16 không thể chứa giá trị trong khoảng 20-29 vì vậy nút lá L1 được bỏ qua.

- Hệ thống quy lui xét nút tiếp theo là F16, đọc tiếp giá trị nút tiếp theo là F20, và thấy rằng khối bắt đầu bởi nút F16 không thể chứa giá trị trong khoảng 20-29 vì vậy nút lá L2 được bỏ qua.
- Hệ thống quy lui lại xét nút con có giá trị F20 và F29 thấy thỏa mãn và quét lấy giá trị của các nút lá L3 và L4.
- Sau khi quét hết giá trị của các khối có các giá trị bắt đầu là F20 và F29 hệ thống xác định không còn giá trị nào có giá trị trong khoảng từ 20 đến 29 và quay lui lên xét nhánh B2.
- Hệ thống thực hiện việc quét giá trị tương tự với nhánh B2.

8. Index Join Scan



Index Join Scan là phép kết nối một vài cấu trúc chỉ mục cùng chứa tất cả các cột của bảng được tham chiếu trong các truy vấn

Index Join Scan cho phép không cần truy xuất dữ liệu trên bảng để lấy giá trị của cột được tham chiếu trong câu truy vấn, thay vào đó có thể sử dụng cấu trúc chỉ mục để lấy dữ liệu của cột tương ứng.

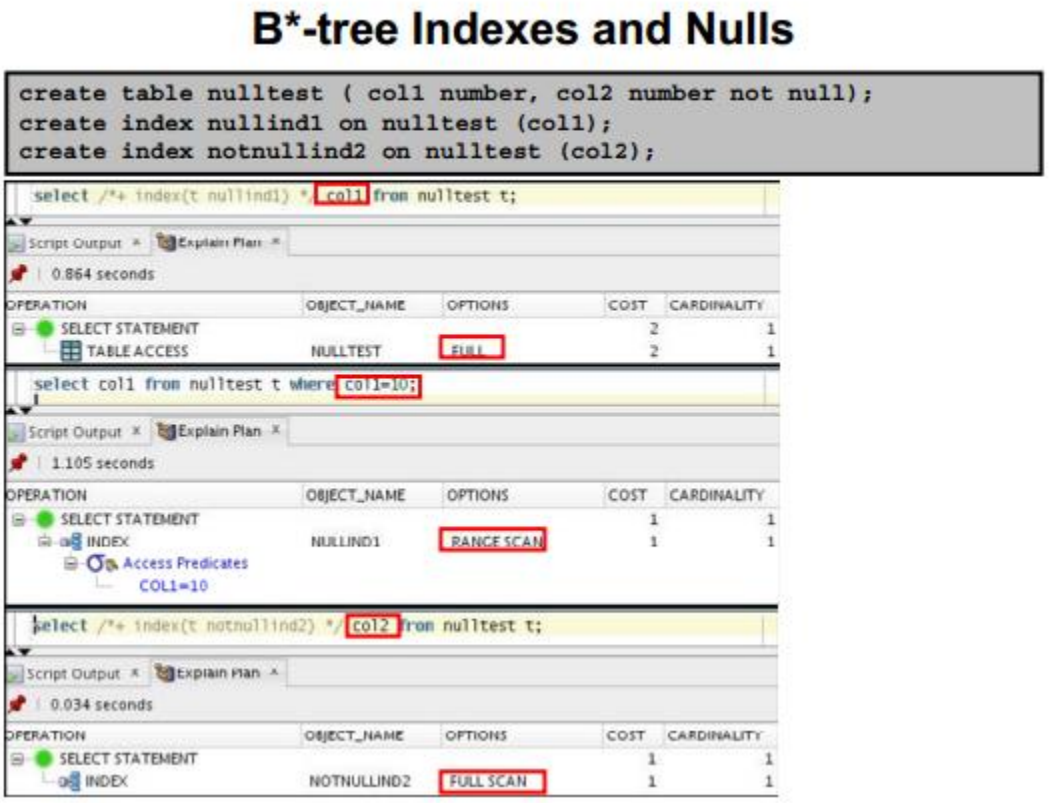
- VD : alter table emp modify (SAL not null, ENAME not null);
  create index I_ENAME on EMP(ename);

create index I_SAL on EMP(sal);

- select /*+ INDEX_JOIN(e) */ ename , sal, from emp e;
- "/*+ INDEX_JOIN(e) */" yêu cầu bộ tối ưu sử dụng phép Index Join Scan do 2 trường ename và sal đã được đánh chỉ mục, nên bộ tối ưu sẽ sử dụng

Index Fast Full Scan để quét toàn bộ các giá trị của 2 trường này và sử dụng ROWID của chúng để làm điều kiện cho phép kết nối (tức 2 node trên 2 cấu trúc chỉ mục có cùng ROWID thì khi đó key value của node đó là thỏa mãn)

