CS 460 – Database Management Systems

Winter semester 2020

2nd Assignment
Due date: 29/11/2020

General instructions

Write a report demonstrating your results using images/screenshots, SQL code, SQL errors/warnings and any other explanation necessary. Include a cover page with your student ID and name. The filename of the deliverable must be in the following form: **ask2_AM** (where AM is your student id number e.g. ask2_1234). Any images/screenshots should be included in your report document. Do NOT submit separate image files. The report document must be in pdf format. Submit a single compressed file (zip/rar/gz etc) if your work contains multiple files. Note that this assignment is to be done **individually**. Cheating in any way, including giving your work to someone else will result in failing this assignment (a mark of zero will be given).
Submit your assignment **electronically** (online) until 29/11/2020 at 23:59 using the course webpage at the eLearn system.

Overview

In this assignment, you will experiment with SQL indexing structures using various configurations. You will be working with the latest Oracle Database Express Edition.

Background

We strongly suggest you read/review the following resources:

- Course slides and tutorials on indexing and index tuning

Tasks

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.

Task 1. Explore Cache Memory

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.

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

D) 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.

Task 2. Explore Indexing Techniques

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;
```

F) 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.

Now 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*.

G) 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.

Now find the execution plans and the execution time 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;
```

i. Drop the index for the *birthYear* field and repeat the query. Do you observe any differentiation between the two queries?

ii. Create a bitmap index on the *birthYear* field. Repeat the query. What differences do you observe using the bitmap index?

H) Drop the *people* table and recreate it, defining a clustered B^+-Tree on the *birthYear* field. Find the execution plan and the execution time for the query in G). Do you find any difference now when compared to the unclustered B^+-Tree index?