# Tutorial Week 8: SQL Tuning

Objective is to understand how SQL tuning done in Oracle. To start first finding how your sql is executed and how much time it is taken for execution. You have to use this time “SH” schema or user. Please unlock and set password for “SH” user using your Enterprise Manager.

In your current session, type following:

set echo on

/

set timing on

/

set autotrace on

/

After that before you do each part of this tutorial, flush the cached data using:

Alter system flush shared\_pool

/

Alter system flush buffer\_cache

/

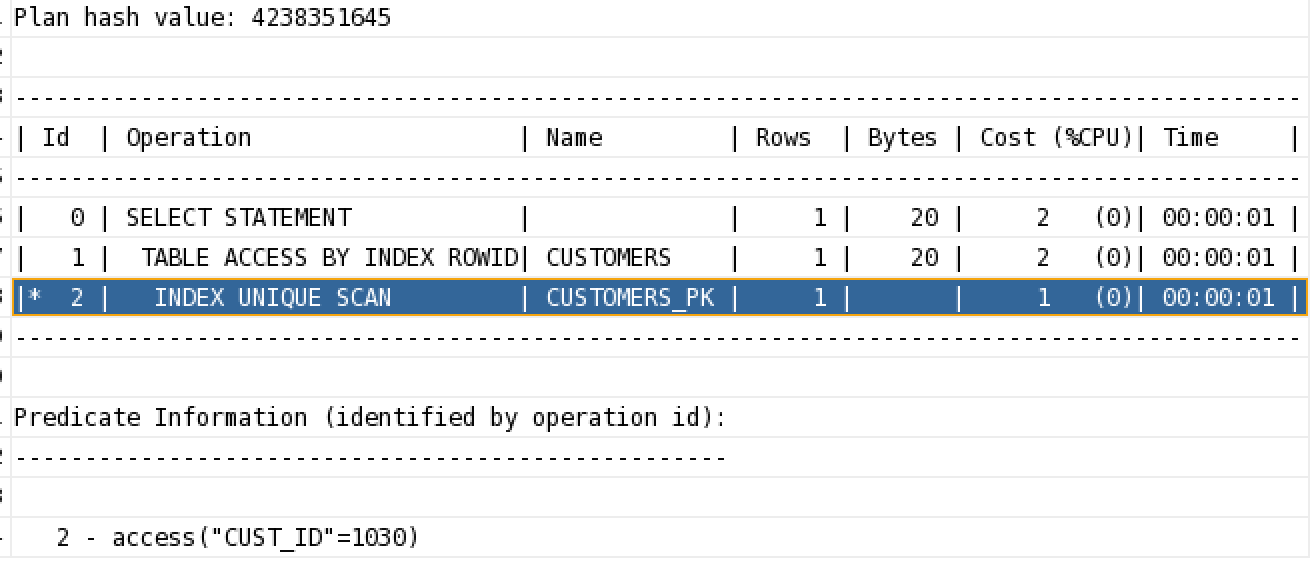
## Q1. Type following select statements and compare their execution plans and time. What is difference you find?

explain plan for select cust\_first\_name, cust\_last\_name from customers where cust\_id = 1030;

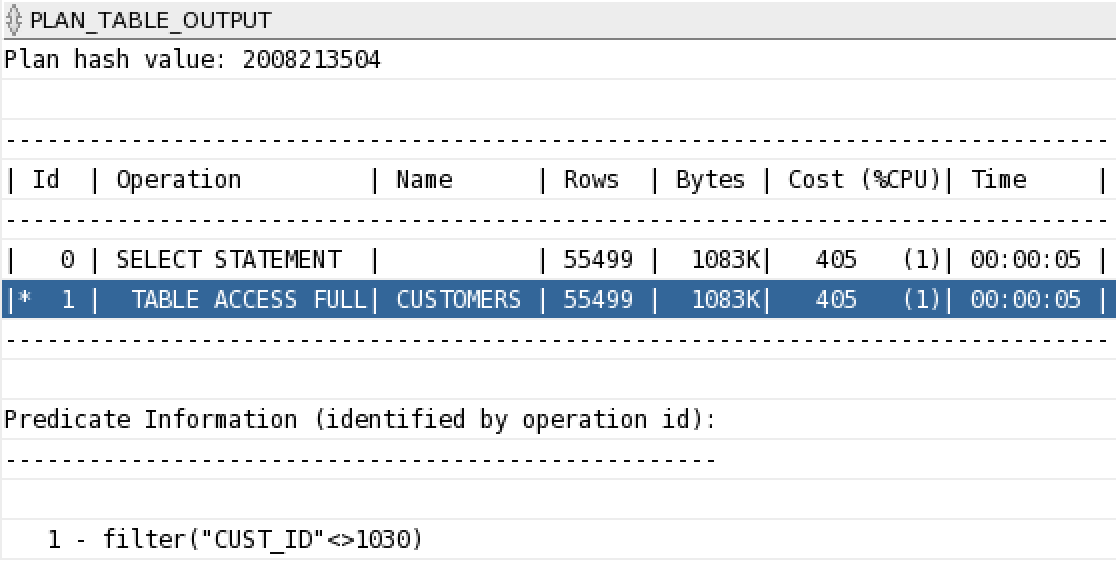
select \* from table(dbms\_xplan.display);

index is better than full scan

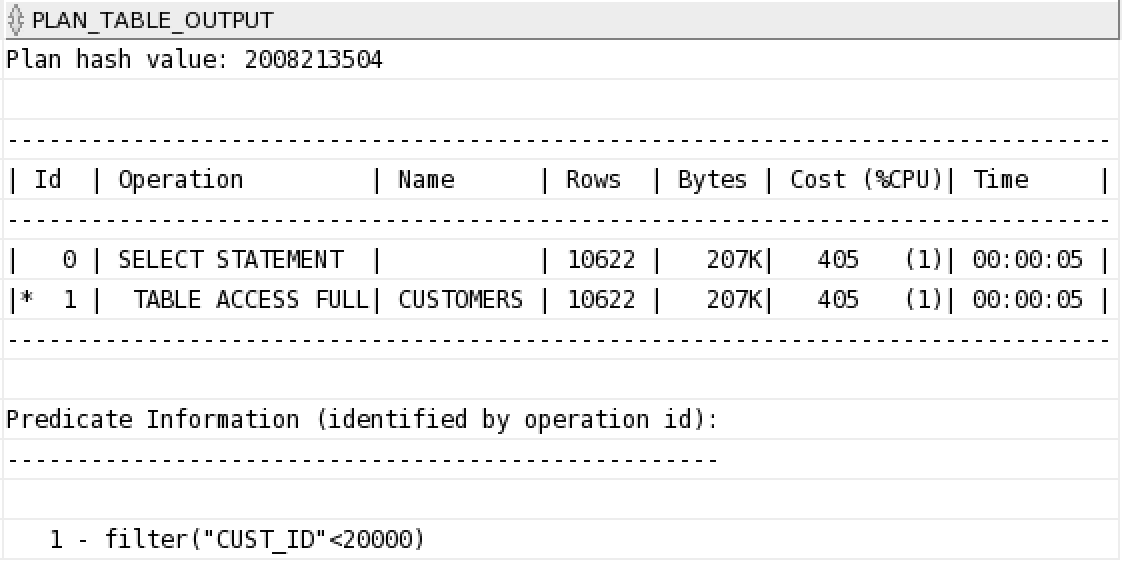
* select cust\_first\_name, cust\_last\_name from customers where cust\_id = 1030;