9. B*-tree Indexes and Nulls

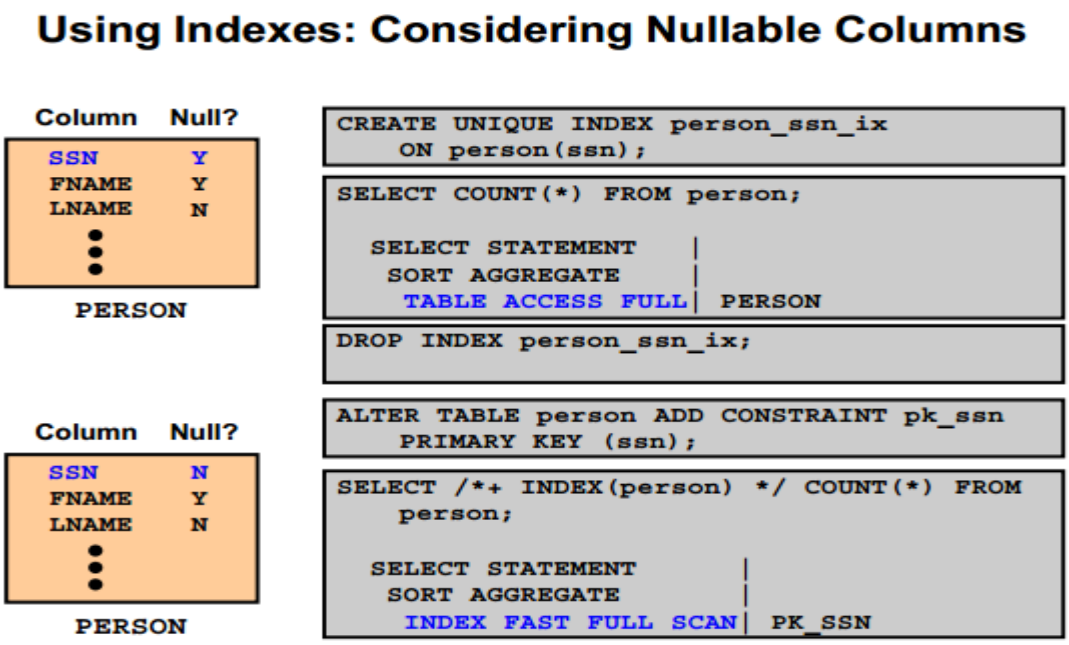


Đây là lỗi phổ biến khi sử dụng cấu trúc B*-tree Indexes trên cột có thể có giá trị NULL xuất hiện. Cấu trúc B*-tree Indexes không cho phép lưu trữ giá trị NULL vì vậy cột được đánh chỉ mục sử dụng B*-tree Indexes sẽ không được sử dụng thực hiện câu truy vấn trừ khi có điều kiện loại bỏ giá trị NULL trong câu truy vấn

- Ở trong câu truy vấn đầu tiên do COL1 có thể chứa giá trị NULL do vậy cấu trúc chỉ mục không được sử dụng do đó bộ tối ưu đã lựa chọn chiến lược FULL SCAN TABLE cho câu truy vấn này

- Tuy nhiên ở câu truy vấn thứ 2, điều kiện col1 = 10 đã loại bỏ giá trị NULL có thể loại bỏ giá trị NULL từ dữ liệu trả về của cột trong câu truy vấn, do đó cấu trúc chỉ mục sẽ vẫn được thực hiện, và ở đây bộ tối ưu sử dụng chiến lược Index Range Scan
- Ở truy vấn thứ 3, do COL2 chứa điều kiện NOT NULL nên cấu trúc chỉ mục tại COL2 sẽ được sử dụng cho câu truy vấn, và ở đây bộ tối ưu sử dụng chiến lược Index Full Scan Index.

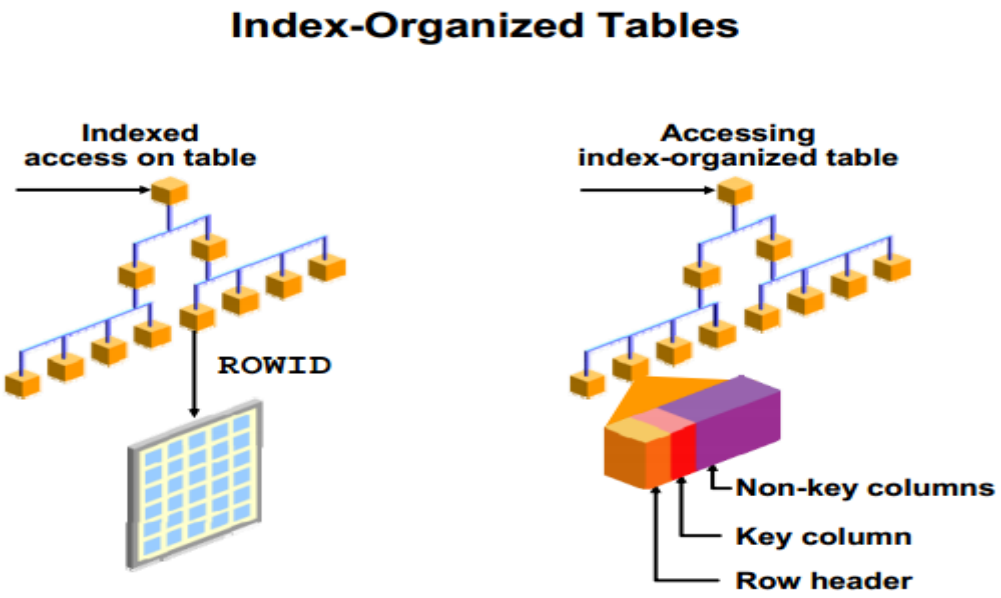
10. Using Indexes: Considering Nullable Columns



