CSCI 4707

Lab 4 Analysis

Rain Xuanyu Zhang (zhan2223, 4642453)

Hao Wang (wang5167, 5086487)

**Part 1**

(\*All the query running times listed below are the average time after 8-10 trials\*)

(\*Since the query result does not matter, to make the running time best displayable on the screen, we wrapped

SELECT COUNT(\*) FROM --query-- AS q

out of all the queries. The COUNT statement will have negligible effect on the overall query running time. \*)

(\*All the CREATE and DROP index statements has been written in the sql script file. This analysis document will only contain the select statements and the corresponding time efficiency. \*)

1. No Index

select age from student where age= 20 time = 00:00:00.026355

select year from student where year = 4 time = 00:00:00.027019

select gpa from student where gpa>3 time = 00:00:00.032992

Since no index is applied to any of the attributes, all select statements are executed by using scan over all records. The execution time increases as the number of returned records increases. In that sense, WHERE statement will determine how much records to be returned, and thus how much time will be spend on retrieving all qualified records.

2. B+ tree

select age from student where age=20 time = 00:00:00.014952

select year from student where year = 4 time = 00:00:00.018239

select gpa from student where gpa>3 time = 00:00:00.021597

The running times for each query are substantially reduced due to the application of the B+ tree index on all three of those attributes, comparing with above no-index approach. This reduction of running time is especially obvious on equality search, but when it comes to range search that involves in most part of the overall data, the time efficiency will be just the same as running on scan.

3. Hash-index

select \* from student where age=20 time = 00:00:00.017857

select year from student where year = 4 time = 00:00:00.022371

select gpa from student where gpa>3 time = 00:00:00.031892

Hash index is mostly used on equality search and the search criteria’s order has to comply with hash indexes’ prefix order. Here, the time efficiency of hash index is, apparently, more efficient than scan but almost the same as B+ tree index approach.

4.Hash-index and B-tree for range

(1) B-tree

select age from student where age > 1 time = 00:00:00.035786

select age from student where age > 10 time = 00:00:00.033814

select age from student where age > 50 time = 00:00:00.018606

(2) Hash-index

select age from student where age > 1 time = 00:00:00.051346

select age from student where age > 10 time = 00:00:00.041011

select age from student where age > 50 time = 00:00:00.029515

As we can see above, the run time for each range search with Hash-index is longer than the run time for same range search with B-tree. Because Hash-index is not optimized for range search, it is reasonable that it cost longer to range search with Hash-index. Moreover, it is obvious than the run time of range search and the amount of qualified data are positive correlation for B-tree and Hash-index. If the number of qualified data becomes less, the run time also becomes less.

5.Hash-index and B-tree for equal and range

(1)B-tree

select age from student where age = 20 time = 00:00:00.016136

select age from student where age>10 and age<20 time = 00:00:00.017036

(2)Hash-index

select age from student where age = 20 time = 00:00:00.014286

select age from student where age>10 and age<20 time = 00:00:00.031833

For equality search, we can know that the Hash-index is a little faster than B-tree. As for range search, B-tree is better than Hash-index. These data prove what we learn which is that for equality search, it takes 2D time using Hash, while it takes D\*(1+log F0.15B) time using B-tree and for range search, it takes BD time to run range query using Hash, while it takes D\*(logF0.15B+#matching records) time to run range query using B-tree.

6.

(1)Original

select \* from student where age>15 and sex=’male’ time = 00:00:00.051809

(2)B-tree only for age

select \* from student where age>15 and sex=’male’ time = 00:00:00.037687

(3)B-tree for age and Hash-index for sex

select \* from student where age>15 and sex=’male’ time = 00:00:00.013173

(4)Hash-index for age and sex

select \* from student where age>15 and sex=’male’ time = 00:00:00.016825

The hybrid indexing strategy that utilize B-tree is for range search and Hash-index for equality search is the best among these above four cases. Only B-tree or Hash-index is faster than no index applied. It shows that sequential scan (no index) is slower than B-tree or Hash-index or hybrid. In addition, the hybrid indexing which B-tree used for range search and Hash-index for equality search is better than only B-tree or Hash-index.

**Part 2**

1.

(1) without DISTINCT

select \* from student where age<30 time = 00:00:00.032001

(2) with DISTINCT

select distinct \* from student where age<30 time = 00:00:00.372044

DISTINCT key word will trigger the sorting/hashing algorithm that is built-in in the PostgreSQL. The difference is quite obvious by either using or not using DISTINCT.

2.

(1) using Where

select age, avg(gpa) as avegpa from student where age>22 group by age time = 00:00:00.063205

(2) using Having

select age, avg(gpa) as avegpa from student group by age having age>22 time = 00:00:00.090370

As we can see, the query with where is faster than the query with having since Where clause restricts records before grouping and Having clause restricts records after grouping.

3.

(1)using join and where

select distinct m.dname from major m join student s on s.sid = m.sid and s.age<30

time = 00:00:01.927912

Analysis:

HashAggregate  (cost=46304.12..46304.22 rows=10 width=3) (actual time=1659.718..1659.721 rows=10 loops=1)

   ->  Hash Join  (cost=5004.09..45344.54 rows=383831 width=3) (actual time=81.796..1595.136 rows=384640 loops=1)

         Hash Cond: (m.sid = s.sid)

         ->  Seq Scan on major m  (cost=0.00..17586.51 rows=1219151 width=7) (actual time=0.045..76.671 rows=1219151 loops=1)

         ->  Hash  (cost=3971.00..3971.00 rows=62967 width=4) (actual time=71.984..71.984 rows=63077 loops=1)

               Buckets: 4096  Batches: 4  Memory Usage: 559kB

               ->  Seq Scan on student s  (cost=0.00..3971.00 rows=62967 width=4) (actual time=0.017..45.036 rows=63077 loops=1)

                     Filter: (age < 30)

                     Rows Removed by Filter: 136923

 Total runtime: 1659.796 ms

(2)using in and nested query

select distinct m.dname from major m where m.sid in (select s.sid from student s where s.age<30)

time = 00:00:01.725931

Analysis:

HashAggregate  (cost=42733.11..42733.21 rows=10 width=3) (actual time=3248.803..3248.805 rows=10 loops=1)

   ->  Hash Semi Join  (cost=5004.09..41745.90 rows=394886 width=3) (actual time=81.234..3197.020 rows=384640 loops=1)

         Hash Cond: (m.sid = s.sid)

         ->  Seq Scan on major m  (cost=0.00..17586.51 rows=1219151 width=7) (actual time=0.020..65.346 rows=1219151 loops=1)

         ->  Hash  (cost=3971.00..3971.00 rows=62967 width=4) (actual time=59.937..59.937 rows=63077 loops=1)

               Buckets: 4096  Batches: 4  Memory Usage: 559kB

               ->  Seq Scan on student s  (cost=0.00..3971.00 rows=62967 width=4) (actual time=0.008..18.882 rows=63077 loops=1)

                     Filter: (age < 30)

                     Rows Removed by Filter: 136923

 Total runtime: 3248.851 ms

Those two queries have almost the same execution time by calculating the average after running eight times. From the analysis scripts below, outputted by adding EXPLAIN ANALYZE statement in front of each queries, we can see that the only difference in querying strategy between those two queries execution is that JOIN&WHERE used hash join and IN&nested used hash semi join. If data is very large, JOIN&WHERE query will be better since the cost of building hash index only take a little part of total time and hash index speeds up search. Otherwise, data is small, IN&nested query is better since hash semi join is enough to help us.