Assignment 5

Join Tuning

Database Tuning

Gruppe J
Brezovic Ivica, 11702570
Demir Cansu, 11700525
Mrazovic Mirna, 11700383

May 16, 2020

Experimental Setup

Describe your experimental setup in a few lines.

Das Java-Programm stellt eine Verbindung zur lokalen Datenbank oder zur Server Datenbank her. Es initialisiert die Tabellen und Indexes für jede Aufgabe (1-4). Daraufhin löscht es die Tabellen und Indexes, wann immer es erforderlich ist. Die ResultSets mit der Ausgabe von EXPLAIN ANALYZE werden abgerufen und auf die Konsole ausgegeben.

Join Strategies Proposed by System

Response times
On localhost

Indexes	Join Strategy Q1	Join Strategy Q2
no index	Merge Join	Hash Join
unique non-clustering on `Publ.pubID`	Hash Join	Nested Loop Join
clustering on `Publ.pubID` and `Auth.pubID`	Hash Join	Nested Loop Join

Response times
On server

Indexes	Join Strategy Q1	Join Strategy Q2
no index	Merge Join	Hash Join
unique non-clustering on `Publ.pubID`	Hash Join	Nested Loop Join
clustering on `Publ.pubID` and `Auth.pubID`	Hash Join	Nested Loop Join

Discussion Discuss your observations. Is the choice of the strategy expected? How does the system come to this choice?

Um zu entscheiden, mit welcher Strategie ein Join ausgeführt werden soll, führt das System Äquivalenztransformationen an der Query durch, um eine Reihe von äquivalenten Kandidatenabfragen zu generieren. Beispielsweise kann die Reihenfolge von Joins, Auswahlen usw. neu geordnet werden. Das Ziel ist in der Regel, die Anzahl oder die Größe der Zwischenergebnisse so weit wie möglich zu reduzieren. Die Kandidaten-Querys werden dann "annotated", was bedeutet, dass mögliche Querypläne generiert werden. Dies beinhaltet die Prüfung auf zutreffende Indexes, die Auswahl eines Join-Algorithmus und die Berücksichtigung von Statistiken. Dann werden die Kosten für die Ausführung geschätzt, um den besten Queryplan auszuwählen.

Bei der Einrichtung ohne Index hat das DBMS keine andere Wahl, als teure Operationen wie Sortieren, Hashing oder sogar den Aufbau einer temporären Indexstruktur durchzuführen, um die Verknüpfung durchzuführen.

Für einen unique non-clustering index wird die Query 1 gehasht und dann eine Hash-Join verwendet, um alle IDs nachzuschlagen. Der erstellte Index wird im Queryplan überhaupt nicht verwendet. Das DBMS schätzt die Kosten des Plans um eine Größenordnung niedriger ein als der alte Merge-Join-Plan. Hashing ist in der Regel schneller als ein B+ Baum-Lookup.

Query 2 wird ausgeführt, indem Auth erneut nach dem Namensattribut gefiltert wird und dann der Index auf Publ.pubID verwendet wird, um die verbleibenden Zeilen nachzuschlagen, deren Anzahl auf 792 geschätzt wird.

Indexed Nested Loop Join

Response times
On localhost

Indexes	Response time Q1 [ms]	Response time Q2 [ms]
index on `Publ.pubID`	15338.856 ms	768.958 ms
index on `Auth.pubID`	7968.559 ms	8190.180 ms
index on `Publ.pubID` and `Auth.pubID`	9631.952 ms	544.132 ms

Response times
On server

Indexes	Response time Q1 [ms]	Response time Q2 [ms]
index on `Publ.pubID`	42608.860 ms	675.785 ms
index on `Auth.pubID`	19094.566 ms	68233.062 ms
index on `Publ.pubID` and `Auth.pubID`	18953.848 ms	617.987 ms

Query plans
On localhost

Index on `Publ.pubID` (Q1/Q2):

```
Q1:
Gather  (cost=1000.43..225501.12 rows=509162 width=184) (actual
   ↪ time=0.992..15215.912 rows=3095201 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Nested Loop  (cost=0.43..173584.92 rows=212151
                 ↪ width=184) (actual time=0.503..14612.300 rows
                 ↪ =1031734 loops=3)
                    ->  Parallel Seq Scan on "Auth"  (cost
                       ↪ =0.00..28919.51 rows=212151 width=392) (
                       ↪ actual time=0.404..380.405 rows=1031734
                       ↪ loops=3)
                    ->  Index Scan using publ_pubid_idx on "Publ"
                       ↪  (cost=0.43..0.67 rows=1 width=90) (
                       ↪ actual time=0.013..0.013 rows=1 loops
                       ↪ =3095201)
                          Index Cond: (("pubID")::text = ("Auth"."
                             ↪ pubID")::text)
            Planning Time: 0.403 ms
            Execution Time: 15338.856 ms


Q2:
Gather  (cost=1000.43..38755.10 rows=2546 width=68) (actual time
   ↪ =86.372..768.872 rows=183 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Nested Loop  (cost=0.43..37500.50 rows=1061
                 ↪ width=68) (actual time=38.398..371.275 rows=61
                 ↪  loops=3)
                    ->  Parallel Seq Scan on "Auth"  (cost
                       ↪ =0.00..29449.89 rows=1061 width=276) (
                       ↪ actual time=37.535..367.509 rows=61
                       ↪ loops=3)
                          Filter: ((name)::text = 'Divesh
                             ↪ Srivastava'::text)
                          Rows Removed by Filter: 1031673
                    ->  Index Scan using publ_pubid_idx on "Publ"
                       ↪  (cost=0.43..7.58 rows=1 width=90) (
                       ↪ actual time=0.058..0.059 rows=1 loops
                       ↪ =183)
                          Index Cond: (("pubID")::text = ("Auth"."
                             ↪ pubID")::text)
            Planning Time: 0.241 ms
            Execution Time: 768.958 ms
```

