Schema 1

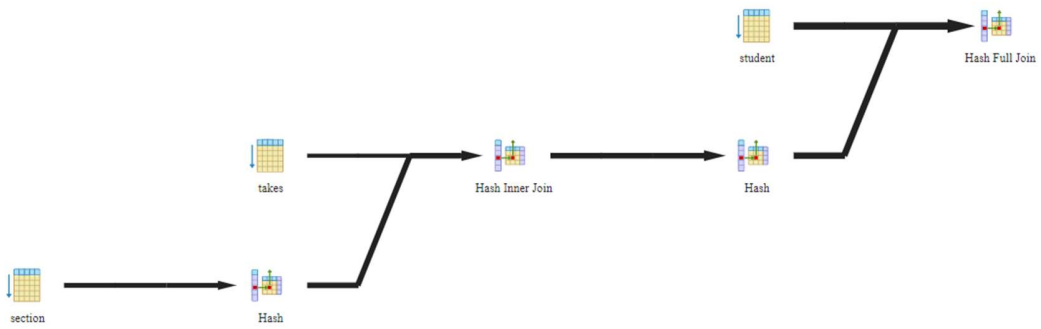
Query 1:

"Display a list of all students in the CSEN department, along with the course sections, if any, that they
have taken in Semester 1 2019; all course sections from Spring 2019 must be displayed, even if no student from the CSEN department has taken the course section.

```
select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester = 1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:



And the analysis output was as the following:

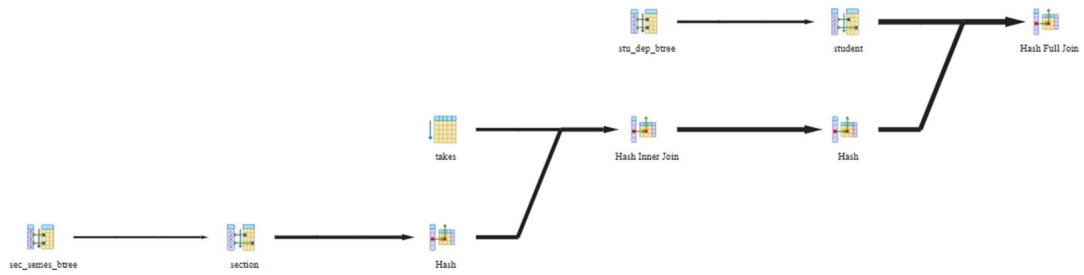
"Hash Full Join (cost=129.14..1563.84 rows=1243 width=67) (actual time=1.508..12.492 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Seq Scan on student (cost=0.00..1430.00 rows=1243 width=27) (actual time=0.030..10.564 rows=1200 loops=1)"
"   Filter: ((department)::text = 'CSEN'::text)"
"   Rows Removed by Filter: 70800"

" -> Hash (cost=126.01..126.01 rows=250 width=40) (actual time=1.465..1.469 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 26kB"
" -> Hash Join (cost=71.13..126.01 rows=250 width=40) (actual time=0.672..1.380 rows=250 loops=1)"
" Hash Cond: (t.section_id = s.section_id)"
" -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.023..0.286 rows=3000 loops=1)"
" -> Hash (cost=68.00..68.00 rows=250 width=28) (actual time=0.632..0.633 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 23kB"
" -> Seq Scan on section s (cost=0.00..68.00 rows=250 width=28) (actual time=0.030..0.569 rows=250 loops=1)"
" Filter: ((semester = 1) AND (year = 2019))"
" Rows Removed by Filter: 2750"
"Planning Time: 0.505 ms"
"Execution Time: 12.619 ms"

B+Tree:

We created B+tree indexes on *section(semester), student(department)* and *takes(section_id)*

PgAdmin decided not to use the index on the *takes* table and the plan was as the following:

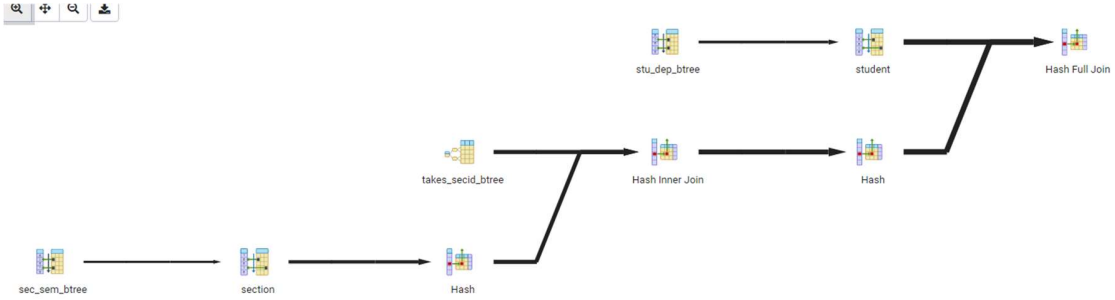


And the analysis output was as follows:

"Hash Full Join (cost=112.03..662.27 rows=1243 width=67) (actual time=2.088..4.037 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Bitmap Heap Scan on student (cost=17.93..563.46 rows=1243 width=27) (actual time=0.550..1.910 rows=1200 loops=1)"
" Recheck Cond: ((department)::text = 'CSEN'::text)"
" Heap Blocks: exact=530"
" -> Bitmap Index Scan on stu_dep_btree (cost=0.00..17.61 rows=1243 width=0) (actual time=0.345..0.345 rows=1200 loops=1)"

" Index Cond: ((department)::text = 'CSEN'::text)"
" -> Hash (cost=90.98..90.98 rows=250 width=40) (actual time=1.516..1.519 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 26kB"
" -> Hash Join (cost=36.09..90.98 rows=250 width=40) (actual time=0.238..1.385 rows=250 loops=1)"
" Hash Cond: (t.section_id = s.section_id)"
" -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.018..0.423 rows=3000 loops=1)"
" -> Hash (cost=32.97..32.97 rows=250 width=28) (actual time=0.197..0.199 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 23kB"
" -> Bitmap Heap Scan on section s (cost=6.22..32.97 rows=250 width=28) (actual time=0.045..0.150 rows=250 loops=1)"
" Recheck Cond: (semester = 1)"
" Filter: (year = 2019)"
" Heap Blocks: exact=23"
" -> Bitmap Index Scan on sec_semes_btree (cost=0.00..6.16 rows=250 width=0) (actual time=0.029..0.029 rows=250 loops=1)"
" Index Cond: (semester = 1)"
"Planning Time: 0.642 ms"
"Execution Time: 4.339 ms"

Then we disabled seqscan flag to force Postgres to use our index and the result was as the following:



And the analysis output was as follows :

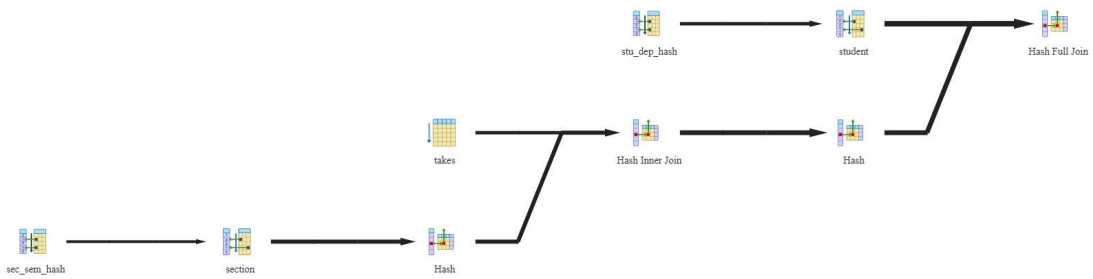
"Hash Full Join (cost=174.31..724.55 rows=1243 width=67) (actual time=2.165..4.026 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Bitmap Heap Scan on student (cost=17.93..563.46 rows=1243 width=27) (actual time=0.329..1.645 rows=1200 loops=1)"
" Recheck Cond: ((department)::text = 'CSEN'::text)"
" Heap Blocks: exact=530"

" -> Bitmap Index Scan on stu_dep_btree (cost=0.00..17.61 rows=1243 width=0) (actual time=0.223..0.224 rows=1200 loops=1)"
" Index Cond: ((department)::text = 'CSEN'::text)"
" -> Hash (cost=153.26..153.26 rows=250 width=40) (actual time=1.802..1.807 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 26kB"
" -> Hash Join (cost=36.37..153.26 rows=250 width=40) (actual time=0.465..1.699 rows=250 loops=1)"
" Hash Cond: (t.section_id = s.section_id)"
" -> Index Scan using takes_secid_btree on takes t (cost=0.28..109.28 rows=3000 width=12) (actual time=0.039..0.727 rows=3000 loops=1)"
" -> Hash (cost=32.97..32.97 rows=250 width=28) (actual time=0.399..0.402 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 23kB"
" -> Bitmap Heap Scan on section s (cost=6.22..32.97 rows=250 width=28) (actual time=0.115..0.311 rows=250 loops=1)"
" Recheck Cond: (semester = 1)"
" Filter: (year = 2019)"
" Heap Blocks: exact=23"
" -> Bitmap Index Scan on sec_sem_btree (cost=0.00..6.16 rows=250 width=0) (actual time=0.098..0.098 rows=250 loops=1)"
" Index Cond: (semester = 1)"
"Planning Time: 0.850 ms"
"Execution Time: 4.400 ms"

<u>Explanation: the engine decided not to use the index on takes(section_id) because of the full outer join. All the rows in the takes table will be checked so there is no need for an index and sequential scan is the best technique.</u>

Hash:

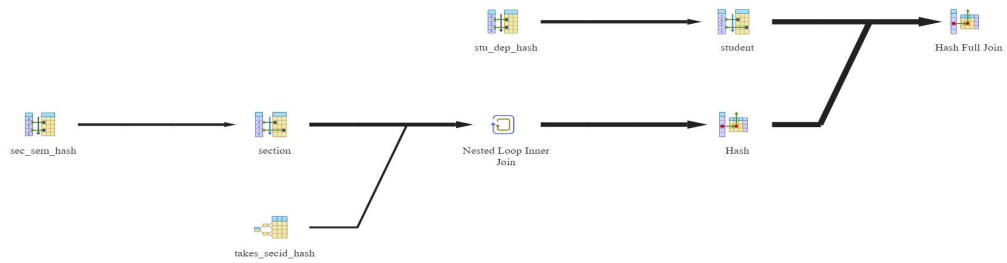
We created Hash indexes on *section(semester), student(department)* and *takes(section_id)* PgAdmin decided not to use the index on the *takes* table and the plan was as the following:



And the analysis output was as the following:

"Hash Full Join (cost=135.46..685.70 rows=1243 width=67) (actual time=1.448..3.382 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Bitmap Heap Scan on student (cost=37.63..583.17 rows=1243 width=27) (actual time=0.295..1.730 rows=1200 loops=1)"
"   Recheck Cond: ((department)::text = 'CSEN'::text)"
"   Heap Blocks: exact=530"
"   -> Bitmap Index Scan on stu_dep_hash (cost=0.00..37.32 rows=1243 width=0) (actual time=0.186..0.186 rows=1200 loops=1)"
"   Index Cond: ((department)::text = 'CSEN'::text)"
" -> Hash (cost=94.70..94.70 rows=250 width=40) (actual time=1.133..1.141 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Hash Join (cost=39.81..94.70 rows=250 width=40) (actual time=0.267..1.035 rows=250 loops=1)"
"   Hash Cond: (t.section_id = s.section_id)"
"   -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.017..0.304 rows=3000 loops=1)"
"   -> Hash (cost=36.69..36.69 rows=250 width=28) (actual time=0.231..0.233 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 23kB"
"   -> Bitmap Heap Scan on section s (cost=9.94..36.69 rows=250 width=28) (actual time=0.060..0.174 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Filter: (year = 2019)"
"   Heap Blocks: exact=23"
"   -> Bitmap Index Scan on sec_sem_hash (cost=0.00..9.88 rows=250 width=0) (actual time=0.045..0.045 rows=250 loops=1)"
"   Index Cond: (semester = 1)"
"Planning Time: 0.615 ms"
"Execution Time: 3.607 ms"

Then we disabled seq scan to force Postgres to use the created index and it created a pkey index on *takes* table so we disabled it and the plan was:



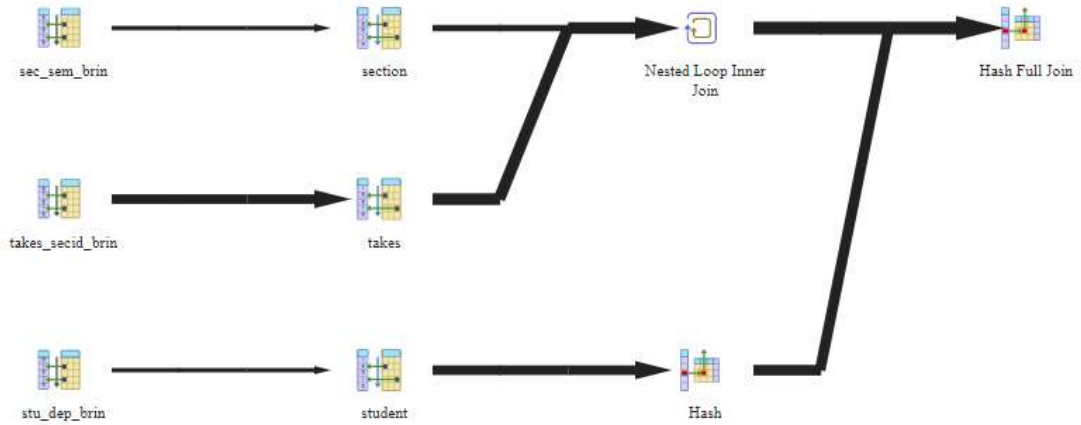
And the analysis output was as the following:

"Hash Full Join (cost=224.32..774.56 rows=1243 width=67) (actual time=2.258..4.819 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Bitmap Heap Scan on student (cost=37.63..583.17 rows=1243 width=27) (actual time=0.477..2.410 rows=1200 loops=1)"
"   Recheck Cond: ((department)::text = 'CSEN'::text)"
"   Heap Blocks: exact=530"
"   -> Bitmap Index Scan on stu_dep_hash (cost=0.00..37.32 rows=1243 width=0) (actual time=0.317..0.317 rows=1200 loops=1)"
"   Index Cond: ((department)::text = 'CSEN'::text)"
" -> Hash (cost=183.56..183.56 rows=250 width=40) (actual time=1.753..1.765 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Nested Loop (cost=9.94..183.56 rows=250 width=40) (actual time=0.144..1.532 rows=250 loops=1)"
"   -> Bitmap Heap Scan on section s (cost=9.94..36.69 rows=250 width=28) (actual time=0.096..0.382 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Filter: (year = 2019)"
"   Heap Blocks: exact=23"
"   -> Bitmap Index Scan on sec_sem_hash (cost=0.00..9.88 rows=250 width=0) (actual time=0.069..0.069 rows=250 loops=1)"
"   Index Cond: (semester = 1)"
"   -> Index Scan using takes_secid_hash on takes t (cost=0.00..0.58 rows=1 width=12) (actual time=0.003..0.003 rows=1 loops=250)"
"   Index Cond: (section_id = s.section_id)"
"Planning Time: 0.896 ms"
"Execution Time: 5.223 ms"

Explanation: The cost before disabling the seq scan was better for the same reason as the B+Tree query where the takes table is fully scanned and there is no need for any index.

BRIN

We created BRIN indexes on section(semester), student(department), and takes(section_id) PgAdmin decided not to use any of them and preferred to use seqscan over the BRIN index so we disabled seqscan and it used all the BRIN indexes and the result was as follows :



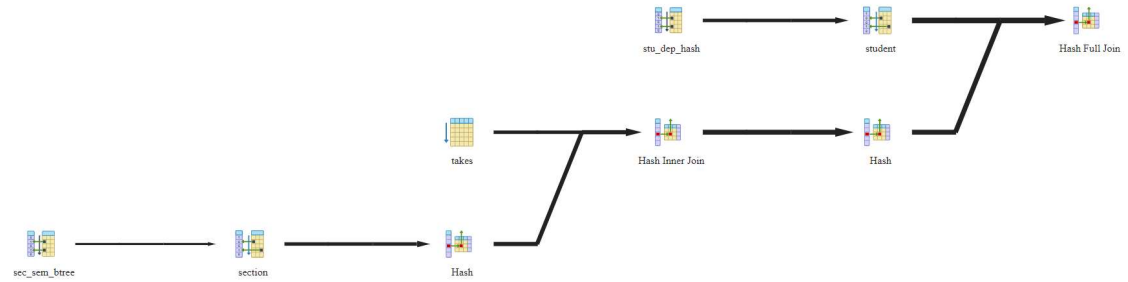
"Hash Full Join (cost=4470.14..761924.32 rows=1243 width=67) (actual time=14.657..139.923 rows=1450 loops=1)"
" Hash Cond: (t.student_id = student.id)"
" -> Nested Loop (cost=3012.13..760465.66 rows=250 width=40) (actual time=0.124..124.908 rows=250 loops=1)"
"   -> Bitmap Heap Scan on section s (cost=12.09..80.09 rows=250 width=28) (actual time=0.066..0.651 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Rows Removed by Index Recheck: 2750"
"   Filter: (year = 2019)"
"   Heap Blocks: lossy=23"
"   -> Bitmap Index Scan on sec_sem_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.055..0.056 rows=230 loops=1)"
"   Index Cond: (semester = 1)"
"   -> Bitmap Heap Scan on takes t (cost=3000.03..3041.53 rows=1 width=12) (actual time=0.254..0.485 rows=1 loops=250)"
"   Recheck Cond: (section_id = s.section_id)"
"   Rows Removed by Index Recheck: 2999"
"   Heap Blocks: lossy=4250"
"   -> Bitmap Index Scan on takes_secid_brin (cost=0.00..3000.03 rows=3000 width=0) (actual time=0.016..0.016 rows=170 loops=250)"
"   Index Cond: (section_id = s.section_id)"
" -> Hash (cost=1442.47..1442.47 rows=1243 width=27) (actual time=14.511..14.512 rows=1200 loops=1)"

" Buckets: 2048 Batches: 1 Memory Usage: 91kB"
" -> Bitmap Heap Scan on student (cost=12.47..1442.47 rows=1243 width=27) (actual time=0.097..14.212 rows=1200 loops=1)"
" Recheck Cond: ((department)::text = 'CSEN'::text)"
" Rows Removed by Index Recheck: 70800"
" Heap Blocks: lossy=530"
" -> Bitmap Index Scan on stu_dep_brin (cost=0.00..12.16 rows=72000 width=0) (actual time=0.071..0.071 rows=5300 loops=1)"
" Index Cond: ((department)::text = 'CSEN'::text)"
"Planning Time: 0.709 ms"
"Execution Time: 140.393 ms"

Explanation: since there is no aggregate functions or query with exact value or range ona very small percentage of the stored data in the table so BRIN is very costly and seqscan is better in this case

Mixed:

We created Hash index on *student(department)* and B+Tree index on *section(semester)* and the plan was as the following:



And the analysis output was as the following:

"Hash Full Join (cost=131.74..681.98 rows=1243 width=67) (actual time=1.412..3.117 rows=1450 loops=1)"
" Hash Cond: (student.id = t.student_id)"
" -> Bitmap Heap Scan on student (cost=37.63..583.17 rows=1243 width=27) (actual time=0.322..1.578 rows=1200 loops=1)"
" Recheck Cond: ((department)::text = 'CSEN'::text)"
" Heap Blocks: exact=530"
" -> Bitmap Index Scan on stu_dep_hash (cost=0.00..37.32 rows=1243 width=0) (actual time=0.209..0.209 rows=1200 loops=1)"
" Index Cond: ((department)::text = 'CSEN'::text)"
" -> Hash (cost=90.98..90.98 rows=250 width=40) (actual time=1.074..1.078 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 26kB"
" -> Hash Join (cost=36.09..90.98 rows=250 width=40) (actual time=0.249..0.988 rows=250 loops=1)"

" Hash Cond: (t.section_id = s.section_id)"
" -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.018..0.296 rows=3000 loops=1)"
" -> Hash (cost=32.97..32.97 rows=250 width=28) (actual time=0.216..0.218 rows=250 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 23kB"
" -> Bitmap Heap Scan on section s (cost=6.22..32.97 rows=250 width=28) (actual time=0.048..0.170 rows=250 loops=1)"
" Recheck Cond: (semester = 1)"
" Filter: (year = 2019)"
" Heap Blocks: exact=23"
" -> Bitmap Index Scan on sec_sem_btree (cost=0.00..6.16 rows=250 width=0) (actual time=0.034..0.035 rows=250 loops=1)"
" Index Cond: (semester = 1)"
"Planning Time: 0.446 ms"
"Execution Time: 3.370 ms"

Conclusion:

We saw that the query is not performing well with BRIN and its best performance was when we created B+tree indexes and Hash indexes so we mixed them up but using only B+tree indexes was slightly better.

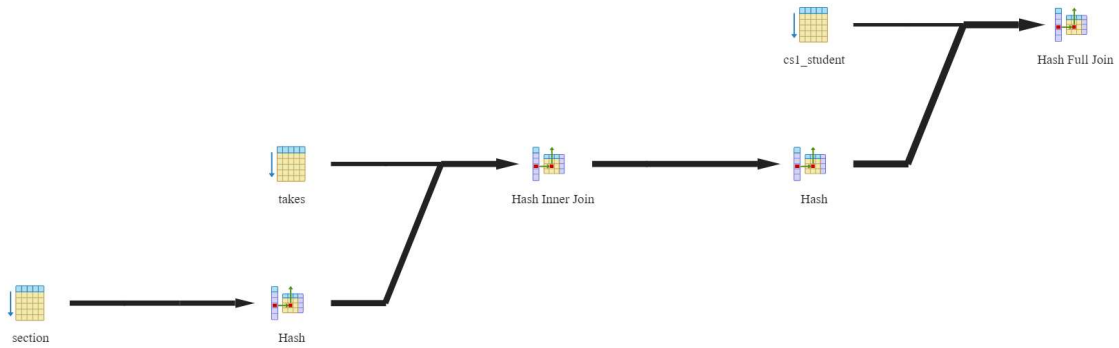
Query 1 Optimized:

We have created a Materialized view over the most expensive select subquery of the query and used it in the optimized query

