Carnegie Mellon University in Qatar

Database Applications

15-415 - Spring 2020

Problem Set 4

Out: April 07, 2020

Due: April 15, 2020

1 B^+ External Sorting [20 Points]

Suppose that you just finished inserting several records into a heap file and now want to sort those records. Assume that the DBMS uses external sort and makes efficient use of the available buffer space when it sorts a file. Below is some potentially useful information about the newly loaded file and the DBMS software available to operate on it:

- The number of records in the file is 4500.
- The sort key for the file is 4 bytes long.
- You can assume that rids are 8 bytes long and page ids are 4 bytes long.
- Each record is a total of 48 bytes long.
- The page size is 512 bytes.
- Each page has 12 bytes of control information on it.
- Four buffer pages are available.

4pts (a) How many sorted sub-files will there be after the initial pass of the sort? How long will each sub-file be?

2pts (b) How many passes (including the initial pass just considered) are required to sort this file?

2pts (c) What is the total I/O cost for sorting this file?

4pts (d) What is the largest file, in terms of the number of records, can you sort with just four buffer pages in two passes? How would your answer change if you had 257 buffer pages?

8pts (e) Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

i. The index uses Alternative (1) for data entries.

ii. The index uses Alternative (2) and is unclustered (assume the worst-case).

2 Evaluating Relational Operators [20 points]

Consider a join operation between two relations R and S : $R \bowtie_{R.a=S.b} S$. Read the information below about the two relations to be joined, then answer the following questions.

Assume the cost metric as the number of I/Os and ignore the cost of writing out the result.

- Relation R contains 10,000 tuples and has 10 tuples per page.
- Relation S contains 2000 tuples and also has 10 tuples per page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- 52 buffer pages are available.

8pts (a) If unclustered B^+ indexes existed on $R.a$ and $S.b$, which would incur a lower cost; an index nested loop join or a block nested loop join? Explain (assume the worst-case).

3pts (b) Would your answer change if only five buffer pages were available?

6pts (c) Would your answer change if S contained only 10 tuples instead of 2000?

3pts (d) If $R.a$ is a foreign key that refers to $S.b$, which of R or S would you choose to be the inner relation? Explain.

Assignment continues on the next pages

3 Query Optimization in PostgreSQL [60 Points]

The purpose of this question is to expose the query plans elected by PostgreSQL and observe how indices affect its choice of query plans. For your convenience, **Part 3.1** presents a brief summary of the steps involved in query optimization (from the course lectures). In **Part 3.2**, you must run the given queries on PostgreSQL, analyze the execution plans elected, and answer the given questions. In these questions, we refer to the *estimated total cost* and the *actual total cost*; the former is in arbitrary units, while the latter is in milliseconds. Both are straightforwardly obtained from the output.

3.1 Summary of Optimization

As shown in Figure 1 below, given a query, the Query Optimizer is the DBMS component responsible for generating a subset of all possible plans, evaluating each such plan, and finally electing a (sub) optimal plan.

The sub-component responsible for generating plans is the **Plan Generator**. For a given query, the plan generator does the following:

- By convention, it produces all the left-deep relational algebra trees.
- For each tree produced in (1), it produces the *extended* relational algebra trees (by enumerating the different algorithms for each operator as well as access paths for each relation).

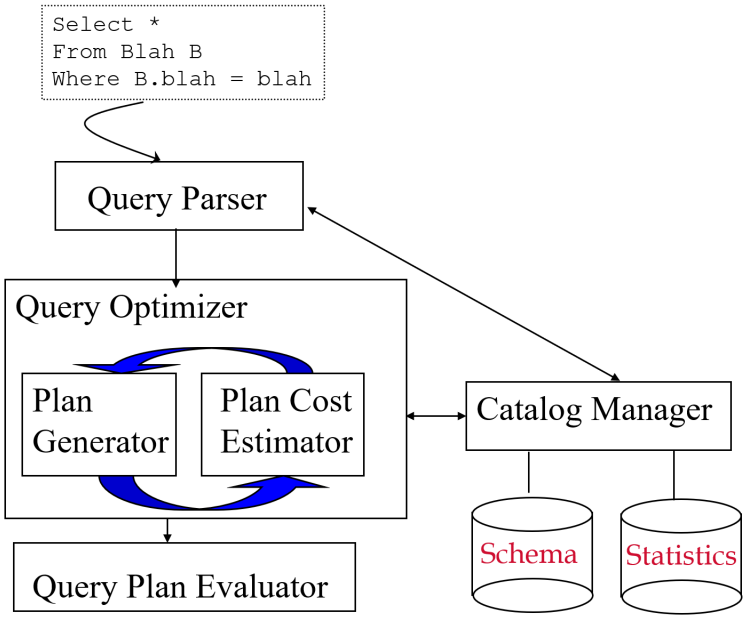

Figure 1: Components of a DBMS.

The sub-component that cooperates very closely with the Plan Generator is the **Plan Cost Estimator**. For every extended relational algebra tree (a.k.a. a query plan) produced by the Generator, the **Plan Cost Estimator** estimates its cost by leveraging statistics from the **System Catalog** about the size of relations and subsequently computing the cost of the algorithms involved.