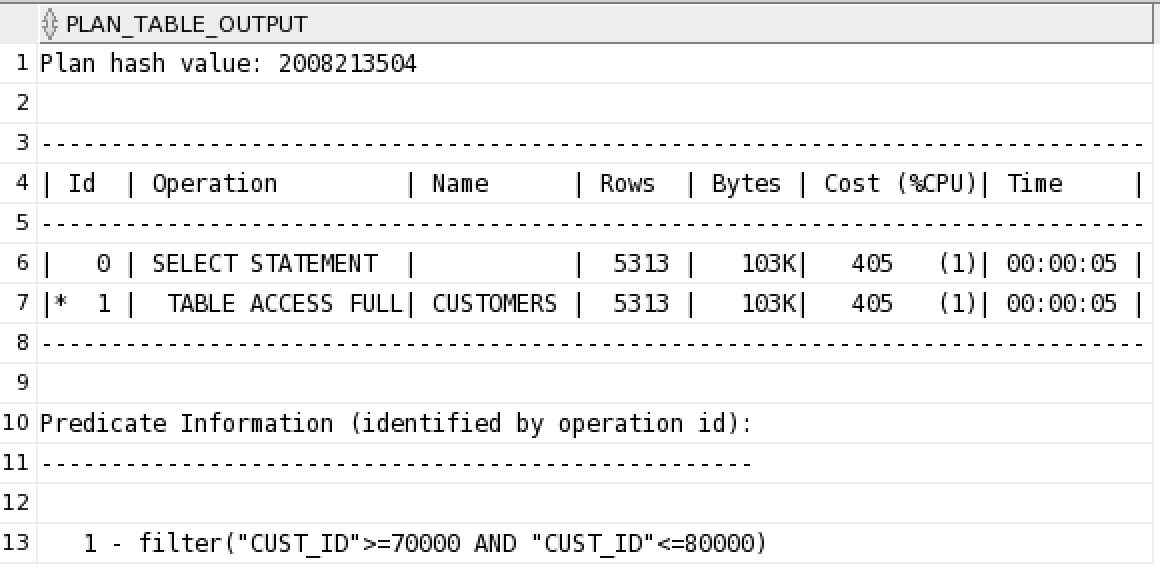
* select cust\_first\_name, cust\_last\_name from customers where cust\_id <> 1030;



* select cust\_first\_name, cust\_last\_name from customers where cust\_id < 20000;



* select cust\_first\_name, cust\_last\_name from customers where cust\_id between 70000 and 80000



## Q2. Create an index on the CUST\_CREDIT\_LIMIT column of the CUSTOMERS table. After that, compare the execution plan and timing of following queries

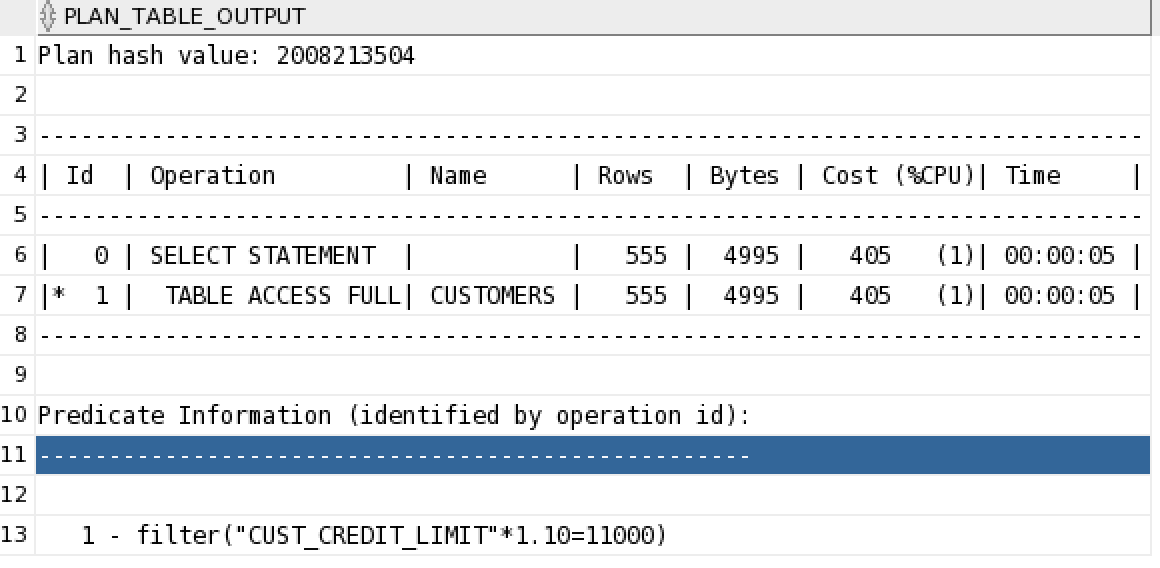
Create index iname on customers (cust\_credit\_limit)

