TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP LỚN

MÔN THIẾT KẾ VÀ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 8: Other Optimizer Operators

Giáo viên hướng dẫn: TS. Trần Việt Trung

Sinh viên thực hiện:

Nguyễn Huy Bộ - 20111183

Nguyễn Huy Hòa - 20111597

Đỗ Văn Tiệp - 20112322

Nguyễn Trường Sơn - 20112078

Hà nội, tháng 5 năm 2015

Mục lục

I. Mục tiêu

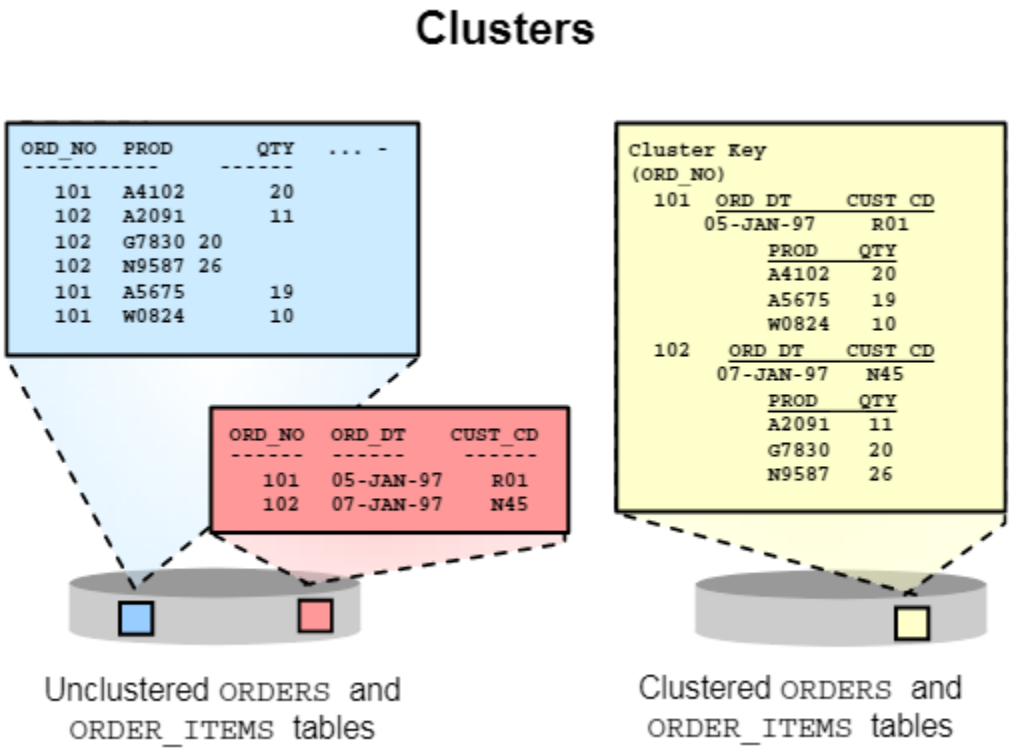
Sau khi đọc xong chương này chúng ta có thể:

- Mô tả được các thao tác khai thác SQL đối với:
- Clusters
- In-List
- Sort
- Filters
- Set Operations
- Khai thác kết quả bộ nhớ Cache

Chương này sẽ giúp các bạn hiểu được các kế hoạch thực thi mà sử dụng các thao tác phổ biến của các cách thức truy cập khác nhau.

II. Nội dung

Clusters



Clusters là một cách thức tùy chọn đối với việc lưu trữ dữ liệu bảng. Một **Cluster** là một nhóm các bảng chia sẻ các **data block** tương tự bởi vì chúng chia sẻ những cột chung và thường xuyên được sử dụng cùng nhau.

Cho ví dụ.

Bảng ORDERS and ORDER_ITEMS chia sẻ cột ORDER_ID. Khi bạn đánh cluster hai bảng đó, hệ thống vật lý lưu trữ tất cả các hàng cho mỗi đơn hàng từ cả hai bảng đó ở trong khối dữ liệu tương tự.

Cluster Index

Một **cluster index** là một Index xác định cụ thể cho một **cluster .** Một **index** như vậy có chứa một mục cho mỗi **giá trị khóa cluster.** Để định vị trí một hàng trong một **cluster**, cluster index được sử dụng để tìm **key value** mà chỉ tới khối dữ liệu liên kết với **cluster key value.** Vì thế hệ thống truy cập một hàng đã cho với một mức tối thiểu 2 lần vào ra.

Hash clusters:

Hashing là một cách tùy chọn của việc lưu trữ bảng dữ liệu để cải thiện hiệu năng của truy vấn dữ liệu. Để sử dụng Hashing, bạn khởi tạo một **hash cluster** và load các bảng tới cluster đó.Hệ thống vật lý lưu trữ các hàng của một bảng bên trong một hash cluster và truy vấn chúng theo như kết quả của hàm **hash.** Khóa của một hash cluster (chỉ như khóa của một cluster index) có thể là một cột đơn hoặc là khóa hỗn hợp. Để tìm hoặc lưu trữ một hàng trong một hash cluster, hệ thống áp dụng hàm hash tới giá trị khóa của mỗi hàng. Giá trị hash tương ứng với một **data block** trong khối cluster đó, mà hệ thống khi đó đọc và viết thay...

Chú ý. Hash cluster là một lựa chọn tốt hơn khi dùng một indexed table hoặc một index cluster khi một bảng là được truy vấn thường xuyên với các truy vấn tương đương.

Khi nào Clusters hữu ích?

Clusters index cho phép dữ liệu hàng từ một hoặc nhiều bảng mà chia sẻ một Cluster key value để được lưu trữ trong cùng Block . Bạn có thể xác định những hàng đó dùng một cluster index, mà có một mục trên cluster key value và không đối với mỗi hàng. Vì thế mà chỉ số index đó là nhỏ hơn và ít tốn chi phí hơn để truy cập đối với việc tìm nhiều hàng. Các hàng với cùng khóa Key là trong cùng một nhóm nhỏ Blocks. Điều này có nghĩa rằng trong một index cluster, yếu tố clustering là rất tốt và cung cấp clustering đối với dữ liệu từ nhiều bảng chia sẻ cùng khóa Key .

