

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 Kiêm	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 mẽ từ năm 1996 nhưng sự phát triển 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 một 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 tiên 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 đối 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

Lời nói đầu.....	2
Phân công công việc .....	4
I. Main Structures and Access Paths.....	5
II. Cấu trúc lưu trữ dạng bảng và các phương pháp truy xuất dữ liệu.....	6
1. Full Table Scan.....	6
2. ROWID Scan.....	8
3. Sample Table Scans.....	9
III. Cấu trúc chỉ mục và các phương pháp truy xuất dữ liệu.....	10
1. Khái quát về index trong Oracle.....	10
2. Giới thiệu về B*-tree indexes .....	10
3. Index Unique Scan.....	12
4. Index Range Scan .....	13
5. Index Full Scan.....	15
6. Index Skip Scan.....	17
7. Index Join Scan.....	18
8. B*-tree Indexes and Nulls .....	19
9. Using Indexes: Considering Nullable Columns .....	20
10. Index-Organized Tables .....	21
11. Bitmap Indexes.....	23
12. Invisible Index .....	26
13. Các nguyên tắc sử dụng Index .....	27
Lời cảm ơn .....	28
Tài liệu tham khảo: .....	29

## Phân công công việc

Thành viên	Công việc
Lê Quyết Thắng	Dịch chương I → II
Tạ Công Sơn	Dịch chương III 1 → III 5
Nguyễn Khắc Nhất	Dịch chương III 6 → III 9
Vũ Mạnh Kiểm	Dịch chương III 10 → III 13

# I. Main Structures and Access Paths

Oracle sử dụng 2 cấu trúc lưu trữ đó là : *cấu trúc dạng bảng (table)* và *cấu trúc chỉ mục (indexes)*. Với mỗi cấu trúc Oracle cung cấp các phương pháp truy xuất dữ liệu khác nhau tùy thuộc vào ngữ cảnh để có thể tối ưu câu truy vấn.

## Main Structures and Access Paths

Structures	Access Paths
Tables	1. Full Table Scan 2. Rowid Scan 3. Sample Table Scan
Indexes	4. Index Scan (Unique) 5. Index Scan (Range) 6. Index Scan (Full) 7. Index Scan (Fast Full) 8. Index Scan (Skip) 9. Index Scan (Index Join) 10. Using Bitmap Indexes 11. Combining Bitmap Indexes

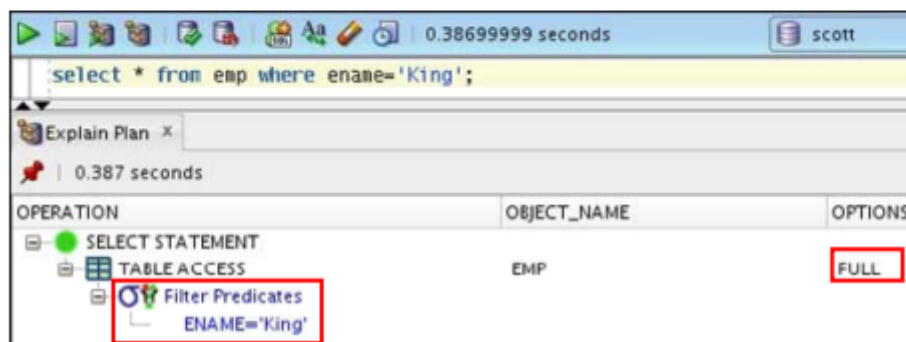
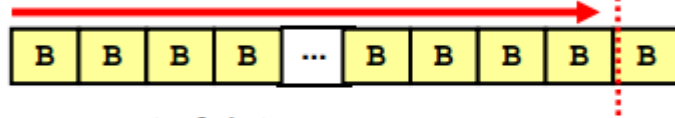
Cấu trúc lưu trữ và các phương pháp truy xuất dữ liệu tương ứng với từng phương pháp

## II. Cấu trúc lưu trữ dạng bảng và các phương pháp truy xuất dữ liệu.

### 1. Full Table Scan

#### Full Table Scan

- Performs multiblock reads  
(here `DB_FILE_MULTIBLOCK_READ_COUNT = 4`)
- Reads all formatted blocks below the high-water mark <sup>HWM</sup>
- May filter rows
- Is faster than index range scans for large amount of data



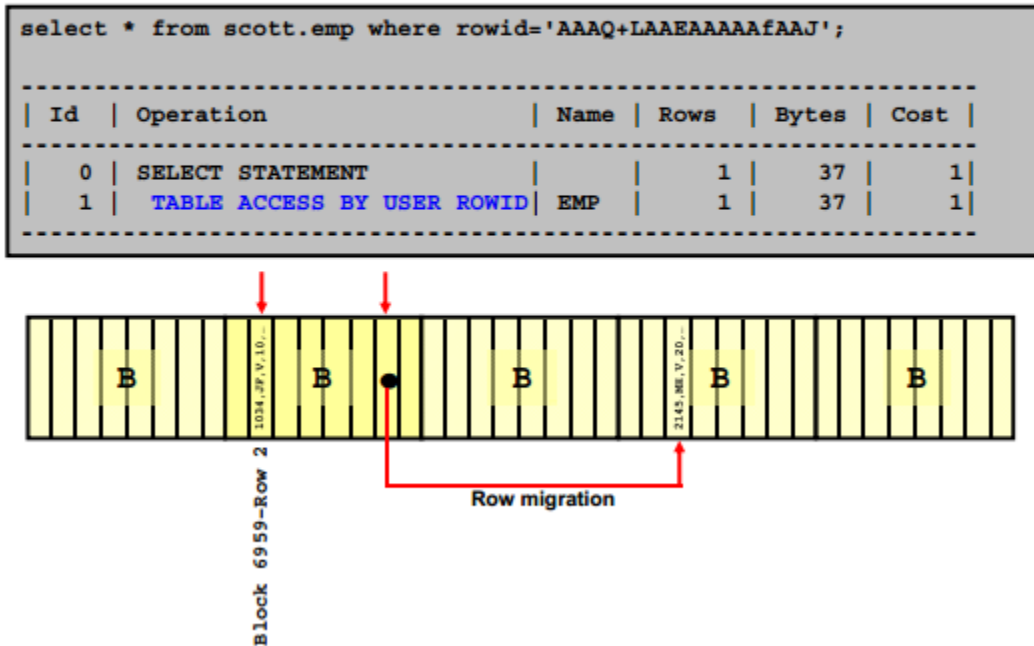
OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
ENAME='King'		

- Full table scan (FTS) là thao tác 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. Đối với thao tác này mỗi block chỉ được đọc duy nhất một lần.  
(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.)
- Bởi vì một FTS đọc tất cả block trong bảng, nó đọc các block liên tiếp 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. Số Block tối đa có thể đọc trong một lần truy xuất dữ liệu được thiết lập bởi biến `DB_FILE_MULTIBLOCK_READ_COUNT`.