Một số câu truy vấn đơn giản như đếm số hàng trong bảng (sử dụng hàm COUNT) sử dụng cấu trúc chỉ mục thường hiệu quả hơn so với việc scan toàn bảng. Tuy nhiên cấu trúc chỉ mục B*-tree indexes không được sử dụng cho câu truy vấn đối với các cột có chứa giá trị NULL

- Ở trong câu truy vấn đầu tiên SELECT COUNT(*) FROM person; do giá trị của cột ssn trong bảng person có thể chứa giá trị NULL, do đó mặc dù cột ssn đã được đánh chỉ mục nhưng nó sẽ không được sử dụng cho câu truy vấn và vì vậy bộ tối ưu đã lựa chọn việc Full Scan Table cho câu truy vấn này
- Ở câu truy vấn thứ 2 SELECT /*+ INDEX(person) */ COUNT(*) FROM person; do trường ssn trong bảng person được xét làm khóa chính (PRIMARY KEY) mà một trường khi trở thành khóa chính thì không thể chứa giá trị NULL và giá trị đó phải là giá trị duy nhất, do đó cấu trúc chỉ mục trên trường ssn lúc này sẽ được sử dụng cho câu truy vấn, do đó chiến lược được bộ tối ưu sử dụng ở đây là INDEX FAST FULL SCAN

11. Index-Organized Tables



Một tổ chức chỉ mục bảng (IOT) là một bảng lưu trữ vật lý kết nối với nhau tạo thành cấu trúc chỉ mục. Giá trị khóa (cho bảng và chỉ mục B*-tree) được lưu trữ trong phân đoạn giống nhau. Một IOT bao gồm :

- Giá trị khóa chính
- Giá trị cột khác (không phải khóa chính) trong cùng dòng

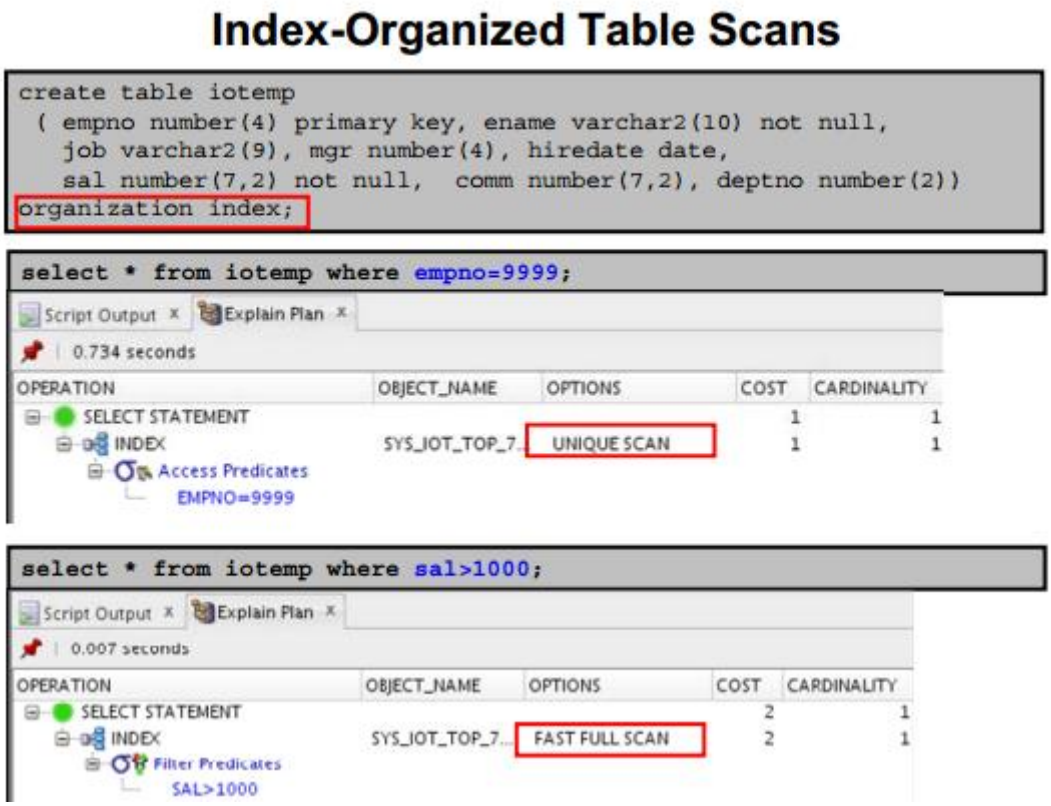
Cấu trúc B*-tree, nó được xây dựng dựa trên khóa chính của bảng, được tổ chức theo cách thức giống như chỉ mục. Những khối lá trong cấu trúc này bao gồm những dòng thay cho những dòng ROWIDS. Biện pháp này với những dòng trong IOT luôn luôn duy trì trong trật tự của khóa chính.