Index nhỏ hơn và nhóm blocks nhỏ hơn giảm chi phí truy cập bằng cách giảm block thăm tới cache buffer. Index clusters hữu ích khi kích thước các bảng là không biết trước. Bởi vì một cluster bucket chỉ được tạo ra sau khi một cluster key value được sử dụng. Chúng cũng hữu ích đối với tất cả các thao tác lọc và tìm kiếm. Chú ý rằng quét toàn bộ bảng không thực hiện tốt trên bảng trong nhiều bảng cluster khi nó có nhiều blocks hơn bảng sẽ có nếu đã tạo như một đống bảng .

Hash clusters cho phép dữ liệu hàng từ một hoặc nhiều bảng mà chia sẻ một cluster key value để được lưu trữ trong cùng block . Bạn có thể xác định vị các hàng đó bằng việc sử dụng hệ thống cung cấp hoặc người dùng cung cấp hàm hashing hoặc sử dụng cluster key value.

Điều này đã được phân tán đều làm cho việc truy cập vào một hàng nhanh hơn so với sử dụng index clusters . Các hàng với cùng các cluster key values băm tới cùng các cluster bucket và có thể được lưu trữ trong cùng khối hoặc các nhóm nhỏ block

Điều này có nghĩa rằng trong một Hash cluster yếu tố clustering cũng rất tốt và một hàng có lẽ được truy cập bởi chính Key của nó với một block duy nhất mà không cần một index .

Hash clusters nó phân bổ tất cả việc lưu trữ đối với tất cả hash buckets khi cluster được tạo ra. Vì vậy chúng có thể lãng phí không gian. Chúng cũng không thực hiện tốt hơn trên các tìm kiếm

tương đương khác hoặc không tương đương. Giống như index cluster nếu chúng chứa nhiều bảng, quyets toàn bộ là đắt hơn đối với cùng lý do.

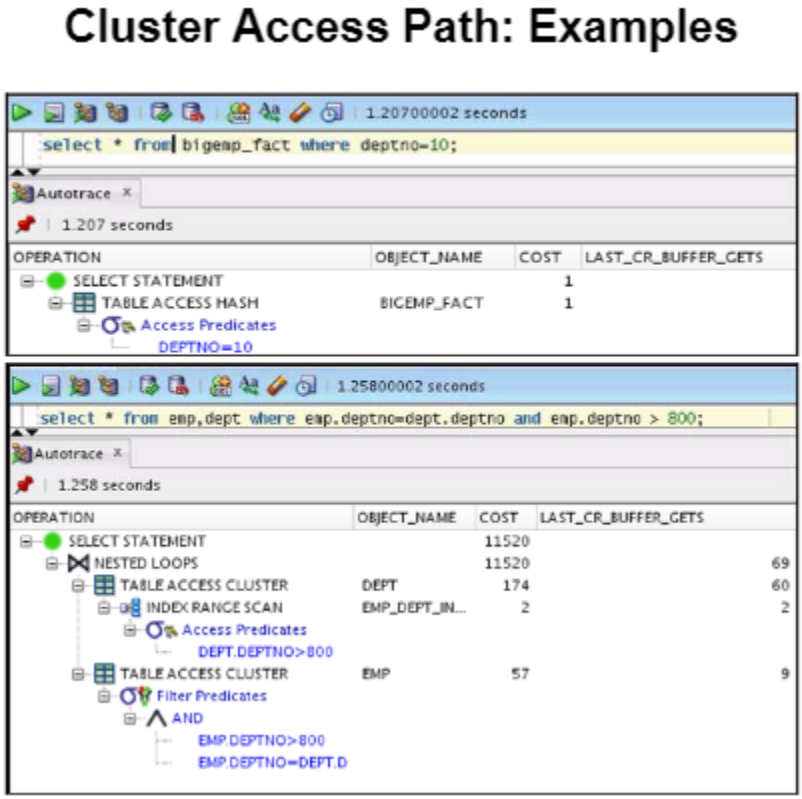
Single-table hash clusters giống như hash cluster nhưng được tối ưu trong cấu trúc block đối với việc truy cập tới một bảng đơn, qua việc cung cấp truy cập nhanh nhất có thể tới một hàng khác hơn là dùng bộ lọc "rowid". Khi chúng chỉ có một bảng , quét toàn bộ , nếu chúng xảy ra, chi phí cũng nhiều như chúng trong một heap table.

Sorted hash cluster được thiết kế để giảm chi phí truy cập dữ liệu bằng cách sử dụng một thuật toán hashing trên hash key. Việc truy cập hàng đầu tiên phù hợp với khóa băm có lẽ ít chi phí hơn so với sử dụng một IOT đối với một bảng lớn. Bởi vì nó tiết kiệm chi phí B*-tree thăm.

Tất cả các hàng phù hợp trên một hash key cụ thể(ví dụ Account number) được lưu trữ trong cluster đó theo thứ tự sắp xếp khóa hoặc các khóa(ví dự Phone calls). Vì thế nó loại bỏ sự cần thiết của xử lý sắp xếp theo thứ tự bởi tài khoản. Những cluster đó là rất tốt đối với việc báo cáo hàng loạt, thanh toán và tương tự như vậy.

Cluster Access Path: Examples

Ví dụ.



Các ví dụ trong slide cho thấy hai đường dẫn truy cập khác nhau.

Trong ví dụ đầu. Một hash Scan được xác định để định vị hàng trong một hash cluster dựa trên một hash value. Trong một hash cluster, tất cả các hàng cùng hash value được lưu trữ trong cùng các khối. Để thực hiện một hash Scan, Hệ thống đầu tiên có được hash value bằng cách áp dụng một hsah function tới một cluster key value đặc biệt bởi câu lệnh. Hệ thống khi đó sẽ quét những Block data có chứa những hàng cùng với giá trị hash value đó.