- Bộ tối ưu hóa sử dụng FTS trong các trường hợp sau:
- *Thiếu Index*: nếu câu truy vấn không tồn tại trường nào trong các bảng được tham chiếu được đánh chỉ mục, khi đó thao tác FTS sẽ được bộ tối ưu lựa chọn
  - *Khối lượng truy xuất dữ liệu trong bảng là lớn*: khi lượng dữ liệu truy xuất trong một bảng là lớn bộ tối ưu sẽ sử dụng thao tác FTS ngay cả khi bảng đó tồn tại trường đã được đánh chỉ mục.
  - *Bảng có kích thước nhỏ*: nếu một bảng có kích thước nhỏ, giả sử kích thước của bảng chỉ chiếm 3 block, trong khi số block có thể đọc tối đa trong một lần là 4 block, khi đó thao tác FTS sẽ được bộ tối ưu lựa chọn ngay cả khi bảng đã được đánh chỉ mục.
  - *Độ tương đồng cao*: Nếu một bảng có độ tương đồng cao thì sẽ có những giá trị được lặp lại nhiều lần do đó một nút của index sẽ trở tới nhiều hàng.

## 2. ROWID Scan

### ROWID Scan

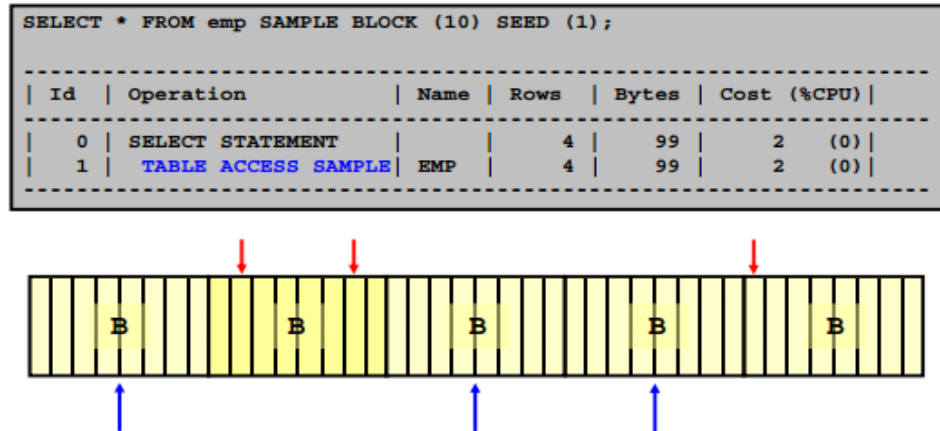


- Rowid của một hàng xác định file dữ liệu và block dữ liệu chứa trong hàng và vị trí của hàng trong block đó.
- Để truy nhập bảng bằng rowid, hệ thống cần xác định rowid của các hàng được chọn hoặc từ mệnh đề WHERE hay thông qua index scan một hay nhiều index của bảng. Hệ thống sẽ xác định vị trí các hàng được chọn trong bảng dựa theo rowid của chúng.



### 3. Sample Table Scans

#### Sample Table Scans



- Sample table scan (STS) là thao tác cho phép lấy một phần dữ liệu trong các bảng. Cách truy nhập này được sử dụng khi mệnh đề FROM bao gồm mệnh đề SAMPLE hoặc SAMPLE BLOCK.

VD : `SELECT * FROM emp SAMPLE BLOCK (10);`

Thực hiện lấy ra 10 block bất kỳ từ bảng emp.

### III. Cấu trúc chỉ mục và các phương pháp truy xuất dữ liệu

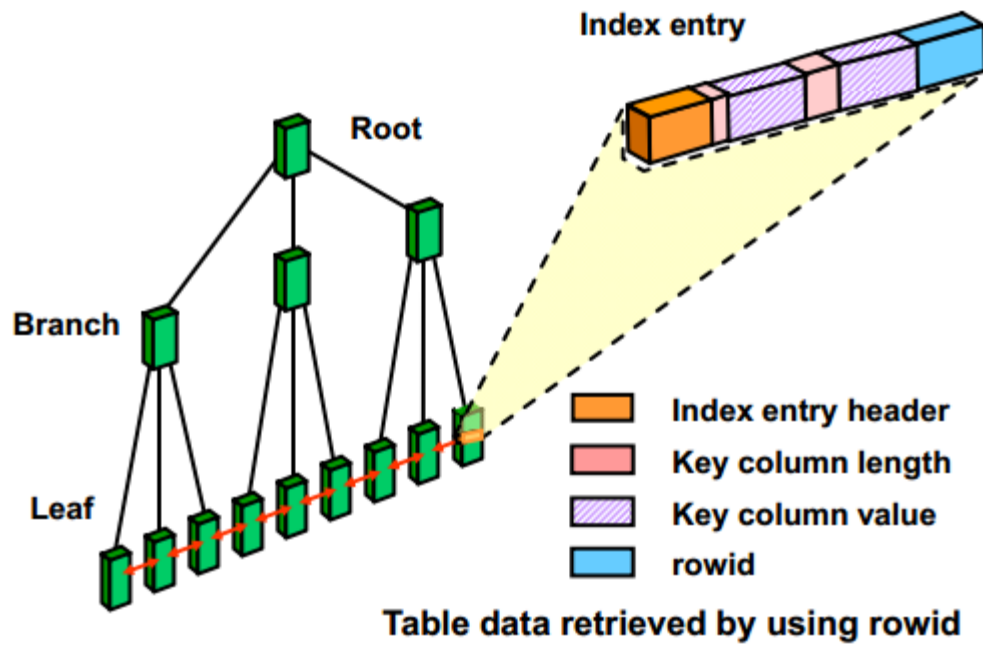
#### 1. Khái quát về index trong Oracle

- Một index là một đối tượng không bắt buộc của CSDL 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 truy vấn SQL, hoặc sử dụng các index để thực thi các ràng buộc toàn vẹn.
- Oracle sử dụng các kỹ thuật sau trong việc lưu trữ index :
  - *B\*-tree indexes* - kỹ thuật mặc định và được sử dụng phổ biến nhất
  - *Bitmap Indexes*
  - *Cluster Indexes*

