CS460 – Database Management Systems
Winter 2016
Project
Submission Deadline: 16/12/2016

Create the following tables (tables.sql) and insert the data in the accompanying files (distributedBy.sql – 143.153 tuples, movie.sql – 193.781 tuples, people.sql - 289.221 tuples, plays.sql – 810.692 tuples, producedBy.sql – 202.868 tuples). Initially there are not defined indexes, keys or clusters for the aforementioned tables. In addition create the table PLAN_TABLE, where you will save the execution plans of each assignment's query. Each time you create a table and you load tuples to that tables independent of whether you create new indexes or not you should always keep table's statistics updated.

Note: The cache memory that is used by the Oracle Database is 41.943.040 bytes long, split into 5.120 buffers of size 8.192 bytes each.

Task 1 – Cache memory usage for I/O minimization

A) Since the large amount of I/Os slow down the response time and increase CPU usage, the performance of the Oracle Server can be significantly improved when most of the requested pages are already in cache.

The measure of this performance is the *hit ratio* of the cache and corresponds to the number of the pages available in cache divided by the total number of pages that have been accessed. When the cache of a DBMS is really small, then system's performance degrades due to many I/Os. A systems needs to be tuned when the *hit ratio* is lower than 80%.

Some of the suggested improvements for tuning are the following:

- Add more cache pages
- Use different cache buffer pools, for individualizing pages according to their access method.
- Transfer one or more tables in cache (if there is enough space available).

A) For the following queries calculate the disk physical reads.

Query 1:

```
SELECT      year, count(*)
FROM        movie
GROUP  BY   year
ORDER  BY   year  desc;
```

Query 2:

```
SELECT      companyID, count(*)
FROM        distributedBy
GROUP  BY   companyID;
```

B) Calculate the cache *hit ratio* for executing the aforementioned queries. Do we need tuning? Justify your answer.

Γ) Calculate the size of the tables in the database. Which of them could be loaded to the cache for improving DBMS performance?

Δ) Load movie in the cache. Now execute the aforementioned queries in the given order calculating after each one the number of pages loaded by the disk. What observations do you make? Now execute again the first query and calculate the number of pages that are read by the disk. What observations do you now make? Drop movie and create the table again, loading from scratch his data.

Tast 2 – Alternative SQL Queries

Initially, the optimizer_mode has the value *choose.* As such the Oracle optimizer chooses, depending on the existence or not of statistics, to optimize queries based on *cost* or based on *rules*.

A) We are looking for the movies *IDs* for which we do not have any information about their distribution. Let the two following equivalent queries:

Query 1:

```
SELECT movieID
FROM movie MINUS
SELECT movieID
FROM distributedBy;
```

Query 2:

```
SELECT movieID FROM movie
WHERE movieID NOT IN

    (SELECT movieID FROM
     distributedBy);
```

i) Use the EXPLAIN PLAN command to identify Oracle's execution plans for the two aforementioned queries that the optimizer selects. Draw graphically those two plans (as trees) and describe them.

ii) Before executing the queries, based on the constructed query execution plans, order the queries according to the expected execution time. Justify your answer. Then find the execution time for these two queries and compare the results with your expectations.

iii) Suggest ways to improve the execution times of these two queries, e.g. using indexes and clusters. Identify whether they succeed these techniques.

B) We are looking for the *IDs* of the movies that include actors born after the year x, where x = 1930, 1950, 1970, 1990, 1995. We have the following two equivalent queries:

1η επερώτηση:

```
SELECT pl.movieID
FROM people p, plays pl
WHERE p.personID = pl.personID and p.birthYear > x;
```

2η επερώτηση:

```
SELECT pl.movieID
FROM plays pl
WHERE pl.personID IN
      (SELECT p.personID
       FROM people p
       WHERE p.birthYear > x);
```

i) Identify the execution plan and show that they do not depend on the specific *x* value. Compare their execution time and identify the fastest query. Why it is the fastest one?

ii) How we can rephrase those queries in order to better use the available memory and improve their execution time?

Tast 3 – Cost based optimization

Use the command ALTER SESSION to set the parameter OPTIMIZER_MODE to ALL_ROWS. Using this parameter value the optimizer is set to use a cost based optimization method, trying to identify the best resource allocation for query execution.