The first diagram is no index on Cust\_Credit\_limit

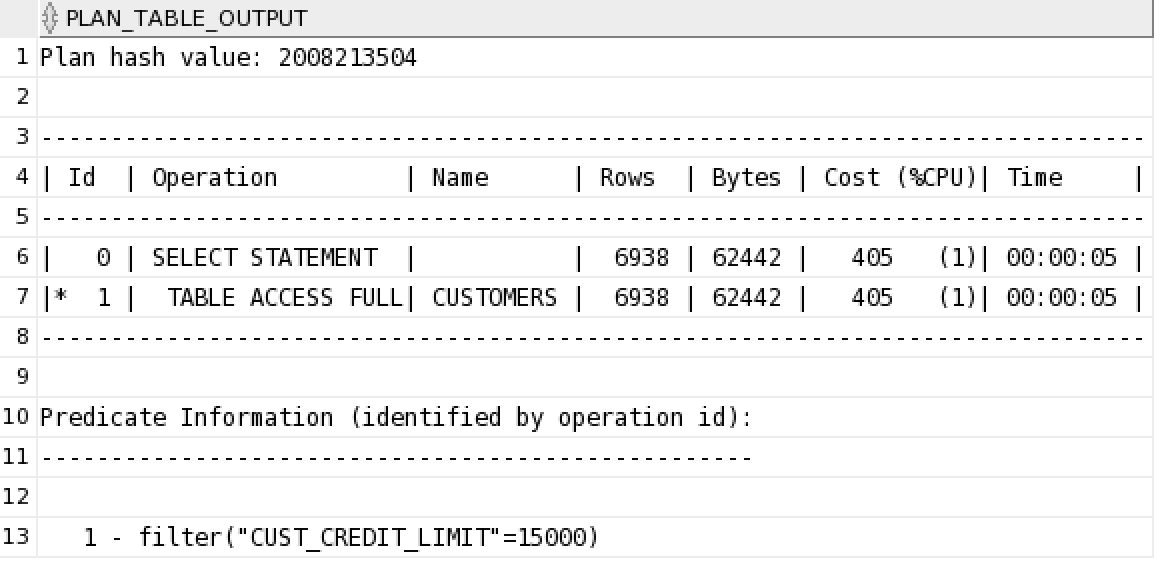
Calculation before would not used index

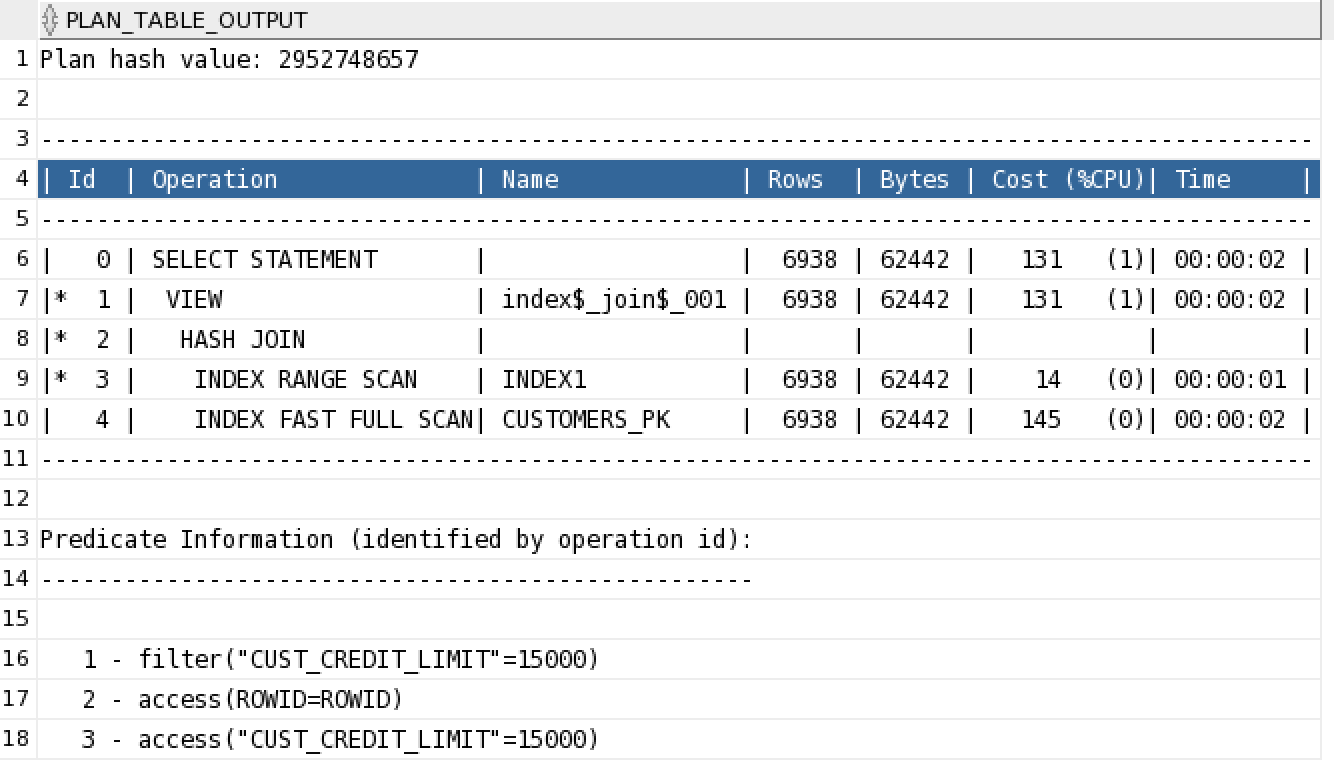
Should calculate first before set value

select cust\_id from customers where cust\_credit\_limit\*1.10 = 11000



select cust\_id from customers where cust\_credit\_limit = 30000/2

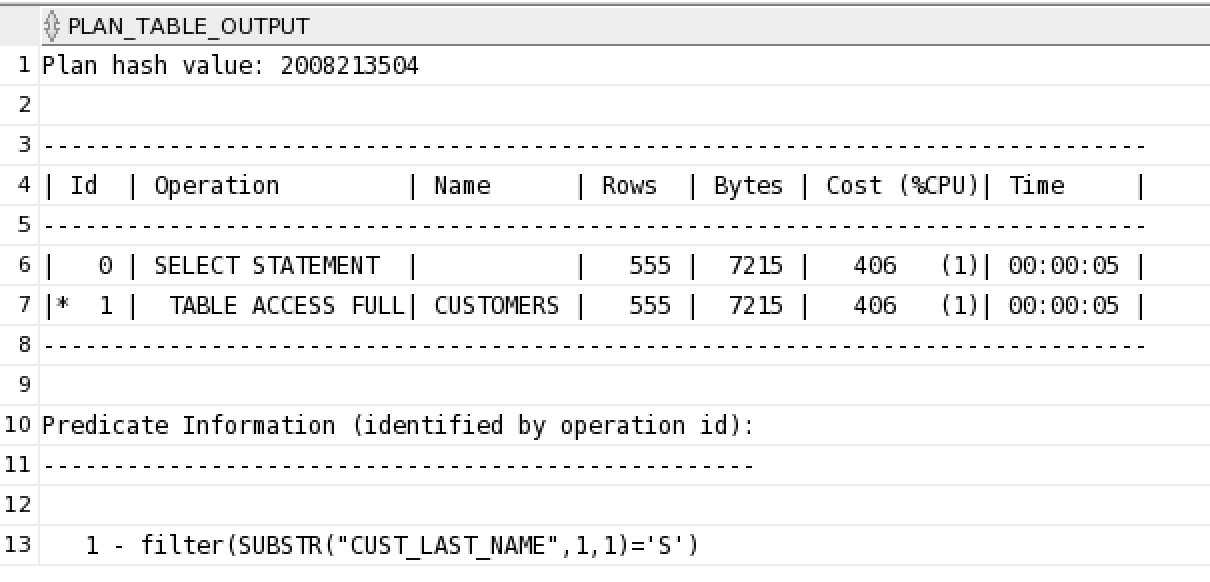




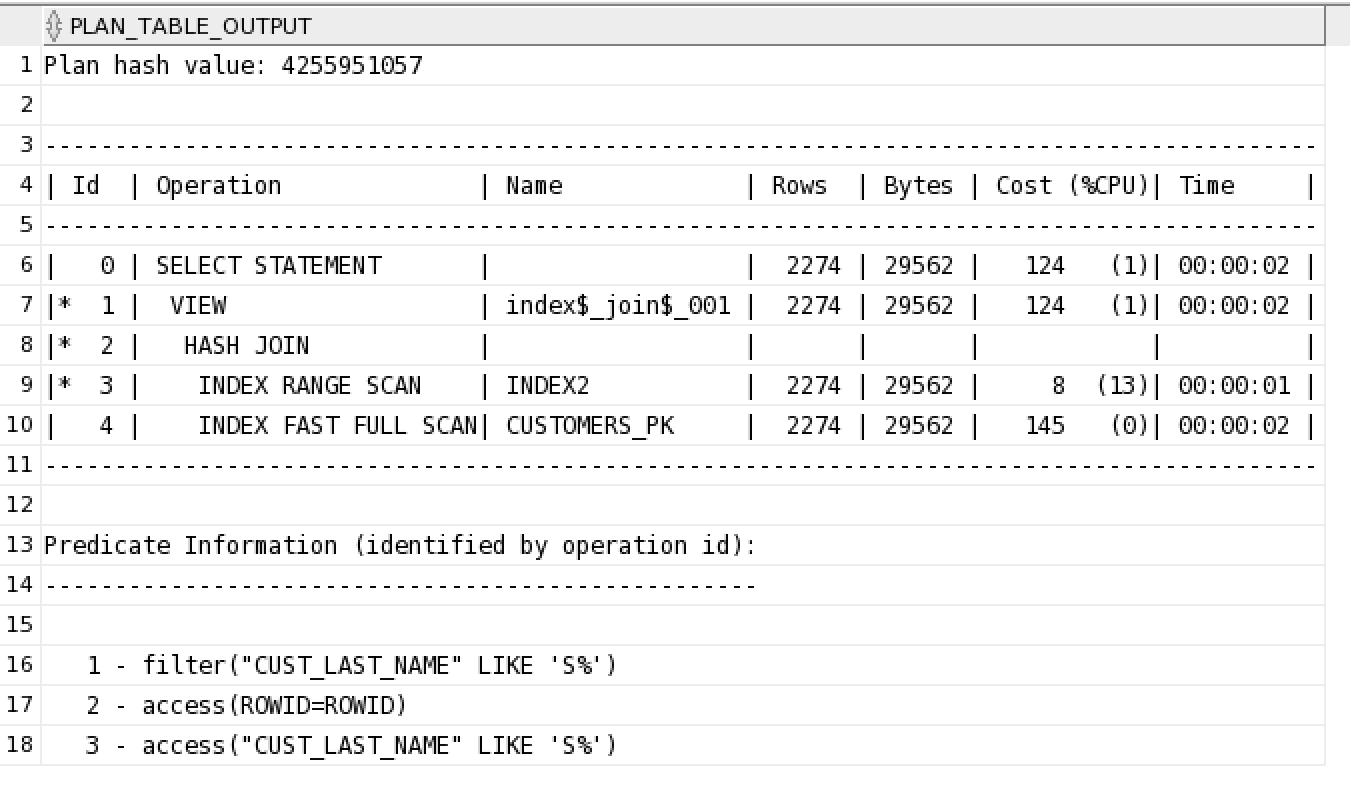
## q3. Create another index on the CUST\_LAST\_NAME column of the CUSTOMERS table. After that, compare