```
create materialized view CS1_student as

select *
from student
where
department = 'CSEN'


select *
from CS1_student
full outer join
(select *
from takes t  inner join section s
on t.section_id = s.section_id
where semester = 1
and
year = 2019)  as sem1_student
on CS1_student.id = sem1_student.student_id;
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:

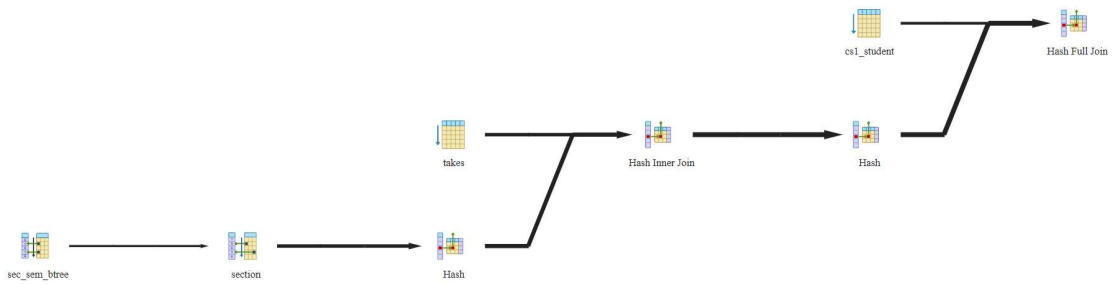


And the analysis output was as the following:

"Hash Full Join (cost=129.14..155.64 rows=1200 width=66) (actual time=1.267..1.836 rows=1450 loops=1)"
" Hash Cond: (cs1_student.id = t.student_id)"
" -> Seq Scan on cs1_student (cost=0.00..21.00 rows=1200 width=26) (actual time=0.023..0.159 rows=1200 loops=1)"
" -> Hash (cost=126.01..126.01 rows=250 width=40) (actual time=1.234..1.236 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Hash Join (cost=71.13..126.01 rows=250 width=40) (actual time=0.447..1.148 rows=250 loops=1)"
"   Hash Cond: (t.section_id = s.section_id)"
"   -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.023..0.280 rows=3000 loops=1)"
"   -> Hash (cost=68.00..68.00 rows=250 width=28) (actual time=0.406..0.407 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 23kB"
"   -> Seq Scan on section s (cost=0.00..68.00 rows=250 width=28) (actual time=0.015..0.362 rows=250 loops=1)"
"   Filter: ((semester = 1) AND (year = 2019))"
"   Rows Removed by Filter: 2750"
"Planning Time: 0.402 ms"
"Execution Time: 1.939 ms"

B+Tree:

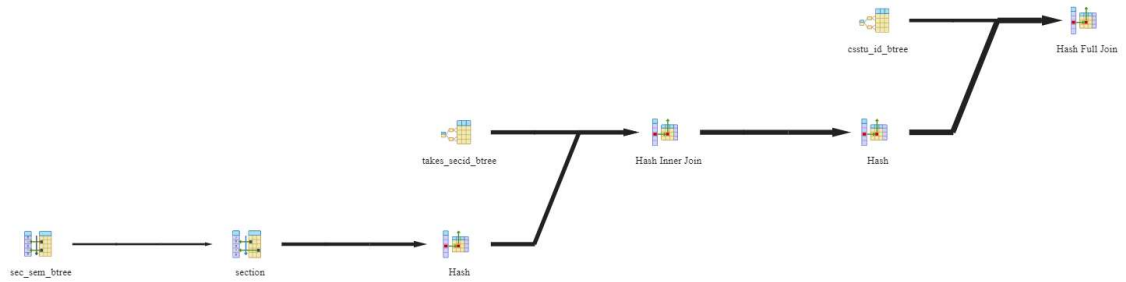
We created B+tree indexes on *section(semester), CS1_student (id)* and *takes(section_id)*
PgAdmin decided not to use the indexes but the one on the *section* table, and the plan was as the following:



And the analysis output was as follows:

"Hash Full Join (cost=97.82..124.32 rows=1200 width=66) (actual time=1.010..1.620 rows=1450 loops=1)"
" Hash Cond: (cs1_student.id = t.student_id)"
" -> Seq Scan on cs1_student (cost=0.00..21.00 rows=1200 width=26) (actual time=0.016..0.120 rows=1200 loops=1)"
" -> Hash (cost=94.70..94.70 rows=250 width=40) (actual time=0.988..0.991 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Hash Join (cost=39.81..94.70 rows=250 width=40) (actual time=0.185..0.915 rows=250 loops=1)"
"   Hash Cond: (t.section_id = s.section_id)"
"   -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.015..0.300 rows=3000 loops=1)"
"   -> Hash (cost=36.69..36.69 rows=250 width=28) (actual time=0.161..0.163 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 23kB"
"   -> Bitmap Heap Scan on section s (cost=9.94..36.69 rows=250 width=28) (actual time=0.032..0.129 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Filter: (year = 2019)"
"   Heap Blocks: exact=23"
"   -> Bitmap Index Scan on sec_sem_hash (cost=0.00..9.88 rows=250 width=0) (actual time=0.021..0.022 rows=250 loops=1)"
"   Index Cond: (semester = 1)"
"Planning Time: 0.391 ms"
"Execution Time: 1.746 ms"

Then we disabled seqscan flag to force Postgres to use our index and the result was as the following:



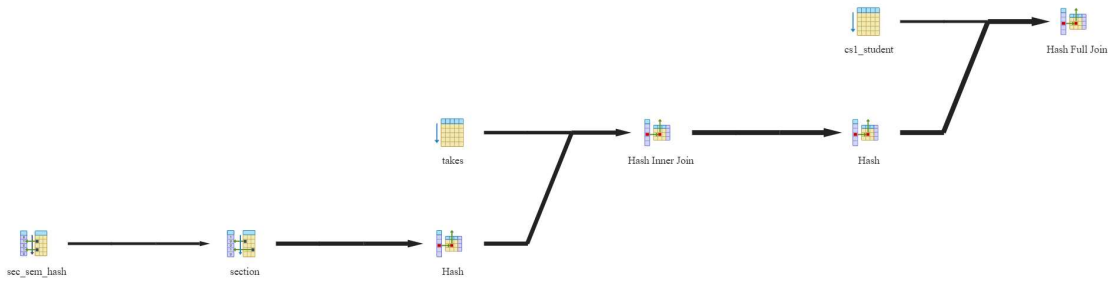
And the analysis output was as follows:

"Hash Full Join (cost=156.66..216.16 rows=1200 width=66) (actual time=1.380..2.039 rows=1450 loops=1)"
" Hash Cond: (cs1_student.id = t.student_id)"
" -> Index Scan using csstu_id_btree on cs1_student (cost=0.28..54.28 rows=1200 width=26) (actual time=0.024..0.266 rows=1200 loops=1)"
" -> Hash (cost=153.26..153.26 rows=250 width=40) (actual time=1.342..1.344 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Hash Join (cost=36.37..153.26 rows=250 width=40) (actual time=0.219..1.252 rows=250 loops=1)"
"   Hash Cond: (t.section_id = s.section_id)"
"   -> Index Scan using takes_secid_btree on takes t (cost=0.28..109.28 rows=3000 width=12) (actual time=0.020..0.622 rows=3000 loops=1)"
"   -> Hash (cost=32.97..32.97 rows=250 width=28) (actual time=0.180..0.181 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 23kB"
"   -> Bitmap Heap Scan on section s (cost=6.22..32.97 rows=250 width=28) (actual time=0.041..0.130 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Filter: (year = 2019)"
"   Heap Blocks: exact=23"
"   -> Bitmap Index Scan on sec_sem_btree (cost=0.00..6.16 rows=250 width=0) (actual time=0.028..0.028 rows=250 loops=1)"
"   Index Cond: (semester = 1)"
"Planning Time: 0.484 ms"
"Execution Time: 2.249 ms"

Explanation: Using the B+Tree without disabling the seqscan had lower cost because the B+Tree index was useless on the other tables.

Hash:

We created Hash indexes on *section(semester), CS1_student (id)* and *takes(section_id)* PgAdmin decided not to use the indexes but the one on the *section* table, and the plan was as the following:

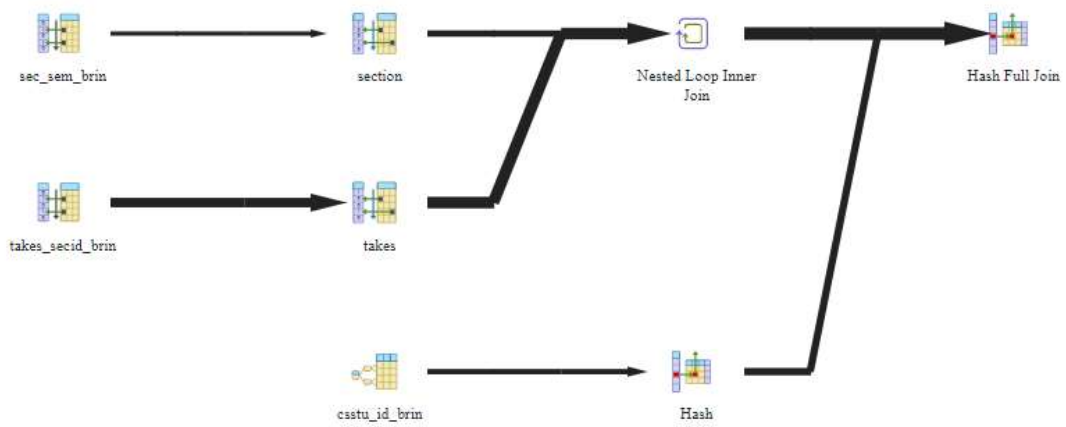


And the analysis output was as follows:

"Hash Full Join (cost=97.82..124.32 rows=1200 width=66) (actual time=1.041..1.568 rows=1450 loops=1)"
" Hash Cond: (cs1_student.id = t.student_id)"
" -> Seq Scan on cs1_student (cost=0.00..21.00 rows=1200 width=26) (actual time=0.015..0.120 rows=1200 loops=1)"
" -> Hash (cost=94.70..94.70 rows=250 width=40) (actual time=1.014..1.016 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 26kB"
"   -> Hash Join (cost=39.81..94.70 rows=250 width=40) (actual time=0.210..0.921 rows=250 loops=1)"
"   Hash Cond: (t.section_id = s.section_id)"
"   -> Seq Scan on takes t (cost=0.00..47.00 rows=3000 width=12) (actual time=0.016..0.281 rows=3000 loops=1)"
"   -> Hash (cost=36.69..36.69 rows=250 width=28) (actual time=0.185..0.186 rows=250 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 23kB"
"   -> Bitmap Heap Scan on section s (cost=9.94..36.69 rows=250 width=28) (actual time=0.038..0.150 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Filter: (year = 2019)"
"   Heap Blocks: exact=23"
"   -> Bitmap Index Scan on sec_sem_hash (cost=0.00..9.88 rows=250 width=0) (actual time=0.026..0.026 rows=250 loops=1)"
"   Index Cond: (semester = 1)"
"Planning Time: 0.379 ms"
"Execution Time: 1.681 ms"

BRIN

We created BRIN indexes on *section(semester), CS1_student (id)* and *takes(section_id)* PgAdmin decided not to use any of them and preferred to use seqscan over the BRIN index so we disabled seqscan and it used all the BRIN indexes and the result was as follows :



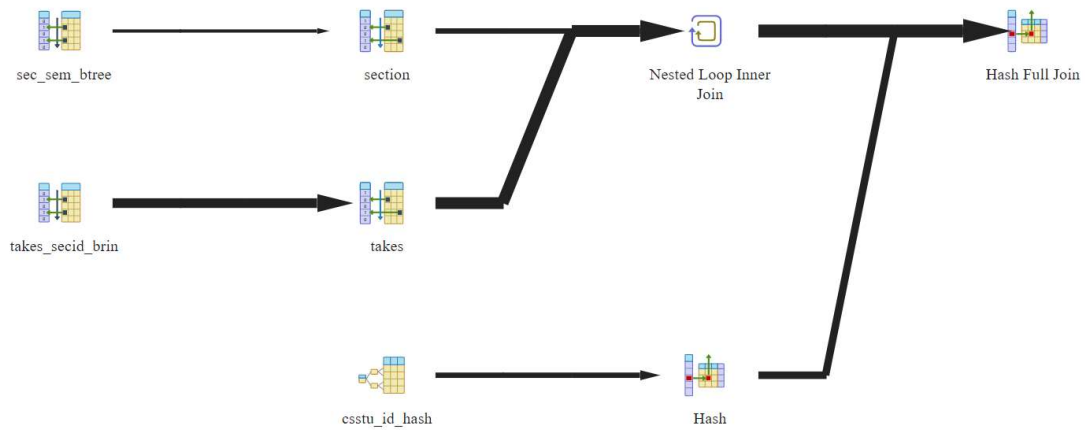
And the analysis output was as follows:

"Hash Full Join (cost=3081.40..760536.87 rows=1200 width=66) (actual time=0.608..122.334 rows=1450 loops=1)"
" Hash Cond: (t.student_id = cs1_student.id)"
" -> Nested Loop (cost=3012.13..760465.66 rows=250 width=40) (actual time=0.097..121.369 rows=250 loops=1)"
"   -> Bitmap Heap Scan on section s (cost=12.09..80.09 rows=250 width=28) (actual time=0.047..0.589 rows=250 loops=1)"
"   Recheck Cond: (semester = 1)"
"   Rows Removed by Index Recheck: 2750"
"   Filter: (year = 2019)"
"   Heap Blocks: lossy=23"
"   -> Bitmap Index Scan on sec_sem_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.036..0.036 rows=230 loops=1)"
"   Index Cond: (semester = 1)"
"   -> Bitmap Heap Scan on takes t (cost=3000.03..3041.53 rows=1 width=12) (actual time=0.245..0.471 rows=1 loops=250)"
"   Recheck Cond: (section_id = s.section_id)"
"   Rows Removed by Index Recheck: 2999"
"   Heap Blocks: lossy=4250"
"   -> Bitmap Index Scan on takes_secid_brin (cost=0.00..3000.03 rows=3000 width=0) (actual time=0.015..0.015 rows=170 loops=250)"
"   Index Cond: (section_id = s.section_id)"
" -> Hash (cost=54.28..54.28 rows=1200 width=26) (actual time=0.500..0.501 rows=1200 loops=1)"

" Buckets: 2048 Batches: 1 Memory Usage: 91kB"
" -> Index Scan using csstu_id_brin on cs1_student (cost=0.28..54.28 rows=1200 width=26) (actual time=0.018..0.261 rows=1200 loops=1)"
"Planning Time: 0.630 ms"
"Execution Time: 122.763 ms"

Mixed:

We created B+Tree index on *section(semester), Hash index on CS1_student (id)* and BRIN on *takes(section_id)* and we disabled the seqscan flag and the plan was as the following:



And the analysis output was as follows:

"Hash Full Join (cost=3075.53..760489.75 rows=1200 width=66) (actual time=0.552..216.430 rows=1450 loops=1)"
" Hash Cond: (t.student_id = cs1_student.id)"
" -> Nested Loop (cost=3006.25..760418.53 rows=250 width=40) (actual time=0.076..215.222 rows=250 loops=1)"
" -> Bitmap Heap Scan on section s (cost=6.22..32.97 rows=250 width=28) (actual time=0.029..0.282 rows=250 loops=1)"
" Recheck Cond: (semester = 1)"
" Filter: (year = 2019)"
" Heap Blocks: exact=23"
" -> Bitmap Index Scan on sec_sem_btree (cost=0.00..6.16 rows=250 width=0) (actual time=0.016..0.017 rows=250 loops=1)"
" Index Cond: (semester = 1)"
" -> Bitmap Heap Scan on takes t (cost=3000.03..3041.53 rows=1 width=12) (actual time=0.343..0.841 rows=1 loops=250)"
" Recheck Cond: (section_id = s.section_id)"
" Rows Removed by Index Recheck: 2999"
" Heap Blocks: lossy=4250"

" -> Bitmap Index Scan on takes_secid_brin (cost=0.00..3000.03 rows=3000 width=0) (actual time=0.021..0.021 rows=170 loops=250)"
" Index Cond: (section_id = s.section_id)"
" -> Hash (cost=54.28..54.28 rows=1200 width=26) (actual time=0.459..0.460 rows=1200 loops=1)"
" Buckets: 2048 Batches: 1 Memory Usage: 91kB"
" -> Index Scan using csstu_id_hash on cs1_student (cost=0.28..54.28 rows=1200 width=26) (actual time=0.015..0.252 rows=1200 loops=1)"
"Planning Time: 0.444 ms"
"Execution Time: 216.936 ms"