Index on `Auth.pubID` (Q1/Q2):

```
Q1:
Gather  (cost=1000.55..738109.50 rows=3095201 width=83) (actual
   ↪ time=0.867..7849.964 rows=3095201 loops=1)
             Workers Planned: 2
             Workers Launched: 2
             ->  Nested Loop  (cost=0.56..427589.40 rows=1289667
                ↪ width=83) (actual time=0.526..7289.373 rows
                ↪ =1031734 loops=3)
                   ->  Parallel Seq Scan on "Publ"  (cost
                      ↪ =0.00..27500.39 rows=513839 width=90) (
                      ↪ actual time=0.423..149.982 rows=411071
                      ↪ loops=3)
                   ->  Index Scan using auth_pubid_idx on "Auth"
                      ↪  (cost=0.56..0.75 rows=3 width=38) (
                      ↪ actual time=0.016..0.016 rows=3 loops
                      ↪ =1233214)
                         Index Cond: (("pubID")::text = ("Publ"."
                            ↪ pubID")::text)
          Planning Time: 0.860 ms
          Execution Time: 7968.559 ms


Q2:
Gather  (cost=1000.55..422168.91 rows=25 width=68) (actual time
   ↪ =1129.260..8190.122 rows=183 loops=1)
             Workers Planned: 2
             Workers Launched: 2
             ->  Nested Loop  (cost=0.56..421166.41 rows=10 width
                ↪ =68) (actual time=833.964..8071.213 rows=61
                ↪ loops=3)
                   ->  Parallel Seq Scan on "Publ"  (cost
                      ↪ =0.00..27500.39 rows=513839 width=90) (
                      ↪ actual time=0.478..161.415 rows=411071
                      ↪ loops=3)
                   ->  Index Scan using auth_pubid_idx on "Auth"
                      ↪  (cost=0.56..0.76 rows=1 width=23) (
                      ↪ actual time=0.019..0.019 rows=0 loops
                      ↪ =1233214)
                         Index Cond: (("pubID")::text = ("Publ"."
                            ↪ pubID")::text)
                         Filter: ((name)::text = 'Divesh
                            ↪ Srivastava'::text)
                         Rows Removed by Filter: 3
          Planning Time: 0.171 ms
          Execution Time: 8190.180 ms
```

Index on `Auth.pubID` and `Auth.pubID` (Q1/Q2):

```
Q1:
Gather  (cost=1000.55..752240.07 rows=3095201 width=82) (actual
   ↪ time=0.885..9481.923 rows=3095201 loops=1)
             Workers Planned: 2
             Workers Launched: 2
             ->  Nested Loop  (cost=0.56..441719.97 rows=1289667
                ↪ width=82) (actual time=0.726..8769.791 rows
                ↪ =1031734 loops=3)
                   ->  Parallel Seq Scan on "Publ"  (cost
                      ↪ =0.00..27500.39 rows=513839 width=89) (
                      ↪ actual time=0.443..226.634 rows=411071
                      ↪ loops=3)
                   ->  Index Scan using auth_pubid_idx on "Auth"
                      ↪  (cost=0.56..0.77 rows=4 width=38) (
                      ↪ actual time=0.019..0.020 rows=3 loops
                      ↪ =1233214)
                         Index Cond: (("pubID")::text = ("Publ"."
                            ↪ pubID")::text)
          Planning Time: 0.326 ms
          Execution Time: 9631.952 ms


Q2:
Gather  (cost=1000.43..44005.79 rows=24 width=67) (actual time
   ↪ =144.746..544.073 rows=183 loops=1)
             Workers Planned: 2
             Workers Launched: 2
             ->  Nested Loop  (cost=0.43..43003.39 rows=10 width
                ↪ =67) (actual time=57.159..412.715 rows=61
                ↪ loops=3)
                   ->  Parallel Seq Scan on "Auth"  (cost
                      ↪ =0.00..42918.84 rows=10 width=23) (
                      ↪ actual time=56.050..405.636 rows=61
                      ↪ loops=3)
                         Filter: ((name)::text = 'Divesh
                            ↪ Srivastava'::text)
                         Rows Removed by Filter: 1031673
                   ->  Index Scan using publ_pubid_idx on "Publ"
                      ↪  (cost=0.43..8.45 rows=1 width=89) (
                      ↪ actual time=0.111..0.112 rows=1 loops
                      ↪ =183)
                         Index Cond: (("pubID")::text = ("Auth"."
                            ↪ pubID")::text)
          Planning Time: 0.170 ms
          Execution Time: 544.132 ms
```