the execution plan and timing of following queries

there is some problem with the single quota when copy and paste

select cust\_id from customers where substr(cust\_last\_name,1,1) = ‘S’

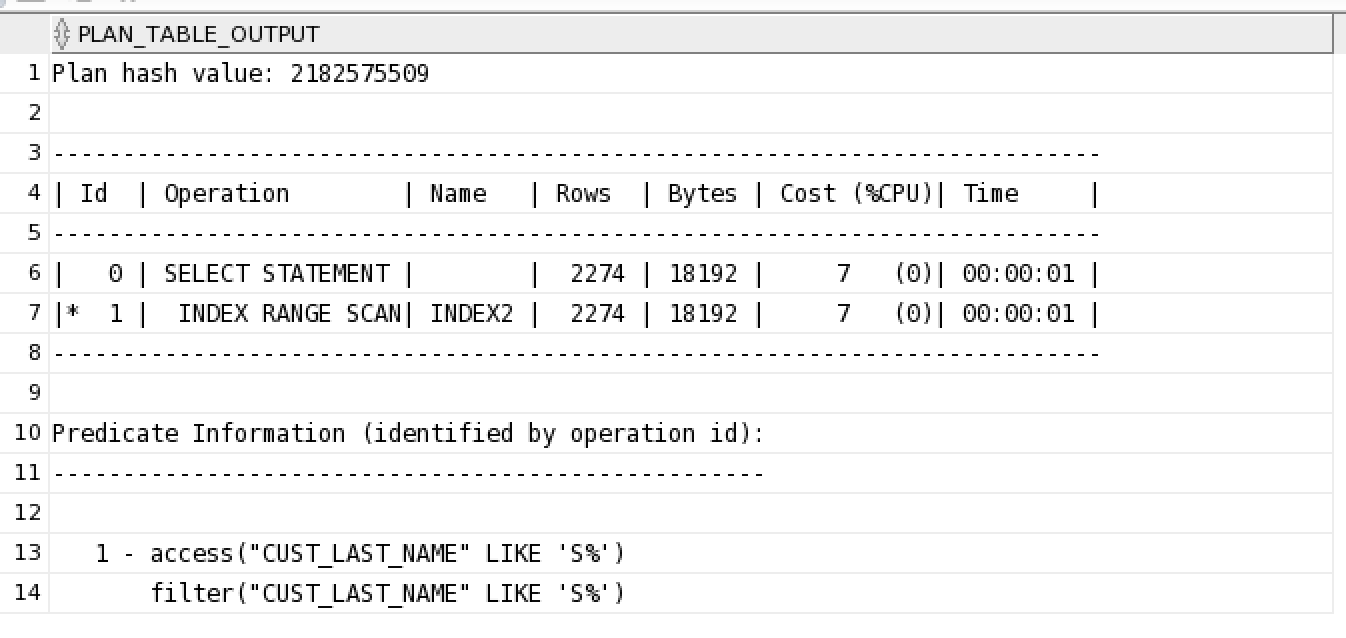


select cust\_id from customers where cust\_last\_name like **‘**S%’



select cust\_last\_name from customers where cust\_last\_name like ‘S%’

use like and select the index term is the fastest way



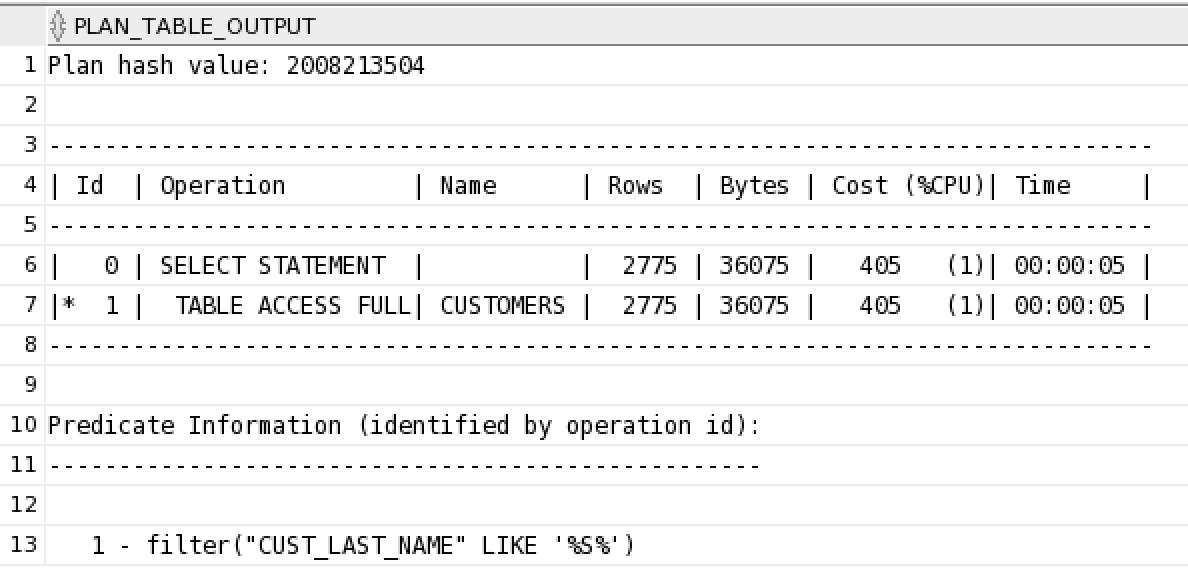
## Q4. Check the execution plan and reason why index is not used.