Bạn có thể khởi tạo thêm chỉ số trong IOTs. Khóa chính có thể là khóa phức hợp. Bởi vì đa phần những dòng của IOT có thể bị bỏ trống dày đặc và lưu trữ hiệu quả của cấu trúc B*-tree, bạn có thể lưu trữ một phần của bản ghi trong phân đoạn khác, nó được gọi là vùng tràn.

Tổ chức chỉ mục bảng cung cấp nhanh khóa cơ sở truy cập tới dữ liệu bảng cho những câu truy vấn liên quan đến độ chính xác và phạm vi tìm kiếm. Những thay đổi tới kết quả dữ liệu bảng chỉ được cập nhật trong cấu trúc chỉ mục. Cũng có yêu cầu lưu trữ được giảm xuống bởi vì cột khóa không bị trùng lặp trong bảng và chỉ mục. Còn lại những cột không phải là khóa được lưu trữ trong cấu trúc chỉ mục. IOTs đặc biệt hữu ích khi bạn sử dụng ứng dụng mà cần phải truy xuất cơ sở dữ liệu trên khóa chính và chỉ có một vài, tương đối ngắn những cột không phải khóa.

Note : các mô tả đề cập ở đây là chính xác nếu phân đoạn tràn tồn tại. Phân đoạn tràn nên được sử dụng với những bản ghi dài.

Index-Organized Table Scans :

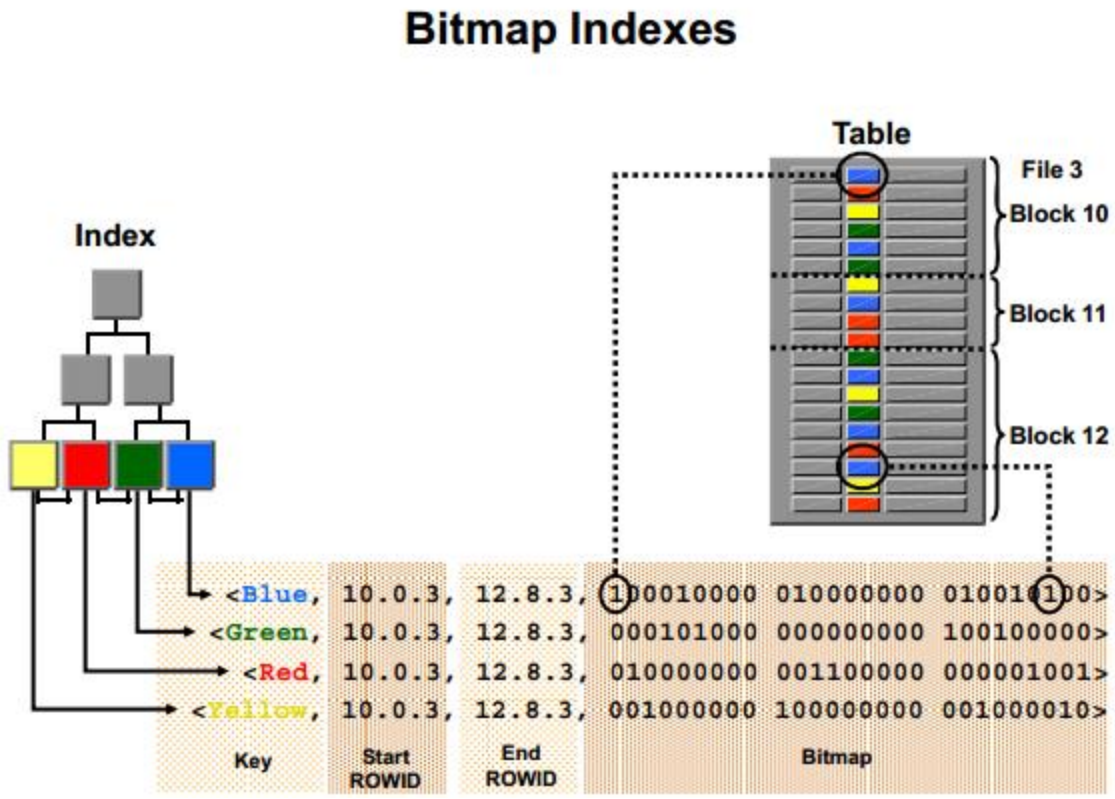


Tổ chức chỉ mục bảng được đánh chỉ mục. Họ sử dụng những đường dẫn truy cập giống nhau mà bạn nhìn thấy cho những chỉ mục bình thường.