Query plans
On server

Index on `Publ.pubID` (Q1/Q2):

```
Q1:
Nested Loop  (cost=0.43..379086.33 rows=509162 width=183) (actual
   ↪ time=0.141..42296.657 rows=3095201 loops=1)
              ->  Seq Scan on "Auth"  (cost=0.00..31889.62 rows
                 ↪ =509162 width=392) (actual time=0.032..976.389
                 ↪  rows=3095201 loops=1)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..0.67 rows=1 width=89) (actual time
                 ↪ =0.012..0.012 rows=1 loops=3095201)
                    Index Cond: (("pubID")::text = ("Auth"."pubID"
                       ↪ )::text)
            Planning time: 0.642 ms
            Execution time: 42608.860 ms


Q2:
Nested Loop  (cost=0.43..52480.96 rows=2546 width=67) (actual time
   ↪ =59.574..675.713 rows=183 loops=1)
              ->  Seq Scan on "Auth"  (cost=0.00..33162.53 rows
                 ↪ =2546 width=276) (actual time=59.517..671.897
                 ↪ rows=183 loops=1)
                    Filter: ((name)::text = 'Divesh Srivastava'::
                       ↪ text)
                    Rows Removed by Filter: 3095018
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..7.58 rows=1 width=89) (actual time
                 ↪ =0.018..0.019 rows=1 loops=183)
                    Index Cond: (("pubID")::text = ("Auth"."pubID"
                       ↪ )::text)
            Planning time: 0.173 ms
            Execution time: 675.785 ms
```

Index on `Auth.pubID` (Q1/Q2):

```
Q1:
Nested Loop  (cost=0.43..874605.70 rows=3095201 width=83) (actual
   ↪ time=0.068..18813.458 rows=3095201 loops=1)
              ->  Seq Scan on "Publ"  (cost=0.00..34694.14 rows
                 ↪ =1233214 width=90) (actual time=0.016..379.763
                 ↪  rows=1233214 loops=1)
              ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                 ↪ =0.43..0.64 rows=4 width=38) (actual time
                 ↪ =0.013..0.014 rows=3 loops=1233214)
                    Index Cond: (("pubID")::text = ("Publ"."pubID"
                       ↪ )::text)
            Planning time: 0.379 ms
            Execution time: 19094.566 ms


Q2:
Nested Loop  (cost=0.00..544139.25 rows=24 width=68) (actual time
   ↪ =5943.751..68232.931 rows=183 loops=1)
              Join Filter: (("Auth"."pubID")::text = ("Publ"."
                 ↪ pubID")::text)
              Rows Removed by Join Filter: 225677979
              ->  Seq Scan on "Publ"  (cost=0.00..34694.14 rows
                 ↪ =1233214 width=90) (actual time=0.034..380.497
                 ↪  rows=1233214 loops=1)
              ->  Materialize  (cost=0.00..65488.13 rows=24 width
                 ↪ =23) (actual time=0.000..0.019 rows=183 loops
                 ↪ =1233214)
                    ->  Seq Scan on "Auth"  (cost=0.00..65488.01
                       ↪ rows=24 width=23) (actual time
                       ↪ =52.825..616.512 rows=183 loops=1)
                          Filter: ((name)::text = 'Divesh
                             ↪ Srivastava'::text)
                          Rows Removed by Filter: 3095018
            Planning time: 0.279 ms
            Execution time: 68233.062 ms
```

Index on `Auth.pubID` and `Auth.pubID` (Q1/Q2):

```
Q1:
Nested Loop  (cost=0.43..840692.31 rows=3095201 width=82) (actual
   ↪ time=0.066..18678.916 rows=3095201 loops=1)
              ->  Seq Scan on "Publ"  (cost=0.00..34694.14 rows
                 ↪ =1233214 width=89) (actual time=0.018..369.927
                 ↪  rows=1233214 loops=1)
              ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                 ↪ =0.43..0.62 rows=3 width=38) (actual time
                 ↪ =0.013..0.014 rows=3 loops=1233214)
                    Index Cond: (("pubID")::text = ("Publ"."pubID"
                       ↪ )::text)
            Planning time: 0.371 ms
            Execution time: 18953.848 ms
```

```
Q2:
Nested Loop  (cost=0.43..65690.93 rows=24 width=67) (actual time
   ↪ =53.610..617.922 rows=183 loops=1)
              ->  Seq Scan on "Auth"  (cost=0.00..65488.01 rows=24
                 ↪  width=23) (actual time=53.490..611.828 rows
                 ↪ =183 loops=1)
                    Filter: ((name)::text = 'Divesh Srivastava'::
                       ↪ text)
                    Rows Removed by Filter: 3095018
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..8.45 rows=1 width=89) (actual time
                 ↪ =0.031..0.031 rows=1 loops=183)
                    Index Cond: (("pubID")::text = ("Auth"."pubID"
                       ↪ )::text)
            Planning time: 0.200 ms
            Execution time: 617.987 ms
```

Discussion Discuss your observations. Are the response times expected? Why (not)?