% in the front is like the substr to match

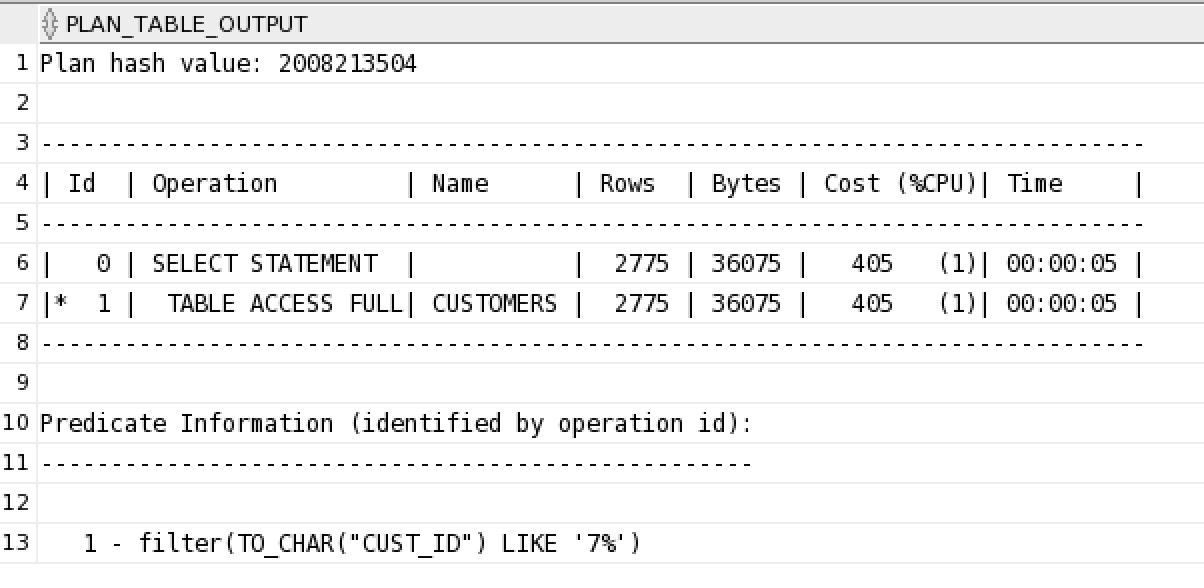
so there is no index used

not use index, use the string instead

select cust\_id from customers where cust\_last\_name like ‘%S%’



select cust\_last\_name from customers where cust\_id like ‘7%’

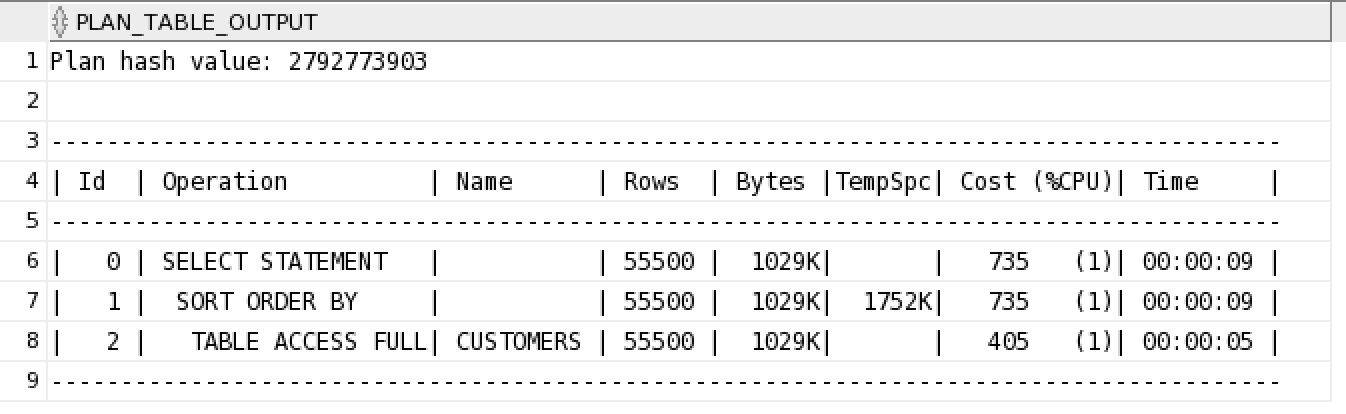


## Q5. In order by clause, in general there is sorting done on all the rows. Try to run following select statement and again check the execution plan:

All with indexes means no indexes

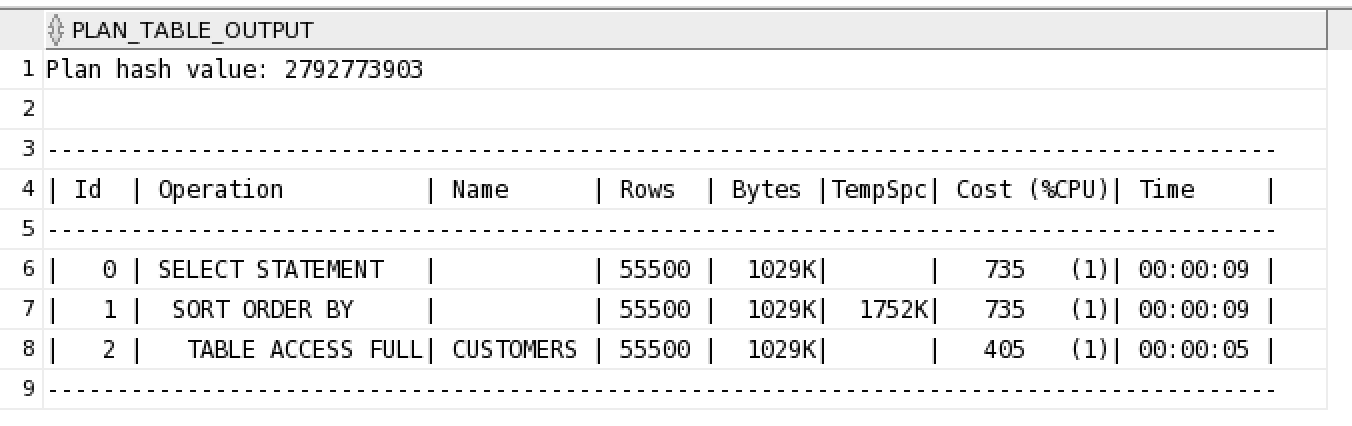
The cust\_credit\_limit has null value so that the index would not work

select cust\_first\_name, cust\_last\_name, cust\_credit\_limit from customers order by cust\_credit\_limit;



Now create an appropriate index on a column and again run above select statement. Do you find any change in performance and execution plan?