Sự khác biệt lớn từ một bảng tổ chức heap đó là không cần truy cập cả chỉ mục và bảng để truy xuất dữ liệu chỉ mục.

Note: SYS_IOT_TOP_75664 là tên hệ thống được tạo của phân đoạn được sử dụng để lưu trữ của cấu trúc IOT. Bạn có thể truy xuất link giữa tên bảng và phân đoạn từ USER_INDEXES với những cột : INDEX_NAME, INDEX_TYPE, TABLE_NAME.

12. 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 bitmap 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 lượng 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 trừ khi hàng đó không tồn tại. Nội dung của vị trí trong bitmap có một giá trị đặc biệt cho biết giá trị của của hàng trong các 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.

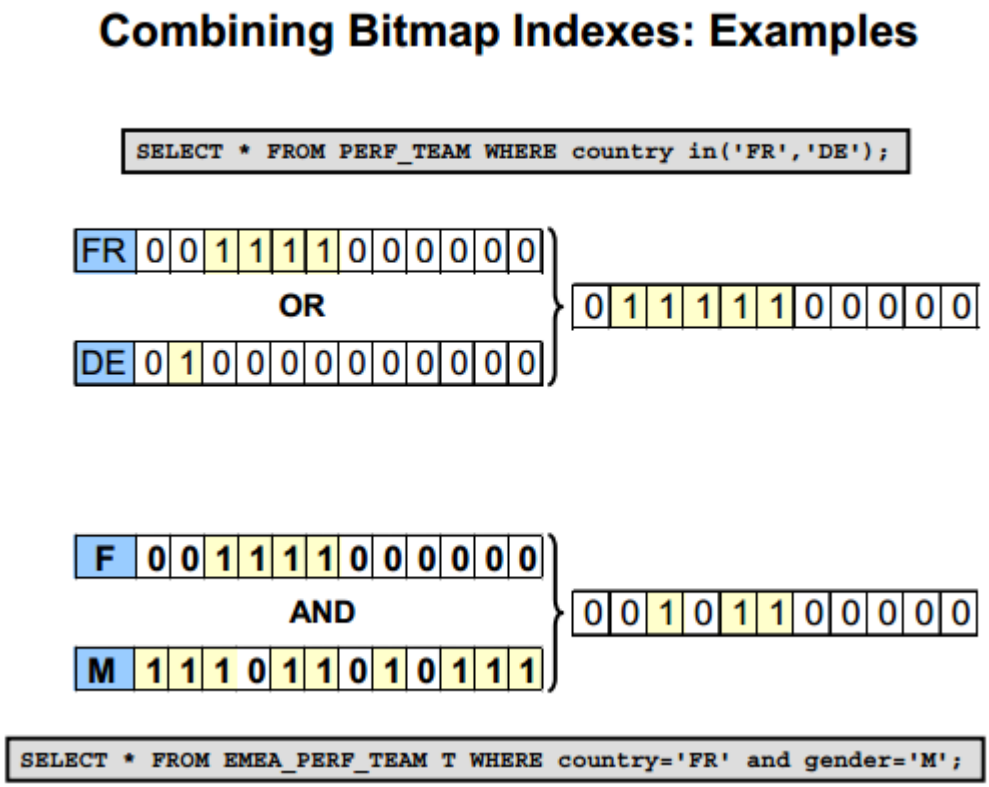
Bitmap Index Access: Examples

```
SELECT * FROM PERF_TEAM WHERE country='FR';
-------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID | PERF_TEAM |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS|           |       |       |
|   3 |    BITMAP INDEX SINGLE VALUE | IX_B2     |       |       |
-------------------------------------------------------------------
Predicate:  3 - access("COUNTRY"='FR')
```

```
SELECT * FROM PERF_TEAM WHERE country>'FR';
-------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID | PERF_TEAM |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS|           |       |       |
|   3 |    BITMAP INDEX RANGE SCAN   | IX_B2     |       |       |
-------------------------------------------------------------------
Predicate: 3 - access("COUNTRY">'FR') filter("COUNTRY">'FR')
```

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 truy 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

Combining Bitmap Index Access Paths

```
SELECT * FROM PERF_TEAM WHERE country in ('FR','DE');
-------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes |
|   0 | SELECT STATEMENT              |            |     1 |    45 |
|   1 |  INLIST ITERATOR              |            |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID | PERF_TEAM  |     1 |    45 |
|   3 |    BITMAP CONVERSION TO ROWIDS|            |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE | IX_B2      |       |       |
Predicate: 4 - access("COUNTRY"='DE' OR "COUNTRY"='FR')
```

```
SELECT * FROM PERF_TEAM WHERE country='FR' and gender='M';
-------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes |
|   0 | SELECT STATEMENT              |            |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | PERF_TEAM  |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS|            |       |       |
|   3 |    BITMAP AND                 |            |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE| IX_B1      |       |       |
|   5 |     BITMAP INDEX SINGLE VALUE| IX_B2      |       |       |
Predicate: 4 - access("GENDER"='M') 5 - access("COUNTRY"='FR')
```

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.