#### 2. Giới thiệu về B\*-tree indexes

- Mỗi cây B\*-tree index đều có 1 root block 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 của các hàng.
- Con trỏ block trước và sau kết nối các block lá với nhau để chúng có thể dịch chuyển từ trái qua phải hoặc ngược lại – **đây là đặc điểm khác biệt giữa B\*-tree index và B<sup>+</sup>-tree index**
- Index luôn được cân bằng và chúng phát triển từ trên xuống
- Có thể tái tổ chức index bằng cách sử dụng câu lệnh: ALTER INDEX ... REBUILD|COALESCE.
- Hệ thống có thể trực tiếp truy nhập các hàng sau khi lấy được địa chỉ (ROWID) từ index trong block lá.

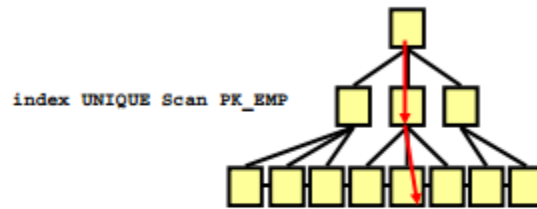
## Normal B\*-tree Indexes



Hình 2. Cấu trúc của một cây B\*-tree indexes

### 3. Index Unique Scan

#### Index Unique Scan



```
create unique index PK_EMP on EMP(empno)

select * from emp where empno = 9999;
```

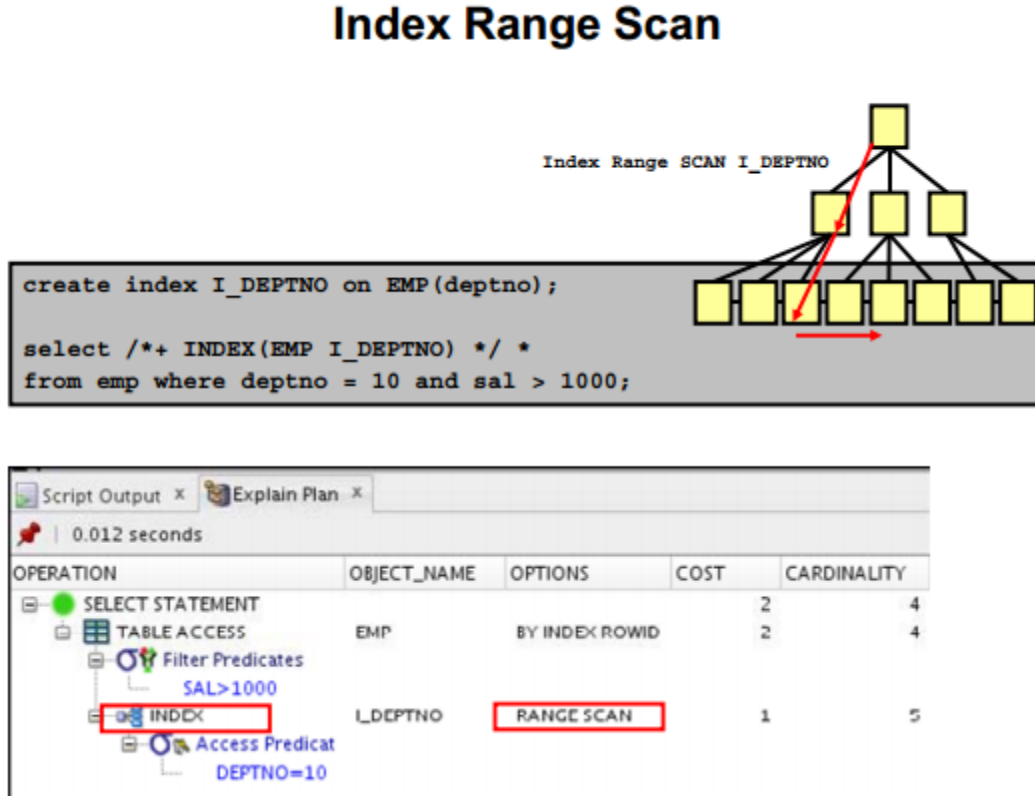
Explain Plan - X

0.024 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
TABLE ACCESS	EMP	BY INDEX ROWID	1	1
INDEX	PK_EMP	UNIQUE SCAN	0	1
Access Predicat				
EMPNO=7839				

- Index Unique Scan là thao tác được bộ tối ưu thực hiện khi trường được đánh chỉ mục là *khóa chính* hoặc trường được đánh chỉ mục đảm bảo điều kiện không chứa giá trị NULL và không có sự lặp lại dữ liệu (một giá trị xuất hiện tối đa một lần). Khi đó dữ liệu đầu tiên thỏa mãn điều kiện trong mệnh đề WHERE cũng đảm bảo là giá trị duy nhất.