Điều thứ hai giả định rằng một cluster index đã được sử dụng để cluster cả hai bảng EMP và DEPT . Trong trường hợp này, một cluster scan được sử dụng để truy vấn, từ một bảng đã lưu trữ trong một index clustered . Tất cả các hàng đó có cùng cluster key value . Trong một Indexed cluster, Tất cả các hàng với cùng cluster key value được lưu trữ trong cùng các block data. Để thực hiện một cluster scan , hệ thống đầu tiên có được ROWID một trong các hàng được chọn bằng cách scanning cluster index đó.Hệ thống khi đó sẽ xác định vị trí các hàng dựa trên giá trị ROWID này.

Chú ý bạn thấy ví dụ làm thế nào để tạo cluster trong tiết học này.

Sorting Operators

Thao tác sắp xếp trả kết quả khi người dùng chỉ rõ một thao tác mà yêu cầu sắp xếp.

Các thao tác thường gặp bao gồm sau đây.

SORT AGGREGATE không liên quan đến một sắp xếp.Nó truy vấn một hàng đơn là kết quả của việc áp dụng một nhóm hàm tới một nhóm các hàng được chọn. Các thao tác như COUNT và MIN được xem như là SORT AGGREGATE .

SORT UNIQUE sắp xếp đầu ra các hàng mà được loại bỏ các bản sao. Nó xảy ra nếu người dùng xác định một mệnh đề DISTINCT hoặc một thao tác yêu cầu các giá trị duy nhất cho bước tiếp theo.

SORT JOIN xảy ra trong một sort-merge join, nếu các hàng cần được sắp xếp theo khóa **key join**

SORT GROUP BY được sử dụng khi tập hợp được tính cho các nhóm khác nhau trong **data.** Sắp xếp được yêu cầu để tách riêng các hàng thành các nhóm khác nhau.

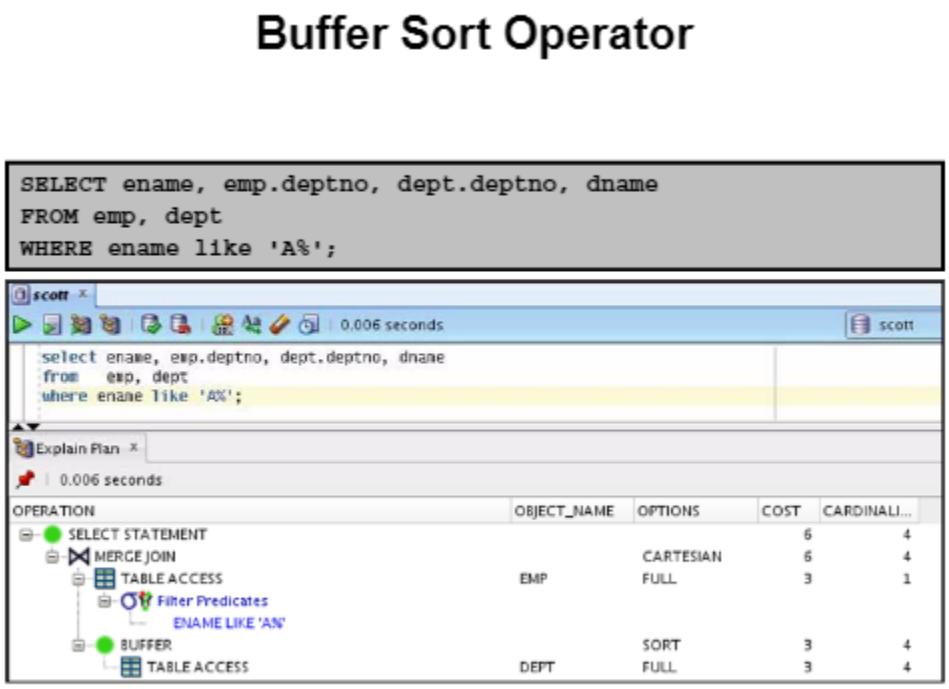
SORT ORDER BY được yêu cầu khi câu lệnh xác định một ORDER BY mà không thể đáp ứng bằng một trong các **Indexes**

HASH GROUP BY băm một tập các hàng thành các nhóm đối với một truy vấn mệnh đề GROUP BY.

HASH UNIQUE băm một tập hợp các hàng để loại bỏ các bản trung lặp , nó sảy ra nếu người dùng xác định mệnh đề DISTINCT hoặc nếu một thao tác yêu cầu các giá trị duy nhất cho bước tiếp theo. Điều này nó tương tự như SORT UNIQUE.

Chú ý: Nhiều thao tác SQL nó gây ra nhiều sắp xếp tiềm ẩn. Chẳng hạn như DISTINCT, GROUP BY , UNION, MINUS , INTERSECT . Nếu bạn muốn có các hàng theo thứ tự hãy sử dụng mệnh đề ORDER BY.

