Assignment 4

Index Tuning – Selection

Database Tuning

A4

Balint Peter, 12213073

Günter Lukas, 12125639

Ottino David, 51841010

May 19, 2025

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.

Experimental Setup

How do you send the queries to the database? How do you measure the execution time for a sequence of queries?

Data Set: publ.tsv, size: 118 MB and auth.tsv, size: 139 MB

Schemes: P{name, pubid} A{pubid, type, title, booktitle, year, publisher}

We use: Python3

DBMS: PostgreSQL 14.7

When we filled our tables we avoided key or foreign key constraints to avoid conflicts with later index creation. Both are running local on the same machine to avoid network latencies. We measure the time of the individual queries with the built in “time” library of python and the overall running time of the whole script the with “time” function of bash

Clustering B^+ Tree Index

Point Query Repeat the following query multiple times with different conditions for `pubID`.

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

Which conditions did you use?

We extracted the first 100 unique values of the attribute `pubID` from the table `Publ`. The table is randomly sorted to avoid always selecting the same first values. Those values are then used in the query. We avoid using the same queries directly after each other since

Postgres could cache the results, therefore consecutive executions of the same query will in general lead to faster runtimes.

```
SELECT pubID FROM Publ
WHERE pubID IS NOT NULL
GROUP BY pubID
ORDER BY RANDOM()
LIMIT 100;
```

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.00020982424418131512 seconds

Calculated throughput: 4765 queries per second

Query plan (for one of the queries):

```
Index Scan using idx_pubid_clustered on publ  (cost=0.43..8.45 rows=1 width=118)
(actual time=0.088..0.088 rows=1 loops=1)
  Index Cond: ((pubid)::text = 'conf/agents/Sengers98'::text)
```

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

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

SELECT * FROM Publ WHERE pubID IN (...)
```

Which conditions did you use?

Again, we selected 100 random values from `Publ` and ran the queries with those values. For the multipoint query with IN-Predicate we also selected 100 random values and split them up in groups of 10, which then are used for the IN-selection.

```
SELECT booktitle FROM Publ
WHERE booktitle IS NOT NULL
GROUP BY booktitle
ORDER BY RANDOM()
LIMIT 100;
```

Show the runtime results and compute the throughput.

Average runtime over all queries (multipoint): 0.13089529355367024 seconds

Average runtime over all queries (multipoint with IN): 0.00035497546195983887 seconds

Calculated throughput (multipoint): 8.8893 queries per second

Calculated throughput (multipoint with IN): 2 817 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30921.08 rows=181 width=118)
(actual time=91.466..131.932 rows=128 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=75 width=118)
      (actual time=77.534..89.322 rows=43 loops=3)
          Filter: ((booktitle)::text = 'W4A'::text)

Index Scan using idx_pubid_clustered on publ  (cost=0.43..51.70 rows=10 width=118)
(actual time=0.028..0.130 rows=10 loops=1)
  Index Cond: ((pubid)::text = ANY ('{journals/eik/Burkhard76a,...}'::text[]))
```

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?

Once again 100 random values have been select from `publ`.

```
SELECT year FROM Publ
WHERE year IS NOT NULL
GROUP BY year
ORDER BY RANDOM()
LIMIT 100;
```

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.1545756498972575 seconds

Calculated throughput: 6.46 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30922.58 rows=196 width=118)
(actual time=45.647..120.119 rows=203 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=82 width=118)
        (actual time=44.298..82.342 rows=68 loops=3)
           Filter: ((year)::text = '1960'::text)
           Rows Removed by Filter: 411003
```

Non-Clustering 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 Repeat the following query multiple times with different conditions for `pubID`.

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

Which conditions did you use?

Same as before.

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.00019787152608235676 seconds

Caculated throughput: 5053 queries per second

Query plan (for one of the queries):

```
Index Scan using idx_pubid_hash on publ  (cost=0.00..8.02 rows=1 width=118)
(actual time=0.015..0.015 rows=1 loops=1)
  Index Cond: ((pubid)::text = 'journals/entcs/AhoniemiL07'::text)
```

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

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

