Database Tuning – Assignment 4

Index Tuning

A2
Baumgartner Dominik, 0920177
Dafir Thomas Samy, 1331483
Schörgnhofer Kevin, 1421082

November 29, 2016

Notes:

- Do not forget to run `ANALYZE tablename` after creating or changing a table.
- Use `EXPLAIN ANALYZE` for the query plans that you display in the report.

1 Experimental Setup

We send our queries with a small java program to the database server. The java program was running on the computers of the RÜR.
The time measurement starts before we start sending the queries of one type and one table (different tables for different indexes) and stops afterwards. We repeat this for each query type and for each table.

2 Clustered B$^+$-Tree Index

Point Query

```
SELECT * FROM Publ WHERE pubID = ...
```

For this query we created a text file containing about 20000 randomly selected pubids from the publ.tsv file. Queries were then executed for each entry in that file.

Runtime: 65155ms.
Queries: 19728. Throughput: about 302 queries/sec.
Query plan (for one of the queries):

```
Index Scan using pubidcb on publ_cb  (cost=0.43..8.45 rows=1 width=113) (actual time=0.061..0.061
Index Cond: ((pubid)::text = 'books/idea/encyclopediaDB2005/Manjunath05'::text)
Planning time: 0.314 ms
Execution time: 0.079 ms
```

Multipoint Query – Low Selectivity Repeat the following query multiple times with different conditions for `booktitle`.

```
SELECT * FROM Publ WHERE booktitle = ...
```

Which conditions did you use?
We used a equality condition where the index matches a given string from the booktitle row of the publ.tsv file, like 'Software Engineering (Workshops)'.

Show the runtime results and compute the throughput.
For the Clustered B^+-Tree Index we achieved a runtime from $93660ms$ to $101960ms$ with 7190 searched values. This leads to a throughput of $70{,}5\frac{queries}{second}$ to $76{,}8\frac{queries}{second}$.

Query plan (for one of the queries):

```
EXPLAIN SELECT * FROM publ_cb WHERE booktitle = 'Software Engineering (Workshops)';

 Index Scan using booktitlecb on publ_cb  (cost=0.43..14.60 rows=181 width=112)
   Index Cond: ((booktitle)::text = 'Software Engineering (Workshops)'::text)
(2 rows)
```

Multipoint Query – High Selectivity Repeat the following query multiple times with different conditions for `year`.

```
SELECT * FROM Publ WHERE year = ...
```

Which conditions did you use?
The conditions were imported from a file which contains the years from 1920-2020 repeated up to a total amount of 1500 years.

Show the runtime results and compute the throughput.
Runtime (ms): 86316 Throughput (q/s): 17.37800639510635
Query plan (for one of the queries):

```
Bitmap Heap Scan on publ_cb  (cost=45.98..6695.42 rows=2265 width=112) (actual time=1.081..6.685
  Recheck Cond: ((year)::text = '1986'::text)
  Heap Blocks: exact=160
  ->  Bitmap Index Scan on yearcb  (cost=0.00..45.41 rows=2265 width=0) (actual time=1.049..1.04
        Index Cond: ((year)::text = '1986'::text)
Planning time: 0.484 ms
Execution time: 9.974 ms
```

3 Non-Clustered B$^+$-Tree Index

Note: Make sure the data is not physically ordered by the indexed attributes due to the clustering index that you created before.

Point Query

```
SELECT * FROM Publ WHERE pubID = ...
```

For this query we created a text file containing about 20000 randomly selected pubids from the publ.tsv file. Queries were then executed for each entry in that file.

Runtime: 64573ms.
Queries: 19728. Throughput: about 305 queries/sec.
Query plan (for one of the queries):

```
Index Scan using pubidb on publ_b  (cost=0.43..8.45 rows=1 width=112) (actual time=0.071..0.072 r
Index Cond: ((pubid)::text = 'books/idea/encyclopediaDB2005/Manjunath05'::text)
Planning time: 0.405 ms
Execution time: 0.092 ms
```

Multipoint Query – Low Selectivity Repeat the following query multiple times with different conditions for `booktitle`.

```
SELECT * FROM Publ WHERE booktitle = ...
```

Which conditions did you use?
We used a equality condition where the index matches a given string from the booktitle row of the publ.tsv file, like 'Software Engineering (Workshops)'.