Buffer Sort Operator



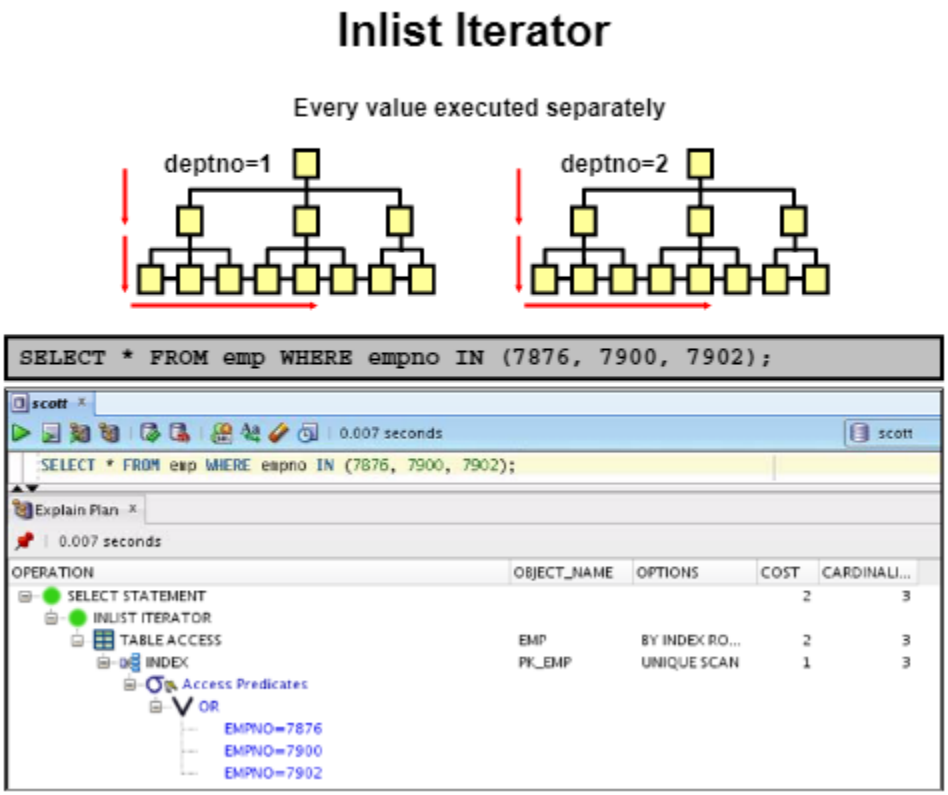
Buffer Sort Operator sử dụng bảng tạm thời hoặc một khu vực sắp xếp trong bộ nhớ để lưu trữ dữ liệu trung gian. Tuy nhiên, dữ liệu không cần thiết được sắp xếp trước.

Buffer Sort Operator là cần thiết nếu có một thao tác mà cần tất cả dữ liệu đầu vào trước khi nó có thể bắt đầu. Vì vậy, Buffer Sort sử dụng cơ chế đệm của một loại sắp xếp truyền thống nhưng nó không sắp xếp chính nó. Hệ thống này đơn giản là bộ đệm dữ liệu, trong User Global Area (UGA) hoặc Program Global Area (PGA) để tránh nhiều bảng quét lại các khối dữ liệu thực. Các cơ chế sắp xếp được tái sử dụng, bao gồm việc trao đổi đĩa khi không đủ vùng bộ nhớ sắp xếp có sẵn, nhưng mà không phân loại[sắp xếp] dữ liệu.

Sự khác nhau giữa bảng tạm thời và buffer sort như sau:

- Một bảng tạm thời sử dụng System Global Area (SGA) .
- Một loại đệm sử dụng UGA .

Inlist Iterator



Nó được sử dụng khi một truy vấn có chứa một mệnh đề IN với các giá trị hoặc nhiều vị từ tương đương trên cùng một cột liên kết với nhiều OR.

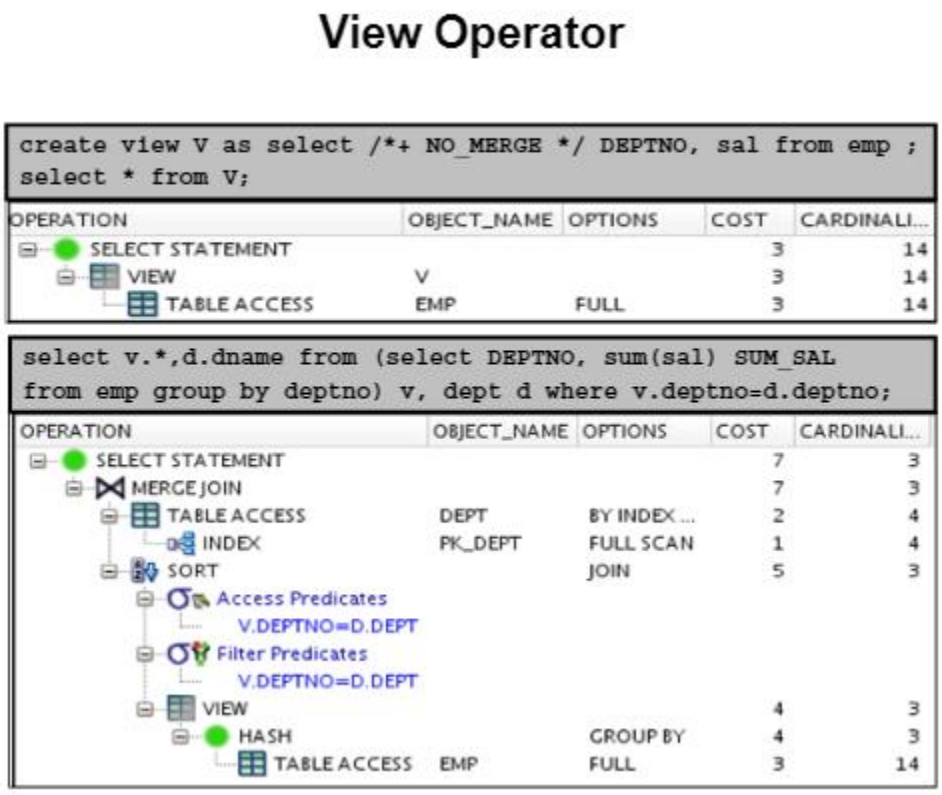
Thao tác INLIST INTERATOR lặp lại thông qua danh sách giá trị đã liệt kê, và mỗi giá trị được thực thi riêng biệt.Kế hoạch thực hiện là giống hệt với kết quả của một câu lệnh với mệnh đề tương đương thay vì IN, ngoại trừ một bước bổ sung. Bước phụ xảy ra khi Inlist Iterator cấp dữ liệu các mệnh đề tương đương với các giá trị duy nhất trong danh sách.

Bạn có thể xem các toán tử này như là For Loop lệnh trong PL/SQL. Trong ví dụ trong slide ,bạn lặp chỉ số thăm dò trên hai giá trị : 1 và 2 .

Như vậy, nó là một chức năng sử dụng **index**, được quét cho mỗi giá trị trong danh sách. Một sử lý thay thế khác là UNION ALL của mỗi giá trị hoặc một FILTER của các giá trị đối nghịch tất cả các hàng, điều đó là hiệu quả hơn đáng kể.