## 4. Index Range Scan

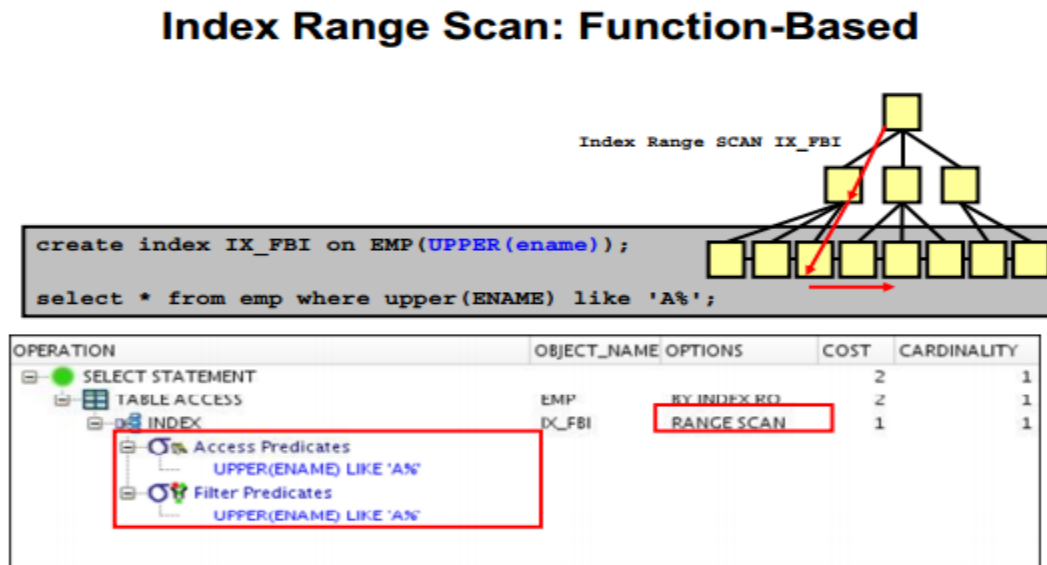


Hình 3. Index Range Scan

- Index Range Scan là thao tác phổ biến nhất trong việc truy cập dữ liệu có chọn lọc. Dữ liệu trả về sẽ được sắp 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)
- Range scan có thể sử dụng cấu trúc chỉ mục unique or nonunique indexes. Range scan có thể tránh được việc sắp 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 sắp 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.

Ví dụ trong Hình 3 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.

### *Index Range Scan: Function-Based*

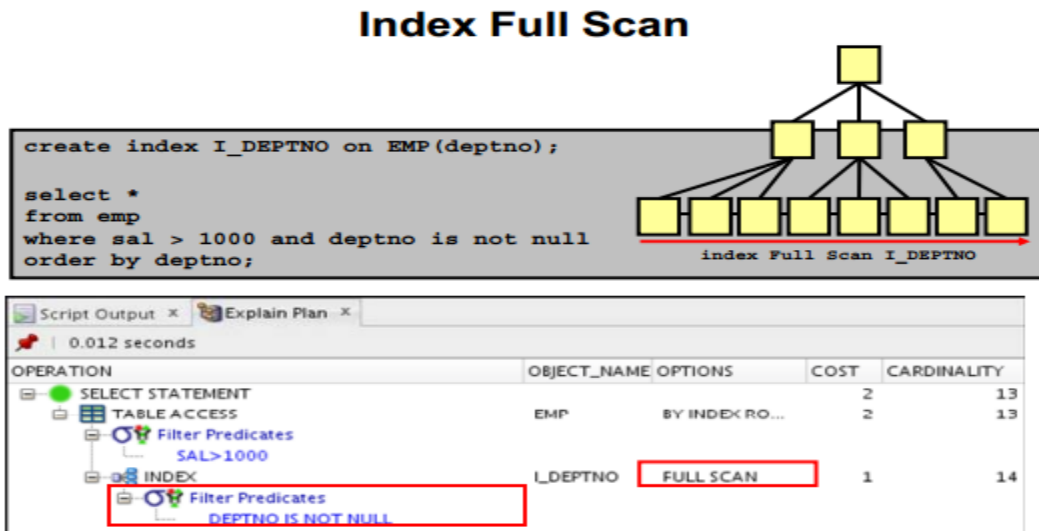


- Một chỉ mục Function-Based có thể được lưu trữ như một cấu trúc B\*-tree hay bitmap. Những chỉ mục này bao gồm các cột được biến đổi bởi một hàm (như hàm UPPER) hoặc bao gồm các biểu thức (như col1 + col2).
- Cấu trúc chỉ mục Function-Based phép dữ liệu được trả về khi sử dụng cấu trúc chỉ mục khi 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.

## 5. 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:

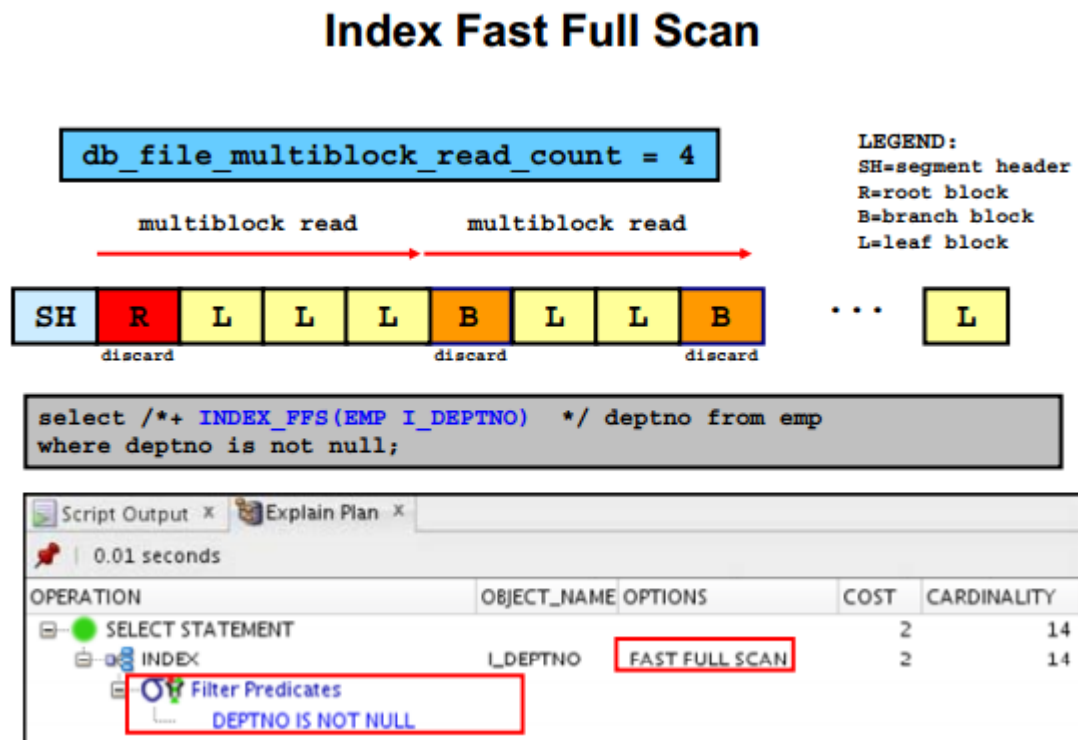
- Tất cả các cột trong bảng được tham chiếu trong câu truy vấn đều được đánh chỉ mục.
- Có ít nhất một cột được đánh chỉ mục là NOT NULL.