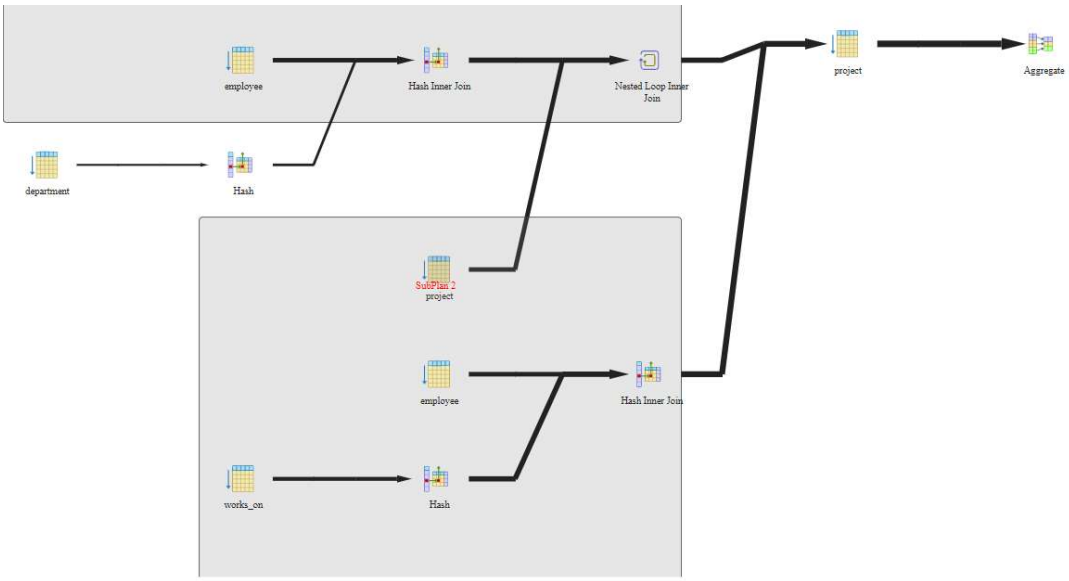
Schema 2

Query 2:

```
select distinct pnumber
from project
where pnumber  in
(select pnumber
from project, department d, employee e
where e.dno=d.dnumber
and
d.mgr_snn=ssn
and
e.lname='employee1' )
or
pnumber  in
(select pno
from works_on, employee
where essn=ssn  and lname='employee1' );
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:



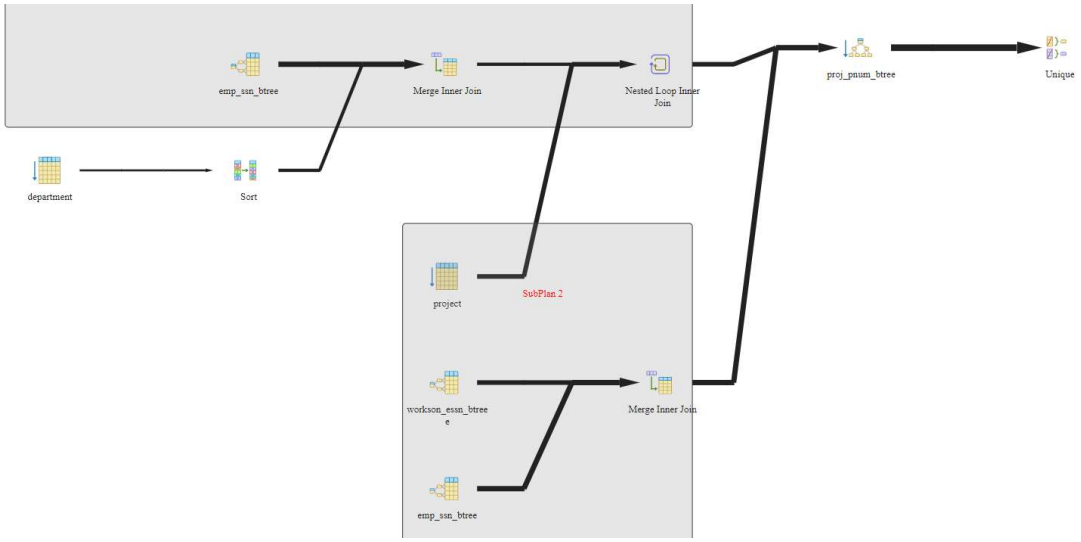
And the analysis output was:

"HashAggregate (cost=1805.13..1874.13 rows=6900 width=4) (actual time=15.632..15.863 rows=600 loops=1)"
" Group Key: project.pnumber"
" Batches: 1 Memory Usage: 241kB"
" -> Seq Scan on project (cost=1517.88..1787.88 rows=6900 width=4) (actual time=11.965..15.305 rows=600 loops=1)"

" Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
" Rows Removed by Filter: 8600"
" SubPlan 1"
" -> Nested Loop (cost=5.75..790.35 rows=613 width=4) (actual time=4.538..4.541 rows=0 loops=1)"
" -> Hash Join (cost=5.75..474.35 rows=1 width=0) (actual time=4.535..4.537 rows=0 loops=1)"
" Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
" -> Seq Scan on employee e (cost=0.00..463.00 rows=1066 width=8) (actual time=0.016..4.158 rows=1066 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 14934"
" -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.077..0.078 rows=150 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 14kB"
" -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.016..0.040 rows=150 loops=1)"
" -> Seq Scan on project project_1 (cost=0.00..224.00 rows=9200 width=4) (never executed)"
" SubPlan 2"
" -> Hash Join (cost=251.50..724.50 rows=600 width=4) (actual time=3.441..7.115 rows=600 loops=1)"
" Hash Cond: (employee.ssn = works_on.essn)"
" -> Seq Scan on employee (cost=0.00..463.00 rows=1066 width=4) (actual time=0.059..3.321 rows=1066 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 14934"
" -> Hash (cost=139.00..139.00 rows=9000 width=8) (actual time=3.206..3.207 rows=9000 loops=1)"
" Buckets: 16384 Batches: 1 Memory Usage: 480kB"
" -> Seq Scan on works_on (cost=0.00..139.00 rows=9000 width=8) (actual time=0.030..1.335 rows=9000 loops=1)"
"Planning Time: 0.566 ms"
"Execution Time: 16.408 ms"

B+Tree:

We created B+tree indexes on *employee(ssn), project(pnumber),works_on(essn) and department(dnumber)* the engine decided to use all of them except for the one on the department table and the plan was as the following:

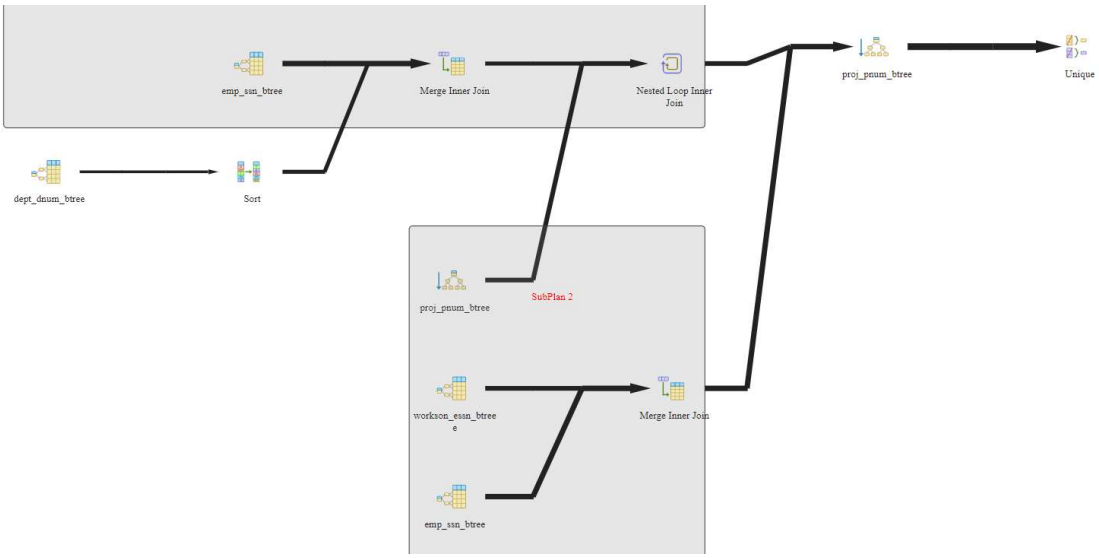


And the analysis output was as follows:

"Unique (cost=1072.72..1385.97 rows=6900 width=4) (actual time=5.973..8.683 rows=600 loops=1)"
" -> Index Only Scan using proj_pnum_btree on project (cost=1072.72..1368.72 rows=6900 width=4) (actual time=5.971..8.526 rows=600 loops=1)"
"   Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"   Rows Removed by Filter: 8600"
"   Heap Fetches: 0"
"   SubPlan 1"
"   -> Nested Loop (cost=9.21..332.95 rows=613 width=4) (actual time=0.176..0.178 rows=0 loops=1)"
"   -> Merge Join (cost=9.21..16.95 rows=1 width=0) (actual time=0.175..0.177 rows=0 loops=1)"
"   Merge Cond: (e.ssn = d.mgr_snn)"
"   Join Filter: (d.dnumber = e.dno)"
"   Rows Removed by Join Filter: 10"
"   -> Index Scan using emp_ssn_btree on employee e (cost=0.29..730.28 rows=1066 width=8) (actual time=0.023..0.065 rows=11 loops=1)"
"   Filter: (lname = 'employee1'::bpchar)"
"   Rows Removed by Filter: 154"
"   -> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.080..0.089 rows=150 loops=1)"
"   Sort Key: d.mgr_snn"
"   Sort Method: quicksort Memory: 32kB"

" -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.018..0.041 rows=150 loops=1)"
" -> Seq Scan on project p (cost=0.00..224.00 rows=9200 width=4) (never executed)"
" SubPlan 2"
" -> Merge Join (cost=0.57..736.45 rows=600 width=4) (actual time=0.037..5.507 rows=600 loops=1)"
" Merge Cond: (works_on.essn = employee.ssn)"
" -> Index Scan using workson_essn_btree on works_on (cost=0.29..295.29 rows=9000 width=8) (actual time=0.013..1.790 rows=9000 loops=1)"
" -> Index Scan using emp_ssn_btree on employee (cost=0.29..730.28 rows=1066 width=4) (actual time=0.018..2.681 rows=601 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 8414"
"Planning Time: 0.851 ms"
"Execution Time: 8.883 ms"

Then we disabled the seqscan flag so that we can force the engine to use our indexes and the plan was as the following:



And the analysis plan was as follows:

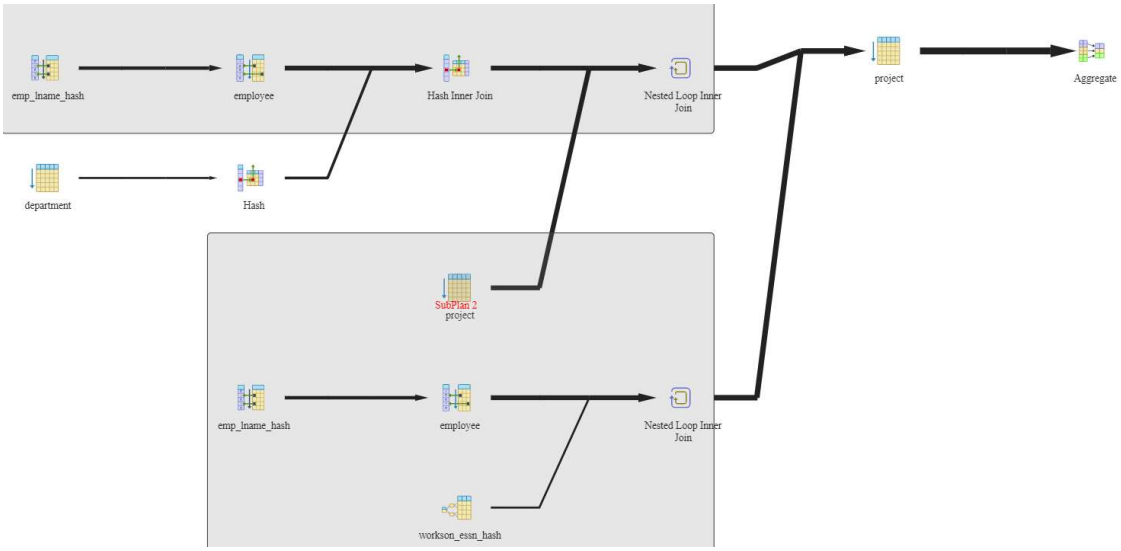
"Unique (cost=1110.90..1424.15 rows=6900 width=4) (actual time=104.192..106.711 rows=600 loops=1)"
" -> Index Only Scan using proj_pnum_btree on project (cost=1110.90..1406.90 rows=6900 width=4) (actual time=104.190..106.570 rows=600 loops=1)"
" Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
" Rows Removed by Filter: 8600"
" Heap Fetches: 0"

" SubPlan 1"
" -> Nested Loop (cost=21.39..371.13 rows=613 width=4) (actual time=0.364..0.366 rows=0 loops=1)"
" -> Merge Join (cost=21.10..28.85 rows=1 width=0) (actual time=0.363..0.364 rows=0 loops=1)"
" Merge Cond: (e.ssn = d.mgr_snn)"
" Join Filter: (d.dnumber = e.dno)"
" Rows Removed by Join Filter: 10"
" -> Index Scan using emp_ssn_btree on employee e (cost=0.29..730.28 rows=1066 width=8) (actual time=0.040..0.140 rows=11 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 154"
" -> Sort (cost=20.82..21.19 rows=150 width=8) (actual time=0.156..0.174 rows=150 loops=1)"
" Sort Key: d.mgr_snn"
" Sort Method: quicksort Memory: 32kB"
" -> Index Scan using dept_dnum_btree on department d (cost=0.14..15.39 rows=150 width=8) (actual time=0.019..0.101 rows=150 loops=1)"
" -> Index Only Scan using proj_pnum_btree on project p (cost=0.29..250.28 rows=9200 width=4) (never executed)"
" Heap Fetches: 0"
" SubPlan 2"
" -> Merge Join (cost=0.57..736.45 rows=600 width=4) (actual time=0.063..6.313 rows=600 loops=1)"
" Merge Cond: (works_on.essn = employee.ssn)"
" -> Index Scan using workson_essn_btree on works_on (cost=0.29..295.29 rows=9000 width=8) (actual time=0.023..1.932 rows=9000 loops=1)"
" -> Index Scan using emp_ssn_btree on employee (cost=0.29..730.28 rows=1066 width=4) (actual time=0.029..3.142 rows=601 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 8414"
"Planning Time: 1.330 ms"
"Execution Time: 106.959 ms"

Explanation: as it is shown before disabling the seqscan it was better and the B+Tree index on the department table made no difference but it also increased the cost of the query.

Hash:

We created Hash indexes on *employee(lname), project(pnumber),works_on(essn) and department(dnumber)* the engine decided to use only employee(lname) and works_on(essn) and the plan was as the following:



And the analysis output was as follows:

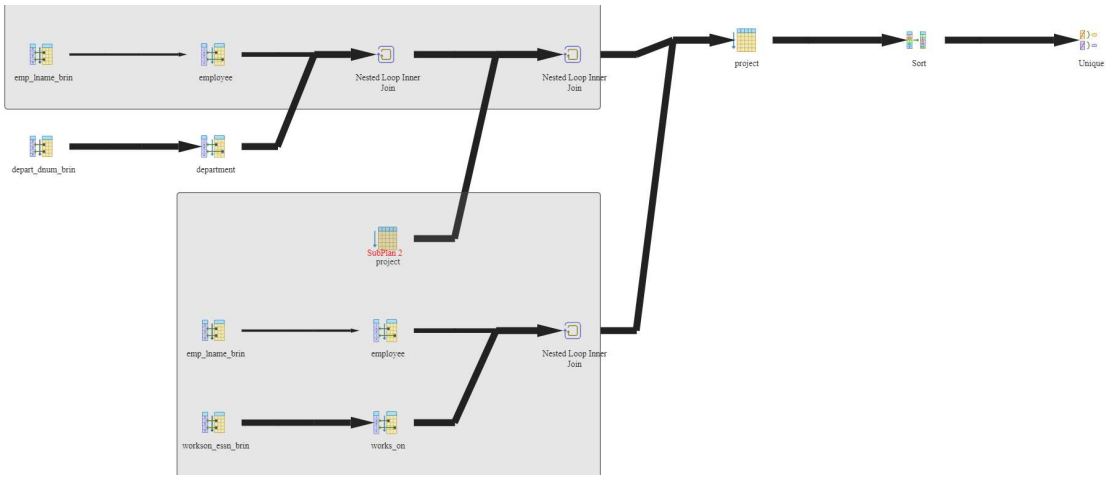
"HashAggregate (cost=1604.12..1673.12 rows=6900 width=4) (actual time=9.803..10.045 rows=600 loops=1)"
" Group Key: project.pnumber"
" Batches: 1 Memory Usage: 241kB"
" -> Seq Scan on project (cost=1316.87..1586.87 rows=6900 width=4) (actual time=5.389..9.413 rows=600 loops=1)"
"   Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"   Rows Removed by Filter: 8600"
"   SubPlan 1"
"   -> Nested Loop (cost=42.01..639.93 rows=613 width=4) (actual time=1.656..1.661 rows=0 loops=1)"
"   -> Hash Join (cost=42.01..323.93 rows=1 width=0) (actual time=1.655..1.659 rows=0 loops=1)"
"   Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
"   -> Bitmap Heap Scan on employee e (cost=36.26..312.59 rows=1066 width=8) (actual time=0.273..1.147 rows=1066 loops=1)"
"   Recheck Cond: (lname = 'employee1'::bpchar)"
"   Heap Blocks: exact=263"
"   -> Bitmap Index Scan on emp_lname_hash (cost=0.00..35.99 rows=1066 width=0) (actual time=0.179..0.180 rows=1066 loops=1)"
"   Index Cond: (lname = 'employee1'::bpchar)"
"   -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.135..0.136 rows=150 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"
"   -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.023..0.056 rows=150 loops=1)"
"   -> Seq Scan on project project_1 (cost=0.00..224.00 rows=9200 width=4) (never executed)"

" SubPlan 2"
" -> Nested Loop (cost=36.26..673.90 rows=600 width=4) (actual time=0.212..3.319 rows=600 loops=1)"
" -> Bitmap Heap Scan on employee (cost=36.26..312.59 rows=1066 width=4) (actual time=0.196..1.020 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Heap Blocks: exact=263"
" -> Bitmap Index Scan on emp_lname_hash (cost=0.00..35.99 rows=1066 width=0) (actual time=0.135..0.135 rows=1066 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
" -> Index Scan using workson_essn_hash on works_on (cost=0.00..0.33 rows=1 width=8) (actual time=0.001..0.002 rows=1 loops=1066)"
" Index Cond: (essn = employee.ssn)"
"Planning Time: 1.042 ms"
"Execution Time: 10.814 ms"

We tried to disable seqscan to force the engine to use all the hash indexes but it was significantly inefficient because there is a distinct function that needs either btree or seqscan because it depends on sorting.

BRIN

We created BRIN indexes on *employee(lname), works_on(essn)* and *department(dnumber)* the engine decided not to use any of them so we disabled the seqscan flag to test the performance of the brin index and the plan was as the following:



And the analysis output was as follows:

"Unique (cost=20027405465.29..20027405499.79 rows=6900 width=4) (actual time=1575.148..1575.400 rows=600 loops=1)"
" -> Sort (cost=20027405465.29..20027405482.54 rows=6900 width=4) (actual time=1575.146..1575.204 rows=600 loops=1)"

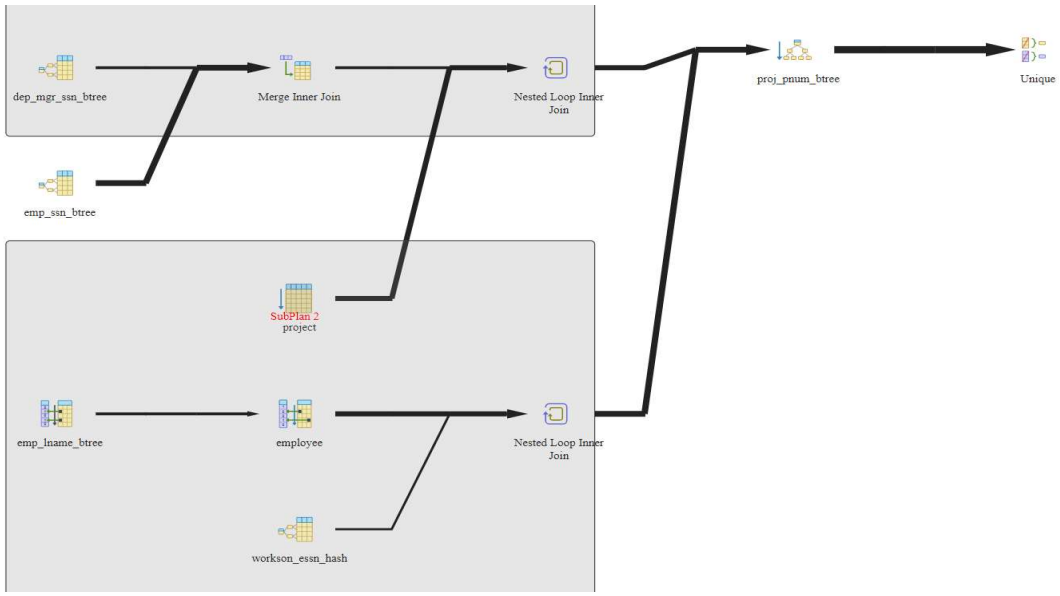
" Sort Key: project.pnumber"
" Sort Method: quicksort Memory: 53kB"
" -> Seq Scan on project (cost=20027404755.33..20027405025.33 rows=6900 width=4) (actual time=1569.605..1574.922 rows=600 loops=1)"
" Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
" Rows Removed by Filter: 8600"
" SubPlan 1"
" -> Nested Loop (cost=10000012804.39..10013643770.90 rows=613 width=4) (actual time=126.282..126.286 rows=0 loops=1)"
" -> Nested Loop (cost=12804.39..13643454.90 rows=1 width=0) (actual time=126.282..126.284 rows=0 loops=1)"
" -> Bitmap Heap Scan on employee e (cost=12.36..475.36 rows=1066 width=8) (actual time=0.062..8.808 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Rows Removed by Index Recheck: 14934"
" Heap Blocks: lossy=263"
" -> Bitmap Index Scan on emp_lname_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.047..0.047 rows=2630 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
" -> Bitmap Heap Scan on department d (cost=12792.03..12798.28 rows=1 width=8) (actual time=0.089..0.089 rows=0 loops=1066)"
" Recheck Cond: (dnumber = e.dno)"
" Rows Removed by Index Recheck: 149"
" Filter: (e.ssn = mgr_snn)"
" Rows Removed by Filter: 1"
" Heap Blocks: lossy=2132"
" -> Bitmap Index Scan on depart_dnum_brin (cost=0.00..12792.03 rows=150 width=0) (actual time=0.029..0.029 rows=20 loops=1066)"
" Index Cond: (dnumber = e.dno)"
" -> Seq Scan on project project_1 (cost=10000000000.00..10000000224.00 rows=9200 width=4) (never executed)"
" SubPlan 2"
" -> Nested Loop (cost=12804.39..13760981.40 rows=600 width=4) (actual time=0.243..1441.888 rows=600 loops=1)"
" -> Bitmap Heap Scan on employee (cost=12.36..475.36 rows=1066 width=4) (actual time=0.074..7.555 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"

" Rows Removed by Index Recheck: 14934"
" Heap Blocks: lossy=263"
" -> Bitmap Index Scan on emp_lname_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.060..0.060 rows=2630 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
" -> Bitmap Heap Scan on works_on (cost=12792.03..12908.53 rows=1 width=8) (actual time=0.649..1.324 rows=1 loops=1066)"
" Recheck Cond: (essn = employee.ssn)"
" Rows Removed by Index Recheck: 5065"
" Heap Blocks: lossy=29400"
" -> Bitmap Index Scan on workson_essn_brin (cost=0.00..12792.03 rows=9000 width=0) (actual time=0.028..0.028 rows=276 loops=1066)"
" Index Cond: (essn = employee.ssn)"
"Planning Time: 0.872 ms"
"Execution Time: 1580.322 ms"

As we can see the cost is 20027405499.79 which is too high but this is not the actual cost because when we disabled the seqscan flag some of the operation like the projection on the project table so the actual cost if we disabled the seqscan only for the indexes used it whould have been about 27405499.79

Mixed:

We created B+Tree indexes on on *employee(lname) and employee(ssn)* and Hash index on department(mgr_snn) and Hash index on works_on(essn) the plan was as the following:



And the analysis output was as follows:

"Unique (cost=996.56..1309.81 rows=6900 width=4) (actual time=2.603..5.610 rows=600 loops=1)"
" -> Index Only Scan using proj_pnum_btree on project (cost=996.56..1292.56 rows=6900 width=4) (actual time=2.602..5.448 rows=600 loops=1)"
"   Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"   Rows Removed by Filter: 8600"
"   Heap Fetches: 0"
"   SubPlan 1"
"   -> Nested Loop (cost=0.43..339.05 rows=613 width=4) (actual time=0.115..0.117 rows=0 loops=1)"
"   -> Merge Join (cost=0.43..23.05 rows=1 width=0) (actual time=0.115..0.116 rows=0 loops=1)"
"   Merge Cond: (d.mgr_snn = e.ssn)"
"   Join Filter: (d.dnumber = e.dno)"
"   Rows Removed by Join Filter: 10"
"   -> Index Scan using dep_mgr_ssn_btree on department d (cost=0.14..15.39 rows=150 width=8) (actual time=0.005..0.033 rows=150 loops=1)"
"   -> Index Scan using emp_ssn_btree on employee e (cost=0.29..730.28 rows=1066 width=8) (actual time=0.014..0.058 rows=11 loops=1)"
"   Filter: (lname = 'employee1'::bpchar)"
"   Rows Removed by Filter: 154"
"   -> Seq Scan on project project_1 (cost=0.00..224.00 rows=9200 width=4) (never executed)"
"   SubPlan 2"
"   -> Nested Loop (cost=16.55..654.19 rows=600 width=4) (actual time=0.142..2.293 rows=600 loops=1)"
"   -> Bitmap Heap Scan on employee (cost=16.55..292.87 rows=1066 width=4) (actual time=0.131..0.679 rows=1066 loops=1)"
"   Recheck Cond: (lname = 'employee1'::bpchar)"
"   Heap Blocks: exact=263"
"   -> Bitmap Index Scan on emp_lname_btree (cost=0.00..16.28 rows=1066 width=0) (actual time=0.085..0.085 rows=1066 loops=1)"
"   Index Cond: (lname = 'employee1'::bpchar)"
"   -> Index Scan using workson_essn_hash on works_on (cost=0.00..0.33 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=1066)"
"   Index Cond: (essn = employee.ssn)"
"Planning Time: 1.104 ms"
"Execution Time: 5.723 ms"

As shown in the analysis output the mixed indexes is the best one.

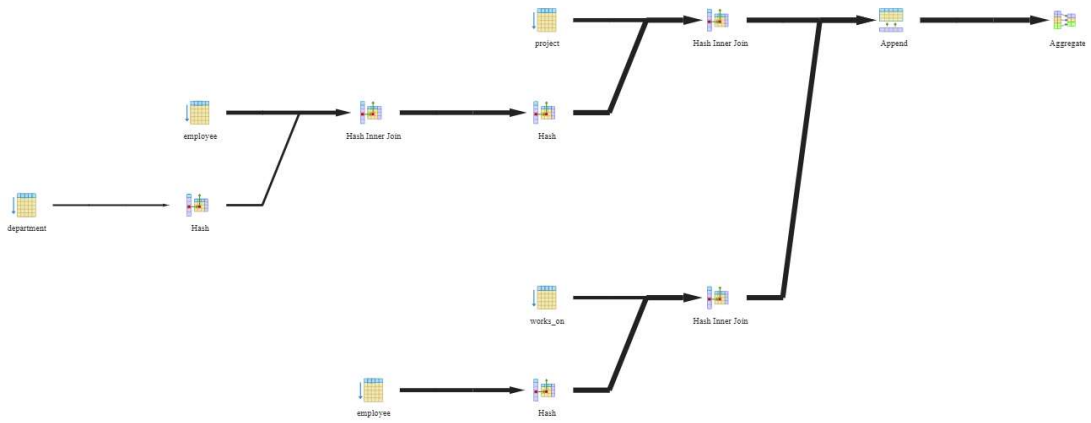
Query 2 Optimized:

Since the query is selecting from two queries and then applying distinct operator on the result we performed the two sub queries and performed set union between them

