Shiyi Li ID: 4576123

Zhengbin Chen ID: 4656187

PART 1

We add statement EXPLAIN ANALYZE just right before all the queries, this statement will get rid of the printing time. Only give us the correct execution time.

ALL the statement to create the B-tree index and HASH index should be outside of timer table, so it would not have any effection on the time.

1.

SELECT S.age

FROM student S

WHERE S.age = 20; 5452 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 25ms | 26ms | 26ms | 26ms | 25ms | 25.6ms |

SELECT S.year

FROM student S

WHERE S.year = 5; 40051 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 30ms | 32ms | 31ms | 32ms | 33ms | 31.6ms |

SELECT S.gpa

FROM student S

WHERE S.gpa = 4; 66745 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 36ms | 35ms | 37ms | 36ms | 35ms | 35.8ms |

As we can see the time data, when the number of qualified data is small, the execution time would be small, and when the qualified data is larger, the execution time would be larger. The time increases 6 ms if data increase 40051-5452=34599. So time goes up 1ms / 5767 selected data. Because the only difference between those three queries is WHERE clause, so we can say in sequential scan method, when the number of qualified data increased, the time we need to get the data will be increased.

2.

SELECT S.age

FROM student S

WHERE S.age = 20; 5452 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 4.7ms | 4.6ms | 5.1ms | 4.9ms | 4.5ms | 4.76ms |

SELECT S.year

FROM student S

WHERE S.year = 5; 40051elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 21ms | 22ms | 20ms | 21ms | 21ms | 21ms |

SELECT S.gpa

FROM student S

WHERE S.gpa = 4; 66745elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 25ms | 24ms | 22ms | 25ms | 25ms | 24.2ms |

First, the creation of B –tree index statement and dropping index statement should be outside of the timer table, so it will not affect the execution time. When the qualified data is small, B-tree index will work efficiently, which will be much faster than scan. It just needs to search from root to leave once, and then get the data by pointer, since the data are chained. However, if the qualified data is super large, then it would have similar time as scan because this is unclustered index, which would cause a lot I/Os to get the data from different pages. The time increases 16.24 ms if data increase 40051-5452=34599. So time goes up 1ms / 2131 selected data. So the time increasing speed is much faster than scan. Since our qualified data is not large enough in this case, so our time would just be faster than scan.

3.

SELECT S.age

FROM student S

WHERE S.age = 20; 5452 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 5.3ms | 5.6ms | 4.9ms | 5.0ms | 5.0ms | 5.16ms |

SELECT S.year

FROM student S

WHERE S.year = 5; 40051elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 22ms | 23ms | 21ms | 20ms | 19ms | 21ms |

SELECT S.gpa

FROM student S

WHERE S.gpa = 4; 66745elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 25ms | 26ms | 25ms | 24ms | 26ms | 25.2ms |

First, the creation of HASH index statement and dropping index statement should be outside of the timer table, so it will not affect the execution time. We know that HASH would work more efficient than scan because it used INDEX, which will do much better job than sequential scan. We know that HASH will be best job on Equality search. In this case the qualified data is not very large, so it does not show really big difference with B-tree. However, theoretically, HASH is fastest for equality search on key.

4.

**Hash**

SELECT S.age

FROM student S

WHERE S.age > 1; 200000 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 40ms | 41ms | 40ms | 39ms | 40ms | 40ms |

SELECT S.age

FROM student S

WHERE S.age > 20; 184048 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 39ms | 40ms | 38ms | 39ms | 40ms | 39.2ms |

SELECT S.age

FROM student S

WHERE S.age >50; 26385 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 25ms | 24ms | 26ms | 25ms | 25ms | 25ms |

**B-tree**

SELECT S.age

FROM student S

WHERE S.age > 1; 200000 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 41ms | 40ms | 40ms | 41ms | 39ms | 40.2ms |

SELECT S.age

FROM student S

WHERE S.age > 20; 184048 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 39ms | 38ms | 40ms | 38ms | 38ms | 38.6ms |

SELECT S.age

FROM student S

WHERE S.age > 50; 26385 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 12ms | 11ms | 13ms | 12ms | 11ms | 12ms |

From the record, we can see that both B-Tree and HASH have some similarities, which that when the qualified data is getting less and less, the execution is also less and less. base on our record, we know that when the qualified data is large, both B-tree and HASH have very close time result. This is because the DBMS query plan used sequential scan method. when the data is small, the DBMS query plan use B-tree index, which is much faster.