Show the runtime results and compute the throughput.
For the Non-Clustered B^+-Tree Index we achieved a runtime from $99160ms$ to $113470ms$ with 7190 searched values. This leads to a throughput of $63{,}4\frac{queries}{second}$ to $72{,}5\frac{queries}{second}$.

Query plan (for one of the queries):

```
EXPLAIN SELECT * FROM publ_b WHERE booktitle = 'Software Engineering (Workshops)';

 Index Scan using booktitleb on publ_b  (cost=0.43..577.16 rows=181 width=113)
   Index Cond: ((booktitle)::text = 'Software Engineering (Workshops)'::text)
(2 rows)
```

Multipoint Query – High Selectivity Repeat the following query multiple times with different conditions for `year`.

```
SELECT * FROM Publ WHERE year = ...
```

Which conditions did you use?
The conditions were imported from a file which contains the years from 1920-2020 repeated up to a total amount of 1500 years.

Show the runtime results and compute the throughput.
Runtime (ms): 83157 Throughput (q/s): 18.038168765106967
Query plan (for one of the queries):

```
Bitmap Heap Scan on publ_b  (cost=55.01..7010.65 rows=2398 width=112) (actual time=2.148..34.510
  Recheck Cond: ((year)::text = '1986'::text)
  Heap Blocks: exact=3258
  ->  Bitmap Index Scan on yearb  (cost=0.00..54.41 rows=2398 width=0) (actual time=1.620..1.620
        Index Cond: ((year)::text = '1986'::text)
Planning time: 0.301 ms
Execution time: 38.380 ms
```

4 Non-Clustered Hash Index

Note: Make sure the data is not physically ordered by the indexed attributes due to the clustering index that you created before.

Point Query

```
SELECT * FROM Publ WHERE pubID = ...
```

For this query we created a text file containing about 20000 randomly selected pubids from the publ.tsv file. Queries were then executed for each entry in that file.

Runtime: 64503ms.
Queries: 19728. Throughput: about 305 queries/sec.
Query plan (for one of the queries):

```
Index Scan using pubidh on publ_h  (cost=0.00..8.02 rows=1 width=113) (actual time=0.037..0.038 r
Index Cond: ((pubid)::text = 'books/idea/encyclopediaDB2005/Manjunath05'::text)
Planning time: 0.368 ms
Execution time: 0.060 ms
```

Multipoint Query – Low Selectivity Repeat the following query multiple times with different conditions for `booktitle`.

```
SELECT * FROM Publ WHERE booktitle = ...
```

Which conditions did you use?
We used a equality condition where the index matches a given string from the booktitle row of the publ.tsv file, like 'Software Engineering (Workshops)'.

Show the runtime results and compute the throughput.
For the Non-Clustered Hash Index we achieved a runtime from $124470ms$ to $113470ms$ with 7190 searched values. This leads to a throughput of $57{,}8\frac{queries}{second}$ to $76{,}9\frac{queries}{second}$.

Query plan (for one of the queries):

```
EXPLAIN SELECT * FROM publ_h WHERE booktitle = 'Software Engineering (Workshops)';

 Bitmap Heap Scan on publ_h  (cost=5.37..668.34 rows=177 width=112)
   Recheck Cond: ((booktitle)::text = 'Software Engineering (Workshops)'::text)
   ->  Bitmap Index Scan on booktitleh  (cost=0.00..5.33 rows=177 width=0)
         Index Cond: ((booktitle)::text = 'Software Engineering (Workshops)'::text)
(4 rows)
```

Multipoint Query – High Selectivity Repeat the following query multiple times with different conditions for `year`.

```
SELECT * FROM Publ WHERE year = ...
```

Which conditions did you use?
The conditions were imported from a file which contains the years from 1920-2020 repeated up to a total amount of 1500 years.