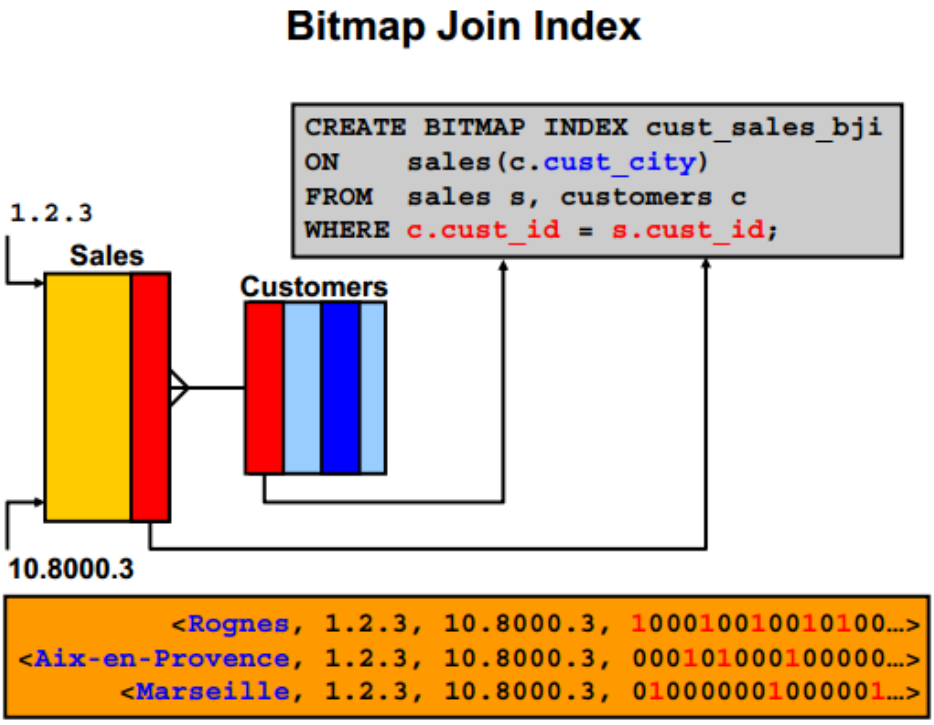
Bitmap Operations

- BITMAP CONVERSION:
  - TO ROWIDS
  - FROM ROWIDS
  - COUNT
- BITMAP INDEX:
  - SINGLE VALUE
  - RANGE SCAN
  - FULL SCAN
- BITMAP MERGE
- BITMAP AND/OR
- BITMAP MINUS
- BITMAP KEY ITERATION

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 bitmap (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 bitmap 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 này sao đó được trộn vào trong toán tử BITMAP MERGE.



Thêm một điều về bitmap index trên một bảng đơn, bạn có thể tạo một bitmap join index. Một bitmap join index là một bitmap index của sự kết nối của 2 hay nhiều hơn các bảng. Bitmap join index là một cách hiệu quả hơn trong việc giảm 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.

