**Advanced Databases - LAB Week 10**

**Indexes and Query Optimization**

Dr. Pierpaolo Dondio

In this lab we will experiment with query optimization.

Create the following tables:

1. Persons
2. Jobs
3. Jobs-persons

A list of persons is connected many-to-many to a list of jobs (the table jobs-person is the relation table).

drop table persons;

create table persons(

person\_id integer,

person\_name varchar(20),

person\_surname varchar(20),

person\_age integer not null,

person\_wealth integer,

person\_weight float

);

drop table jobs\_person;

create table jobs\_person(

jobs\_id integer,

person\_id integer,

start\_date date,

end\_date date);

drop table jobs;

create table jobs(

jobs\_id integer,

job\_description varchar(200),

salary integer

);

Execute the following sql 3 commands block **ONE BY ONE** to fill the tables with random data. The commands are also in the *populate.sql* (but execute them one by one!!)

/\* 1. populate table persons \*/

declare v\_p\_id number;

v\_p\_name varchar2(20);

v\_p\_surname varchar2(20);

v\_p\_age integer;

p\_wealth float;

p\_weight float;

BEGIN

FOR i IN 1..10000 LOOP

select DBMS\_RANDOM.STRING('a', 20) into v\_p\_name from dual;

select DBMS\_RANDOM.STRING('a', 20) into v\_p\_surname from dual;

SELECT TRUNC(DBMS\_RANDOM.VALUE(18, 100)) into v\_p\_age FROM DUAL;

SELECT TRUNC(DBMS\_RANDOM.VALUE(0,10000000)) into p\_wealth FROM DUAL;

SELECT trunc(DBMS\_RANDOM.VALUE(40, 120),2) into p\_weight FROM DUAL;

insert into persons values(i,v\_p\_name,v\_p\_surname,v\_p\_age,p\_wealth,p\_weight);

END LOOP;

end;

/\* 2. populate table jobs \*/

declare j\_id number;

j\_description varchar2(100);

j\_salary float;

BEGIN

FOR i IN 1..10000 LOOP

select DBMS\_RANDOM.STRING('a', 100) into j\_description from dual;

SELECT TRUNC(DBMS\_RANDOM.VALUE(0,100000)) into j\_salary FROM DUAL;

insert into jobs values(i,j\_description,j\_salary);

END LOOP;

end;

/\* 3. populate table jobs-persons \*/

declare j\_id number;

p\_id integer;

start\_date date;

end\_date date;

st integer;

en integer;

BEGIN

FOR i IN 1..10000 LOOP

FOR j in 1..15 LOOP

SELECT TRUNC(DBMS\_RANDOM.VALUE(0,1000000)) into p\_id FROM DUAL;

SELECT TRUNC(DBMS\_RANDOM.VALUE(0, 1000)) into st FROM DUAL;

SELECT TRUNC(DBMS\_RANDOM.VALUE(0, 2000)) into en FROM DUAL;

SELECT TO\_DATE(TRUNC(DBMS\_RANDOM.VALUE(2452641,2452641+st)),'J') into start\_date FROM DUAL;

SELECT TO\_DATE(TRUNC(DBMS\_RANDOM.VALUE(2452641+st,2452641+st+en)),'J') into end\_date FROM DUAL;

insert into jobs\_person values(i,p\_id,start\_date,end\_date);