**5.**

**Hash**

SELECT S.age

FROM student S

WHERE S.age = 20; 5452 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 4.4ms | 4.5ms | 4.7ms | 4.6ms | 4.3ms | 4.5ms |

SELECT S.age

FROM student S

WHERE S.age > 10 AND S.age < 20; 10698 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 10ms | 11ms | 12ms | 11ms | 12ms | 11.2ms |

**B-tree**

SELECT S.age

FROM student S

WHERE S.age = 20; 5452 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 5.4ms | 5.3ms | 5.3ms | 5.4ms | 5.4ms | 5.36ms |

SELECT S.age

FROM student S

WHERE S.age > 10 AND S.age < 20; 10698 elements selected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 10ms | 9.6ms | 9.8ms | 11ms | 9.2ms | 9.92ms |

From the result we can know that Hash can get equality query faster while B-tree can get inequality query faster.

From what we learnt from class and textbook we know that It only takes 2D time to run equality query using Hash, while it take D(1+logf0.15B) time to run equality query using B+tree. This explain why Hash is faster when we do query S.age = 20 (D is the time reading or writing a page-usually 15ms) (C is the time dealing with a data-usually 100ms).

We also learnt that it takes BD time to run range query using Hash, while it take D(logf0.15B+#matching records) time to run range query using B+Tree. This explain why Hash has similar time with B+Tree. Because in the above query (log f0.15B + #matching records) ≈ B. We can also know that the Hash will be faster if they are a lot of matching records.

**6.**

CREATE INDEX Sage ON student USING hash(age);

CREATE INDEX Ssex ON student USING hash(sex);

CREATE INDEX Sage ON student(age);

CREATE INDEX Ssex ON student(sex);

SELECT S.sid

FROM student S

WHERE S.age > 15 AND S.sex = ‘male’ ;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TIRAL 5 | AVG |
| 0 ,0 | 0.064 | 0.065 | 0.060 | 0.067 | 0.062 | 0.0636ms |
| 0, h | 0.059 | 0.063 | 0.068 | 0.060 | 0.066 | 0.0632ms |
| 0, b | 0.058 | 0.060 | 0.063 | 0.054 | 0.063 | 0.0596ms |
| h, 0 | 0.055 | 0.064 | 0.059 | 0.067 | 0.066 | 0.0622ms |
| b, 0 | 0.067 | 0.063 | 0.068 | 0.058 | 0.067 | 0.0646ms |
| b, h | 0.068 | 0.060 | 0.064 | 0.055 | 0.063 | 0.0620ms |
| h, b | 0.055 | 0.068 | 0.060 | 0.063 | 0.058 | 0.0608ms |
| h, h | 0.067 | 0.063 | 0.060 | 0.056 | 0.068 | 0.0628ms |
| b, b | 0.068 | 0.064 | 0.067 | 0.060 | 0.064 | 0.0646ms |

0 : does not use index

h : hash index used

b : b-tree index used

From the time data, we can see that the time does not change much using different index combinations. this is because the DMBS used sequential scan to retrieve the data due to the among of qualified data is very large. so the unclustered index does not help. if the qualified data is small, then the index combinations will be a lot helpful.

PART 2

1.

SELECT S.age

FROM student S

WHERE S.age < 30; 63077 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 30ms | 32ms | 29ms | 30ms | 31ms | 30.4ms |

SELECT DISTINCT S.age

FROM student S

WHERE S.age < 30; 12 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 132ms | 128ms | 135ms | 133ms | 132ms | 132ms |

Adding DISTINCT will be much slower than without DISTINCT because adding DISTINCT is just like adding more condition to it. It will do the same part to find the all the qualified data, then DISTINCT will do the extra work that to get rid of duplicated ones. It will probably sort all the attribution and get rid of the duplicate data. This takes a lot of extra time, so DISTINCT will be much slower.