A) Consider the following query over the tables *movie* and *plays*, including the *movieID* in the WHERE clause. Identify the execution plan and the execution time. What if we include the DISTINCT statement in the query for removing duplicate?

```
SELECT    p.personID
FROM      movie m, plays p
WHERE     p.movieID = m.movieID and m.movieID = 0046790;

SELECT    DISTINCT p.personID
FROM      movie m, plays p
WHERE     p.movieID = m.movieID and m.movieID = 0046790;
```

B) Create a hash cluster for the *movieID* using 1000 search keys and relate it with the *movie* table. To do that you should drop and re-create the movie table to take into advantage of the hash cluster. Identify again the execution plan and the execution time.

C) Drop the table *plays* and create it again relating the field *movieID* with the cluster generated in B) Identify again the execution plan and the execution time.

D) Find the execution plan of the following queries. What observation do you make? Next, create an unclustered B^+-Tree for the *year* field of the table *movie* and one for the *birthYear* of the table *people*. Which is now the execution plan of the following SQL queries?

```
SELECT    movieTitle
FROM      movie
WHERE     year > 1990;

SELECT    personName
FROM      people
WHERE     birthYear > 1945;
```

E) Find the execution plan for the following queries for the table movie including the fields *movieID* and *year* in the WHERE clauses. Which one of the hash cluster on the *movieID* and the B^+-Tree on the field *year* are being used? What are the reasons for the possible different execution plans?

```
SELECT    movieID, movieTitle
FROM      movie
WHERE     movieID != 0046778 and year > 1985;

SELECT    movieID, movieTitle
FROM      movie
WHERE     movieID = 0046778 and year > 1985;
```

Task 4 – Rule based optimization

Up to this point you should have available a hash cluster on the *movieID* used by the *movie* and the *plays* tables. In addition there should be available two unclustered B^+-Tree indexes for the *year* and the *birthYear* fields of the *movie* and *people* tables respectively.

Use the command ALTER SESSION to set the OPTIMIZER_MODE to RULE. Using this command the optimizer uses a rule based optimization method ignoring statistics.

A) Now execute again the (D) of the Task 3. Is there any difference in the execution plan? If yes compare the execution times.

B) Define for the *movieTitle* field a unique unclustered B^+-Tree. Find the query execution plan for the following query. Is the hash cluster used or the unclustered B^+-Tree. Why?

```
SELECT m.movieID, m.movieTitle, p.companyID
FROM movie m, producedBy p
WHERE m.movieID = p.movieID and m.movieID = 0046799 and m.movieTitle = 'Boot
Polish (1954)';
```

Now drop the index on the *movieTitle*.

Task 5 – Compare optimization methods

Up to this point you should have available a hash cluster on the field *movieID* used by the *movie* and the *plays* tables. In addition there are two unclustered B^+-Tree indexes for the *year* and the *birthYear* fields of the *movie* and the *people* tables respectively. Finally the optimizer_mode is set to the RULE value.

A) For each one of the OPTIMIZATION_MODE values (RULE, FIRST_ROWS, ALL_ROWS) find the execution plans and the execution times of the following query.:

```
SELECT /*+ ORDERED USE_HASH(p,pl) */ pl.movieID
FROM people p, plays pl
WHERE p.personID = pl.personID and p.birthYear > 1990;
```

B) Drop the index for the *birthYear* field and repeat (A). For the same mode do you observe any differentiation between the two queries?

C) Create a bitmap index on the *birthYear* field. Repeat (A). What differences do you observe using the bitmap index?

D) Compare the execution times for (A), (B) and (C) for each optimization mode. What differences do you observe?

E) Drop the *people* table and recreate it, defining a clustered B^+-Tree on the *birthYear* field. For each one of the three OPTIMIZATION_MODE (RULE, FIRST_ROWS, ALL_ROWS) find the execution plan and the execution time for the query in (A). Do you find any difference now when compared to the unclustered B^+-Tree index?

Task 6 – Oracle SQL Access Advisor usage

At this point, you will use SQL Access Advisor of Oracle. This tool is used for the automatic analysis of a set of data and queries and for the suggestion of methods that will optimize the access to the data.

A) Drop your database and recreate it. Then, execute the following query:

```
SELECT pl.movieID
FROM people p, plays pl
WHERE p.personID = pl.personID and p.birthYear > 1990;
```

Run SQL Access Advisor (you can find directions here https://oracle-base.com/articles/11g/sql-access-advisor-11gr1#dbms_advisor) and analyze its suggestions for the optimization of the particular query, and the respective charts it generates.

B) Implement the suggestions regarding the materialized views and measure the execution time of the above query. Analyze the results. What differences do you observe, in relation to the approaches of Task 5?

C) What was the cost you "paid" for the specific implementation? Which would finally be your approach to the problem of optimizing the query above?