```
select pnumber
from project, department d, employee e
where e.dno=d.dnumber
and
d.mgr_snn=ssn
and
e.lname='employee1'

UNION
select pno
from works_on, employee
where essn=ssn and lname='employee1' ;
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:



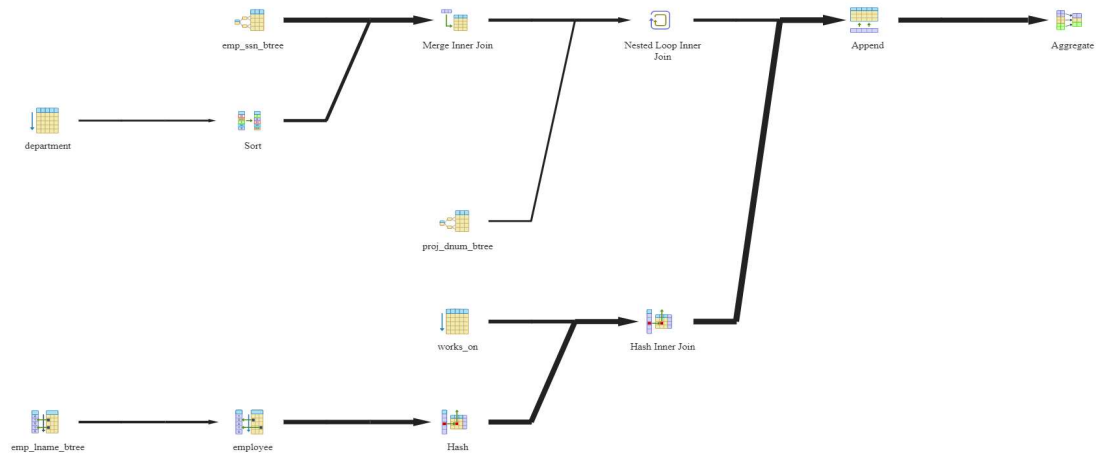
And the analysis output was as follows:

"HashAggregate (cost=1401.52..1407.56 rows=604 width=4) (actual time=8.822..8.915 rows=600 loops=1)"
" Group Key: p.pnumber"
" Batches: 1 Memory Usage: 73kB"
" -> Append (cost=476.77..1400.01 rows=604 width=4) (actual time=6.505..8.624 rows=600 loops=1)"
"   -> Hash Join (cost=476.77..735.88 rows=4 width=4) (actual time=3.273..3.276 rows=0 loops=1)"
"   Hash Cond: (p.dnumber = d.dnumber)"

" -> Seq Scan on project p (cost=0.00..224.00 rows=9200 width=8) (actual time=0.026..0.026 rows=1 loops=1)"
" -> Hash (cost=476.75..476.75 rows=1 width=8) (actual time=3.241..3.244 rows=0 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 8kB"
" -> Hash Join (cost=5.75..476.75 rows=1 width=8) (actual time=3.240..3.242 rows=0 loops=1)"
" Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
" -> Seq Scan on employee e (cost=0.00..463.00 rows=1066 width=8) (actual time=0.017..2.957 rows=1066 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 14934"
" -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.072..0.073 rows=150 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 14kB"
" -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.017..0.041 rows=150 loops=1)"
" -> Hash Join (cost=476.32..655.08 rows=600 width=4) (actual time=3.231..5.294 rows=600 loops=1)"
" Hash Cond: (works_on.essn = employee.ssn)"
" -> Seq Scan on works_on (cost=0.00..139.00 rows=9000 width=8) (actual time=0.017..0.816 rows=9000 loops=1)"
" -> Hash (cost=463.00..463.00 rows=1066 width=4) (actual time=3.197..3.198 rows=1066 loops=1)"
" Buckets: 2048 Batches: 1 Memory Usage: 54kB"
" -> Seq Scan on employee (cost=0.00..463.00 rows=1066 width=4) (actual time=0.015..2.990 rows=1066 loops=1)"
" Filter: (lname = 'employee1'::bpchar)"
" Rows Removed by Filter: 14934"
"Planning Time: 0.461 ms"
"Execution Time: 9.020 ms"

B+Tree:

We created B+tree indexes on *employee(ssn), project(dnumber),employee(lname) and* the and the plan was as the following:



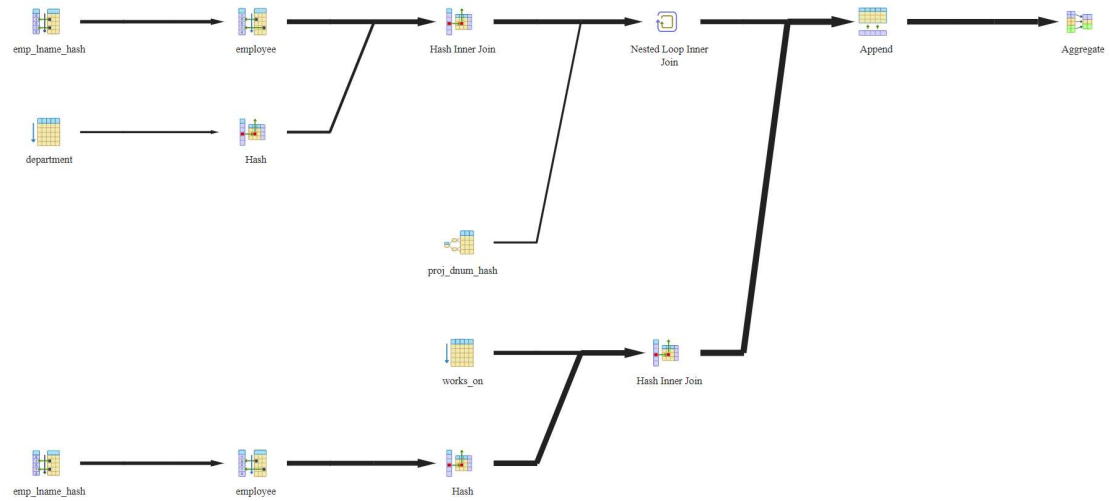
And the analysis output was as follows:

"HashAggregate (cost=515.12..521.16 rows=604 width=4) (actual time=3.445..3.537 rows=600 loops=1)"
" Group Key: p.pnumber"
" Batches: 1 Memory Usage: 73kB"
" -> Append (cost=9.49..513.61 rows=604 width=4) (actual time=1.151..3.269 rows=600 loops=1)"
"   -> Nested Loop (cost=9.49..19.60 rows=4 width=4) (actual time=0.162..0.163 rows=0 loops=1)"
"   Join Filter: (d.dnumber = p.dnumber)"
"   -> Merge Join (cost=9.21..16.95 rows=1 width=8) (actual time=0.161..0.162 rows=0 loops=1)"
"   Merge Cond: (e.ssn = d.mgr_snn)"
"   Join Filter: (d.dnumber = e.dno)"
"   Rows Removed by Join Filter: 10"
"   -> Index Scan using emp_ssn_btree on employee e (cost=0.29..730.28 rows=1066 width=8) (actual time=0.018..0.061 rows=11 loops=1)"
"   Filter: (lname = 'employee1'::bpchar)"
"   Rows Removed by Filter: 154"
"   -> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.069..0.078 rows=150 loops=1)"
"   Sort Key: d.mgr_snn"
"   Sort Method: quicksort Memory: 32kB"
"   -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.018..0.041 rows=150 loops=1)"
"   -> Index Scan using proj_dnum_btree on project p (cost=0.29..1.89 rows=61 width=8) (never executed)"
"   Index Cond: (dnumber = e.dno)"
"   -> Hash Join (cost=306.20..484.95 rows=600 width=4) (actual time=0.989..3.061 rows=600 loops=1)"

" Hash Cond: (works_on.essn = employee.ssn)"
" -> Seq Scan on works_on (cost=0.00..139.00 rows=9000 width=8) (actual time=0.015..0.845 rows=9000 loops=1)"
" -> Hash (cost=292.87..292.87 rows=1066 width=4) (actual time=0.961..0.962 rows=1066 loops=1)"
" Buckets: 2048 Batches: 1 Memory Usage: 54kB"
" -> Bitmap Heap Scan on employee (cost=16.55..292.87 rows=1066 width=4) (actual time=0.143..0.779 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Heap Blocks: exact=263"
" -> Bitmap Index Scan on emp_lname_btree (cost=0.00..16.28 rows=1066 width=0) (actual time=0.097..0.097 rows=1066 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
"Planning Time: 1.148 ms"
"Execution Time: 3.674 ms"

Hash:

We created Hash indexes on *employee(lname), project(pnumber),works_on(essn) and department(dnumber)* the engine decided to use only employee(lname) and works_on(essn) and the plan was as the following:



And the analysis output was as follows:

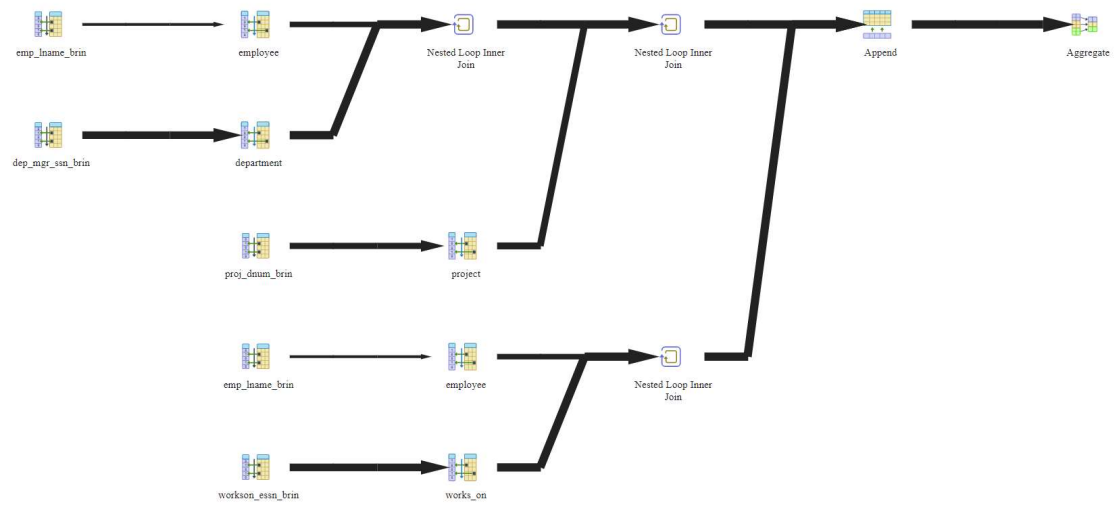
"HashAggregate (cost=844.04..850.08 rows=604 width=4) (actual time=4.137..4.235 rows=600 loops=1)"
" Group Key: p.pnumber"
" Batches: 1 Memory Usage: 73kB"
" -> Append (cost=42.01..842.53 rows=604 width=4) (actual time=1.837..3.950 rows=600 loops=1)"
" -> Nested Loop (cost=42.01..328.81 rows=4 width=4) (actual time=0.956..0.959 rows=0 loops=1)"

" Join Filter: (d.dnumber = p.dnumber)"
" -> Hash Join (cost=42.01..326.34 rows=1 width=8) (actual time=0.956..0.958 rows=0 loops=1)"
" Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
" -> Bitmap Heap Scan on employee e (cost=36.26..312.59 rows=1066 width=8) (actual time=0.164..0.661 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Heap Blocks: exact=263"
" -> Bitmap Index Scan on emp_lname_hash (cost=0.00..35.99 rows=1066 width=0) (actual time=0.113..0.113 rows=1066 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
" -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.084..0.085 rows=150 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 14kB"
" -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.021..0.043 rows=150 loops=1)"
" -> Index Scan using proj_dnum_hash on project p (cost=0.00..1.71 rows=61 width=8) (never executed)"
" Index Cond: (dnumber = e.dno)"
" -> Hash Join (cost=325.91..504.66 rows=600 width=4) (actual time=0.879..2.945 rows=600 loops=1)"
" Hash Cond: (works_on.essn = employee.ssn)"
" -> Seq Scan on works_on (cost=0.00..139.00 rows=9000 width=8) (actual time=0.017..0.850 rows=9000 loops=1)"
" -> Hash (cost=312.59..312.59 rows=1066 width=4) (actual time=0.840..0.841 rows=1066 loops=1)"
" Buckets: 2048 Batches: 1 Memory Usage: 54kB"
" -> Bitmap Heap Scan on employee (cost=36.26..312.59 rows=1066 width=4) (actual time=0.140..0.631 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Heap Blocks: exact=263"
" -> Bitmap Index Scan on emp_lname_hash (cost=0.00..35.99 rows=1066 width=0) (actual time=0.096..0.096 rows=1066 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
"Planning Time: 0.718 ms"
"Execution Time: 4.429 ms"