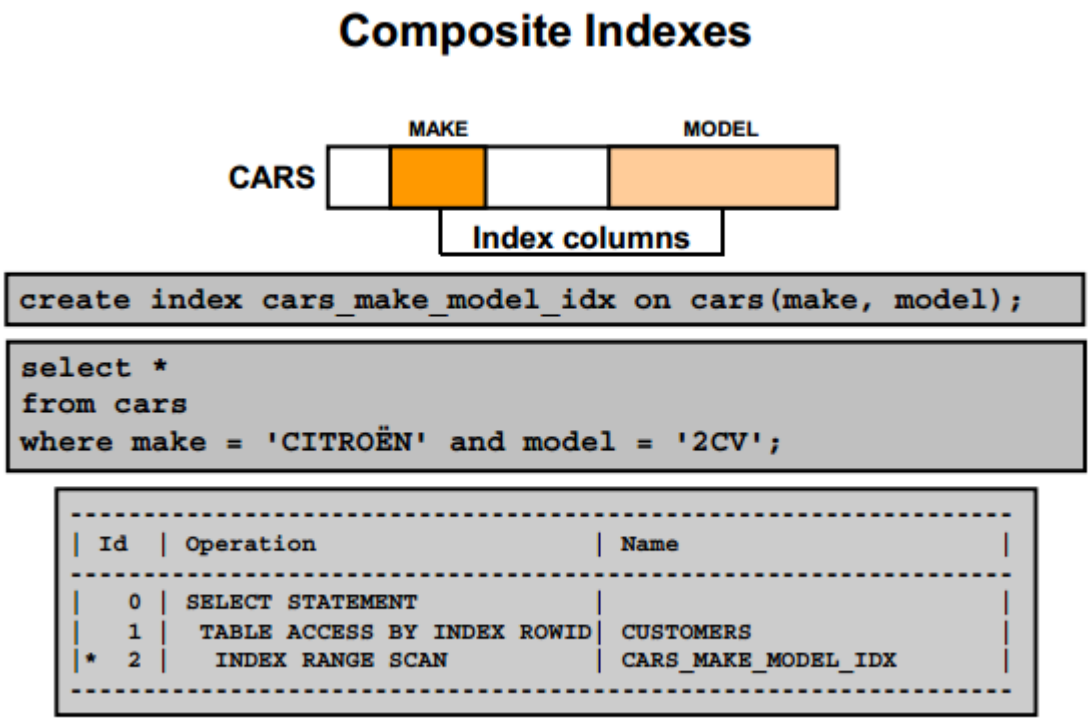
Khóa của index này là cột CUST_CITY của bảng CUSTOMERS. Ví dụ này cho rằng khóa chính của CUSTOMERS được lưu trữ trong bimap index tồ tại thực sự trong dữ liệu của bảng. Cột CUST_ID là khóa chính của bảng CUSTOMERS nhưng cũng là khóa ngoài của bảng SALES, cho dù nó không đòi hỏi.

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 bitmap 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.

13. Composite Indexes



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.

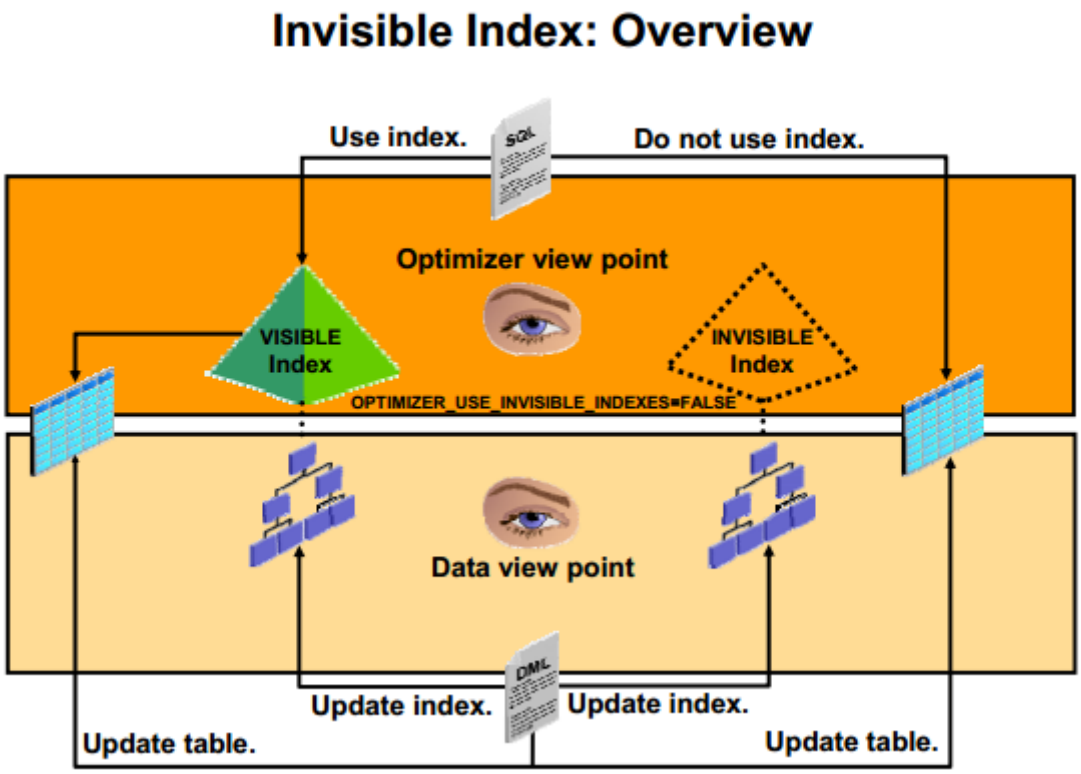
Cải thiện tính chọn lọc: Đôi khi hai hay nhiều cột hoặc biểu thức, mỗi cái với tính chọn lọc nghèo nàn có thể được tổ hợp thành index kết hợp với tính chọn lọc cao hơn.

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 chiếu tới một phần của index. Nó có thể dừng scan index và scan nhanh toàn bộ có thể được thi hành.