Một thao tác full scan có thể được sử dụng để loại bỏ thao tác sắp xếp bởi vì dữ liệu được sắp 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 phương pháp truy suất được sử dụng 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. Fast full scan sẽ truy cập dữ liệu trực tiếp trong cấu trúc chỉ mục mà không cần truy cập đến cấu trúc bảng.

VD : *SELECT /\*+ INDEX\_FFS(departments dept\_id\_pk) \*/ COUNT(\*)*  
*FROM departments;*

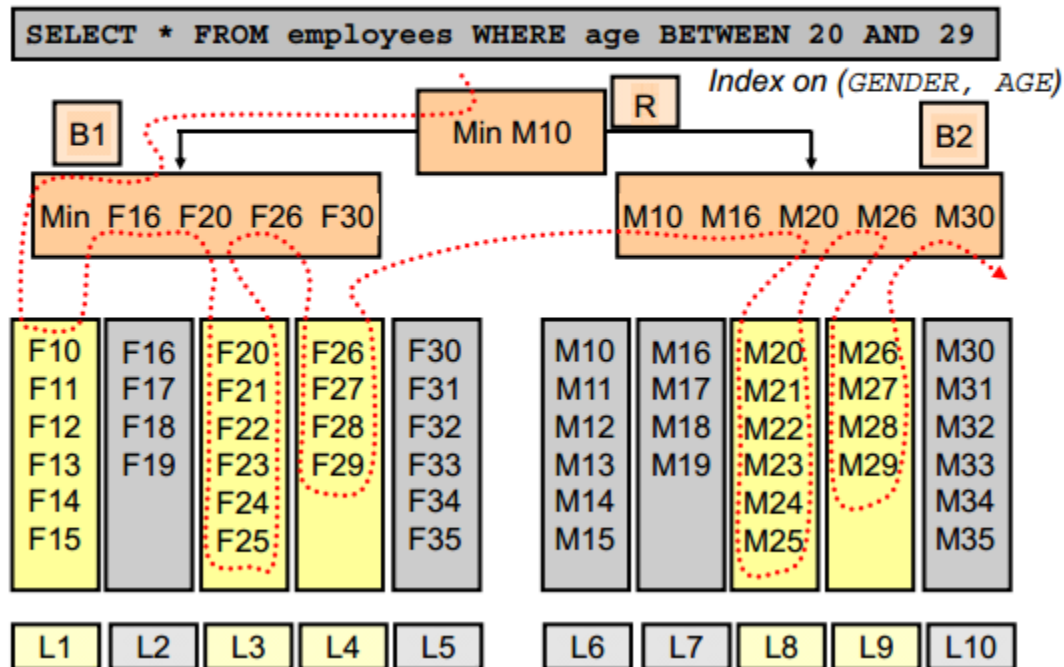
→ Ở đây bảng departments được đánh chỉ mục trên trường thuộc tính dept\_id. Việc đếm số lượng phần tử của bảng departments chỉ cần đếm các nút của cấu trúc chỉ mục.

→ */\*+ INDEX\_FFS(departments dept\_id\_pk) \*/* : hint này bắt buộc tối ưu phải lựa chọn thao tác Index fast full scan để thực hiện câu truy vấn.



## 6. Index Skip Scan

### Index Skip Scan



Hình 4. 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.

## 7. Index Join Scan

### Index Join Scan

```
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;
```

Statement Output x Autotrace x

1.563 seconds

OPERATION	OBJECT_NAME	COST
SELECT STATEMENT		3
VIEW	index\$join\$_001	3
type="db_version"		
11.2.0.1		
HASH JOIN		
Access Predicates		
ROWID=ROWID		
INDEX FAST FULL SCAN	I_ENAME	1
INDEX FAST FULL SCAN	I_SAL	1

Index Join Scan là thao tác 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)

## 8. B\*-tree Indexes and Nulls

### B\*-tree Indexes and Nulls

```
create table nulltest ( col1 number, col2 number not null);
create index nullind1 on nulltest (col1);
create index notnullind2 on nulltest (col2);
```

**Query 1:** `select /*+ index(t nullind1) */ col1 from nulltest t;`  
 Execution time: 0.864 seconds  
 Explain Plan:  

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
TABLE ACCESS	NULLTEST	FULL	2	1

**Query 2:** `select col1 from nulltest t where col1=10;`  
 Execution time: 1.105 seconds  
 Explain Plan:  

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	NULLIND1	RANGE SCAN	1	1
Access Predicates		COL1=10		

**Query 3:** `select /*+ index(t notnullind2) */ col2 from nulltest t;`  
 Execution time: 0.034 seconds  
 Explain Plan:  

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	NOTNULLIND2	FULL SCAN	1	1

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.