Index auf `Publ.pubID`: Da es einen Index auf `Publ.pubID` aber nicht auf `Auth.pubID` gibt, muss der DBS `Publ` als innere Relation verwenden, um einen Indexed Nested Loop Join durchzuführen. Der Grund, warum die erste Query eine sehr lange Laufzeit hat, liegt darin, dass `Auth` die größere Tabelle ist, aber für eine optimale Leistung sollte die kleinere Tabelle die äußere sein. Dies ist auch der Grund, warum die zweite Query extrem schnell ausgeführt wird: Der DBS kann `Auth` scannen und alle Tupel entfernen, die die gesuchte Bedingung nicht erfüllen, was die Größe der äußeren Relation enorm reduziert.

Index auf `Auth.pubID`: Hier ist `Publ` die äußere Query, da wir nur einen Index auf `Auth` haben. Da die kleinere Relation jetzt die äußere Relation ist, ist die erste Query jetzt fast doppelt so schnell. Die zweite Query hat eine ähnliche Laufzeit, da der DBS jedes Tupel aus `Auth` mit `Publ.pubID=Auth.pubID` lesen muss und den Filter erst danach anwenden kann.

Index auf `Publ.pubID` **und** `Auth.pubID`: Wenn zwei Indexes existieren, wird die kleinere Beziehung nach außen gesetzt. Bei der ersten Query wäre dies `Publ` und bei der zweiten Query wäre dies die gefilterte `Auth`. Dies führt zu Laufzeiten, die den "optimalen" Laufzeiten, die wir bereits bei den ersten beiden Varianten hatten, sehr ähnlich sind.

Sort-Merge Join

Response times
On localhost

Indexes	Response time Q1 [ms]	Response time Q2 [ms]
no index	45444.965 ms	6105.353 ms
two non-clustering indexes	3531.287 ms	1767.829 ms
two clustering indexes	2497.417 ms	1464.339 ms

Response times
On server

Indexes	Response time Q1 [ms]	Response time Q2 [ms]
no index	64410.432 ms	18360.732 ms
two non-clustering indexes	5638.950 ms	1972.349 ms
two clustering indexes	4992.177 ms	1709.057 ms

Query plans
On localhost

No index (Q1/Q2):

```
Q1:
Gather  (cost=528031.98..866419.29 rows=3095201 width=82) (actual
   ↪ time=35902.877..45268.947 rows=3095201 loops=1)
            Workers Planned: 2
            Workers Launched: 2
            ->  Merge Join  (cost=527031.98..555899.19 rows
               ↪ =1289667 width=82) (actual time
               ↪ =36058.381..42148.961 rows=1031734 loops=3)
                  Merge Cond: (("Auth"."pubID")::text = ("Publ".
                     ↪ "pubID")::text)
                  ->  Sort  (cost=241118.63..244342.80 rows
                     ↪ =1289667 width=38) (actual time
                     ↪ =11152.305..12698.666 rows=1031734 loops
                     ↪ =3)
                        Sort Key: "Auth"."pubID"
                        Sort Method: external merge  Disk: 49856
                           ↪ kB
                        Worker 0:  Sort Method: external merge
                           ↪ Disk: 48928kB
                        Worker 1:  Sort Method: external merge
                           ↪ Disk: 49504kB
                        ->  Parallel Seq Scan on "Auth"  (cost
                           ↪ =0.00..39694.67 rows=1289667 width
                           ↪ =38) (actual time=0.399..232.974
                           ↪ rows=1031734 loops=3)
                  ->  Materialize  (cost=285913.35..292079.42
                     ↪ rows=1233214 width=89) (actual time
                     ↪ =24897.690..27130.377 rows=1856276 loops
                     ↪ =3)
                        ->  Sort  (cost=285913.35..288996.38
                           ↪ rows=1233214 width=89) (actual
                           ↪ time=24897.680..26761.477 rows
                           ↪ =1229487 loops=3)
                              Sort Key: "Publ"."pubID"
                              Sort Method: external merge  Disk:
                                 ↪  121472kB
                              Worker 0:  Sort Method: external
                                 ↪ merge  Disk: 121472kB
                              Worker 1:  Sort Method: external
                                 ↪ merge  Disk: 121480kB
                              ->  Seq Scan on "Publ"  (cost
                                 ↪ =0.00..34694.14 rows=1233214
```

```
                                        ↪  width=89) (actual time
                                        ↪ =0.533..590.035 rows=1233214
                                        ↪  loops=3)
                    Planning Time: 3.912 ms
                    Execution Time: 45444.965 ms


Q2:
Gather  (cost=169073.54..171642.81 rows=24 width=67) (actual time
    ↪ =5165.088..6098.805 rows=183 loops=1)
                      Workers Planned: 2
                      Workers Launched: 2
                      ->  Merge Join  (cost=168073.54..170640.41 rows=10
                          ↪ width=67) (actual time=5130.756..5936.638 rows
                          ↪ =61 loops=3)
                            Merge Cond: (("Publ"."pubID")::text = ("Auth".
                                ↪ "pubID")::text)
                            ->  Sort  (cost=102584.98..103869.58 rows
                                ↪ =513839 width=89) (actual time
                                ↪ =4280.191..4895.216 rows=409986 loops=3)
                                  Sort Key: "Publ"."pubID"
                                  Sort Method: external merge  Disk: 39328
                                      ↪ kB
                                  Worker 0:  Sort Method: external merge
                                      ↪ Disk: 42184kB
                                  Worker 1:  Sort Method: external merge
                                      ↪ Disk: 40080kB
                                  ->  Parallel Seq Scan on "Publ"  (cost
                                      ↪ =0.00..27500.39 rows=513839 width
                                      ↪ =89) (actual time=0.406..161.864
                                      ↪ rows=411071 loops=3)
                            ->  Sort  (cost=65488.56..65488.62 rows=24
                                ↪ width=23) (actual time=733.850..733.878
                                ↪ rows=183 loops=3)
                                  Sort Key: "Auth"."pubID"
                                  Sort Method: quicksort  Memory: 39kB
                                  Worker 0:  Sort Method: quicksort
                                      ↪ Memory: 39kB
                                  Worker 1:  Sort Method: quicksort
                                      ↪ Memory: 39kB
                                  ->  Seq Scan on "Auth"  (cost
                                      ↪ =0.00..65488.01 rows=24 width=23)
                                      ↪ (actual time=51.083..732.761 rows
                                      ↪ =183 loops=3)
                                        Filter: ((name)::text = 'Divesh
                                            ↪ Srivastava'::text)
                                        Rows Removed by Filter: 3095018
                    Planning Time: 0.249 ms
                    Execution Time: 6105.353 ms
```