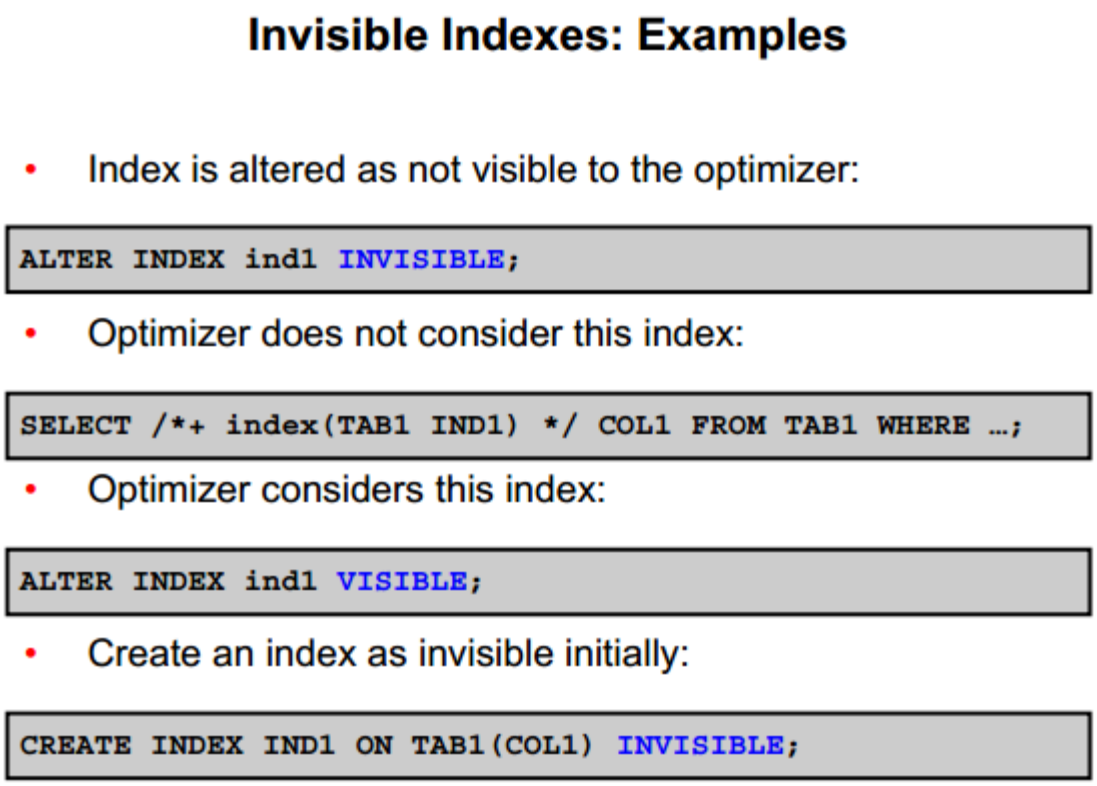
14. Invisible Index



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.

15. 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 khi tạo index:
  - Tạo index nếu bạn thường xuyên muốn nhận ít hơn 15% số hàng trong một bảng lớn.
  - Để cải thiện hiệu năng khi nối nhiều bảng, dùng index 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 nhất 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.

16. 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 dễ 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.

Lơi cảm ơn

Bài báo cáo trên là công sức của cả nhóm sau một thời gian dài tìm hiểu về các phép toán tối ưu trên hệ quản trị cơ sở dữ liệu Oracle. Về mặt nội dung, nhóm hy vọng đã có thể truyền tải tới thầy và các bạn những kiến thức và hình ảnh trực quan nhất về các chỉ mục nghiên cứu.

Tuy đã rất cố gắng thực hiện đề tài nhưng nội dung nghiên cứu của nhóm vẫn còn những thiếu sót và hạn chế, rất mong nhận được sự quan tâm của thầy và các bạn để nhóm có thể bổ sung và hoàn thiện hơn, không chỉ phục vụ kiến thức môn học mà còn có thể áp dụng vào thực tế sau này.

Nhóm chúng em xin chân thành cảm ơn thầy Trần Việt Trung đã giúp đỡ nhóm về mặt tài liệu và kiến thức nền tảng. Hy vọng chúng em sẽ được tiếp thu nhiều hơn các kiến thức về công nghệ thông tin từ thầy.

Cảm ơn thầy và các bạn đã đọc và theo dõi!

Hà Nội, tháng 5 năm 2015

Nhóm sinh viên thực hiện đề tài.
Nhóm 3

Tài liệu tham khảo:

1. Oracle Database 11g: SQL Tuning Workshop
   https://drive.google.com/file/d/0B2NXgbj910FEUUctM2JrY0NxZ3c/view

2. Kiến trúc và quản lý cơ sở dữ liệu Oracle.
   https://cdn.fbsbx.com/hphotos-xpa1/v/t59.2708-21/11207447_747047482074715_25225199_n.pdf/kien_truc_va_quan_ly_co_so_du_lieu_oracle_s6gNSqA2LR_20140208082543_65671.pdf?oh=7791d534ee55ccc8f4343dcd69e93ff3&oe=555D07D5&dl=1