## 9. Using Indexes: Considering Nullable Columns

### Using Indexes: Considering Nullable Columns

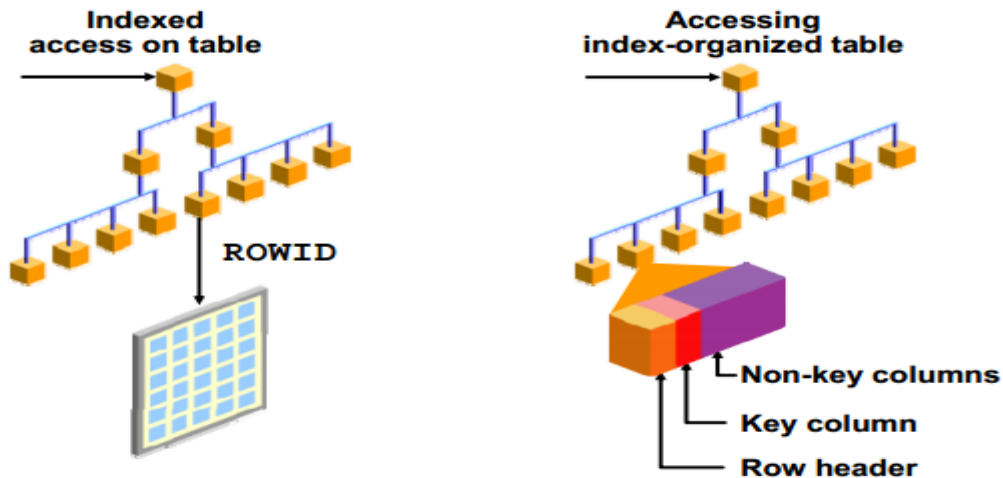
<table> <tr> <th>Column</th><th>Null?</th></tr> <tr> <td>SSN</td><td>Y</td></tr> <tr> <td>FNAME</td><td>Y</td></tr> <tr> <td>LNAME</td><td>N</td></tr> <tr> <td>⋮</td><td></td></tr> </table> <p>PERSON</p>	Column	Null?	SSN	Y	FNAME	Y	LNAME	N	⋮		<pre>CREATE UNIQUE INDEX person_ssn_ix ON person(ssn);</pre> <pre>SELECT COUNT(*) FROM person;</pre> <table> <tr> <td>SELECT STATEMENT</td><td></td></tr> <tr> <td>SORT AGGREGATE</td><td></td></tr> <tr> <td>TABLE ACCESS FULL</td><td>PERSON</td></tr> </table> <pre>DROP INDEX person_ssn_ix;</pre>	SELECT STATEMENT		SORT AGGREGATE		TABLE ACCESS FULL	PERSON
Column	Null?																
SSN	Y																
FNAME	Y																
LNAME	N																
⋮																	
SELECT STATEMENT																	
SORT AGGREGATE																	
TABLE ACCESS FULL	PERSON																
<table> <tr> <th>Column</th><th>Null?</th></tr> <tr> <td>SSN</td><td>N</td></tr> <tr> <td>FNAME</td><td>Y</td></tr> <tr> <td>LNAME</td><td>N</td></tr> <tr> <td>⋮</td><td></td></tr> </table> <p>PERSON</p>	Column	Null?	SSN	N	FNAME	Y	LNAME	N	⋮		<pre>ALTER TABLE person ADD CONSTRAINT pk_ssn PRIMARY KEY (ssn);</pre> <pre>SELECT /*+ INDEX(person) */ COUNT(*) FROM person;</pre> <table> <tr> <td>SELECT STATEMENT</td><td></td></tr> <tr> <td>SORT AGGREGATE</td><td></td></tr> <tr> <td>INDEX FAST FULL SCAN</td><td>PK_SSN</td></tr> </table>	SELECT STATEMENT		SORT AGGREGATE		INDEX FAST FULL SCAN	PK_SSN
Column	Null?																
SSN	N																
FNAME	Y																
LNAME	N																
⋮																	
SELECT STATEMENT																	
SORT AGGREGATE																	
INDEX FAST FULL SCAN	PK_SSN																

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

## 10. Index-Organized Tables

### 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 index 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

Một index organized table lưu trữ toàn bộ dữ liệu cho bảng trong 1 cấu trúc B\*-tree, B\*-tree index được xây dựng dựa trên khóa chính của bảng và được tổ chức giống như index. Với biện pháp này IOT luôn luôn duy trì trong trật tự của khóa chính.

Khóa chính có thể là khóa phức hợp. Nếu tạo index organized table mà không có khóa chính sẽ sinh ra lỗi. Bạn có thể lưu trữ một phần của bản ghi trong phân đoạn khác, nó được gọi là vùng tràn.

Truy xuất một index organized table thì chỉ các khối index cần được đọc bởi vì toàn bộ bảng có sẵn trong phần lá của node.

IOT cung cấp nhanh việc 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 trên bảng như : thêm, xóa hay cập nhật các hàng dẫn đến kết quả cập nhật trong cấu trúc index. Yêu cầu lưu trữ được giảm xuống bởi vì cột khóa không bị trùng lặp trong bảng và index. Còn lại những cột không phải là khóa được lưu trữ trong cấu trúc index. 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 :*

## Index-Organized Table Scans

```
create table iotemp
( empno number(4) primary key, ename varchar2(10) not null,
  job varchar2(9), mgr number(4), hiredate date,
  sal number(7,2) not null, comm number(7,2), deptno number(2))
organization index;
```

```
select * from iotemp where empno=9999;
```

Script Output x Explain Plan x

0.734 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	SYS_IOT_TOP_7...	UNIQUE SCAN	1	1
Access Predicates		EMPNO=9999		

```
select * from iotemp where sal>1000;
```