Việc tối ưu sử dụng một INLIST ITERATOR khi một mệnh đề IN là chỉ định với các giá trị, và tối ưu hóa tìm 1 **index** chọn lọc cho cột đó. Nếu có nhiều mệnh dề OR sử dụng cùng **index**, việc tối ưu lựa chọn thao tác này hơn CONCATENATION hoặc UNION ALL, bởi vì nó hiệu quả hơn.

View Operator



Mỗi truy vấn tạo ra một tập biến của dữ liệu trong các hình thức của 1 bảng. Một khung nhìn đơn giản là cấp cho một tên đối với tập dữ liệu này. Khi Views tham chiếu đến 1 truy vấn, hệ thống có thể xử lý chúng trong 2 cách. Nếu 1 số điều kiện được đáp ứng, chúng có thể được sáp nhập vào truy vấn chính. Điều đó có nghĩa các View text được viết lại như Join với các bảng khác trong truy vấn. Lần này cũng có thể để lại như quan điểm đơn và lựa chọn từ trực tiếp như trong trường hợp của 1 bảng. Vị từ cũng có thể được đẩy vào hoặc lấy ra miễn là thỏa mãn điều kiện nhất định.

Khi một khung nhìn không được sáp nhập, bạn có thể thấy toán tử VIEW. Các thao tác được thực hiện riêng biệt. Tất cả các hàng từ Khung nhìn đó được trả về, và các thao tác tiếp theo có thể được thực hiện.

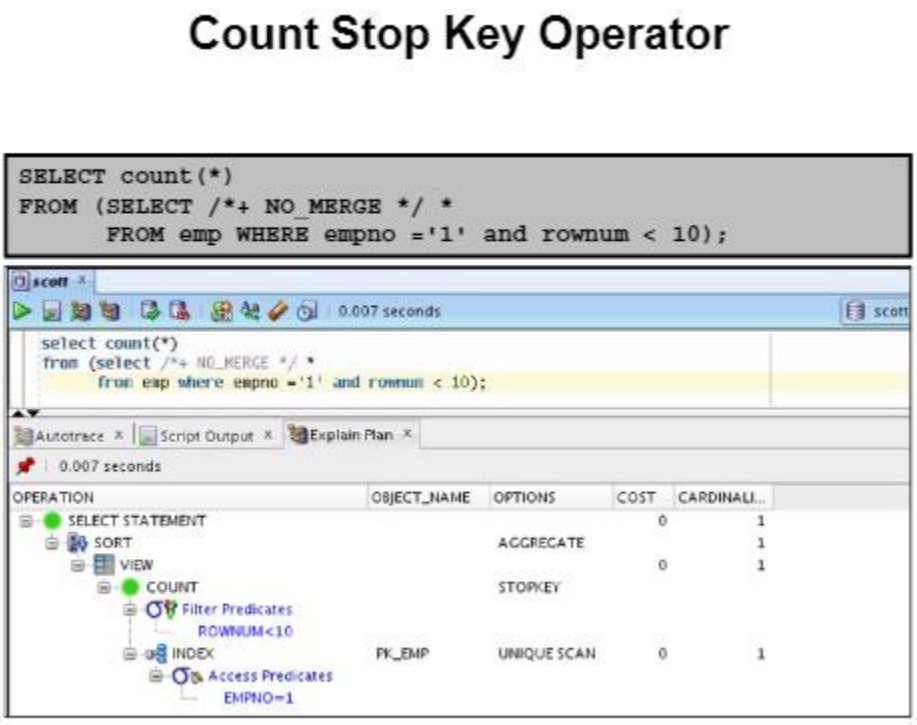
Đội khi một Khung nhìn[View] không thể được sáp nhập và phải thực hiện một cách độc lập trong 1 khối truy vấn tách biệt. Trong trường hợp này, bạn cũng có thể nhìn VIEW toán tử trong kế hoạch giải thích. Từ khóa VIEW chỉ ra rằng Khung nhìn đó[View] được thực hiện như 1 khối truy vấn tách biệt.

Ví dụ, VIEW chứa GROUP BY chức năng không thể sáp nhập.

Ví dụ thứ 2 trong slide cho thấy 1 không thể được sáp nhập bên trong VIEW.

Một **Inline VIEW** về cơ bản 1 truy vấn bên trong mệnh đề FROM của câu lệnh của bạn. Cơ bản, thao tác này thu thập tất cả các hàng từ 1 khối truy vấn trước khi chúng có thể được xử lý bởi thao tác cao hơn trong kế hoạch.

Count Stop Key Operator (khóa dừng đếm)



ví dụ:

SELECT COUNT(*)