Hash index decreased the cost compared to the raw query but no as efficient as the B+Tree index

BRIN

We created BRIN indexes on *employee(lname), works_on(essn), department(mgr_snn)* and *department(dnumber)* the engine decided not to use any of them so we disabled the seqscan flag to test the performance of the brin index and the plan was as the following:



And the analysis output was as follows:

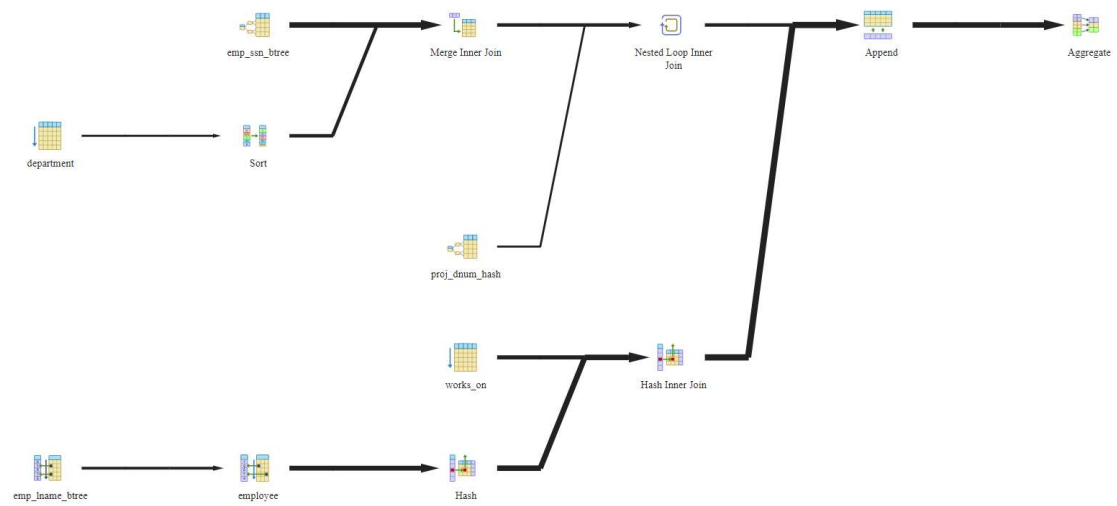
"HashAggregate (cost=27406366.56..27406372.60 rows=604 width=4) (actual time=942.197..942.294 rows=600 loops=1)"
" Group Key: p.pnumber"
" Batches: 1 Memory Usage: 73kB"
" -> Append (cost=14604.47..27406365.05 rows=604 width=4) (actual time=30.939..883.692 rows=600 loops=1)"
"   -> Nested Loop (cost=14604.47..13645374.59 rows=4 width=4) (actual time=30.838..30.840 rows=0 loops=1)"
"   -> Nested Loop (cost=12804.39..13643454.90 rows=1 width=8) (actual time=30.837..30.839 rows=0 loops=1)"
"   -> Bitmap Heap Scan on employee e (cost=12.36..475.36 rows=1066 width=8) (actual time=0.063..4.186 rows=1066 loops=1)"
"   Recheck Cond: (lname = 'employee1'::bpchar)"
"   Rows Removed by Index Recheck: 14934"
"   Heap Blocks: lossy=263"
"   -> Bitmap Index Scan on emp_lname_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.050..0.050 rows=2630 loops=1)"
"   Index Cond: (lname = 'employee1'::bpchar)"
"   -> Bitmap Heap Scan on department d (cost=12792.03..12798.28 rows=1 width=8) (actual time=0.015..0.015 rows=0 loops=1066)"
"   Recheck Cond: (mgr_snn = e.ssn)"
"   Rows Removed by Index Recheck: 1"
"   Filter: (e.dno = dnumber)"
"   Rows Removed by Filter: 0"
"   Heap Blocks: lossy=20"

"   -> Bitmap Index Scan on dep_mgr_ssn_brin (cost=0.00..12792.03 rows=150 width=0) (actual time=0.015..0.015 rows=0 loops=1066)"
"   Index Cond: (mgr_snn = e.ssn)"
"   -> Bitmap Heap Scan on project p (cost=1800.08..1919.08 rows=61 width=8) (never executed)"
"   Recheck Cond: (dnumber = d.dnumber)"
"   -> Bitmap Index Scan on proj_dnum_brin (cost=0.00..1800.06 rows=9200 width=0) (never executed)"
"   Index Cond: (dnumber = d.dnumber)"
"   -> Nested Loop (cost=12804.39..13760981.40 rows=600 width=4) (actual time=0.100..852.678 rows=600 loops=1)"
"   -> Bitmap Heap Scan on employee (cost=12.36..475.36 rows=1066 width=4) (actual time=0.052..4.225 rows=1066 loops=1)"
"   Recheck Cond: (lname = 'employee1'::bpchar)"
"   Rows Removed by Index Recheck: 14934"
"   Heap Blocks: lossy=263"
"   -> Bitmap Index Scan on emp_lname_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.043..0.043 rows=2630 loops=1)"
"   Index Cond: (lname = 'employee1'::bpchar)"
"   -> Bitmap Heap Scan on works_on (cost=12792.03..12908.53 rows=1 width=8) (actual time=0.396..0.784 rows=1 loops=1066)"
"   Recheck Cond: (essn = employee.ssn)"
"   Rows Removed by Index Recheck: 5065"
"   Heap Blocks: lossy=29400"
"   -> Bitmap Index Scan on workson_essn_brin (cost=0.00..12792.03 rows=9000 width=0) (actual time=0.017..0.017 rows=276 loops=1066)"
"   Index Cond: (essn = employee.ssn)"
"Planning Time: 0.723 ms"
"Execution Time: 945.106 ms"

As shown the cost is very high compared to the raw query and the others with indexes of course as the BRIN is not the perfect option for this case as it is not compatible with hash join.

Mixed:

We created B+Tree indexes on on *employee(lname) and employee(ssn)* and Hash index on project(dnumber) the plan was as the following:



And the analysis output was as follows:

"HashAggregate (cost=514.94..520.98 rows=604 width=4) (actual time=3.399..3.495 rows=600 loops=1)"
" Group Key: p.pnumber"
" Batches: 1 Memory Usage: 73kB"
" -> Append (cost=9.21..513.43 rows=604 width=4) (actual time=1.058..3.209 rows=600 loops=1)"
"   -> Nested Loop (cost=9.21..19.43 rows=4 width=4) (actual time=0.182..0.184 rows=0 loops=1)"
"   Join Filter: (d.dnumber = p.pnumber)"
"   -> Merge Join (cost=9.21..16.95 rows=1 width=8) (actual time=0.181..0.183 rows=0 loops=1)"
"   Merge Cond: (e.ssn = d.mgr_snn)"
"   Join Filter: (d.dnumber = e.dno)"
"   Rows Removed by Join Filter: 10"
"   -> Index Scan using emp_ssn_btree on employee e (cost=0.29..730.28 rows=1066 width=8) (actual time=0.027..0.068 rows=11 loops=1)"
"   Filter: (lname = 'employee1'::bpchar)"
"   Rows Removed by Filter: 154"
"   -> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.082..0.091 rows=150 loops=1)"
"   Sort Key: d.mgr_snn"
"   Sort Method: quicksort Memory: 32kB"
"   -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.017..0.048 rows=150 loops=1)"
"   -> Index Scan using proj_dnum_hash on project p (cost=0.00..1.71 rows=61 width=8) (never executed)"
"   Index Cond: (dnumber = e.dno)"

" -> Hash Join (cost=306.20..484.95 rows=600 width=4) (actual time=0.875..2.977 rows=600 loops=1)"
" Hash Cond: (works_on.essn = employee.ssn)"
" -> Seq Scan on works_on (cost=0.00..139.00 rows=9000 width=8) (actual time=0.017..0.854 rows=9000 loops=1)"
" -> Hash (cost=292.87..292.87 rows=1066 width=4) (actual time=0.835..0.836 rows=1066 loops=1)"
" Buckets: 2048 Batches: 1 Memory Usage: 54kB"
" -> Bitmap Heap Scan on employee (cost=16.55..292.87 rows=1066 width=4) (actual time=0.179..0.629 rows=1066 loops=1)"
" Recheck Cond: (lname = 'employee1'::bpchar)"
" Heap Blocks: exact=263"
" -> Bitmap Index Scan on emp_lname_btree (cost=0.00..16.28 rows=1066 width=0) (actual time=0.121..0.121 rows=1066 loops=1)"
" Index Cond: (lname = 'employee1'::bpchar)"
"Planning Time: 1.275 ms"
"Execution Time: 3.680 ms"

The Mixed indxes was the best choice

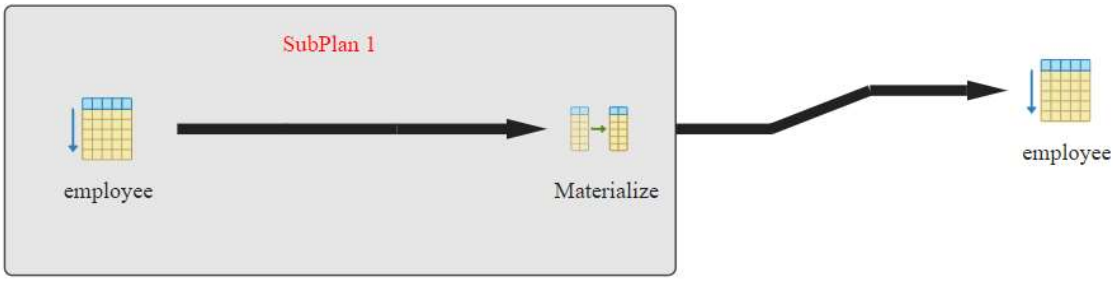
Query 3:

Select the names of employees whose salary is greater than the salary of all the employees in
department 5

```
select lname, fname
from employee
where salary > all (
select salary
from employee
where dno=5 );
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:



And the analysis output was:

"Seq Scan on employee (cost=0.00..3710883.00 rows=8000 width=42) (actual time=4.272..20.187 rows=600 loops=1)"
" Filter: (SubPlan 1)"
" Rows Removed by Filter: 15400"
" SubPlan 1"
"   -> Materialize (cost=0.00..463.54 rows=107 width=4) (actual time=0.000..0.001 rows=5 loops=16000)"
"   -> Seq Scan on employee employee_1 (cost=0.00..463.00 rows=107 width=4) (actual time=0.013..4.174 rows=107 loops=1)"
"   Filter: (dno = 5)"
"   Rows Removed by Filter: 15893"
"Planning Time: 0.110 ms"
"Execution Time: 20.246 ms"

B+Tree:

We created B+tree index on *employee(dno)* and the plan was as the following:

And the analysis output was as follows:

"Seq Scan on employee (cost=5.11..1623025.23 rows=8000 width=42) (actual time=0.348..17.361 rows=600 loops=1)"
" Filter: (SubPlan 1)"
" Rows Removed by Filter: 15400"
" SubPlan 1"
"   -> Materialize (cost=5.11..207.67 rows=107 width=4) (actual time=0.000..0.000 rows=5 loops=16000)"
"   -> Bitmap Heap Scan on employee employee_1 (cost=5.11..207.13 rows=107 width=4) (actual time=0.053..0.273 rows=107 loops=1)"
"   Recheck Cond: (dno = 5)"
"   Heap Blocks: exact=107"
"   -> Bitmap Index Scan on emp_dno_btree (cost=0.00..5.09 rows=107 width=0) (actual time=0.030..0.030 rows=107 loops=1)"

" Index Cond: (dno = 5)"
"Planning Time: 0.349 ms"
"Execution Time: 17.467 ms"

The seqscan on employee table cannot be indexed because the filter is applied over subplan1 as shown in the figure.

Hash:

We created a Hash index on *employee(dno)* and the plan was as the following:

And the analysis output was as follows:

"Seq Scan on employee (cost=4.83..1623024.94 rows=8000 width=42) (actual time=0.217..16.895 rows=600 loops=1)"
" Filter: (SubPlan 1)"
" Rows Removed by Filter: 15400"
" SubPlan 1"
" -> Materialize (cost=4.83..207.38 rows=107 width=4) (actual time=0.000..0.000 rows=5 loops=16000)"
" -> Bitmap Heap Scan on employee employee_1 (cost=4.83..206.85 rows=107 width=4) (actual time=0.050..0.156 rows=107 loops=1)"
" Recheck Cond: (dno = 5)"
" Heap Blocks: exact=107"
" -> Bitmap Index Scan on emp_dno_hash (cost=0.00..4.80 rows=107 width=0) (actual time=0.027..0.027 rows=107 loops=1)"
" Index Cond: (dno = 5)"
"Planning Time: 0.372 ms"
"Execution Time: 16.980 ms"

Almost the same cost as the B+Tree but still far better than using only seqscan.

BRIN:

We created BRIN index on *employee(dno)* and Postgres decided not to use it so we disabled the seqscan flag and the plan was as the following:

And the analysis output was as follows:

"Seq Scan on employee (cost=10000000012.12..10003710895.12 rows=8000 width=42) (actual time=5.138..21.301 rows=600 loops=1)"
" Filter: (SubPlan 1)"
" Rows Removed by Filter: 15400"
" SubPlan 1"
"   -> Materialize (cost=12.12..475.66 rows=107 width=4) (actual time=0.000..0.001 rows=5 loops=16000)"
"   -> Bitmap Heap Scan on employee employee_1 (cost=12.12..475.12 rows=107 width=4) (actual time=0.090..5.045 rows=107 loops=1)"
"   Recheck Cond: (dno = 5)"
"   Rows Removed by Index Recheck: 15893"
"   Heap Blocks: lossy=263"
"   -> Bitmap Index Scan on emp_dno_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.079..0.079 rows=2630 loops=1)"
"   Index Cond: (dno = 5)"
"Planning Time: 0.402 ms"
"Execution Time: 21.385 ms"

We can see that the cost is equal to 10003710895 and the reason of this number is that the plan has unreplaceable seqscan operation and we disabled it so if the plan was the BRIN on employee(dno) and the seqscan on the employee over the sub plan the cost would have been 3710895.

Mixed:

Since the plan has only one option for applying indexes so we don't have the ability to get a mixed indexes query.

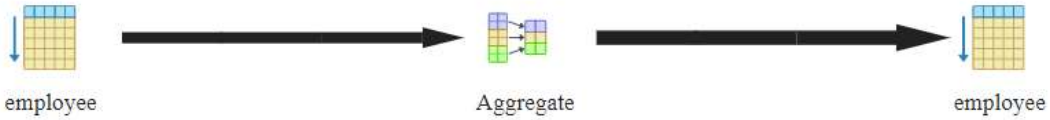
Query 3 Optimized:

Since the engine collects statistics about the data in the schemas and the range of the values of the salary column is saved so instead of traversing the whole table and check that if the selected is larger than them we can compare it only with their max.