```
SELECT * FROM Publ WHERE pubID IN (...)
```

Which conditions did you use?

See above

Show the runtime results and compute the throughput.

Average runtime over all queries (multipoint): 0.11964255650838217 seconds

Average runtime over all queries (multipoint with IN): 0.0004431605339050293 seconds

Calculated throughput (multipoint): 8.3582 queries per second

Calculated throughput (multipoint with IN): 2256 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30921.08 rows=181 width=118)
(actual time=30.014..121.291 rows=345 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=75 width=118)
        (actual time=45.154..81.109 rows=115 loops=3)
           Filter: ((booktitle)::text = 'PROLAMAT'::text)
           Rows Removed by Filter: 410956
```

```
Index Scan using idx_pubid_nonclustered on publ  (cost=0.43..51.70 rows=10 width=118)
(actual time=0.032..0.114 rows=10 loops=1)
  Index Cond: ((pubid)::text = ANY ('{journals/bell/Labrogere08,...}'::text[]))
```

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?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.1524947452545166 seconds

Query plan (for one of the queries): 6.5576 queries per second

```
Gather  (cost=1000.00..31096.18 rows=1932 width=118)
(actual time=4.400..157.862 rows=1531 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=805 width=118)
        (actual time=30.366..72.276 rows=510 loops=3)
           Filter: ((year)::text = '1973'::text)
           Rows Removed by Filter: 410561
```

Non-Clustering 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 Repeat the following query multiple times with different conditions for pubID.

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

Which conditions did you use?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.00010968208312988282 seconds

Calculated throughput: 9117 queries per second

Query plan (for one of the queries):

```
Index Scan using idx_pubid_hash on publ  (cost=0.00..8.02 rows=1 width=118)
(actual time=0.015..0.015 rows=1 loops=1)
  Index Cond: ((pubid)::text = 'journals/entcs/AhoniemiL07'::text)
```

Multipoint Query vs. Multipoint Query IN-Predicate – Low Selectivity Repeat the following query multiple times with different conditions for booktitle.

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

```
SELECT * FROM Publ WHERE pubID IN (...)
```

Which conditions did you use?

Same as above

Show the runtime results and compute the throughput.

Average runtime over all queries (multipoint): 0.11964255650838217 seconds

Average runtime over all queries (multipoint with IN): 0.0006836056709289551 seconds

Calculated throughput (multipoint): 8.3582 queries per second

Calculated thorughput (multipoint with IN): 1462 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30921.08 rows=181 width=118)
(actual time=26.621..155.068 rows=345 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=75 width=118)
        (actual time=11.182..62.757 rows=115 loops=3)
           Filter: ((booktitle)::text = 'PROLAMAT'::text)
           Rows Removed by Filter: 410956

Bitmap Heap Scan on publ  (cost=40.10..79.63 rows=10 width=118)
(actual time=0.281..0.291 rows=10 loops=1)
  Recheck Cond: ((pubid)::text = ANY ('{journals/bell/Labrogere08,...}'::text[]))
  Heap Blocks: exact=10
  ->  Bitmap Index Scan on idx_pubid_hash  (cost=0.00..40.08 rows=10 width=0)
        (actual time=0.255..0.255 rows=10 loops=1)
           Index Cond: ((pubid)::text = ANY ('{journals/bell/Labrogere08,...}'::text[]))
```

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?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.1416764259338379 seconds

Calculated throughput: 7.0583 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..31096.18 rows=1932 width=118)
(actual time=3.121..162.150 rows=1531 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=805 width=118)
       (actual time=22.598..57.898 rows=510 loops=3)
           Filter: ((year)::text = '1973'::text)
           Rows Removed by Filter: 410561
```

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 Repeat the following query multiple times with different conditions for `pubID`.