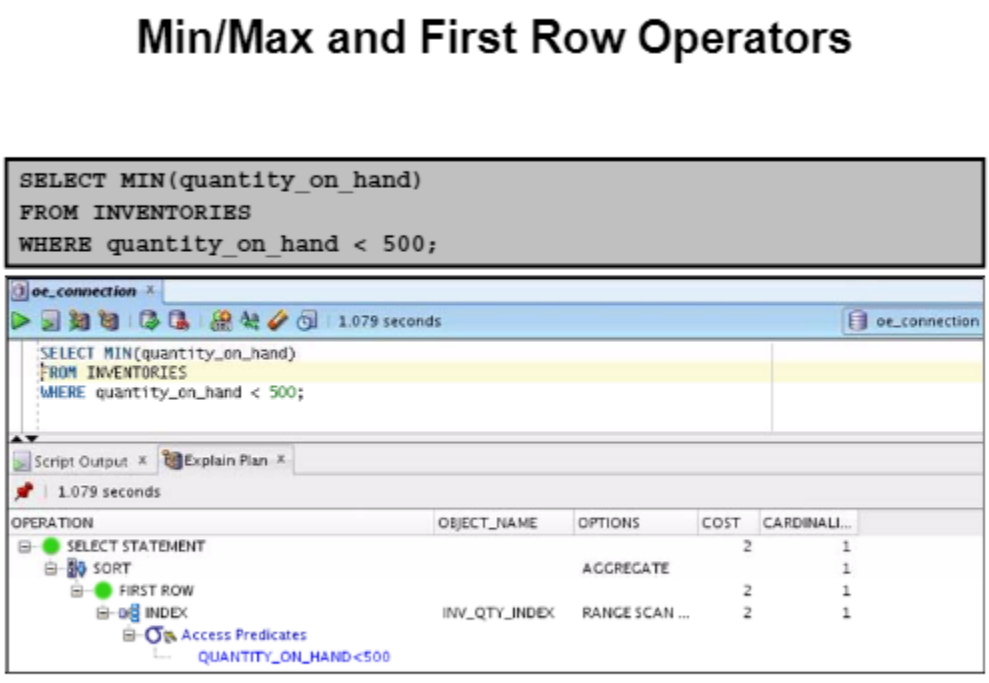
FROM (SELECT /* +NO_MARGE */ *

FROM emp WHERE empno = ’1’ and rownum<10)

Count stopkey là giới hạn số lượng hàng trả về . Các giới hạn được thể hiện bằng biểu thức ROWNUM trong mệnh để WHERE. Nó chấm dứt các thao tác hiện tại khi đếm đủ số lượng đạt được.

Chi phí thực thi truy vấn dựa trên tần xuất xuất hiện của các giá trị mà chúng ta muốn lấy. Nếu giá trị xuất hiện thường xuyên trong bảng thì việc đếm sẽ được thực hiện nhanh chóng. Nếu giá trị hiếm xuất hiện, và không có **indexs** thì hệ thống sẽ đọc hầu hết các khối của bảng trước khi đếm đạt đến yêu cầu.

Min/Max and First Row Operators



Ví dụ:

SELECT MIN(quantity_on_hand)

FROM INVENTORIES

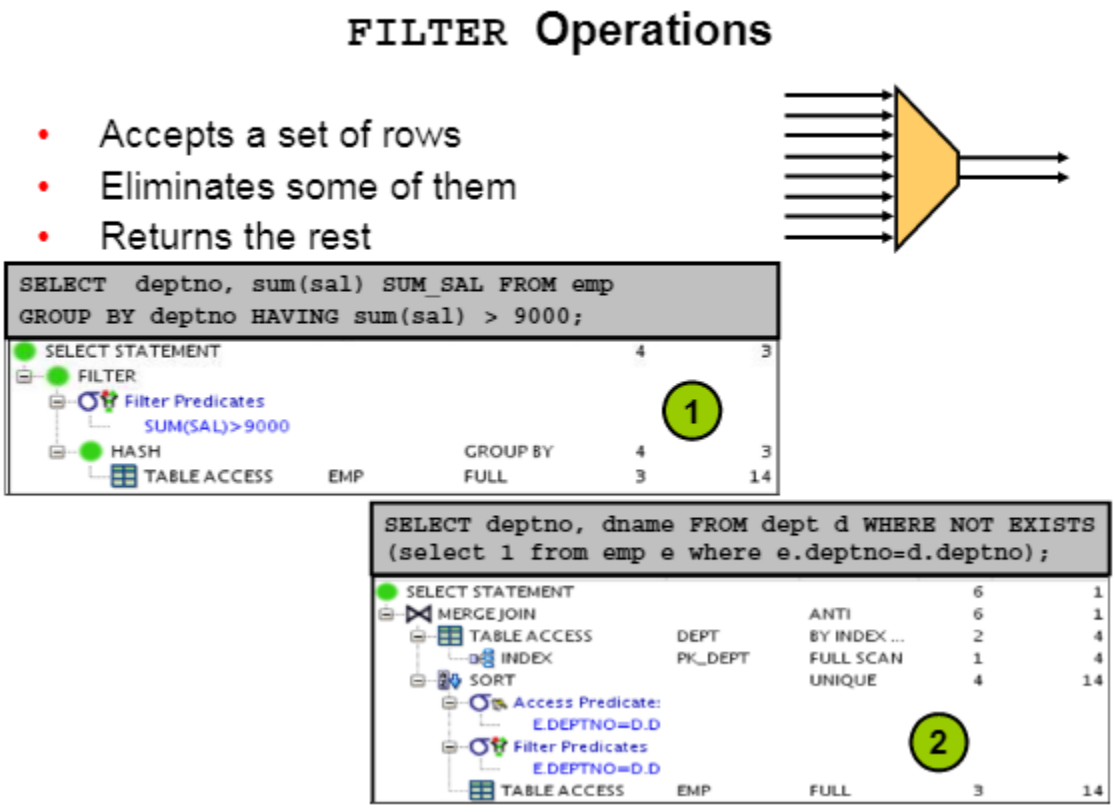
WHERE quantity_on_hand < 500;

Truy vấn FIRSTROW chỉ lấy hàng đầu tiên được chọn bởi truy vấn. Nó dừng lại khi hàng đầu tiên được trả về. Điều này tối ưu hóa trong Oracle 8, và nó làm việc với các **Index** quét phạm vi và quét toàn bộ.

Trong slide ví dụ được giả sử rằng có một **index** trên cột quantity_on_hand

Other N-Array Operations

FILTER Operations



Ví dụ 1:

```
SELECT deptno, sum(sal) SUM_SAL

FROM emp

GROUP BY deptno

HAVING sum(sal) > 9000;
```

Ví dụ 2:

```
SELECT deptno, dname

FROM dept d

WHERE NOT EXISTS (select 1 from emp e where e.deptno=d.deptno);
```

A FILTER operation là bất kì thao tác nào loại bỏ các hàng trả về bởi một bước khác., nhưng không liên quan tới việc lấy các hàng của nó .Tất cả các thao tác có thể bị lọc, bao gồm các truy vấn con,và các vị từ trong bảng.

Trong ví dụ 1: Bộ lọc áp dụng cho các nhóm được tạo ra bởi thao tác Group by.