```
select lname, fname
from employee
where salary > (
select max (salary)
from employee
where dno=5 );
```

Raw Query:

When we analyze this raw query without any indexes we got this plan:



And the analysis output was as follows:

"Seq Scan on employee (cost=463.28..926.28 rows=5333 width=42) (actual time=5.075..9.224 rows=600 loops=1)"
" Filter: (salary > $0)"
" Rows Removed by Filter: 15400"
" InitPlan 1 (returns $0)"
"   -> Aggregate (cost=463.27..463.28 rows=1 width=4) (actual time=5.048..5.049 rows=1 loops=1)"
"   -> Seq Scan on employee employee_1 (cost=0.00..463.00 rows=107 width=4) (actual time=0.014..5.020 rows=107 loops=1)"
"   Filter: (dno = 5)"
"   Rows Removed by Filter: 15893"
"Planning Time: 0.466 ms"
"Execution Time: 9.308 ms"

B+Tree:

We created B+tree index on *employee(salary)* and the plan was as the following:

And the analysis output was as follows:

"Index Scan using emp_salary_btree on employee (cost=9.48..587.88 rows=5333 width=42) (actual time=0.799..1.202 rows=600 loops=1)"

" Index Cond: (salary > $1)"
" InitPlan 2 (returns $1)"
"   -> Result (cost=9.19..9.20 rows=1 width=4) (actual time=0.781..0.782 rows=1 loops=1)"
"   InitPlan 1 (returns $0)"
"   -> Limit (cost=0.29..9.19 rows=1 width=4) (actual time=0.776..0.777 rows=1 loops=1)"
"   -> Index Scan Backward using emp_salary_btree on employee employee_1 (cost=0.29..952.96 rows=107 width=4) (actual time=0.773..0.773 rows=1 loops=1)"
"   Index Cond: (salary IS NOT NULL)"
"   Filter: (dno = 5)"
"   Rows Removed by Filter: 813"
"Planning Time: 0.571 ms"
"Execution Time: 1.307 ms"

Hash:

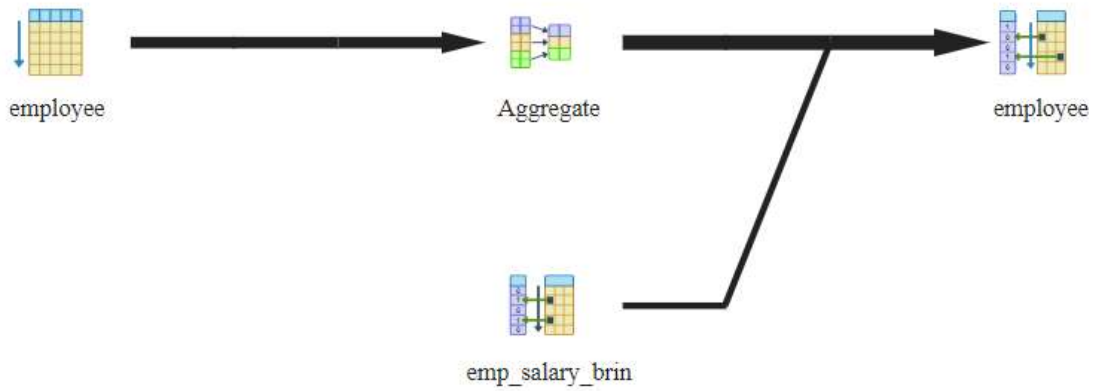
We created Hash index on *employee(dno)* and the plan was as the following:

And the analysis output was as follows:

"Seq Scan on employee (cost=207.12..670.12 rows=5333 width=42) (actual time=0.279..3.939 rows=600 loops=1)"
" Filter: (salary > $0)"
" Rows Removed by Filter: 15400"
" InitPlan 1 (returns $0)"
"   -> Aggregate (cost=207.11..207.12 rows=1 width=4) (actual time=0.259..0.261 rows=1 loops=1)"
"   -> Bitmap Heap Scan on employee employee_1 (cost=4.83..206.85 rows=107 width=4) (actual time=0.121..0.246 rows=107 loops=1)"
"   Recheck Cond: (dno = 5)"
"   Heap Blocks: exact=107"
"   -> Bitmap Index Scan on emp_dno_hash (cost=0.00..4.80 rows=107 width=0) (actual time=0.099..0.099 rows=107 loops=1)"
"   Index Cond: (dno = 5)"
"Planning Time: 0.295 ms"
"Execution Time: 4.033 ms"

BRIN:

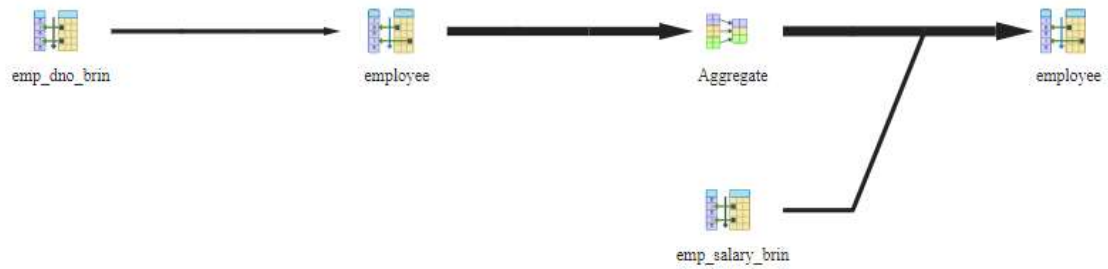
We created BRIN indexes on *employee(dno)* and *employee(salary)* Postgres decided to use the first one only and the plan was as the following:



And the analysis output was as follows:

"Bitmap Heap Scan on employee (cost=476.65..824.66 rows=5333 width=42) (actual time=3.136..5.169 rows=600 loops=1)"
" Recheck Cond: (salary > $0)"
" Rows Removed by Index Recheck: 7592"
" Heap Blocks: lossy=135"
" InitPlan 1 (returns $0)"
"   -> Aggregate (cost=463.27..463.28 rows=1 width=4) (actual time=3.041..3.041 rows=1 loops=1)"
"   -> Seq Scan on employee employee_1 (cost=0.00..463.00 rows=107 width=4) (actual time=0.015..3.022 rows=107 loops=1)"
"   Filter: (dno = 5)"
"   Rows Removed by Filter: 15893"
" -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=3.123..3.123 rows=1350 loops=1)"
"   Index Cond: (salary > $0)"
"Planning Time: 0.320 ms"
"Execution Time: 5.278 ms"

We decided to disable the seqscan flag to force it to use the index and the plan was as follows:

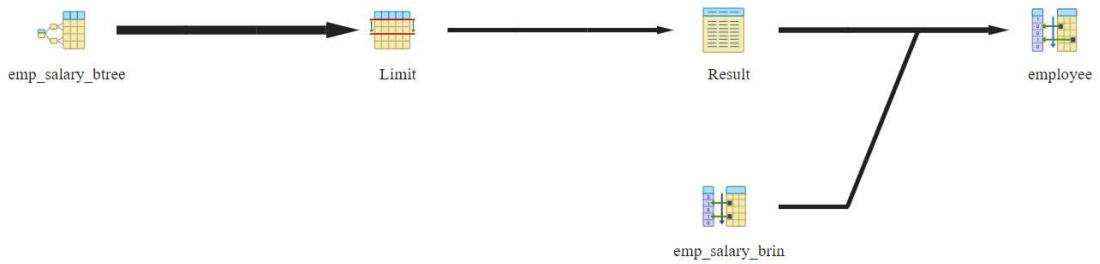


And the analysis output was as follows:

"Bitmap Heap Scan on employee (cost=488.77..836.79 rows=5333 width=42) (actual time=4.029..5.811 rows=600 loops=1)"
" Recheck Cond: (salary > $0)"
" Rows Removed by Index Recheck: 7592"
" Heap Blocks: lossy=135"
" InitPlan 1 (returns $0)"
"   -> Aggregate (cost=475.39..475.40 rows=1 width=4) (actual time=3.981..3.982 rows=1 loops=1)"
"   -> Bitmap Heap Scan on employee employee_1 (cost=12.12..475.12 rows=107 width=4) (actual time=0.041..3.963 rows=107 loops=1)"
"   Recheck Cond: (dno = 5)"
"   Rows Removed by Index Recheck: 15893"
"   Heap Blocks: lossy=263"
"   -> Bitmap Index Scan on emp_dno_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.029..0.029 rows=2630 loops=1)"
"   Index Cond: (dno = 5)"
" -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=4.023..4.023 rows=1350 loops=1)"
"   Index Cond: (salary > $0)"
"Planning Time: 0.309 ms"
"Execution Time: 5.904 ms"

Mixed:

We created B+Tree index on *employee(salary) and BRIN index on employee (salary)* and the plan was as the following:



And the analysis output was as follows:

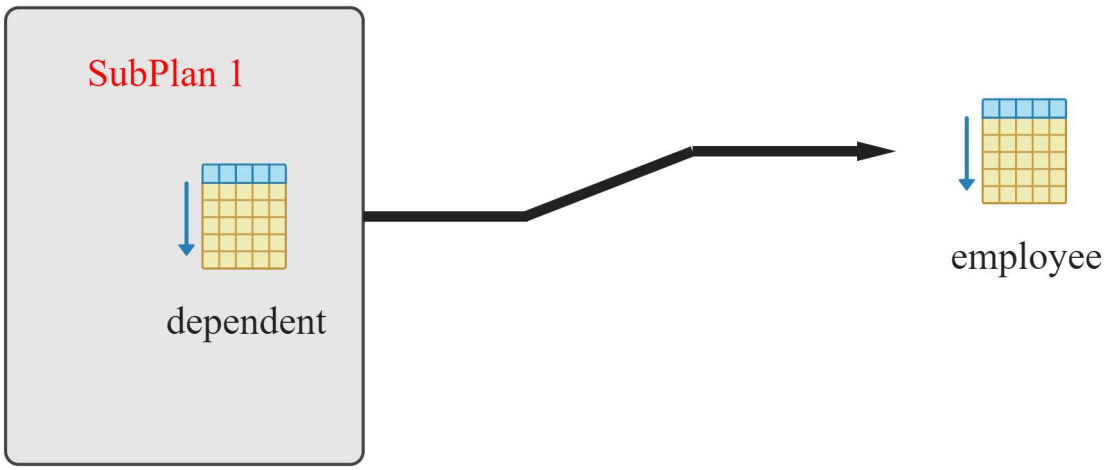
"Bitmap Heap Scan on employee (cost=22.57..370.59 rows=5333 width=42) (actual time=0.448..2.362 rows=600 loops=1)"
" Recheck Cond: (salary > $1)"
" Rows Removed by Index Recheck: 7592"
" Heap Blocks: lossy=135"
" InitPlan 2 (returns $1)"
"   -> Result (cost=9.19..9.20 rows=1 width=4) (actual time=0.409..0.410 rows=1 loops=1)"
"   InitPlan 1 (returns $0)"
"   -> Limit (cost=0.29..9.19 rows=1 width=4) (actual time=0.406..0.407 rows=1 loops=1)"
"   -> Index Scan Backward using emp_salary_btree on employee employee_1 (cost=0.29..952.96 rows=107 width=4) (actual time=0.405..0.405 rows=1 loops=1)"
"   Index Cond: (salary IS NOT NULL)"
"   Filter: (dno = 5)"
"   Rows Removed by Filter: 813"
" -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=0.441..0.441 rows=1350 loops=1)"
"   Index Cond: (salary > $1)"
"Planning Time: 0.381 ms"
"Execution Time: 2.456 ms"

Query 4:

4. Raw Query (without indexes):

