vahidusefzadeh@gmail.com

Join methods

زمانی که یک دستور با چندین شرط join اجرا می شود، عمل join در هر لحظه تنها بین دو جدول آن دستور انجام خواهد شد یعنی به عبارت دیگر، عمل join یک عمل باینری می باشد حال برای تعیین ترتیب join بین جداول، باید حالات مختلفی توسط اوراکل بررسی شود که این کار می تواند وقت زیادی را از سیستم بگیرد برای جلوگیری از این اتلاف وقت، می توان ترتیب پیوند را با استفاده از هینت ORDERED تعیین کرد یعنی با استفاده از این هینت در درون دستور، ترتیب پیوندها بر اساس ترتیب جداول در عبارت from تعیین شوند.

select /*+ ordered */ * from **k,u,v** where v.a=u.a and v.a=k.a

cost:2082837

select /*+ ordered */ * from **v,k,u** where v.a=u.a and v.a=k.a

cost:20834

دلیل اختلاف هزینه بین دو دستور بالا، به اندازه جدول v بر می گردد، که در قسمت روشهای پیوند بین جداول، به آن خواهیم پرداخت.

روش دیگر برای تعیین این اولویت، استفاده از هینت leading می باشد با این هینت به صراحت می توانیم ترتیب جداول را در هنگام join مشخص کنیم و نیازی به تغییر قسمتهای دیگر کد نخواهیم داشت.

select /*+ LEADING(k,u,v) */ * from k,u,v where v.a=u.a and v.a=k.a;

در ادامه سه روش join بین جداول را مورد بررسی قرار خواهیم داد.

Nested loop

دستور زیر را در نظر بگیرید:

select * from v,u where v.a=u.a;

با استفاده از روش Nested loop دستور بالا به شکل زیر قابل تفسیر است:

```
begin
For i in (select * from v) loop
For j in (select * from u where a=i.a) loop
dbms_output.put_line('v.A: '||i.a||'  u.a: '|| j.a);
End loop;
End loop;
end;
```

vahidusefzadeh@gmail.com

در این روش، یکی از جداول در حلقه بیرونی قرار می گیرد که به آن outer table یا جدول بیرونی می گویند و به جدول دیگر هم که در قسمت حلقه درونی قرار می گیرد اصطلاحا inner table یا جدول درونی می گویند به ازای هر سطر موجود در جدول بیرونی، یکبار باید جدول داخلی خوانده شود که optimizer باید جدول بزرگتر را به عنوان جدول درونی و جدول کوچکتر را به عنوان جدول بیرونی یا driving table در نظر بگیرد.

زمانی که نقشه اجرایی دستور خوانده می شود، اولین جدول بعد از nested loop باید به عنوان driving table شناخته شود.

در این روش اگر جدول بیرونی به اندازه ای کوچک باشد که به راحتی بتوان با یک full table scan آن را به حافظه برد و جدول درونی هم ایندکسی بر روی ستون مشروط داشته باشد، هزینه بسیار کاهش می یابد.

این روش معمولا برای دسترسی به driving table از full table scan یا INDEX RANGE SCAN استفاده می کند و برای جدول دوم هم در صورت امکان از INDEX RANGE SCAN بهره می گیرد.

معمولا در شرایط زیر بهینه گر به سراغ این روش می رود:

۱. اندازه جداول کوچک باشد.
۲. بهینه گر در حالت FIRST_ROWS قرار داشته باشد.
۳. بر روی ستونی که در شرط پیوند قرار دارد، ایندکس موجود باشد مخصوصا زمانی که بر روی ستون شرطی جدول درونی، ایندکس unique وجود داشته باشد.

برای اجبار کردن optimizer به استفاده از این روش، می توانیم از هینت use_nl استفاده کنیم.

مثال:

```
exec dbms_stats.set_table_stats(ownname => 'SYS', tabname => 'U', numrows => 3138360,numblks =>
76765,avgrlen => 10);

exec dbms_stats.set_table_stats(ownname => 'SYS', tabname => 'V', numrows => 100,numblks => 1,avgrlen => 6);

exec dbms_stats.set_index_stats(ownname => 'SYS', indname => 'UU', numrows => 3138360,numlblks => 4433);

select  /*+ use_nl(v,u) */ v.a,u.a from v,u where v.a=u.a;
```

table	Count(*)	index
v	100	-
u	3138360	+

method	Cost (%CPU)
HASH JOIN	20832 (1)

vahidusefzadeh@gmail.com

NESTED LOOP	**1102 (0)**
MERGE	33354 (1)

```
-------------------------------------------------------------------------------

| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |

-------------------------------------------------------------------------------

|   0 | SELECT STATEMENT             |      |   789 | 12624 |  1102   (0) | 00:00:14 |

|   1 |  NESTED LOOPS                |      |   789 | 12624 |  1102   (0) | 00:00:14 |

|   2 |   NESTED LOOPS               |      |   800 | 12624 |  1102   (0) | 00:00:14 |

|   3 |    TABLE ACCESS FULL         | V    |   100 |   600 |     2   (0) | 00:00:01 |

|* 4  |    INDEX RANGE SCAN          | UU   |     8 |       |     2   (0) | 00:00:01 |

|   5 |   TABLE ACCESS BY INDEX ROWID| U    |     8 |    80 |    11   (0) | 00:00:01 |

-------------------------------------------------------------------------------
```

Hash join

این روش نسبت به روشهای دیگر کاربرد بسیار بیشتری دارد به طوری که در اکثریت مواقع، این روش توسط بهینه گر انتخاب می شود.