The **Plan Generator**, in conclusion, elects the query plan with the least cost and deems it the **execution plan** for the given query.

3.2 Analyzing Query Optimization in PostgreSQL

Consider the two relations, movies and actors, from the MovieLens database of Project 1, whose schemas are shown below:

```
movies (mid: integer, title: varchar, year: date,
                rating: real, num_ratings: integer)

actors (mid: integer, name: varchar, cast_position: integer)
```

For every relation in a database, PostgreSQL, by default, creates an index on the primary key. The first task is to list all indices using the command `\di` followed by dropping the indices on movies and actors (**hint**: for the default indices use the SQL command: `alter table .. drop constraint ..`)

25pts (a) Consider the following query that finds the name and cast position of the actors that acted in the movie with `mid = 1000`, and answer the following questions

```
SELECT  name, cast_position
FROM    actors
WHERE   mid = 100
```

We will use the `EXPLAIN`[1] [2] and `ANALYZE` SQL commands to expose the execution plan for the given query. Using `EXPLAIN` and `ANALYZE`, execute the given query.

1. What is the execution plan? Also, include the output of PostgreSQL.
2. What is the estimated total cost of the plan?
3. What is the actual total cost of the plan?

Next, build an index on the field `mid` (**hint**: use the SQL command: `create index ..`) and re-run the given query.

4. What is the execution plan? Also, include the output of PostgreSQL.
5. What is the estimated total cost of the plan with the built index?
6. What is the actual total cost of the plan with the built index?
7. Did the estimated and actual costs change? (Yes/No)
8. How did the costs change? (increased/decreased/unchanged)
9. Explain why the costs remained the same or changed with the built index.
10. Can you deduce the type of index built? (Yes/No)
11. Which type of tree was built? Explain. (tree-based/hash-based/either)

[1] Read about the syntax of EXPLAIN at https://www.postgresql.org/docs/current/static/sql-explain.html
[2] Read about the output of EXPLAIN at https://www.postgresql.org/docs/current/static/performance-tips.html

Finally, drop the index built above (**hint**: use the SQL command: `drop index ..`). Build a new index on the fields `mid` and `name` and re-run the given query.

12. What is the execution plan? Also, include the output of PostgreSQL.
13. What is the estimated total cost of the plan?
14. What is the actual total cost of the plan with?
15. Did the estimated and actual costs change? (Yes/No)
16. How did the costs change? (increased/decreased/unchanged)
17. Explain why the costs remained the same or changed.
18. Can you deduce the type of index built? (Yes/No)
19. Which type of tree was built? Explain. (tree-based/hash-based/either)

5pts (b) Consider the following query. Using `EXPLAIN` and `ANALYZE` and given the index on `mid` and `name`, execute the given query.

```
SELECT  name, cast_position
FROM    actors
WHERE   mid > 1000 and mid < 2000
```

1. What is the execution plan? Also, include the output of PostgreSQL.
2. What is the estimated total cost of the plan?
3. What is the actual total cost of the plan?
4. Can you deduce the type of index built? (Yes/No)
5. Which type of tree was built? Explain. (tree-based/hash-based/either)

10pts (c) Consider the following two queries. Using `EXPLAIN` and `ANALYZE` and given the index on `mid` and `name`, execute the given queries.

```
Q1:
        SELECT  name, cast_position
        FROM    actors
        WHERE   mid  > 1000 and cast_position = 1

Q2:
        SELECT  name, cast_position
        FROM    actors
        WHERE   mid  > 6000 and cast_position = 1
```

1. What is the execution plan of each query? Also, include the output of PostgreSQL.
2. What is the estimated total cost of each plan?
3. What is the actual total cost of each plan?
4. Are the execution plans of Q1 and Q2 the same? (Yes/No)
5. Explain why the execution plans are the same or different.

20pts (d) Consider the following query involving a join between the two relations, movies and actors. After dropping the index on mid and name, execute the given query.

```
SELECT  m.mid, a.name
FROM    movies m, actors a
WHERE   a.mid = m.mid
```

1. What is the execution plan? Also, include the output of PostgreSQL.
2. What is the estimated total cost of the plan?
3. Which join algorithm is used in the execution plan?

Next, build an index on the field `m.mid` and re-run the given query.

4. What is the execution plan? Also, include the output of PostgreSQL.
5. What is the estimated total cost of the plan?
6. Which join algorithm is used in the execution plan?
7. Did the execution plan change? (Yes/No)
8. Explain why the execution plan changed or remained the same.

Next, build an index on the field `a.mid` and re-run the given query.

9. What is the execution plan? Also, include the output of PostgreSQL.
10. What is the estimated total cost of the plan?
11. Which join algorithm is used in the execution plan?
12. Did the execution plan change after the creation of the second index? (Yes/No)
13. Explain why the execution plan changed or remained the same after the creation of both indices.

Finally, using the `SET`[3] command, disable the join algorithm used in part (l) above and re-run the given query.

14. What is the execution plan? Also, include the output of PostgreSQL.
15. What is the actual total cost of the plan?
16. Which join algorithm is used in the execution plan?

[3]The SET command can be used to change the behavior of the query planner. For example, `set enable_hash_join = false;` disables the query planner's use of hash join query plans.