Trong ví dụ 2: Bộ lọc được sử dụng cùng một cách như NESTED LOOPS. DEPT được truy cập một lần, và mỗi hàng từ DEPT, EMP được truy cập bởi index DEPTNO. Thao tác này được thực hiện nhiều lần bằng số lượng hàng trong DEPT

Các thao tác được áp dụng, cho mỗi hàng , sau khi DEPT được lấy. Bộ lọc loại bỏ các hàng truy vấn bên trong trả về ít nhất một hàng (select 1 from emp e where e.deptno=d.deptno) là đúng.

Concatenation Operation

Concatenation Operation

```
SELECT * FROM emp WHERE deptno=1 or sal=2;

-----------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     8 |   696 |
|   1 |  CONCATENATION                |          |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP      |     4 |   348 |
|   3 |    INDEX RANGE SCAN           | I_SAL    |     2 |       |
|   4 |   TABLE ACCESS BY INDEX ROWID| EMP      |     4 |   348 |
|   5 |    INDEX RANGE SCAN           | I_DEPTNO |     2 |       |
-----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("SAL"=2)
   4 - filter(LNNVL("SAL"=2))
   5 - access("DEPTNO"=1)
```

Sự liên kết các hàng trả lại bởi hai hoặc nhiều bộ hàng. Việc này giống như UNION ALL và không loại bỏ các hàng trùng lặp.

Nó được sử dụng với nhiều OR. Tuy nhiên, OR không trả lại hàng trùng lặp, vì vậy đối với mỗi thành phần sau thành phần đầu tiên, nó gắn thêm một phủ định của các thành phần trước (LNNVL):

CONCATENATION

- BRANCH 1 - SAL=2

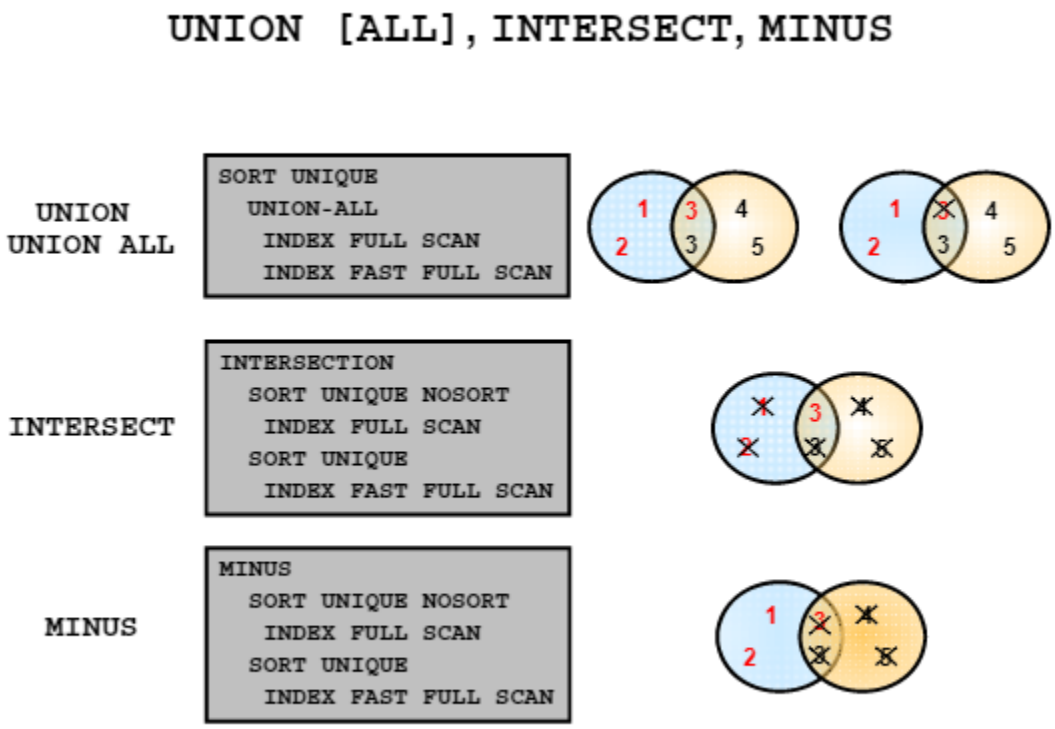
- BRANCH 2 - DEPTNO = 1 AND NOT row in Branch 1

Các hàm LNNVL được tạo ra bởi các mệnh đề OR để xử lý phủ định này.

Hàm LNNVL trả về giá trị TRUE nếu thuộc tính là NULL hoặc FALSE.

Vì vậy, bộ lọc (LNNVL (SAL = 2)) trả về tất cả các hàng mà SAL! = 2 hoặc SAL là NULL.

UNION [ALL], INTERSECT, MINUS



SQL xử lý các hàng trùng lặp với ALL hoặc DISTINCT ở những nơi khác nhau trong ngôn ngữ đó.

ALL giữ các bản sao và DISTINCT loại bỏ chúng.

Dưới đây là một mô tả ngắn gọn về các thao tác của SQL có thể có:

INTERSECTION Thao tác chấp nhận hai bộ hàng và trả lại các giao nhau của các bộ đó, loại bỏ các bản trùng lặp.Tài nguyên Subrow được thực hiện hoặc tối ưu hóa riêng biệt.