در این روش ابتدا سعی می شود جدول کوچکتر شناسایی شود و سپس با الگوریتمی خاص، hash شده و به داخل حافظه برده شود و بعد از آن، جدول hash شده به ازای هر سطر جدول دوم مورد بررسی قرار گیرد.

زمانی که جدول کوچک تر، قابلیت جایگیری کامل را در حافظه داشته باشد کارایی این روش به حداکثر می رسد و زمانی هم که نتوان آن را به طور کامل در حافظه جای داد، باید به قسمتهای کوچکتری تقسیم شود و بعضی از قسمتهای آن در temporary segment قرار گیرد بنابرین مصرف حافظه در این روش نسبتا زیاد می باشد.

در زمان استفاده از هینت ordered یا leading، باید جدول کوچکتر را در ابتدا قرار دهیم تا سرعت اجرای دستور با استفاده از روش hash join بهتر شود.

هینت مربوط به این روش، use_hash می باشد.

معمولا در شرایط زیر بهینه گر به سراغ این روش می رود:

۱. optimizer در حالت ALL_ROWS قرار داشته باشد.

vahidusefzadeh@gmail.com

۲. جداول بزرگ باشند.
۳. یکی از جداول بزرگ و دیگری کوچک باشد.

مثال: اگر در مثال قبلی، ایندکس uu را حذف کنیم، بهینه گر دیگر از روش nested loop استفاده نخواهد کرد و روش انتخابی آن به hash join تغییر خواهد کرد:

table	Count(*)	index
v	100	-
u	3138360	-

method	Cost (%CPU)
HASH JOIN	**20832 (1)**
NESTED LOOP	872K (1)
MERGE	33354 (1)

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |   789 | 12624 | 20832   (1)| 00:04:10 |
|*  1|  HASH JOIN         |      |   789 | 12624 | 20832   (1)| 00:04:10 |
|   2 |   TABLE ACCESS FULL| V    |   100 |   600 |     2   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| U    |  3138K|    29M| 20821   (1)| 00:04:10 |
-------------------------------------------------------------------------------
```

Sort merge join

این روش به طور سنتی به عنوان جایگزین روش nested loop استفاده می شد ولی با آمدن hash join از اوراکل 7.3 به بعد، به دلیل کارایی مطلوبی که روش hash join دارد، این روش به ندرت در مواردی به کار گرفته می شود معمولا در این روش ایندکسها مورد استفاده قرار نمی گیرند و هر دو جدول به صورت full table scan به داخل حافظه آورده می شوند به همین دلیل بهتر است در هنگام اجرای دستور با این روش، از parallel query هم بهره گرفته شود.

```
select   * /*+ use_merge(v,u) parallel(v,2)  parallel(u,2)  */  from u,v where u.a<=v.a;
```

در این روش، دو جدول باید به صورت مرتب شده به داخل حافظه آورده شوند و زمانی که جداول مرتب هستند و هزینه مرتب سازی آنها گران نیست، این روش مناسب می باشد.

vahidusefzadeh@gmail.com

زمانی که حجم جداول بزرگ باشند، این روش معمولا بهتر از nested loop ظاهر می شود ولی به خوبی hash join نیست و زمانی از hash join بهتر کار می کند که ستونهایی که قرار است با هم مقایسه شوند، مرتب باشند و یا نیازی به مرتب کردن آنها نباشد و ضمنا به هر دلیلی sort در دستور نیاز باشد برای مثال در دستور از order by استفاده شده باشد، در این صورت شاید optimizer از این روش است کند.

همچنین درصورتی که از عملگرهای =< ,<= ,> در دستور استفاده شده باشد، باید از این روش استفاده کرد چون روش hash join نمی تواند این کار را انجام دهد و اگر حجم داده بالا باشد، nested loop کارایی مطلوبی ندارد.

هینت مربوط به این روش، use_merge می باشد.

در موارد زیر sort merge join کاربرد دارد:

۱. ایندکسی رو ستونهای پیوندی موجود نباشد.
۲. وقتی query بیشتر اطلاعات هر دو جدول را بر می گرداند.
۳. وقتی full table scan سریعتر از index scan باشد.
۴. خروجی دستور باید به صورت مرتب تولید شوند.

مثال:

```
select   * from u,v where u.a<=v.a;
```

method	Cost (%CPU)
HASH JOIN	-
NESTED LOOP	872K (1)
MERGE	34197 (3)

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT    |      |  313M |  4788M|       | 34197  (3) | 00:06:51 |
|  1 |  MERGE JOIN         |      |  313M |  4788M|       | 34197  (3) | 00:06:51 |
|  2 |   SORT JOIN         |      |  100  |  600  |       |   3  (34)  | 00:00:01 |
|  3 |    TABLE ACCESS FULL| V    |  100  |  600  |       |   2  (0)   | 00:00:01 |
|* 4 |   SORT JOIN         |      | 3138K |   29M |  120M | 33351  (1) | 00:06:41 |
|  5 |    TABLE ACCESS FULL| U    | 3138K |   29M |       | 20821  (1) | 00:04:10 |
-------------------------------------------------------------------------------
```