Script Output x Explain Plan x

0.007 seconds

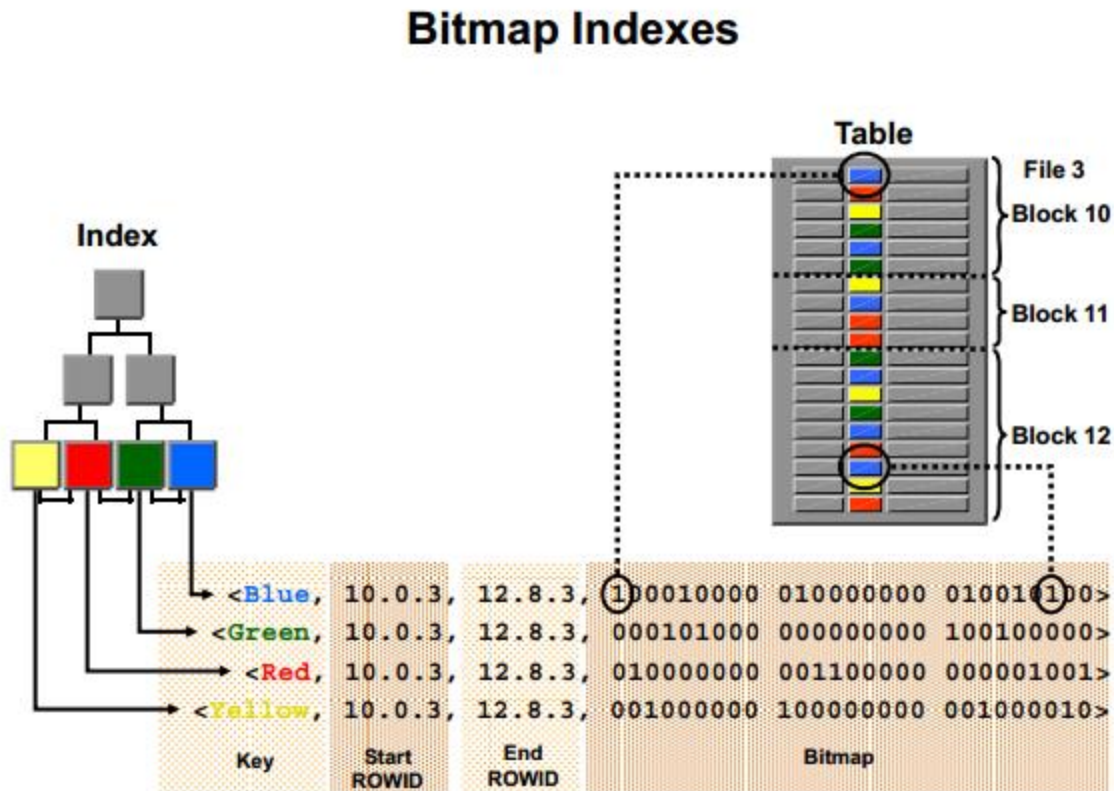
OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
INDEX	SYS_IOT_TOP_7...	FAST FULL SCAN	2	1
Filter Predicates		SAL>1000		

Với IOT được đánh index. Họ sử dụng những đường dẫn truy cập giống nhau mà bạn nhìn thấy cho những index bình thường.

Sự khác biệt lớn từ một heap organized table đó là không cần truy cập cả index và table để truy xuất dữ liệu index.



## 11. Bitmap Indexes



Hình 1: Ví dụ về Bitmap Index

Trong cấu trúc B tree, một nút lá sẽ tương ứng với một hàng. Bitmap index cũng có thể trở tới từng hàng giống B-tree.

Nếu sử dụng nhiều cột để tạo bitmap index, thì mỗi dòng bitmap sẽ là bitmap của tổ hợp giá trị của những hàng đó. Mỗi bitmap header lưu trữ những ROWID bắt đầu và kết thúc. Mỗi vị trí trong một bitmap sẽ đại diện cho một hàng trong bảng. Nếu bitmap có giá trị 1, hàng đó có chứa giá trị tương ứng với bitmap đang được đánh, ngược lại nếu giá trị của bitmap là 0.

Bitmap indexes được sử dụng rộng trong môi trường kho dữ liệu, các môi trường có dữ liệu ổn định, không thường xuyên bị thay đổi, môi trường có số lượng giá trị ít.

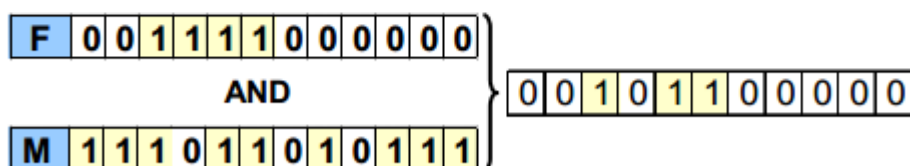
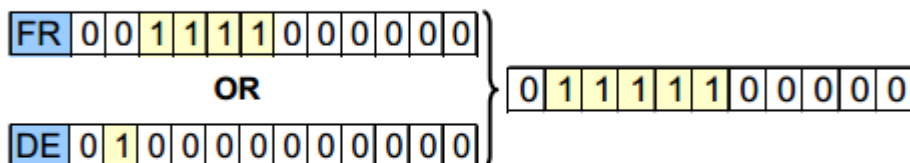
Cần chú ý rằng, khi ta sử dụng một hàng với bitmap index thì ta cũng đồng thời khóa nhiều hàng của bảng đó. Do vậy việc thực hiện đồng thời việc truy suất và cập nhật dữ liệu là hạn chế khi sử dụng bitmap index.

Không giống hầu hết các loại index khác, bitmap indexes bao gồm cả các hàng có giá trị NULL. Việc đánh index 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 hay sử dụng điều kiện IS NOT NULL.

## Combining Bitmap Indexes: Examples

```
SELECT * FROM PERF_TEAM WHERE country in('FR','DE');
```



```
SELECT * FROM EMEA_PERF_TEAM T WHERE country='FR' and gender='M';
```

Hình 2: Thực hiện các phép toán trên bit với bitmap index

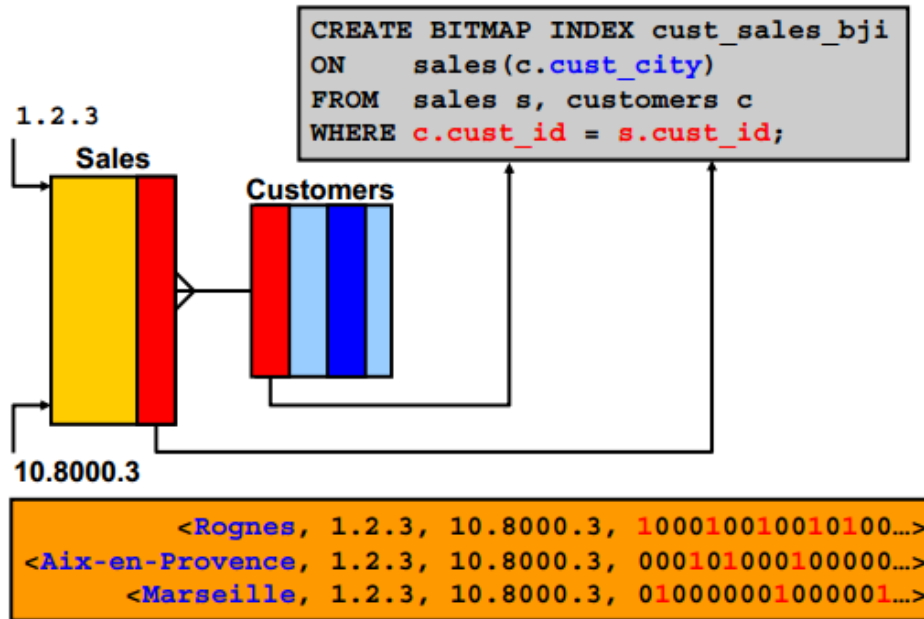
Bitmap indexes có hiệu quả tốt nhất với các truy vấn bao gồm nhiều điều kiện trong mệnh đề WHERE. Việc sử dụng bitmap index giống như việc sơ loại trước các hàng không thỏa mãn điều kiện do đó giúp tránh phải truy cập tới các hàng không thỏa mãn, từ cải thiện thời gian thực hiện yêu cầu.