```
select  e.fname, e.lname

from employee as e

 where e.ssn in (

        select essn from dependent as d

       where e.fname = d.dependent_name

       and  e.sex = d.sex );
```

Plan:



Analyze:

"Seq Scan on employee e (cost=0.00..128483.00 rows=8000 width=42)

(actual time=509.000..1616.692 rows=600 loops=1)"

" Filter: (SubPlan 1)"

" Rows Removed by Filter: 15400"

" SubPlan 1"

"   -> Seq Scan on dependent d (cost=0.00..16.00 rows=1 width=4) (actual time=0.093..0.093 rows=0 loops=16000)"

"   Filter: ((e.fname = dependent_name) AND (e.sex = sex))"

"   Rows Removed by Filter: 589"

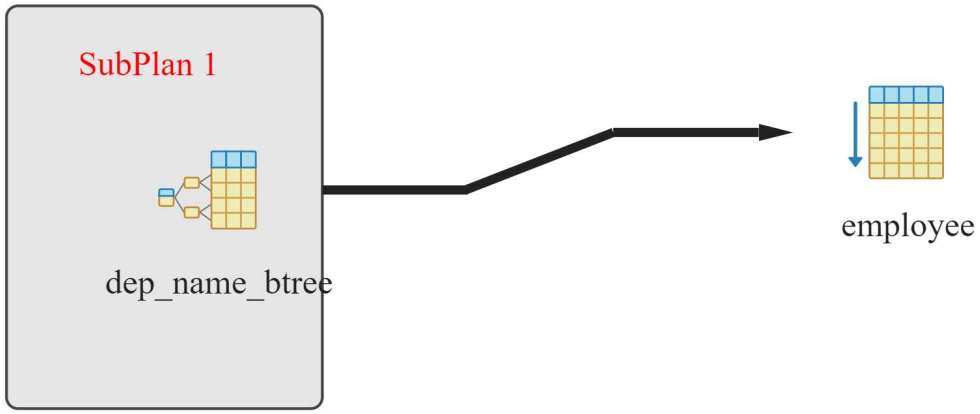
"Planning Time: 0.418 ms"

"Execution Time: 1616.773 ms"

The cost of the query was 128483.00 now we will try different indexes

4. BTree:

Plan:



Analyze:

"Seq Scan on employee e (cost=0.00..69043.00 rows=8000 width=42) (actual time=71.371..122.734 rows=600 loops=1)"

" Filter: (SubPlan 1)"

" Rows Removed by Filter: 15400"

" SubPlan 1"

"   -> Index Scan using dep_name_btree on dependent d (cost=0.28..8.29 rows=1 width=4) (actual time=0.007..0.007 rows=0 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

"   Filter: (e.sex = sex)"

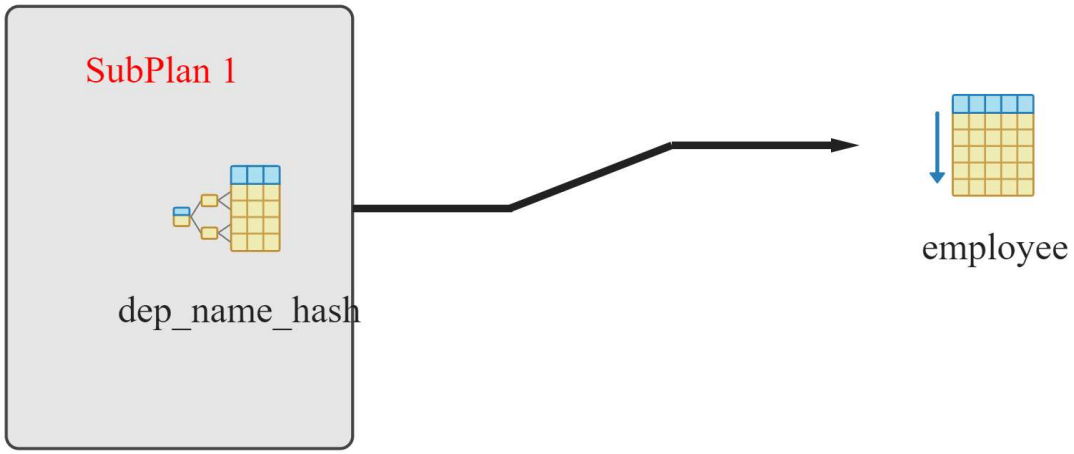
"Planning Time: 1.517 ms"

"Execution Time: 122.843 ms"

The cost of the query was 69043.00 which is better than the row query because we used a B+ tree on the dependent name which increased the speed of the filter (e.fname = d.dependent_name)

4. Hash:

Plan:



Analyze:

"Seq Scan on employee e (cost=0.00..64643.00 rows=8000 width=42) (actual time=6.841..18.753 rows=600 loops=1)"

" Filter: (SubPlan 1)"

" Rows Removed by Filter: 15400"

" SubPlan 1"

"   -> Index Scan using dep_name_hash on dependent d (cost=0.00..8.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

"   Filter: (e.sex = sex)"

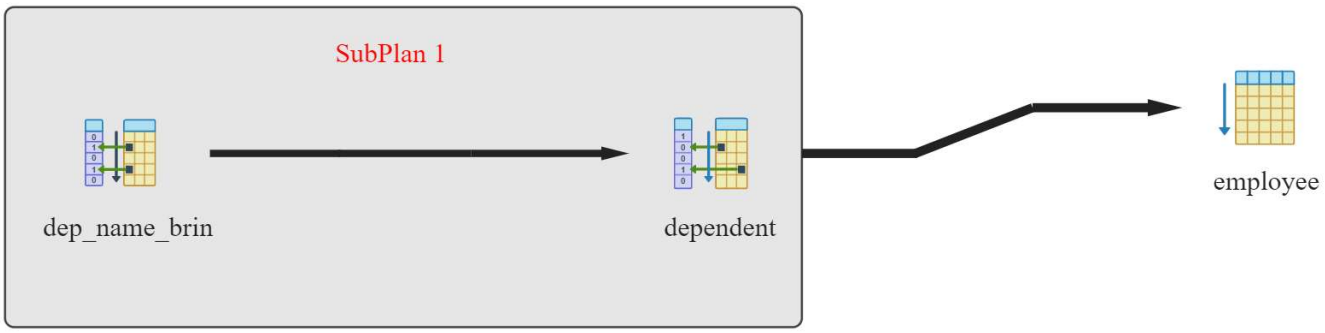
"Planning Time: 0.331 ms"

"Execution Time: 18.819 ms"

The cost of the query was 64643.00 which is better the raw query and a little better than the row query because hash index is the best index when it come to exact value queries like equality (dependent_name = e.fname)

4. Brin:

Plan:



Analyze:

"Seq Scan on employee e (cost=10000000000.00..10000320999.00 rows=8000 width=42) (actual time=128.647..279.654 rows=600 loops=1)"

" Filter: (SubPlan 1)"

" Rows Removed by Filter: 15400"

" SubPlan 1"

"   -> Bitmap Heap Scan on dependent d (cost=12.03..28.03 rows=1 width=4) (actual time=0.012..0.012 rows=0 loops=16000)"

"   Recheck Cond: (e.fname = dependent_name)"

"   Rows Removed by Index Recheck: 14"

"   Filter: (e.sex = sex)"

"   Heap Blocks: lossy=2618"

"   -> Bitmap Index Scan on dep_name_brin (cost=0.00..12.03 rows=600 width=0) (actual time=0.009..0.009 rows=3 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

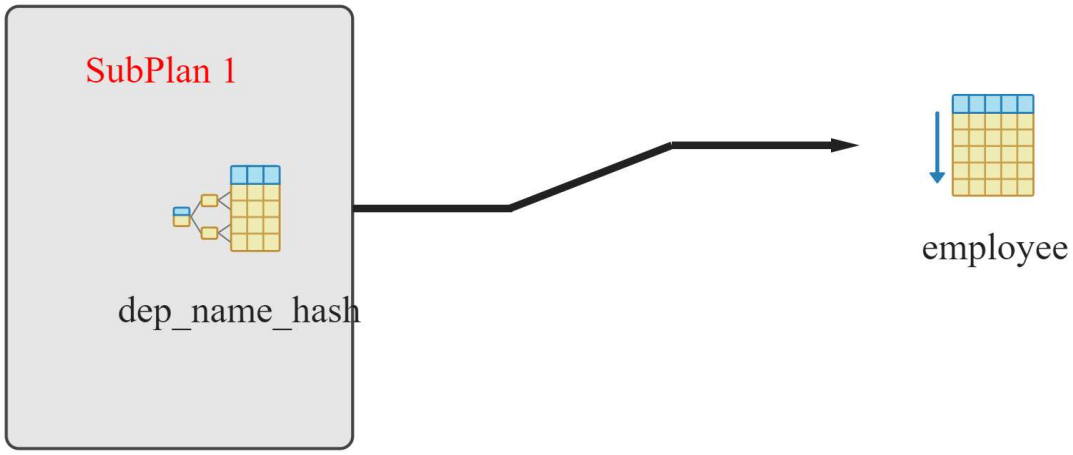
"Planning Time: 0.373 ms"

"Execution Time: 290.882 ms"

We can notice that the cost of the query is 10000320999.00 which is way higher than the other indexes the reason why is that Brin is not a good index to use for this query since brin is better used in range queries so the engine favored seqscan over brin and the only way to force it to use was to shut down seqscan using (set enable_seqscan= off;) and that's why it gave a very high cost, also note that this cost is not real it is the cost because the seqscan is disabled and the real cost is 320999.00 because the engine adds 10000000000 the cost because seqscan is disabled.

4. Mix:

Plan:



Analyze:

"Seq Scan on employee e (cost=0.00..64643.00 rows=8000 width=42) (actual time=6.841..18.753 rows=600 loops=1)"

" Filter: (SubPlan 1)"

" Rows Removed by Filter: 15400"

" SubPlan 1"

"   -> Index Scan using dep_name_hash on dependent d (cost=0.00..8.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

"   Filter: (e.sex = sex)"

"Planning Time: 0.331 ms"

"Execution Time: 18.819 ms"

We can notice here that the mix is the exact same as the hash because there is no mix of indexes possible in this query and that is can be shown if we take a look at the plan we can find only one filtration in the plan which can be optimized using at most one index so multiple indexes is not possible and it is equal to the hash because every time we try several indexes the engine choses the hash index because it is the best one.

Conclusion:

Hash index is the best index to be used in this query since it gives the lowest cost and the reason for that is ability of the Hash index to get exact value queries in a very high speed.

4. Optimised Query (without indexes):

select e.fname, e.lname

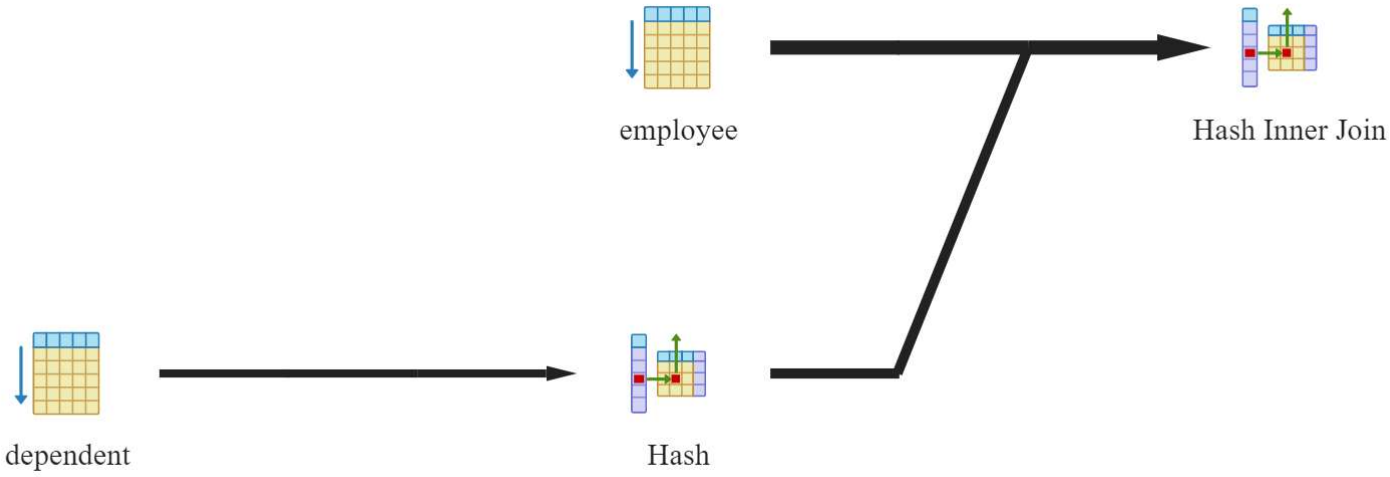
from employee e **inner join** dependent d **on** e.ssn=d.essn

where e.fname = d.dependent_name

and

e.sex = d.sex ;

Plan:



Analyze:

"Hash Join (cost=23.50..572.50 rows=1 width=42) (actual time=1.143..3.660 rows=600 loops=1)"

" Hash Cond: ((e.ssn = d.essn) AND (e.fname = d.dependent_name) AND (e.sex = d.sex))"

" -> Seq Scan on employee e (cost=0.00..423.00 rows=16000 width=48) (actual time=0.009..0.873 rows=16000 loops=1)"

" -> Hash (cost=13.00..13.00 rows=600 width=27) (actual time=0.156..0.158 rows=600 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 43kB"

"   -> Seq Scan on dependent d (cost=0.00..13.00 rows=600 width=27) (actual time=0.008..0.064 rows=600 loops=1)"

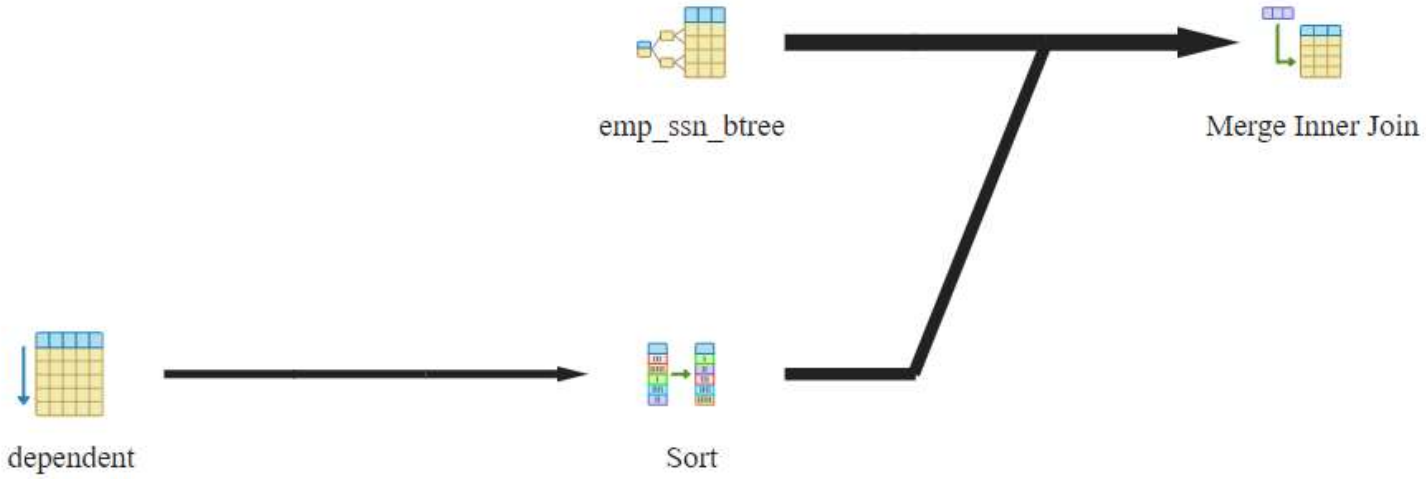
"Planning Time: 0.207 ms"

"Execution Time: 3.704 ms"

In this query we used inner join instead of where-in The cost of this query is 572.50 which is way faster than the initial query

4. Optimised + BTree:

Plan:



Analyze:

"Merge Join (cost=255.36..294.74 rows=1 width=42) (actual time=4.330..6.072 rows=600 loops=1)"

" Merge Cond: (e.ssn = d.essn)"

" Join Filter: ((e.fname = d.dependent_name) AND (e.sex = d.sex))"

" -> Index Scan using emp_ssn_btree on employee e (cost=0.29..690.28 rows=16000 width=48) (actual time=0.028..2.646 rows=5300 loops=1)"

" -> Sort (cost=40.69..42.19 rows=600 width=27) (actual time=0.721..0.837 rows=600 loops=1)"

"   Sort Key: d.essn"

"   Sort Method: quicksort Memory: 71kB"

"   -> Seq Scan on dependent d (cost=0.00..13.00 rows=600 width=27) (actual time=0.051..0.292 rows=600 loops=1)"

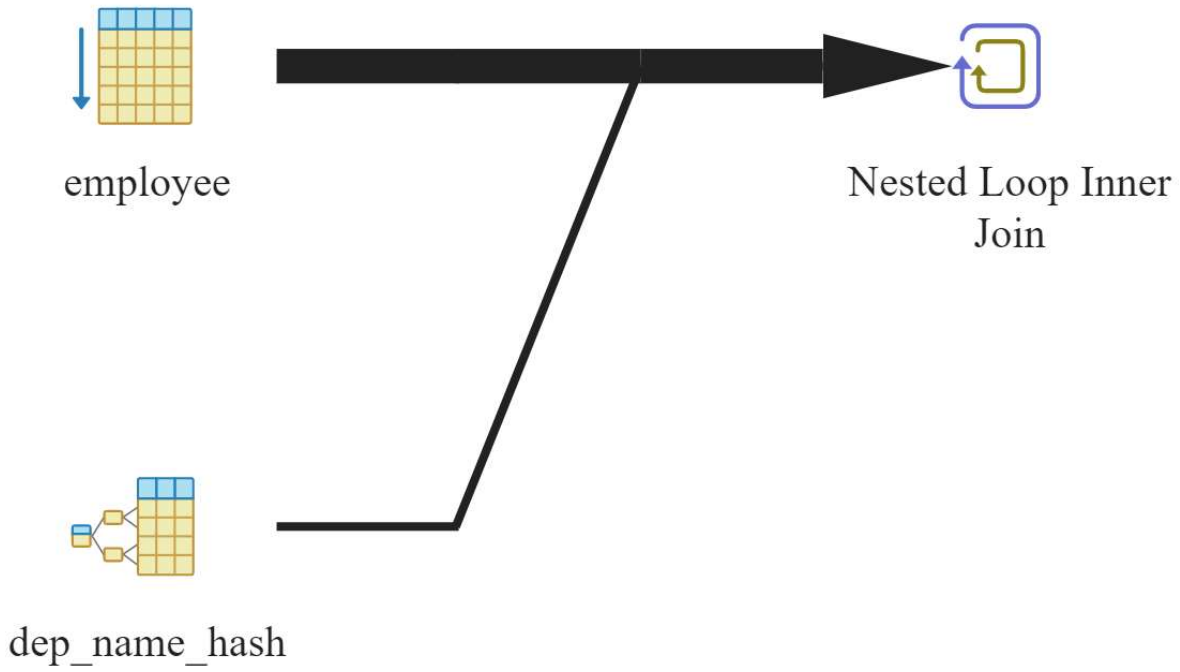
"Planning Time: 0.855 ms"

"Execution Time: 6.216 ms"

The cost of this query is 294.74 which is faster than the raw optimized query because after creating a B+ tree on ssn in the employee column it increases the speed of inner join on (e.ssn=d.essn)

4. Optimised + Hash:

Plan:



Analyze:

"Nested Loop (cost=10000000000.00..10000000987.00 rows=1 width=42) (actual time=6.340..21.795 rows=600 loops=1)"

" -> Seq Scan on employee e (cost=10000000000.00..10000000423.00 rows=16000 width=48) (actual time=0.023..2.040 rows=16000 loops=1)"

" -> Index Scan using dep_name_hash on dependent d (cost=0.00..0.03 rows=1 width=27) (actual time=0.001..0.001 rows=0 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

"   Filter: ((e.ssn = essn) AND (e.sex = sex))"

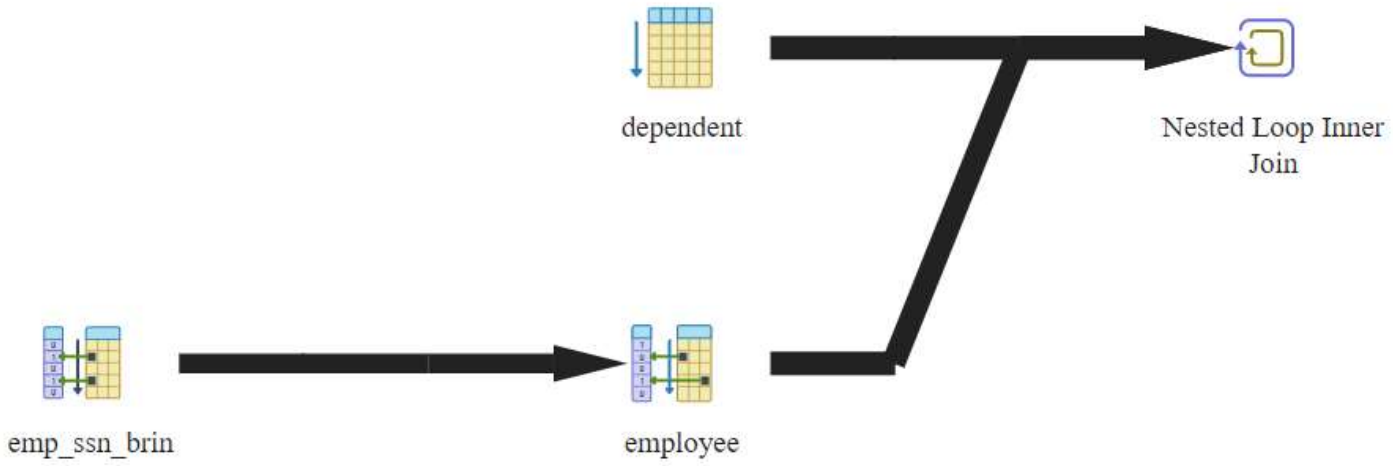
"Planning Time: 0.458 ms"

"Execution Time: 21.862 ms"

The cost of this plan is very high because the engine always preferred seqscan over hash and the only to force the engine to use hash was to disable seqscan, the cost is 10000000987.00 when seqscan is disabled and the real cost is 987.00

4. Optimised + Brin:

Plan:



Analyze:

"Nested Loop (cost=10000007200.03..10004378434.85 rows=1 width=42) (actual time=0.571..443.304 rows=600 loops=1)"

" -> Seq Scan on dependent d (cost=10000000000.00..10000000013.00 rows=600 width=27) (actual time=0.010..0.117 rows=600 loops=1)"

" -> Bitmap Heap Scan on employee e (cost=7200.03..7297.36 rows=1 width=48) (actual time=0.473..0.732 rows=1 loops=600)"

"   Recheck Cond: (ssn = d.essn)"

"   Rows Removed by Index Recheck: 7807"

"   Filter: ((d.dependent_name = fname) AND (d.sex = sex))"

"   Heap Blocks: lossy=76800"

"   -> Bitmap Index Scan on emp_ssn_brin (cost=0.00..7200.03 rows=5333 width=0) (actual time=0.009..0.009 rows=1280 loops=600)"

"   Index Cond: (ssn = d.essn)"

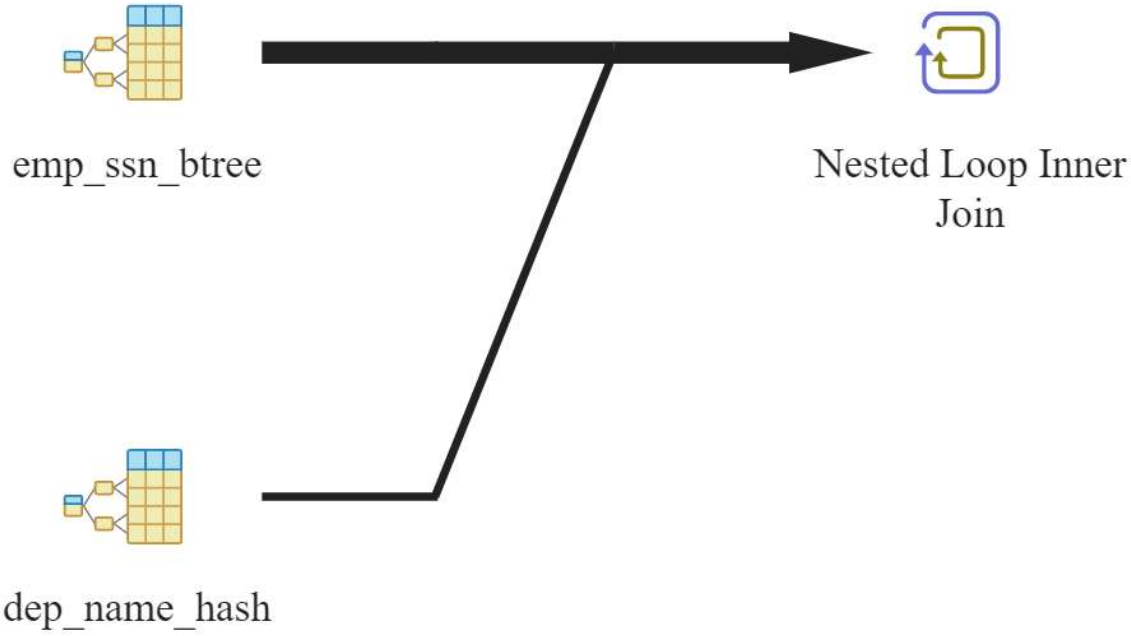
"Planning Time: 0.271 ms"

"Execution Time: 443.777 ms"

Again the cost of this plan is very high because the engine always preferred seqscan over Brin and the only to force the engine to use Brin was to disable seqscan, The cost when seqscan is disabled is 10004378434.85 and the real cost is 4378434.85, we can notice that Hash is faster than Brin even though both of them have the seqscan disabled and a very high cost

4. Optimised + Mix:

Plan:



Analyze:

"Nested Loop (cost=0.29..1254.29 rows=1 width=42) (actual time=3.595..13.042 rows=600 loops=1)"

" -> Index Scan using emp_ssn_btree on employee e (cost=0.29..690.28 rows=16000 width=48) (actual time=0.012..2.544 rows=16000 loops=1)"

" -> Index Scan using dep_name_hash on dependent d (cost=0.00..0.03 rows=1 width=27) (actual time=0.000..0.000 rows=0 loops=16000)"

"   Index Cond: (dependent_name = e.fname)"

"   Filter: ((e.ssn = essn) AND (e.sex = sex))"

"Planning Time: 0.275 ms"

"Execution Time: 13.081 ms"

the cost of this plan is 1254.29 which worse than the optimized raw query and using BTree only query the reason for that is that the engine always preferred using seqscan over Hash and Brin as we saw in the analysis of previous indexes and the only way to force the engine to mix indexes was to disable seqscan so the engine was forced to use the indexes we created and therefore increasing the cost of the query, the best mix of indexes we found is Btree on ssn in employee table and hash index on dependent name.

Conclusion:

The best thing to do to decrease the cost of this query is to use Btree index which is very fast in queries with single term on exact value

Query 5:

5. Raw Query (without indexes):

select fname, lname

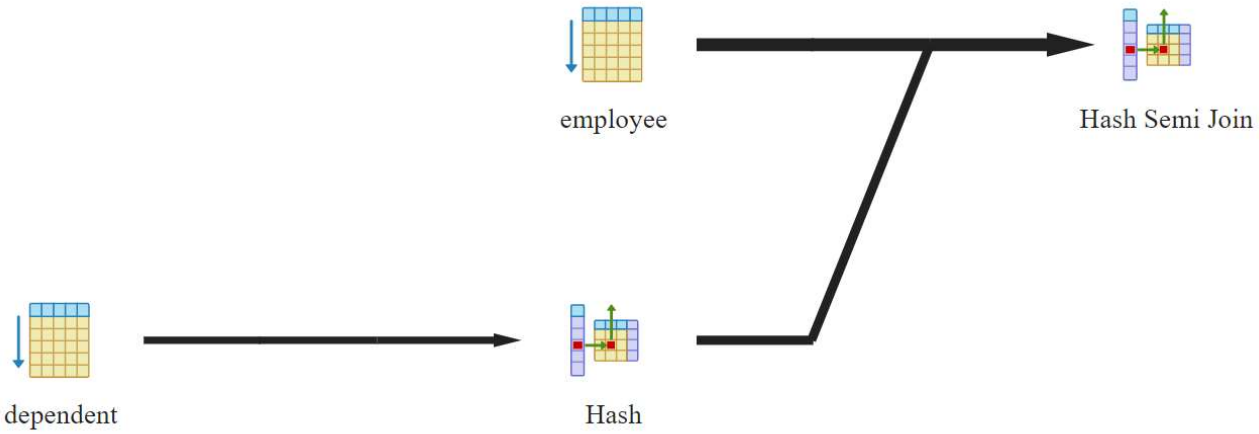
from employee

where exists (**select** *

from dependent

where ssn=essn);

Plan:



Analyze:

"Hash Semi Join (cost=20.50..492.18 rows=600 width=42) (actual time=0.896..2.859 rows=600 loops=1)"

" Hash Cond: (employee.ssn = dependent.essn)"

" -> Seq Scan on employee (cost=0.00..423.00 rows=16000 width=46) (actual time=0.013..1.024 rows=16000 loops=1)"

" -> Hash (cost=13.00..13.00 rows=600 width=4) (actual time=0.124..0.126 rows=600 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 30kB"

"   -> Seq Scan on dependent (cost=0.00..13.00 rows=600 width=4) (actual time=0.009..0.060 rows=600 loops=1)"

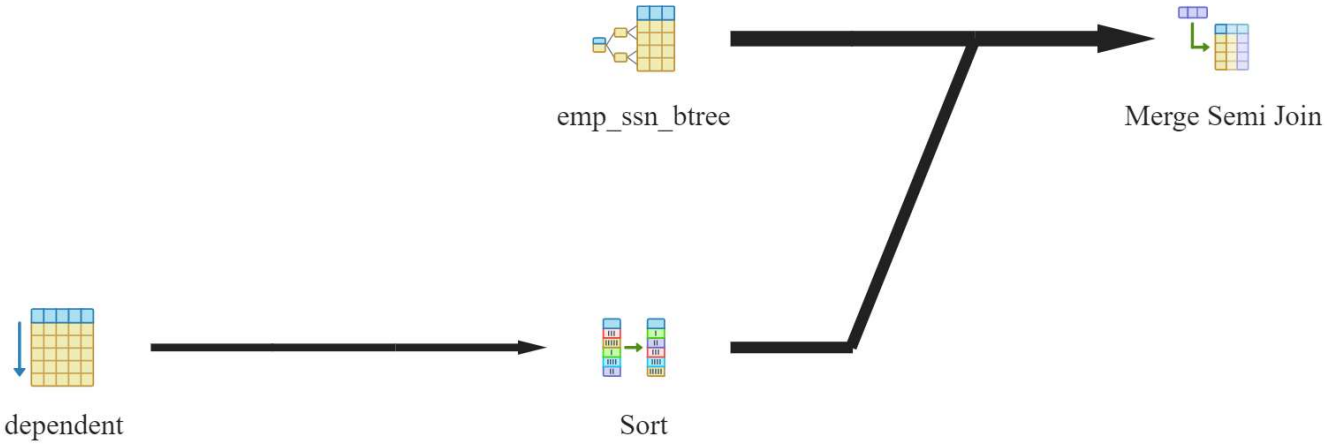
"Planning Time: 0.105 ms"

"Execution Time: 2.898 ms"

The cost of this query is 492.18 now we will try different indexes to decrease the cost.

5. BTree:

Plan:



Analyze:

"Merge Semi Join (cost=255.36..291.74 rows=600 width=42) (actual time=0.909..1.115 rows=600 loops=1)"

" Merge Cond: (employee.ssn = dependent.essn)"

" -> Index Scan using emp_ssn_btree on employee (cost=0.29..690.28 rows=16000 width=46) (actual time=0.018..0.582 rows=5300 loops=1)"

" -> Sort (cost=40.69..42.19 rows=600 width=4) (actual time=0.145..0.166 rows=600 loops=1)"

"   Sort Key: dependent.essn"

"   Sort Method: quicksort Memory: 53kB"

"   -> Seq Scan on dependent (cost=0.00..13.00 rows=600 width=4) (actual time=0.011..0.055 rows=600 loops=1)"

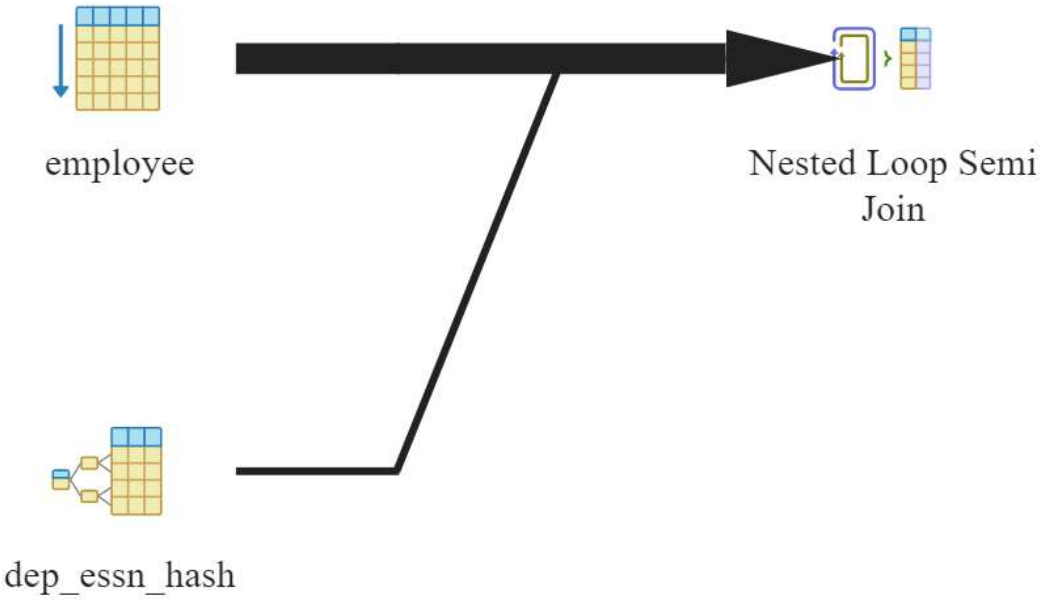
"Planning Time: 0.155 ms"

"Execution Time: 1.148 ms"

The cost of this query is 291.74 which is better than the row query because upon creating a BTree on SSN of the employee, it finds the (employee.ssn = dependent.essn) in a faster way because BTree is fast in single exact value queries.

5. Hash:

Plan:



Analyze:

"Nested Loop Semi Join (cost=10000000000.00..10000000753.00 rows=600 width=42) (actual time=2.365..9.075 rows=600 loops=1)"

" -> Seq Scan on employee e (cost=10000000000.00..10000000423.00 rows=16000 width=46) (actual time=0.011..1.049 rows=16000 loops=1)"

" -> Index Scan using dep_essn_hash on dependent (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=16000)"

"   Index Cond: (essn = e.ssn)"

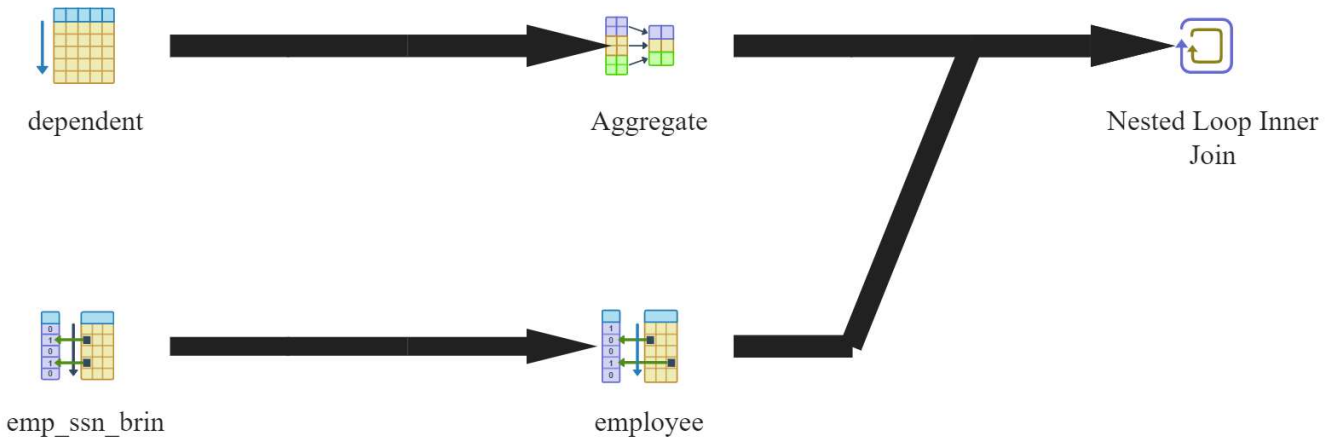
"Planning Time: 0.167 ms"

"Execution Time: 9.115 ms"

The cost of this query is 10000000753.00 which is very high because the engine always preferred seqscan over Hash index so we disabled seqscan to force the engine to use hash index, this cost is not real the engine adds 10 billion to the cost because the seqscan is closed and the real value is 753.00 which is worse than the raw query as we can see.

5. Brin:

Plan:



Analyze:

"Nested Loop (cost=10000007214.53..10004362443.35 rows=600 width=42) (actual time=3.410..501.216 rows=600 loops=1)"

" -> HashAggregate (cost=10000000014.50..10000000020.50 rows=600 width=4) (actual time=0.741..1.079 rows=600 loops=1)"

"   Group Key: dependent.essn"

"   Batches: 1 Memory Usage: 73kB"

"   -> Seq Scan on dependent (cost=10000000000.00..10000000013.00 rows=600 width=4) (actual time=0.035..0.199 rows=600 loops=1)"

" -> Bitmap Heap Scan on employee (cost=7200.03..7270.69 rows=1 width=46) (actual time=0.535..0.826 rows=1 loops=600)"

"   Recheck Cond: (ssn = dependent.essn)"

"   Rows Removed by Index Recheck: 7807"

"   Heap Blocks: lossy=76800"

"   -> Bitmap Index Scan on emp_ssn_brin (cost=0.00..7200.03 rows=5333 width=0) (actual time=0.012..0.012 rows=1280 loops=600)"

"   Index Cond: (ssn = dependent.essn)"

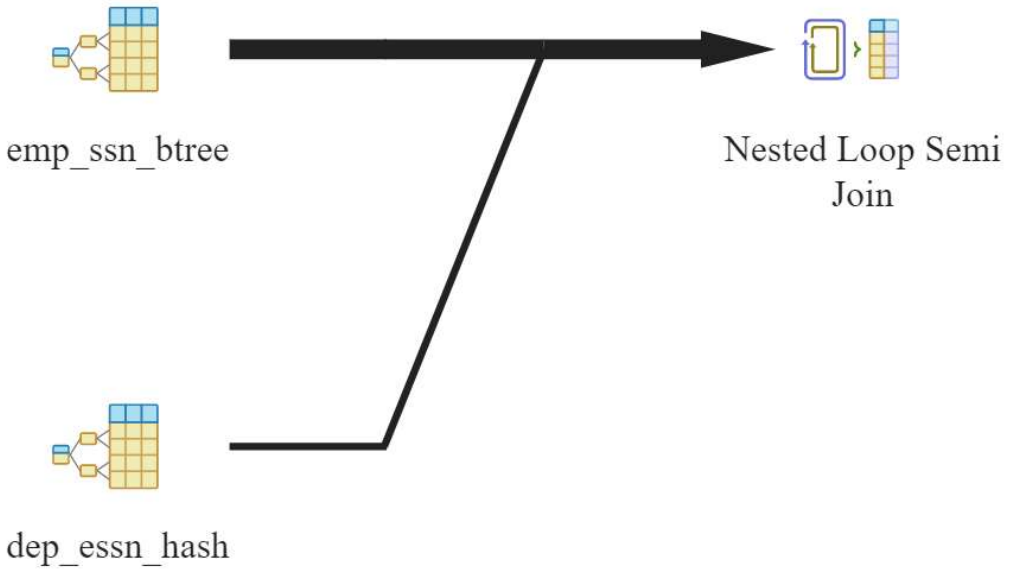
"Planning Time: 0.607 ms"

"Execution Time: 501.769 ms"

Again the cost is very high 10004362443.35 because the engine always preferred seqscan over Brin index so we disabled seqscan to force the engine to use Brin index, this cost is not real the engine adds 10 billion to the cost because the seqscan is closed and the real value is 4362443.35 which is worse than the raw query as we can see and it is also worse than Hash index.

5. Mix:

Plan:



Analyze:

"Nested Loop Semi Join (cost=0.29..1020.28 rows=600 width=42) (actual time=2.727..9.873 rows=600 loops=1)"

" -> Index Scan using emp_ssn_btree on employee (cost=0.29..690.28 rows=16000 width=46) (actual time=0.012..2.220 rows=16000 loops=1)"

" -> Index Scan using dep_essn_hash on dependent (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=16000)"

"   Index Cond: (essn = employee.ssn)"

"Planning Time: 0.251 ms"

"Execution Time: 9.909 ms"

As we can see the cost of this query is 1020.28 which is worse than using Btree alone or the row query the reason for that is that the engine prefers seqscan over hash and brin as we saw in the previous analysis so the only way to force the engine to use a mix of indexes is to close the seqscan at all, the cost was also better than the cost of hash of brin alone.

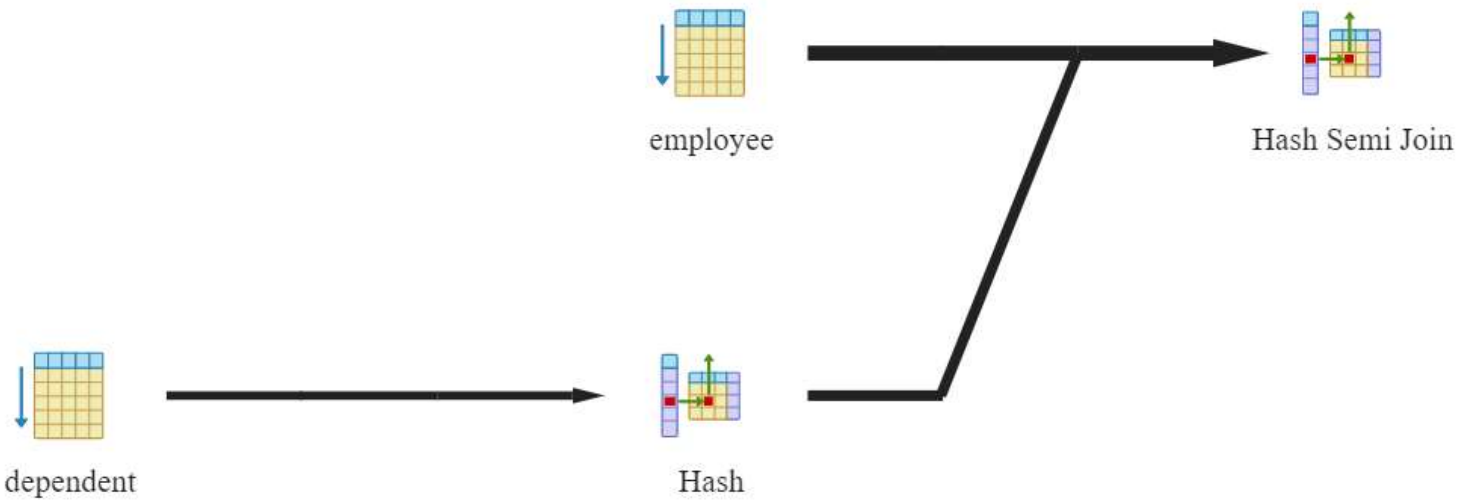
Conclusion:

BTree is the best index to use in this query due to its strength in calculating queries in a single exact value.

5. Equivlant Query (without indexes):