Show the runtime results and compute the throughput.
Runtime (ms): 86386 Throughput (q/s): 17.363924710022456
Query plan (for one of the queries):

```
Bitmap Heap Scan on publ_h  (cost=68.65..6436.08 rows=2149 width=113) (actual time=2.409..32.008
  Recheck Cond: ((year)::text = '1986'::text)
  Heap Blocks: exact=3258
  ->  Bitmap Index Scan on yearh  (cost=0.00..68.12 rows=2149 width=0) (actual time=1.824..1.824
        Index Cond: ((year)::text = '1986'::text)
Planning time: 0.346 ms
Execution time: 35.861 ms
```

5 Table Scan

Note: Make sure the data is not physically ordered by the indexed attributes due to the clustering index that you created before.

Point Query

```
SELECT * FROM Publ WHERE pubID = ...
```

For this query we created a text file containing about 300 randomly selected pubids from the publ.tsv file. Queries were then executed for each entry in that file. The reason we used a different dataset is the enormous amount of time 20000 sequential scan queries would have taken.

Runtime: 75518ms.
Queries: 297.
Throughput: about 3.9 queries/sec.
Query plan (for one of the queries):

```
query plan
```

Multipoint Query – Low Selectivity

```
SELECT * FROM Publ WHERE booktitle = ...
```

Which conditions did you use?
We used a equality condition where the index matches a given string from the booktitle row of the publ.tsv file, like 'Software Engineering (Workshops)'.

Show the runtime results and compute the throughput.
For the Table Scan we achieved a runtime from 197998ms to 201809ms with 719 searched values. This leads to a throughput of 3,6$\frac{queries}{second}$.

Query plan (for one of the queries):

```
EXPLAIN SELECT * FROM publ_s WHERE booktitle = 'Software Engineering (Workshops)';

 Seq Scan on publ_s  (cost=0.00..37843.18 rows=179 width=112)
   Filter: ((booktitle)::text = 'Software Engineering (Workshops)'::text)
(2 rows)
```

Multipoint Query – High Selectivity

```
SELECT * FROM Publ WHERE year = ...
```

Which conditions did you use?
The conditions were imported from a file which contains the years from 1920-2020 repeated up to a total amount of 1500 years.

Show the runtime results and compute the throughput.
Runtime (ms): 485258 Throughput (q/s): 3.0911391465983042
Query plan (for one of the queries):

```
 Seq Scan on publ_s  (cost=0.00..37777.18 rows=2224 width=112) (actual time=0.041..354.600 rows=9
   Filter: ((year)::text = '1986'::text)
   Rows Removed by Filter: 1224032
 Planning time: 0.746 ms
 Execution time: 358.390 ms
```

6 Discussion

Give the throughput of the query types and index types in queries/second.

	clustered	non-clust. B^+-tree	non-clust. hash	table scan
point (`pubID`)	302	305	305	3,9
multipoint (`booktitle`)	73,14	69,37	69,73	3,6
multipoint (`year`)	17,38	18,04	17,36	3,01

6.1 Table Scan

The results for the table scan are as expected for all query types. The database system can obviously not use an index an has to do a sequential scan over the whole table, which is slow, hence the throughput is low.

6.2 Index Scans

The results of the different index scans were nearly identical for each query type, which we kind of expected because of the low complexity of the queries.
The differences of the throughputs of the different query types were also expected:
In the statistics of the DBS the variety of values for each attribute is stored. The single point query (pubID) returns only one tuple, whereas the multipoint queries have to find and return way more tuples (e.g. years about 12000 tuples), which is more expensive. This explains why the single point query hast the highest throughput and the multipoint query for the year the lowest.
Our expectations were, that the clustered B^+-tree would be faster than the non-clustered, which only occured on the first multipoint query (booktitle). This might be caused by the fact, that the non-clustered and the clustered B^+-tree both use a Bitmap Heap Scan which in our opinion makes no sense for the clustered one.
The differences between the throughput of the different index types of the single point query are in our opinion a little bit to small, because one tree traversel should be more expensive than one hash calculation. With about 20000 queries this should probably sum up to a higher difference. This is true for the other two query types as well. Especially for the (year) multipoint query it is weird that the DBS uses for each index a Bitmap Scan, hence does not take any advantage of the clustering.

Time in hours per person: **XXX**

Important: Reference your information sources!