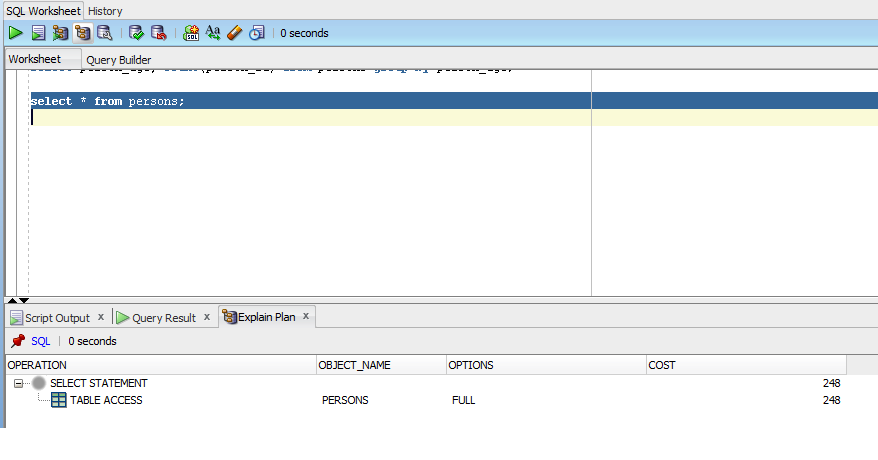
END LOOP;

END LOOP;

end;

There are no keys or indexes defined. The person\_id and the jobs\_id are unique.

Using the oracle function Explain Plain (in SQL developer is the fourth icon from the left in the sql script window or press F10), we3 will analyse how ORACLE executes queries and the cost of each query - a number expressing how much resources and time your query takes.



Execute the following steps to analyse Oracle Index behaviour

1. Check that data are in the three tables. Have a look at the data

There are no indexes or keys defined at this stage.

1. Execute the following query

**Query1.**

*select \* from persons*

And select the explain plain function.

How much is the cost? \_\_\_\_\_\_27\_\_\_- Cardinality 11138\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Was it a full or index scan of the table? \_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why? \_\_\_\_There is no index so it must be a full scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Execute

**Query2 .**

*select \* from persons where person\_id>1000 and person\_id<3000*

Total cost? \_\_\_27\_\_- Cardinality 2171\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or index scan? Why? \_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Any difference with the previous query? \_\_There is a range used\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Define a primary key over *person\_id* (using an ALTER TABLE … ADD CONSTRAINTS statement) Remember that this creates an index on *person\_id* as well. Perform **Query1**

Cost? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_27\_\_\_- Cardinality 11138\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comment the results.

\_\_\_\_\_\_\_\_Index hasn’t made a difference to the scan since its still a full scan \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Perform **Query2**

Cost? \_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_- Cardinality 2171\_\_\_\_\_\_\_\_\_\_index (2 – Cardinality 50)\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_\_\_This is an index scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comment the results.

\_\_\_\_\_\_\_\_\_\_\_The index has now improved the efficiency/cost of this query considerably \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Query1 requires a full scan since it gives back the full unfiltered table

Query2 requires uses the index scan on the primary index *person\_id* to filter the data. Note that range scan refers to accessing an interval of value (range) using an index (therefore oracle finds the starting point using the index and then the scan is sequential over an ordered list. It is therefore faster than full scan)

1. Perform the following

**Query3**

*select \* from persons where person\_id+5>1000 and person\_id<3000*

Check cost and type of scan

Cost: \_\_3 – Cardinality 2179 \_\_\_\_\_\_\_\_\_\_\_\_\_2 – 5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Type of Scan \_\_\_\_\_\_Index Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Query4**

*select \* from persons where person\_id+5>1000 and person\_id\*2<3000*

Check cost and type of scan

Cost: \_\_\_\_\_\_\_\_27 - Cardinality 406\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Type of Scan \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comment the behaviour of Query3 and Query4

\_\_\_\_The query 3 uses the index of the second person\_id. The query 4 cannot use either index since they are both being added/multiplied, so it uses a full scan instead.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In **Query3** the index person\_id is contained in the expression (person\_id+5) so it cannot be used. However, the index can be used in the other where clause (person\_id<20000), so Oracle performs an index access of the table to get the persons with person\_id<20000 and at the same time it filters the condition (person\_id+5>20000)

In **Query4**, none of the index can be used so a full scan is performed

Remember: if an index is used in an expression that affects the ordering of the data , it won’t be used!

1. Execute

**Query5.**

*select person\_age, count(person\_id) from persons group by person\_age;*

Cost? \_\_\_\_\_28 – Cardinality 1118 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comment the results.