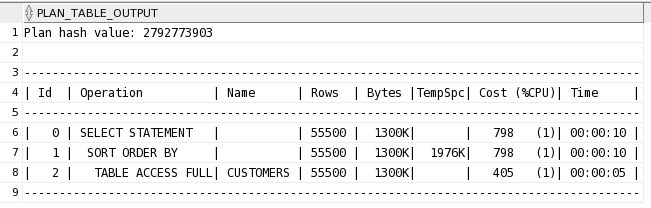
Create index the cust\_credit\_limit



Create an index on cust\_id column of customers table if it does not. Now execute following SQL statement and compare the execution plan of this query with above.

select cust\_first\_name, cust\_last\_name, cust\_credit\_limit from customers order by cust\_id;

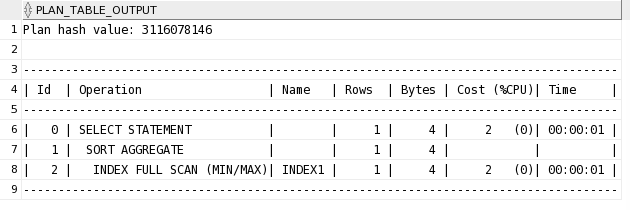
理论上用index,但不知道为什么不能



## Q6. Compare the execution plan for following statements and explain the difference:

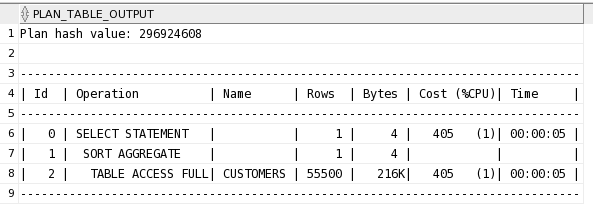
select max(cust\_credit\_limit) from customers;

直接匹配所以可以使用indexes



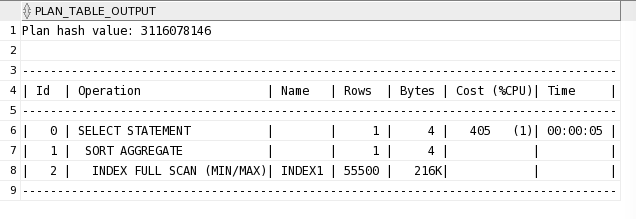
select max(cust\_credit\_limit\*2) from customers;

算术indexes变成了全表扫



select max(cust\_credit\_limit+1000) from customers;

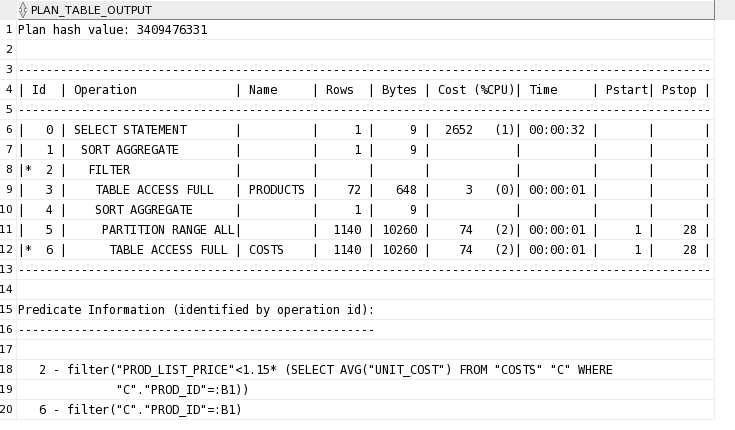
全索引,加法在内外都是全索引搜索



## Q7 Compare the execution plan and time of following two statements:

Select count(\*) from products p where prod\_list\_price<1.15 \* (select avg(unit\_cost) from cost c where c.prod\_id=p.prod\_id);

Select count(\*) from products p where prod\_list\_price<1.15 \* (select avg(unit\_cost) from costs c where c.prod\_id=p.prod\_id);



条件对比时,可以优化为先把数据找出来,再对比

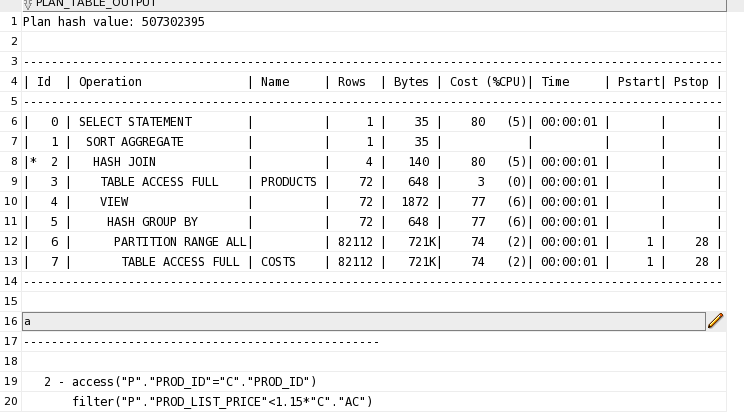
Select count\* from product p, (select prod\_id, Avg(unit\_cost) ac from cost group by prod\_id) c where p.prod\_id=c.prod\_id and p.prod\_list\_price <1.15\*c.ac

Select count(\*) from products p, (select prod\_id, Avg(unit\_cost) ac

from costs group by prod\_id) c

where p.prod\_id=c.prod\_id and p.prod\_list\_price <1.15\*c.ac

;



## Q8. Analyze following SQL statement and identify the best join operation (and join sequence, when relevant), using the hints (USE\_MERGE, USE\_NL, USE\_HASH) to instruct the optimizer.

select c.cust\_last\_name c.cust\_year\_of\_birth, co.country\_name

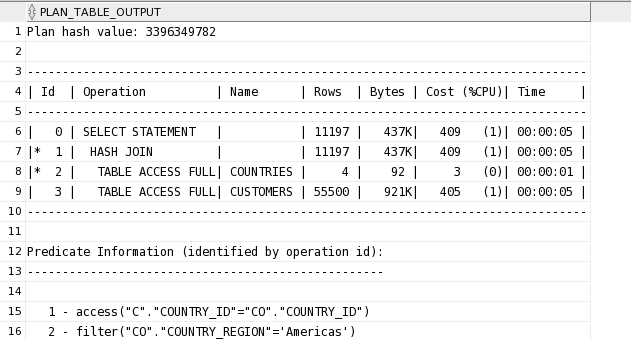
from customers c ,countries co where c.country\_id =co.country\_id and co.country\_region = ‘Americas’;