Bitmap index hiệu quả khi mệnh đề where có các điều kiện:

- IN (value list)
- Có các phép AND, OR.
- **Bitmap join index**



## Bitmap Join Index



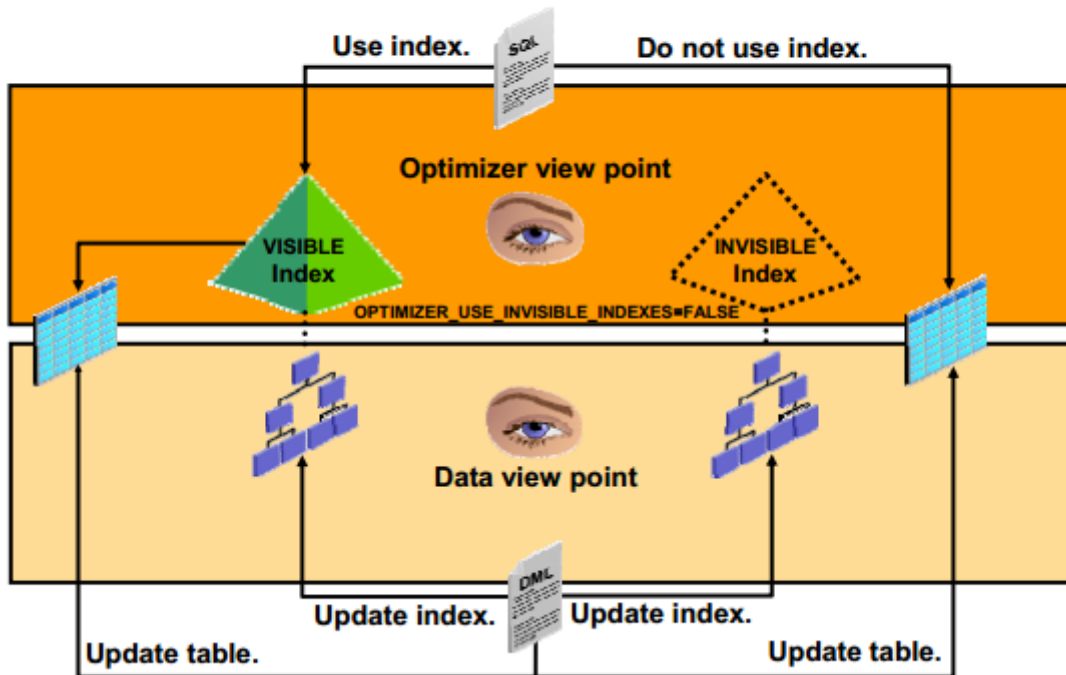
*Hình 3: Ví dụ Bitmap join index*

Bitmap join index là kỹ thuật tạo bitmap index giữa 2 bảng để thực hiện truy vấn trên đó mà không cần join về mặt vật lý giữa 2 bảng.

Trong ví dụ tạo bitmap join index trên hình vẽ, một bitmap index với sự kết hợp của 2 bảng Sale và Customer sẽ được tạo ra, sau đó với mỗi câu truy vấn dựa trên điều kiện của trường c.cust\_city sẽ được thực hiện trên bitmap này mà không cần join thực tế 2 bảng lại.

## 12. Invisible Index

### Invisible Index: Overview



Khi một index được thiết lập với tham số `OPTIMIZED_USE_INVISIBLE_INDEXES` là thì chương trình tối ưu sẽ không sử dụng nó để thực hiện câu truy vấn. Giá trị mặc định của tham số này là `FALSE`.

Việc tạo index vô hình này sẽ giúp chúng ta kiểm chứng được việc sử dụng hay không sử dụng index này cài nào có hiệu quả hơn. Từ đó quyết định xem có sử dụng index đó hay không.

## 13. Các nguyên tắc sử dụng Index

- Tạo index sau khi chèn dữ liệu vào bảng: Dữ liệu thường xuyên được cập nhật vào trong bảng, do việc cập nhật lại index sau khi bảng bị thay đổi là điều cần thiết để đảm bảo index đang chỉ đúng tới các hàng.
- Tạo index nếu các câu truy vấn của bạn thường xuyên chỉ tác động tới dưới 15% số hàng của một bảng lớn.
- Để cải thiện hiệu năng khi nối nhiều bảng, dùng index để nối.
- Bảng nhỏ không cần dùng index.
- Một số cột là ứng cử viên để đánh index khi:
  - Các hàng trong cột có giá trị khác nhau.
  - Có miền giá trị rộng (tốt cho regular index)
  - Có miền giá trị nhỏ (tốt cho bitmap index)
  - Cột có nhiều giá trị NULL, nhưng truy vấn thường chọn các hàng có giá trị khác NULL.
- Hàng không thích hợp để đánh index:
  - 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.
- 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.
- 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.
- Nếu bạn sử dụng cùng một không gian cho bảng và index của nó, nó có thể thuận lợi cho việc duy trì cơ sở dữ liệu, ví dụ như sao lưu cơ sở dữ liệu.

## Lời 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?oh=7791d534ee55ccc8f4343dcd69e93ff3&oe=555D07D5&dl=1](https://cdn.fbsbx.com/hphotos-xpa1/v/t59.2708-21/11207447_747047482074715_25225199_n.pdf?oh=7791d534ee55ccc8f4343dcd69e93ff3&oe=555D07D5&dl=1)