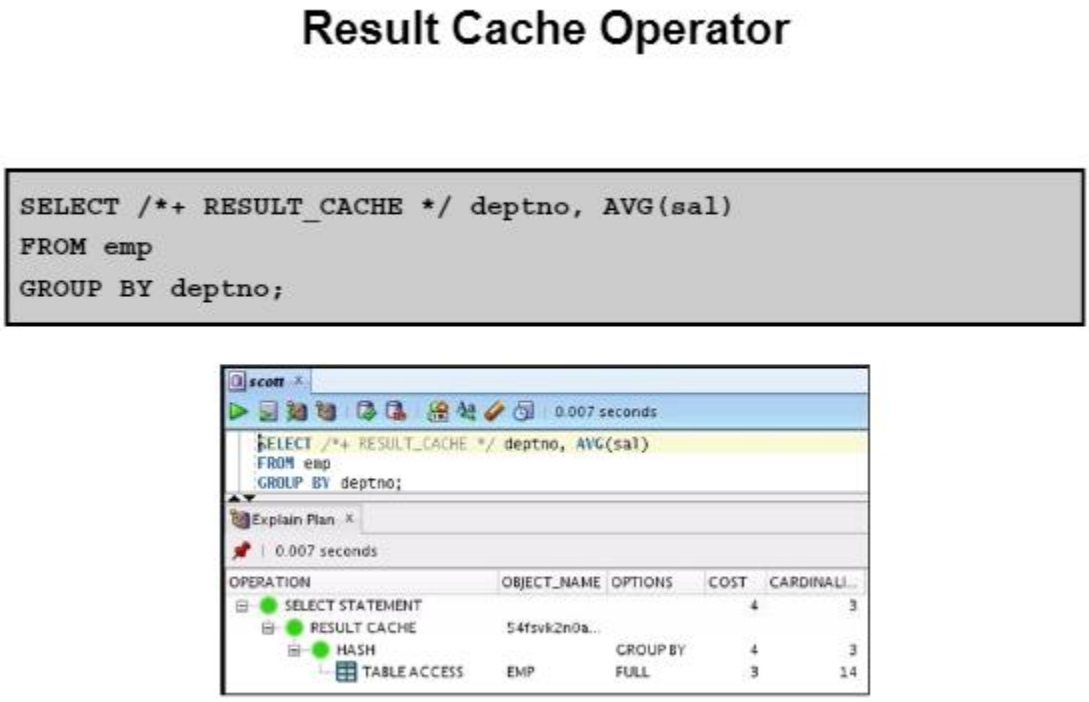
Điều này là tương tự như sắp xếp, hợp nhất, tham gia tiến trình: tất cả các hàng được sắp xếp phù hợp.

MINUS Thao tác chấp nhận hai bộ hàng và trả lại hàng xuất hiện trong bộ thứ nhất, nhưng không phải trong bộ thứ hai, loại bỏ các bản trùng lặp. Tài nguyên Subrow được thực hiện hoặc tối ưu hóa riêng biệt. Tương tự như quá trình INTERSECT. Tuy nhiên, thay vì phù hợp và trả về, đó là phù hợp và loại trừ.

UNION : Thao tác chấp nhận hai bộ hàng và trả lại liên hợp của các bộ, loại bỏ các trùng lặp. Tài nguyên Subrow được thực hiện hoặc tối ưu hóa riêng biệt. Hàng đã lấy được nối và sắp xếp để loại bỏ các hàng trùng lặp.

UNION ALL : Thao tác chấp nhận hai bộ hàng và trả lại liên hợp của các bộ và không loại bỏ trùng lặp.Các thao tác sắp xếp chi phí cao là không cần thiết. Sử dụng UNION ALL nếu bạn biết bạn không phải đối phó với các bản sao.

Result Cache Operator



Bộ nhớ đệm kết quả truy vấn SQL cho thấy bộ nhớ đệm rõ ràng của bộ kết quả truy vấn và đoạn truy vấn trong bộ nhớ cơ sở dữ liệu.

Một bộ nhớ đệm chuyên dụng được lưu trữ trong các **shared pool** có thể được sử dụng để lưu trữ và truy xuất các kết quả đã **cached**.

Các kết quả truy vấn được lưu trữ trong bộ nhớ cache này trở thành không hợp lệ khi dữ liệu trong các đối tượng cơ sở dữ liệu được truy cập bởi các truy vấn bị sửa đổi.

Mặc dù bộ nhớ đệm truy vấn SQL có thể được sử dụng cho bất kỳ truy vấn, báo cáo ứng cử viên tốt là những người cần truy cập vào một số lượng rất lớn các hàng để trả về chỉ có một phần nhỏ trong số đó. Đây là các trường hợp chủ yếu đỗi với các ứng dụng kho dữ liệu.

Nếu bạn muốn sử dụng bộ nhớ cache kết quả truy vấn và các tham số khởi tạo RESULT_CACHE_MODE được thiết lập để MANUAL, bạn phải xác định một cách rõ ràng những gợi ý RESULT_CACHE trong truy vấn của bạn. Điều này giới thiệu các thao tác ResultCache vào kế hoạch thực hiện cho truy vấn. Khi bạn thực hiện truy vấn, các nhà điều hành ResultCache nhìn lên bộ nhớ cache kết quả để kiểm tra xem các kết quả cho truy vấn đã tồn tại trong bộ nhớ cache. Nếu nó tồn tại, kết quả được lấy trực tiếp từ bộ nhớ cache.

Nếu nó chưa tồn tại trong bộ nhớ cache, các truy vấn được thực thi, kết quả trả về là đầu ra, và cũng được lưu giữ trong bộ nhớ cache kết quả.Nếu các tham số khởi tạo RESULT_CACHE_MODE được thiết lập FORCE, và bạn không muốn lưu Kết quả của một truy vấn trong bộ nhớ cache kết quả, khi đó bạn phải sử dụng các gợi ý NO_RESULT_CACHE trong truy vấn của bạn.

III. Kết luận

Qua học phần Thiết kế và quản trị cơ sở dữ liệu, chúng em đã tìm hiểu được thêm nhiều kiến thức mới về thiết kế quản trị cơ sở dữ liệu, bên cạnh đó những kiến thức về tối ưu hóa truy vấn và các phương pháp tối ưu khác cũng rất bổ ích.

Việc làm bài tập lớn cùng với tìm hiểu cuốn sách **Oracle Tunning** cũng cung cấp cho chúng em thêm rất nhiều kiến thức bổ ích. Vì thời gian có hạn nên chúng em mới chỉ tìm hiểu chương 8 Other Optimizer Operators của cuốn sách này. Hy vọng trong thời gian tới, chúng em cũng sẽ dành thời gian để tìm hiểu thêm các phần còn lại của cuốn sách này.

Cuối cùng chúng em cảm ơn thầy đã giới thiệu cuốn sách hay và bổ ích như vậy.