Hint: <http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements006.htm#SQLRF50702>

select c.cust\_last\_name, c.cust\_year\_of\_birth, co.country\_name

from customers c,countries co

where c.country\_id =co.country\_id and co.country\_region = 'Americas';



alter system flush shared\_pool;

alter system flush buffer\_cache;

explain plan for

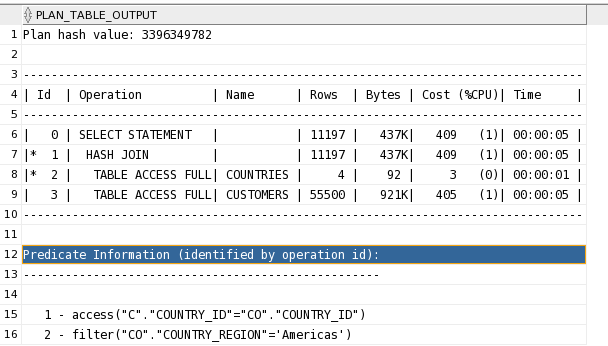
select /\*+ USE\_MERGE(customers countries) \*/ c.cust\_last\_name, c.cust\_year\_of\_birth, co.country\_name

from customers c,countries co

where c.country\_id =co.country\_id and co.country\_region = 'Americas';

select \* from table(dbms\_xplan.display);

使用合并表再查找



## Q9. Analyze the SQL statement

select c.cust\_last\_name s.time\_id, s.prod\_id from customers c, sales s, where c.cust\_id <> s.cust\_id and s.prod\_id = 2595 and s.time\_id = ‘01-JAN-98’

alter system flush shared\_pool;

alter system flush buffer\_cache;

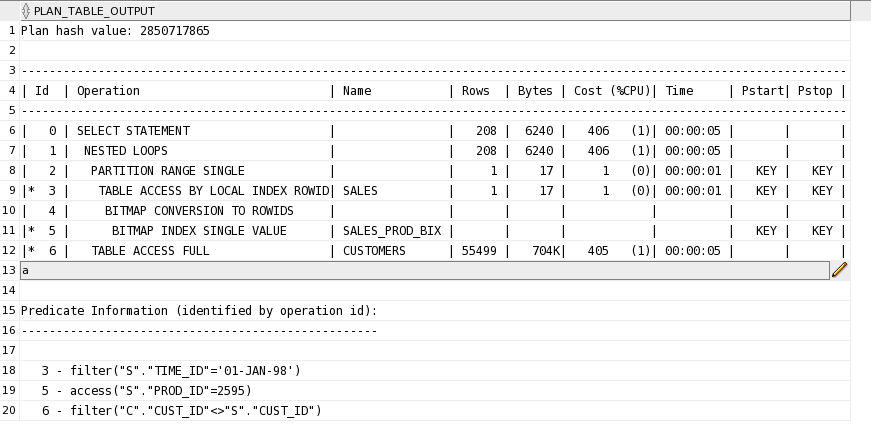
explain plan for

select c.cust\_last\_name, s.time\_id, s.prod\_id

from customers c, sales s

where c.cust\_id <> s.cust\_id and s.prod\_id = 2595 and s.time\_id = '01-JAN-98';

select \* from table(dbms\_xplan.display);



Which join operations can be used to execute this join?

Experiment with different join orders by using an ORDERED hint, then try a LEADING hint,

and find the best choice. There is a significant difference in performance? Do the same with different join operations.

alter system flush shared\_pool;

alter system flush buffer\_cache;

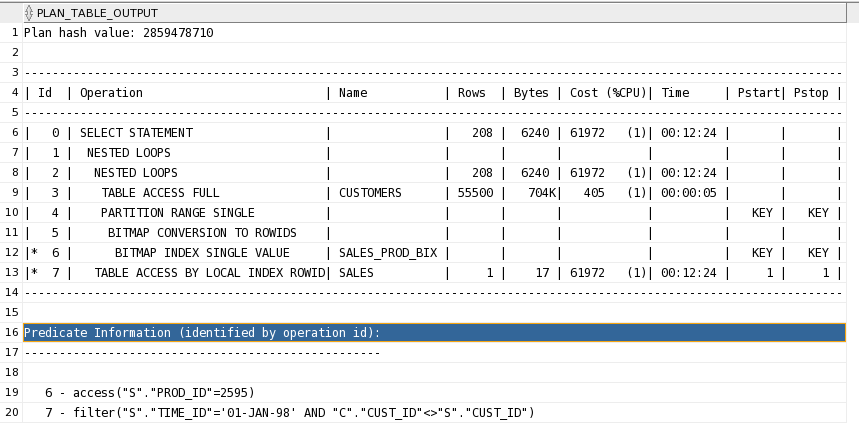
explain plan for

select /\*+ ordered \*/ c.cust\_last\_name, s.time\_id, s.prod\_id

from customers c, sales s

where c.cust\_id <> s.cust\_id and s.prod\_id = 2595 and s.time\_id = '01-JAN-98';

select \* from table(dbms\_xplan.display);



alter system flush shared\_pool;

alter system flush buffer\_cache;

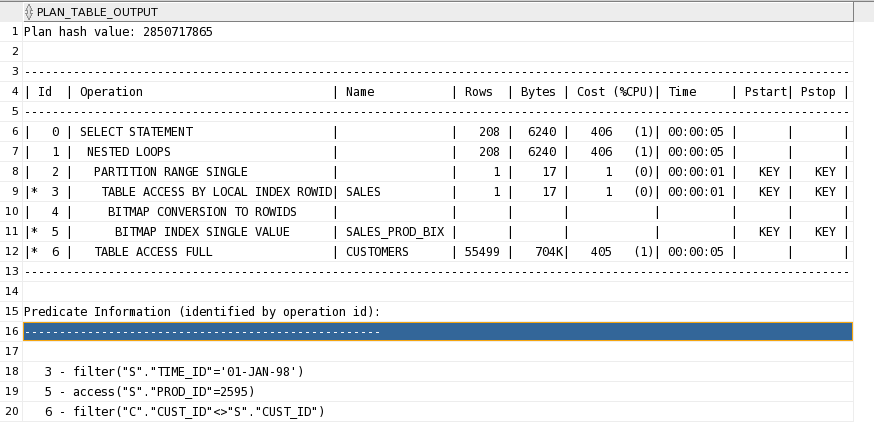
explain plan for

select /\*+ leading \*/ c.cust\_last\_name, s.time\_id, s.prod\_id

from customers c, sales s

where c.cust\_id <> s.cust\_id and s.prod\_id = 2595 and s.time\_id = '01-JAN-98';

select \* from table(dbms\_xplan.display);



alter system flush shared\_pool;

alter system flush buffer\_cache;