**2.**

SELECT S.age, AVG(s.year)

FROM student S

WHERE S.age > 20 AND S.year = 4

GROUP BY S.age; 35 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 64.111ms | 65.840ms | 64.450ms | 64.782ms | 65.807ms | 64.998ms |

HashAggregate (cost=4658.13..4658.23 rows=8 width=8) (actual time=65.671..65.755 rows=35 loops=1)

-> Seq Scan on student s (cost=0.00..4471.00 rows=37426 width=8) (actual time=0.012..53.848 rows=36919 loops=1)

Filter: ((age > 20) AND (year = 4))

Rows Removed by Filter: 163081

Total runtime: 65.807 ms

SELECT S.age, AVG(s.year)

FROM student S

WHERE S.year = 4

GROUP BY S.age

HAVING S.age > 20; 35 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 65.963ms | 64.598ms | 64.527ms | 64.181ms | 66.054ms | 65.0646ms |

HashAggregate (cost=4658.13..4658.23 rows=8 width=8) (actual time=65.975..66.001 rows=35 loops=1)

-> Seq Scan on student s (cost=0.00..4471.00 rows=37426 width=8) (actual time=0.017..54.375 rows=36919 loops=1)

Filter: ((age > 20) AND (year = 4))

Rows Removed by Filter: 163081

Total runtime: 66.054 ms

The output from the statement EXPLAIN ANALYZE tells us that both queries do the same thing, the DBMS gives same query plan for both queries. So the execution time will be the same. there might be a difference, which that the overhead might be different, but it does not do much effect on this case.

3.

SELECT DISTINCT M.dname

FROM major M

JOIN student S

ON M.sid = S.sid

WHERE S.age < 30;

384640 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 1377ms | 1351ms | 1397ms | 1393ms | 1416ms | 1386.8ms |

HashAggregate (cost=46368.04..46368.14 rows=10 width=3) (actual time=1416.442..1416.445 rows=10 loops=1)

-> **Hash Join** (cost=5015.25..45398.51 rows=387812 width=3) (actual time=83.730..1310.258 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=0.022..210.936 rows=1219151 loops=1)

-> Hash (cost=3971.00..3971.00 rows=63620 width=4) (actual time=70.547..70.547 rows=63077 loops=1)

Buckets: 4096 Batches: 4 Memory Usage: 559kB

-> Seq Scan on student s (cost=0.00..3971.00 rows=63620 width=4) (actual time=0.009..41.037 rows=63077 loops=1)

Filter: (age < 30)

Rows Removed by Filter: 136923

Total runtime: 1416.488 ms

SELECT DISTINCT M.dname

FROM major M

WHERE M.sid IN (SELECT S.sid

FROM student S

WHERE S.age < 30);

384640 elements selected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TRIAL 1 | TRIAL 2 | TRIAL 3 | TRIAL 4 | TRIAL 5 | AVG |
| 1346ms | 1375ms | 1354ms | 1386ms | 1352ms | 1362.6ms |

HashAggregate (cost=42768.68..42768.78 rows=10 width=3) (actual time=1354.195..1354.197 rows=10 loops=1)

-> **Hash Semi Join** (cost=5015.25..41778.32 rows=396145 width=3) (actual time=81.168..1247.071 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..214.048 rows=1219151 loops=1)

-> Hash (cost=3971.00..3971.00 rows=63620 width=4) (actual time=70.945..70.945 rows=63077 loops=1)

Buckets: 4096 Batches: 4 Memory Usage: 559kB

-> Seq Scan on student s (cost=0.00..3971.00 rows=63620 width=4) (actual time=0.008..40.648 rows=63077 loops=1)

Filter: (age < 30)

Rows Removed by Filter: 136923

Total runtime: 1354.240 ms

From the records, we can see that the running time is almost same, through the query plan, we can see the only difference is IN version used hash semi join, and JOIN version used hash join. there might be a difference because of overhead, but does not much effect on this case.