Two non-clustering indexes (Q1/Q2):

```
Q1:
Merge Join  (cost=0.98..263785.90 rows=3095201 width=82) (actual
    ↪ time=0.011..3453.195 rows=3095201 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..73130.78 rows=1233214 width=89) (actual
                 ↪  time=0.004..659.540 rows=1233208 loops=1)
              ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                 ↪ =0.56..148957.59 rows=3095201 width=38) (
                 ↪ actual time=0.003..1077.221 rows=3095201 loops
                 ↪ =1)
            Planning Time: 0.645 ms
            Execution Time: 3531.287 ms


Q2:
Merge Join  (cost=43922.35..120060.59 rows=25 width=67) (actual
    ↪ time=742.922..1767.721 rows=183 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..73130.78 rows=1233214 width=89) (actual
                 ↪  time=0.007..585.404 rows=1229958 loops=1)
              ->  Sort  (cost=43921.92..43921.98 rows=25 width=23)
                 ↪  (actual time=612.003..612.047 rows=183 loops
                 ↪ =1)
                    Sort Key: "Auth"."pubID"
                    Sort Method: quicksort  Memory: 39kB
                    ->  Gather  (cost=1000.00..43921.34 rows=25
                       ↪ width=23) (actual time=20.274..610.446
                       ↪ rows=183 loops=1)
                          Workers Planned: 2
                          Workers Launched: 2
                          ->  Parallel Seq Scan on "Auth"  (cost
                             ↪ =0.00..42918.84 rows=10 width=23)
                             ↪ (actual time=40.490..478.516 rows
                             ↪ =61 loops=3)
                                Filter: ((name)::text = 'Divesh
                                   ↪ Srivastava'::text)
                                Rows Removed by Filter: 1031673
            Planning Time: 0.418 ms
            Execution Time: 1767.829 ms
```

Two clustering indexes (Q1/Q2):

```
Q1:
Merge Join  (cost=0.98..248801.80 rows=3095201 width=82) (actual
   ↪ time=0.017..2427.225 rows=3095201 loops=1)
             Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                ↪ ")::text)
             ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                ↪ =0.43..67009.64 rows=1233214 width=89) (actual
                ↪  time=0.006..343.469 rows=1233208 loops=1)
             ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                ↪ =0.56..140088.57 rows=3095201 width=38) (
                ↪ actual time=0.004..643.179 rows=3095201 loops
                ↪ =1)
          Planning Time: 0.774 ms
          Execution Time: 2497.417 ms


Q2:
Merge Join  (cost=43933.22..113956.37 rows=24 width=67) (actual
   ↪ time=529.077..1464.214 rows=183 loops=1)
             Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                ↪ ")::text)
             ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                ↪ =0.43..67009.64 rows=1233214 width=89) (actual
                ↪  time=0.004..454.260 rows=1229958 loops=1)
             ->  Sort  (cost=43932.79..43932.85 rows=24 width=23)
                ↪  (actual time=429.671..429.714 rows=183 loops
                ↪ =1)
                   Sort Key: "Auth"."pubID"
                   Sort Method: quicksort  Memory: 39kB
                   ->  Gather  (cost=1000.00..43932.24 rows=24
                      ↪ width=23) (actual time=45.698..428.159
                      ↪ rows=183 loops=1)
                         Workers Planned: 2
                         Workers Launched: 2
                         ->  Parallel Seq Scan on "Auth"  (cost
                            ↪ =0.00..42929.84 rows=10 width=23)
                            ↪ (actual time=18.926..352.715 rows
                            ↪ =61 loops=3)
                               Filter: ((name)::text = 'Divesh
                                  ↪ Srivastava'::text)
                               Rows Removed by Filter: 1031673
          Planning Time: 0.356 ms
          Execution Time: 1464.339 ms
```