\_\_\_\_\_The index for person id is not used so because of the group by part of the query \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The index on *person\_id* does not help since we are grouping by person\_age, so a full scan is required. Note the extra cost of grouping, executed by Oracle in a quick way by hashing the persons by age during the full scan

1. Define an index on person\_age by executing:

*create index p\_age on persons(person\_age);*

Execute again **Query5**

Cost? \_\_\_\_\_\_\_\_8 – Cardinality 11138 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_Full scan\_\_(FAST)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Even if an index is defined on person\_age, the index is not used, why?

The reason is the following (IMPORTANT!): if a column contains NULL values or it has been defined (with CREATE TABLE) without NOT NULL the index will be ineffective!

1. Drop the table persons.
2. Modify the create table statement adding “*not null*” to the field *person\_age (*and add primary key to the *person\_id* field so you do not need to alter the table afterwards).
3. Populate the table with the sql command used before (page 2, block 1)
4. Execute query5

Cost? \_\_\_27 without p\_age, 8 with p\_age index\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_Full Scan without p\_age, Full Fast Scan with it\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You should see an index-like access (hash, i.e. the type of index create on person\_age – default) and the cost is now reduced

1. Joining two tables

Perform the following query:

*select jobs.jobs\_id,jobs.job\_description, jobs.salary, jobs\_person.person\_id*

*from jobs inner join jobs\_person*

*on jobs.jobs\_id = jobs\_person.jobs\_id*

*where jobs\_person.jobs\_id=34;*

Cost? \_\_\_\_\_260, Cardinality 69\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comment the results.

\_\_\_\_\_\_\_\_The join query uses two full table query scans, and a hash join to speed it up \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note how the query is divided into steps: first the full scans and the hash table used to speed-up the join.

Add indexes on jobs\_id.jobs\_id and jobs\_person.jobs\_id (note that one could be aprimary key and the other a foreign key).

Check again the results.

Did they improve or not? Why?

(you should see a reduction in cost due to the usage of indexes).

1. Reduce the cost of this query as much as you can:

/\* select person name, max salary and job description between 2003 and 2004 \*/

*select p.person\_name, j.salary, j.job\_description*

*from persons p inner join jobs\_person jp on p.person\_id = jp.person\_id*

*inner join jobs j on jp.jobs\_id=j.jobs\_id*

*where jp.start\_date> '01-JAN-2003' and jp.end\_date < '31-DEC-03';*

Cost? \_\_\_\_\_246, Cardinality 27724\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

drop index p\_age;

drop index jp\_end\_date;

drop index jp\_start\_date;

create index p\_age on persons(person\_age);

create index jp\_start\_date on jobs\_person(start\_date);

create index jp\_end\_date on jobs\_person(end\_date);

Cost? \_\_\_\_\_244, Cardinality 27724\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

drop table temp1;

create table temp1 as select p.person\_id, p.person\_name, j.salary, j.job\_description

from persons p inner join jobs\_person jp on p.person\_id = jp.person\_id

inner join jobs j on jp.jobs\_id=j.jobs\_id;

select person\_name, salary, job\_description

from temp1 inner join jobs\_person jp on temp1.person\_id = jp.person\_id

where jp.start\_date> '01-JAN-2003' and jp.end\_date < '31-DEC-03';

Cost? \_\_\_\_\_180, Cardinality 1569\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Alternative – offload whole query to temp1

*create table temp1 as select p.person\_name, j.salary, j.job\_description*

*from persons p inner join jobs\_person jp on p.person\_id = jp.person\_id*

*inner join jobs j on jp.jobs\_id=j.jobs\_id*

*where jp.start\_date> '01-JAN-2003' and jp.end\_date < '31-DEC-03';*

*select \* from temp1;*

Cost? \_\_\_\_\_4, Cardinality 360\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Full or Index? \_\_\_\_\_Full Scan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Use indexes, temporary tables, change the SQL code, split the join – but be sure the result is still equivalent!