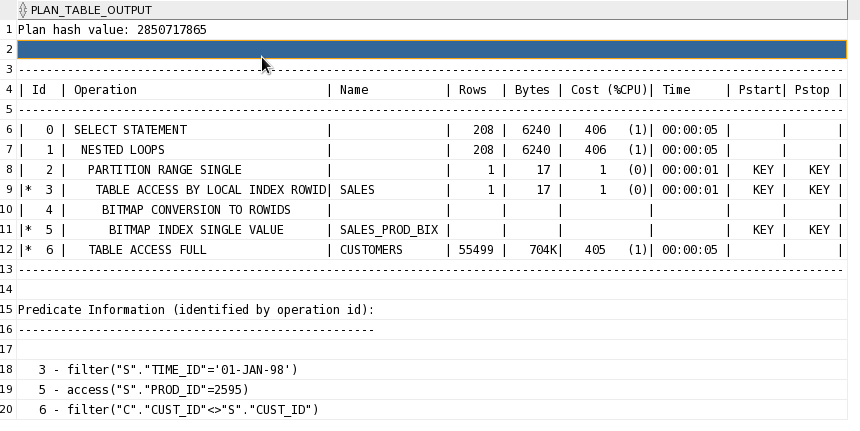
explain plan for

select /\*+ leading(s c) \*/ c.cust\_last\_name, s.time\_id, s.prod\_id

from customers c, sales s

where c.cust\_id <> s.cust\_id and s.prod\_id = 2595 and s.time\_id = '01-JAN-98';

select \* from table(dbms\_xplan.display);



## Q10. Consider following sql statement:

select c1.cust\_first\_name, c1.cust\_last\_name, c1.cust\_year\_of\_birth from customers c1 where c1.cust\_year\_of\_birth = (select max(c2.cust\_year\_of\_birth) from customers c2 where c1.country\_id = c2.country\_id)

This statement retrieves the customers with the oldest birth year in every country.

This type of statement can be found in many real-life situations. What is happening in the

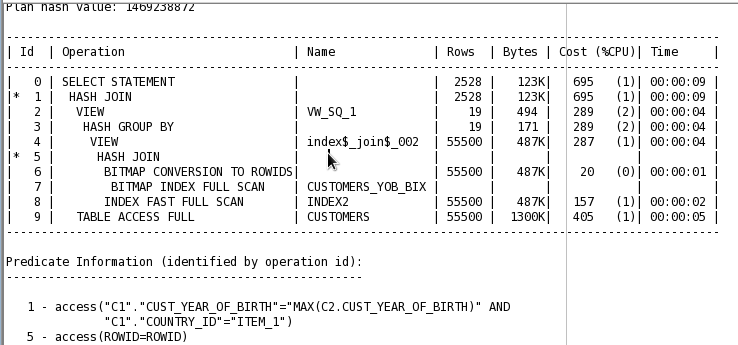
execution plan? Are the results satisfactory?

Create an index on the COUNTRY\_ID column, and measure the performance improvement. Is creating the index a better choice?

Can you rewrite the query to use join operation. Is there any performance improvement.

alter system flush shared\_pool;

alter system flush buffer\_cache;



explain plan for

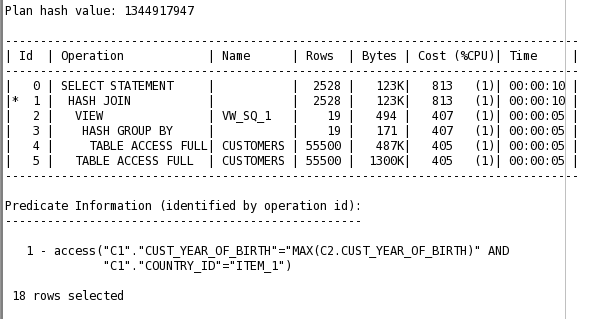
select c1.cust\_first\_name, c1.cust\_last\_name, c1.cust\_year\_of\_birth

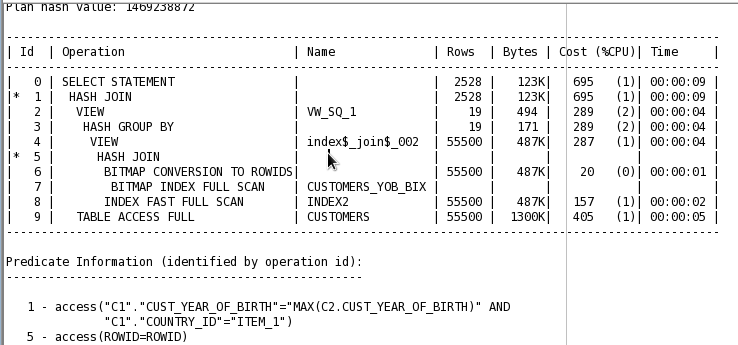
from customers c1

where c1.cust\_year\_of\_birth = (select max(c2.cust\_year\_of\_birth)

from customers c2 where c1.country\_id = c2.country\_id);

select \* from table(dbms\_xplan.display);





## Q11. Create indexes on the following columns:

CUST\_GENDER, CUST\_POSTAL\_CODE,and CUST\_CREDIT\_LIMIT. Check before creating the indexes that are available or not.

The following statement contains WHERE clause with three predicates. Execute this statement, and take notes about the indexes used, the cost of the execution plan, and the amount of I/O performed.

select c.\* from customers c where cust\_gender = ‘M’ and cust\_postal\_code = 40804 and cust\_credit\_limit = 10000;

Run above statement again with different indexes by giving HINT and take notes about the performance results.

Drop above indexes and create a new combined index and run above statement without any hints again. Compare the performance results with previous ones. Which is best?

## Q12 Oracle provides several index types. In this exercise will try to see how they differ:

1. Drop previously created indexes in this tutorial. Create two bitmap indexes on the following columns of the CUSTOMERS table:

cust\_year\_of\_birth

cust\_credit\_limit

Execute following SQL statements and compare their execution plan and performance:

SELECT /\*+ INDEX\_COMBINE(c) \*/ c.\*

FROM customers c

WHERE c.cust\_year\_of\_birth = 1953

OR c.cust\_credit\_limit = 10000;

SELECT c.\*

FROM customers c

WHERE c.cust\_year\_of\_birth = 1953

OR c.cust\_credit\_limit = 10000;

1. Drop all the indexes on the CUSTOMERS table except its primary key index.

After this, create a concatenated B\*-tree index on the following columns of the

CUSTOMERS table, and in the order here:

cust\_last\_name

cust\_first\_name

Compare results of following query before and after creating the index:

SELECT c.cust\_last\_name, c.cust\_first\_name

FROM customers c;

Now run following statement and again compare the results:

SELECT /\*+ INDEX\_JOIN(c cust\_cust\_first\_name\_idx

cust\_cust\_last\_name\_idx) \*/ c.cust\_last\_name

, c.cust\_first\_name

FROM customers c;

1. Drop all the previously created indexes on Customer Table.
   1. Create one B\*-tree index on the following column of the CUSTOMERS table:

cust\_credit\_limit

Execute the following query:

SELECT count(\*) credit\_limit

FROM customers

WHERE cust\_credit\_limit = 10000;

Note down the plan and performance statistics.

* 1. Drop all the indexes on the CUSTOMERS table except its primary key

index. Then, create one bitmap index on the following column of the CUSTOMERS table:

cust\_credit\_limit

Execute the following query:

SELECT count(\*) credit\_limit

FROM customers

WHERE cust\_credit\_limit = 10000;

Note down the plan and performance statistics.

Compare the results of (a) and (b). Which index is better and why?