Query plans
On server

No index (Q1/Q2):

```
Q1:
Merge Join  (cost=846625.43..906942.42 rows=3095201 width=82) (
   ↪ actual time=49829.600..64045.038 rows=3095201 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Sort  (cost=285913.35..288996.38 rows=1233214
                 ↪ width=89) (actual time=14350.912..17364.181
                 ↪ rows=1233213 loops=1)
                    Sort Key: "Publ"."pubID"
                    Sort Method: external merge  Disk: 121384kB
                    ->  Seq Scan on "Publ"  (cost=0.00..34694.14
                       ↪ rows=1233214 width=89) (actual time
                       ↪ =0.022..479.454 rows=1233214 loops=1)
              ->  Materialize  (cost=560711.47..576187.47 rows
                 ↪ =3095201 width=38) (actual time
                 ↪ =35478.670..43614.579 rows=3095201 loops=1)
                    ->  Sort  (cost=560711.47..568449.47 rows
                       ↪ =3095201 width=38) (actual time
                       ↪ =35478.663..42953.844 rows=3095201 loops
                       ↪ =1)
                          Sort Key: "Auth"."pubID"
                          Sort Method: external merge  Disk:
                             ↪ 148128kB
                          ->  Seq Scan on "Auth"  (cost
                             ↪ =0.00..57750.01 rows=3095201 width
                             ↪ =38) (actual time=0.038..694.786
                             ↪ rows=3095201 loops=1)
            Planning time: 0.518 ms
            Execution time: 64410.432 ms



Q2:
Merge Join  (cost=351402.53..357553.86 rows=24 width=67) (actual
   ↪ time=15127.144..18318.343 rows=183 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Sort  (cost=285913.35..288996.38 rows=1233214
                 ↪ width=89) (actual time=14281.937..17077.683
                 ↪ rows=1229958 loops=1)
                    Sort Key: "Publ"."pubID"
                    Sort Method: external merge  Disk: 121384kB
                    ->  Seq Scan on "Publ"  (cost=0.00..34694.14
                       ↪ rows=1233214 width=89) (actual time
                       ↪ =0.054..458.000 rows=1233214 loops=1)
              ->  Sort  (cost=65488.56..65488.62 rows=24 width=23)
                 ↪  (actual time=608.064..608.114 rows=183 loops
                 ↪ =1)
                    Sort Key: "Auth"."pubID"
                    Sort Method: quicksort  Memory: 39kB
                    ->  Seq Scan on "Auth"  (cost=0.00..65488.01
                       ↪ rows=24 width=23) (actual time
```

```
                        ↪ =53.378..607.239 rows=183 loops=1)
                           Filter: ((name)::text = 'Divesh
                              ↪ Srivastava'::text)
                           Rows Removed by Filter: 3095018
              Planning time: 0.383 ms
              Execution time: 18360.732 ms
```

Two non-clustering indexes (Q1/Q2):

```
Q1:
Merge Join  (cost=0.86..262735.88 rows=3095201 width=82) (actual
   ↪ time=0.015..5365.806 rows=3095201 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..72958.98 rows=1233214 width=89) (actual
                 ↪  time=0.006..802.445 rows=1233213 loops=1)
              ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                 ↪ =0.43..148182.48 rows=3095201 width=38) (
                 ↪ actual time=0.004..1457.815 rows=3095201 loops
                 ↪ =1)
              Planning time: 0.738 ms
              Execution time: 5638.950 ms



Q2:
Merge Join  (cost=65488.96..141352.26 rows=23 width=67) (actual
   ↪ time=726.535..1972.272 rows=183 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..72958.98 rows=1233214 width=89) (actual
                 ↪  time=0.006..742.426 rows=1229958 loops=1)
              ->  Sort  (cost=65488.53..65488.59 rows=23 width=23)
                 ↪  (actual time=617.232..617.306 rows=183 loops
                 ↪ =1)
                    Sort Key: "Auth"."pubID"
                    Sort Method: quicksort  Memory: 39kB
                    ->  Seq Scan on "Auth"  (cost=0.00..65488.01
                       ↪ rows=23 width=23) (actual time
                       ↪ =52.832..616.453 rows=183 loops=1)
                          Filter: ((name)::text = 'Divesh
                             ↪ Srivastava'::text)
                          Rows Removed by Filter: 3095018
              Planning time: 0.596 ms
              Execution time: 1972.349 ms
```