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

Which conditions did you use?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries (single select): 0.11055562655131022 seconds

Calculated throughput (single select): 9.0452 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30903.08 rows=1 width=118)
(actual time=98.916..147.098 rows=1 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=1 width=118)
        (actual time=45.902..49.239 rows=0 loops=3)
           Filter: ((pubid)::text = 'journals/entcs/AhoniemiL07'::text)
           Rows Removed by Filter: 411071
```

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

```
SELECT * FROM Publ WHERE booktitle = ...
SELECT * FROM Publ WHERE pubID IN (...)
```

Which conditions did you use?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries (multipoint IN): 0.13194787502288818 seconds

Average runtime over all queries (multipoint): 0.11779035886128743 seconds

Calculated throughput (multipoint IN): 7.5787 queries per second

Calculated throughput (multipoint): 8.4896 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..30921.08 rows=181 width=118)
(actual time=26.307..96.095 rows=345 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=75 width=118)
        (actual time=10.139..54.469 rows=115 loops=3)
           Filter: ((booktitle)::text = 'PROLAMAT'::text)
           Rows Removed by Filter: 410956

Gather  (cost=1000.02..32188.61 rows=10 width=118)
(actual time=16.712..103.601 rows=10 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.03..31187.61 rows=4 width=118)
       (actual time=20.996..60.818 rows=3 loops=3)
           Filter: ((pubid)::text = ANY ('{journals/bell/Labrogere08,...}'::text[]))
           Rows Removed by Filter: 411068
```

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?

Same as above.

Show the runtime results and compute the throughput.

Average runtime over all queries: 0.14277080535888673 seconds

Calculated throughput: 7.0042 queries per second

Query plan (for one of the queries):

```
Gather  (cost=1000.00..31096.18 rows=1932 width=118)
(actual time=3.930..109.379 rows=1531 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on publ  (cost=0.00..29902.98 rows=805 width=118)
       (actual time=32.132..64.346 rows=510 loops=3)
           Filter: ((year)::text = '1973'::text)
           Rows Removed by Filter: 410561
```

Discussion

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

	clustering	non-clust. B^+ tree	non-clust. hash	table scan
point (`pubID`)	4765	5053	9117	9.0452
multipoint (`booktitle`)	8.8893	8.3582	8.3582	8.4896
multipoint-IN (`pubID`)	2817	2256	1462	7.5787
multipoint (`year`)	6.46	6.5576	7.0583	7.0042

Discuss the runtime results for the different index types and the table scan. Are the results expected? Why (not)?

The runtime results are more or less expected. Hash indexes perform great for single point queries and therefore got the best results there, while a table scan has to go through the whole table to return the value which takes a long time.

According to the analyzed query plan, the index is not used for a multipoint query. The performance is always more or less the same, regardless of index, because Postgres prefers to do a parallel sequential scan in this case. This is because an index scan requires several IO operations for each row (look up the row in the index, then retrieve the row from the heap). Whereas a sequential scan only requires a single IO for each row - or even less because a block (page) on the disk contains more than one row, so more than one row can be fetched with a single IO operation.

The multipoint query with the IN-statement was the only surprise during the experiment. Here the clustering B^+ tree had a clearly better performance. In every analyzed query the index was actually used, and to no surprise the Hash-index was outperformed since it is best for single-point and not for the multi-point-query (which is a series of single point queries in this case). However, the clustering index was significantly faster than its non-clustering counterpart. Most likely Postgres orders the values in the IN-statement and is therefore able to take advantage of the physical sorting, but we were unable to find clear evidence for this theory.

The high-selectivity multipoint query again results in a table scan of all the data, therefore the results are more or less the same. Again, Postgres prefers the table scan in this case to reduce operations with the index.

Time Spent on this Assignment

Time in hours per person: **5h**

References

Important: Reference your information sources!

https://www.timescale.com/blog/database-scaling-postgresql-caching-explained
https://stackoverflow.com/questions/5203755/why-does-postgresql-perform-sequential-scan-on-indexed-co