Two clustering indexes (Q1/Q2):

```
Q1:
Merge Join  (cost=0.86..248642.47 rows=3095201 width=82) (actual
    ↪ time=0.017..4709.750 rows=3095201 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..67009.64 rows=1233214 width=89) (actual
                 ↪  time=0.006..562.230 rows=1233213 loops=1)
              ->  Index Scan using auth_pubid_idx on "Auth"  (cost
                 ↪ =0.43..140024.45 rows=3095201 width=38) (
                 ↪ actual time=0.005..1143.752 rows=3095201 loops
                 ↪ =1)
            Planning time: 0.776 ms
            Execution time: 4992.177 ms
```

```
Q2:
Merge Join  (cost=65499.99..135427.94 rows=24 width=67) (actual
    ↪ time=682.630..1708.988 rows=183 loops=1)
              Merge Cond: (("Publ"."pubID")::text = ("Auth"."pubID
                 ↪ ")::text)
              ->  Index Scan using publ_pubid_idx on "Publ"  (cost
                 ↪ =0.43..67009.64 rows=1233214 width=89) (actual
                 ↪  time=0.006..526.286 rows=1229958 loops=1)
              ->  Sort  (cost=65499.56..65499.62 rows=24 width=23)
                 ↪  (actual time=593.923..594.007 rows=183 loops
                 ↪ =1)
                    Sort Key: "Auth"."pubID"
                    Sort Method: quicksort  Memory: 39kB
                    ->  Seq Scan on "Auth"  (cost=0.00..65499.01
                       ↪ rows=24 width=23) (actual time
                       ↪ =54.598..593.698 rows=183 loops=1)
                          Filter: ((name)::text = 'Divesh
                             ↪ Srivastava'::text)
                          Rows Removed by Filter: 3095018
            Planning time: 0.605 ms
            Execution time: 1709.057 ms
```

Discussion Discuss your observations. Are the response times expected? Why (not)?

Ohne Index: Wenn man sich die Querys ohne Indexe ansieht, sieht man, dass die zweite Abfrage viel schneller ist als die erste: Da bei einem Sort-Merge-Join die zu verknüpfenden Tabellen sortiert werden müssen, entsteht ein großer Aufwand für das Sortieren der Tabellen. Während bei beiden Querys `Publ` vollständig sortiert werden muss, muss bei der zweiten Abfrage nur ein Bruchteil von `Auth` sortiert werden. Hier werden alle Tupel, die die Bedingung nicht erfüllen, durch sequentielles Scannen der Tabelle entfernt, so dass die Größe der zweiten zu verknüpfenden Tabelle reduziert wird.

Mit zwei nicht-clusternden Indexes: Da wir jetzt zwei Indexes auf `pubID` haben, braucht der DBS die Tabellen nicht mehr zu sortieren, da er den Index für die Join-Prozedur verwenden kann, was eine enorme Zeitersparnis mit sich bringt, die durch das

Sortieren der Tabellen verschwendet würde. Beim Beantworten der zweiten Query verwendet der DBS wieder den Index auf `Publ.pubID`, aber für `Auth` entfernt er alle Zeilen, die die Bedingung nicht erfüllen, und sortiert die resultierenden Tupel, da es besser ist, dies zu tun, als die Tupel zu verknüpfen und dann alle Zeilen zu entfernen, die die Bedingung nicht erfüllen. Der Grund dafür ist, dass die DBS beim Verbinden der beiden Tabellen tatsächlich auf die Daten zugreifen müsste, da die Indexes nur auf `pubID` stehen.

Mit zwei Clustering-Indexes: Die Strategien sind die gleichen wie bei der Ausführung der Querys mit zwei Nicht-Clustering-Indexes. Es sind kleine Beschleunigungen für die erste und für die zweite Abfrage im Vergleich zu der zuvor erwähnten Varianten zu sehen. Diese kleinen Beschleunigungen könnten durch die fehlende Seitenfragmentierung auf der Platte erklärt werden, die zu einem effizienteren Datenscan-Prozess führt.

Hash Join

Response times
On localhost

Indexes	Response time Q1 [ms]	Response time [ms] Q2
no index	17885.086 ms	612.417 ms

Response times
On server

Indexes	Response time Q1 [ms]	Response time [ms] Q2
no index	5839.772 ms	1211.506 ms

Query plans
On localhost

No Index (Q1/Q2):

```
Q1:
Hash Join  (cost=36015.72..10094673.81 rows=284647016 width=632) (
    ↪ actual time=1063.757..17809.358 rows=3095201 loops=1)
            Hash Cond: (("Auth"."pubID")::text = ("Publ"."pubID"
                ↪ )::text)
            ->  Seq Scan on "Auth"  (cost=0.00..31889.62 rows
                ↪ =509162 width=392) (actual time
                ↪ =0.063..1482.605 rows=3095201 loops=1)
            ->  Hash  (cost=23480.10..23480.10 rows=111810 width
                ↪ =792) (actual time=1039.683..1039.683 rows
                ↪ =1233214 loops=1)
                  Buckets: 8192 (originally 8192)  Batches: 64 (
                      ↪ originally 32)  Memory Usage: 4033kB
                  ->  Seq Scan on "Publ"  (cost=0.00..23480.10
                      ↪ rows=111810 width=792) (actual time
                      ↪ =0.036..392.080 rows=1233214 loops=1)
        Planning Time: 0.235 ms
        Execution Time: 17885.086 ms
```

```
Q2:
Hash Join  (cost=30736.31..104368.78 rows=1423341 width=516) (
   ↪ actual time=329.746..612.326 rows=183 loops=1)
            Hash Cond: (("Publ"."pubID")::text = ("Auth"."pubID"
               ↪ )::text)
            ->  Seq Scan on "Publ"  (cost=0.00..23480.10 rows
               ↪ =111810 width=792) (actual time=0.036..188.583
               ↪  rows=1233214 loops=1)
            ->  Hash  (cost=30704.49..30704.49 rows=2546 width
               ↪ =276) (actual time=304.271..304.271 rows=183
               ↪ loops=1)
                  Buckets: 4096  Batches: 1  Memory Usage: 43kB
                  ->  Gather  (cost=1000.00..30704.49 rows=2546
                     ↪ width=276) (actual time=43.678..304.145
                     ↪ rows=183 loops=1)
                        Workers Planned: 2
                        Workers Launched: 2
                        ->  Parallel Seq Scan on "Auth"  (cost
                           ↪ =0.00..29449.89 rows=1061 width
                           ↪ =276) (actual time=37.093..232.819
                           ↪  rows=61 loops=3)
                              Filter: ((name)::text = 'Divesh
                                 ↪ Srivastava'::text)
                              Rows Removed by Filter: 1031673
          Planning Time: 0.079 ms
          Execution Time: 612.417 ms
```

Query plans
On server

No Index (Q1/Q2):

```
Q1:
Hash Join  (cost=36015.72..10094673.81 rows=284647016 width=632) (
   ↪ actual time=1330.803..5561.947 rows=3095201 loops=1)
            Hash Cond: (("Auth"."pubID")::text = ("Publ"."pubID"
               ↪ )::text)
            ->  Seq Scan on "Auth"  (cost=0.00..31889.62 rows
               ↪ =509162 width=392) (actual time=0.171..853.502
               ↪  rows=3095201 loops=1)
            ->  Hash  (cost=23480.10..23480.10 rows=111810 width
               ↪ =792) (actual time=1330.284..1330.284 rows
               ↪ =1233214 loops=1)
                  Buckets: 8192 (originally 8192)  Batches: 64 (
                     ↪ originally 32)  Memory Usage: 4033kB
                  ->  Seq Scan on "Publ"  (cost=0.00..23480.10
                     ↪ rows=111810 width=792) (actual time
                     ↪ =0.020..629.623 rows=1233214 loops=1)
          Planning time: 0.173 ms
          Execution time: 5839.772 ms
```

```
Q2:
Hash Join  (cost=33194.35..106826.82 rows=1423341 width=516) (
   ↪ actual time=698.197..1211.427 rows=183 loops=1)
            Hash Cond: (("Publ"."pubID")::text = ("Auth"."pubID"
               ↪ )::text)
            ->  Seq Scan on "Publ"  (cost=0.00..23480.10 rows
               ↪ =111810 width=792) (actual time=0.163..301.994
               ↪  rows=1233214 loops=1)
            ->  Hash  (cost=33162.53..33162.53 rows=2546 width
               ↪ =276) (actual time=653.237..653.237 rows=183
               ↪ loops=1)
                  Buckets: 4096  Batches: 1  Memory Usage: 43kB
                  ->  Seq Scan on "Auth"  (cost=0.00..33162.53
                     ↪ rows=2546 width=276) (actual time
                     ↪ =54.074..653.016 rows=183 loops=1)
                        Filter: ((name)::text = 'Divesh
                           ↪ Srivastava'::text)
                        Rows Removed by Filter: 3095018
          Planning time: 0.161 ms
          Execution time: 1211.506 ms
```

Discussion What do you think about the response time of the hash index vs. the response times of sort-merge and index nested loop join for each of the queries? Explain.

Bei Query 1 ist die Hash-Verknüpfung etwa eine Größenordnung schneller als der Sortier-und-Zusammenführungs-Ansatz (sie nimmt nur 9% der Zeit in Anspruch, die für das Sortieren und Zusammenführen benötigt wird). Er ist auch deutlich schneller als der indexierte Nested Loop Join (benötigt nur 33% der Zeit, die der Nested Loop Join benötigt).

Bei Query 2 ist die Hash-Verknüpfung wieder etwa eine Größenordnung schneller als der Sortier-und-Zusammenführungs-Ansatz (die Sortier-und-Zusammenführung dauert nur 6% der Zeit, die für die Sortierung benötigt wird). Im Gegenteil, er ist wesentlich langsamer als der indexierte Nested Loop Join (benötigt 192% der Zeit, die der Nested Loop Join benötigt).

Time Spent on this Assignment

Time in hours per person: **10**

References

Important: Reference your information sources!

https://docs.teradata.com/reader/8mHBBLGP88~HK9Auie2QvQ/YdC7QK_n7uWn~NIv7J6BTw

https://dbmstutorials.com/teradata/teradata-join-strategies.html

https://www.enterprisedb.com/de/blog/hash-indexes-are-faster-btree-indexes