```
select fname, lname
from employee e
where e.ssn in( select essn
                from dependent);
```

Plan:



Analyze:

"Hash Semi Join (cost=20.50..492.18 rows=600 width=42) (actual time=0.851..2.582 rows=600 loops=1)"

" Hash Cond: (e.ssn = dependent.essn)"

" -> Seq Scan on employee e (cost=0.00..423.00 rows=16000 width=46) (actual time=0.013..0.914 rows=16000 loops=1)"

" -> Hash (cost=13.00..13.00 rows=600 width=4) (actual time=0.122..0.123 rows=600 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 30kB"

"   -> Seq Scan on dependent (cost=0.00..13.00 rows=600 width=4) (actual time=0.010..0.062 rows=600 loops=1)"

"Planning Time: 0.184 ms"

"Execution Time: 2.620 ms"

This query has no optimized query because it uses only one filtration using ssn and there no way to make it in another faster way, inner join was tried but it gave a higher cost but when using (where-in) it gives the exact same cost and the same plan when trying this equivalent query with all the indexes it gave the same plans and costs as the initial query

Query 6:

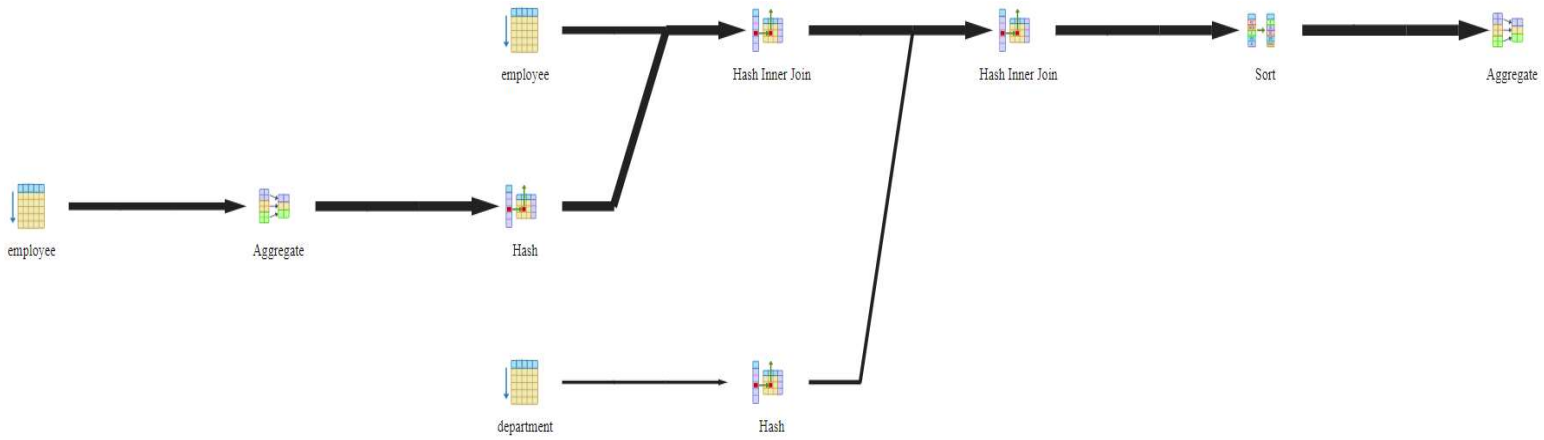
6. Raw Query (without indexes):

select dnumber, count(*) **from** department, employee

where dnumber=dno **and** salary > 40000 **and** dno **in** (

select dno **from** employee **group by** dno **having** count (*) > 5) **group** by dnumber;

Plan:



Analyze:

"GroupAggregate (cost=980.46..982.77 rows=132 width=12) (actual time=17.725..17.891 rows=100 loops=1)"

" Group Key: department.dnumber"

" -> Sort (cost=980.46..980.79 rows=132 width=4) (actual time=17.709..17.750 rows=400 loops=1)"

"   Sort Key: department.dnumber"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=511.38..975.81 rows=132 width=4) (actual time=11.084..17.536 rows=400 loops=1)"

"   Hash Cond: (employee.dno = department.dnumber)"

"   -> Hash Join (cost=506.00..970.08 rows=132 width=8) (actual time=10.969..17.226 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Seq Scan on employee (cost=0.00..463.00 rows=397 width=4) (actual time=0.036..6.019 rows=400 loops=1)"

"   Filter: (salary > 40000)"

"   Rows Removed by Filter: 15600"

"   -> Hash (cost=505.38..505.38 rows=50 width=4) (actual time=10.923..10.925 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> HashAggregate (cost=503.00..504.88 rows=50 width=4) (actual time=10.755..10.856 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   Batches: 1 Memory Usage: 48kB"

"   -> Seq Scan on employee employee_1 (cost=0.00..423.00 rows=16000 width=4) (actual time=0.016..2.661 rows=16000 loops=1)"

"   -> Hash (cost=3.50..3.50 rows=150 width=4) (actual time=0.107..0.108 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> Seq Scan on department (cost=0.00..3.50 rows=150 width=4) (actual time=0.034..0.064 rows=150 loops=1)"

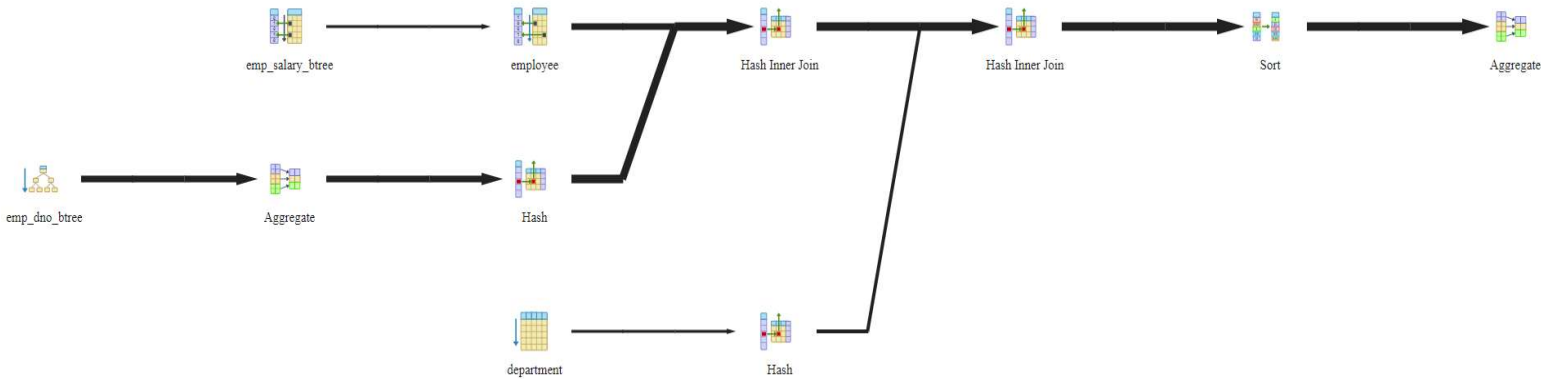
"Planning Time: 0.697 ms"

"Execution Time: 18.002 ms"

The cost of the query was 982.77 now we will try different indexes to get a better cost.

6. BTree:

Plan:



Analyze:

"GroupAggregate (cost=687.85..690.16 rows=132 width=12) (actual time=3.636..3.729 rows=100 loops=1)"

" Group Key: department.dnumber"

" -> Sort (cost=687.85..688.18 rows=132 width=4) (actual time=3.630..3.655 rows=400 loops=1)"

"   Sort Key: department.dnumber"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=400.02..683.20 rows=132 width=4) (actual time=3.253..3.535 rows=400 loops=1)"

"   Hash Cond: (employee.dno = department.dnumber)"

"   -> Hash Join (cost=394.65..676.01 rows=132 width=8) (actual time=3.141..3.332 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Bitmap Heap Scan on employee (cost=7.36..287.65 rows=397 width=4) (actual time=0.030..0.087 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Heap Blocks: exact=10"

"   -> Bitmap Index Scan on emp_salary_btree (cost=0.00..7.26 rows=397 width=0) (actual time=0.024..0.024 rows=400 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=386.66..386.66 rows=50 width=4) (actual time=3.102..3.102 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> GroupAggregate (cost=0.29..386.16 rows=50 width=4) (actual time=0.033..3.073 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   -> Index Only Scan using emp_dno_btree on employee employee_1 (cost=0.29..304.29 rows=16000 width=4) (actual time=0.009..1.629 rows=16000 loops=1)"

"   Heap Fetches: 0"

"   -> Hash (cost=3.50..3.50 rows=150 width=4) (actual time=0.059..0.060 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> Seq Scan on department (cost=0.00..3.50 rows=150 width=4) (actual time=0.019..0.035 rows=150 loops=1)"

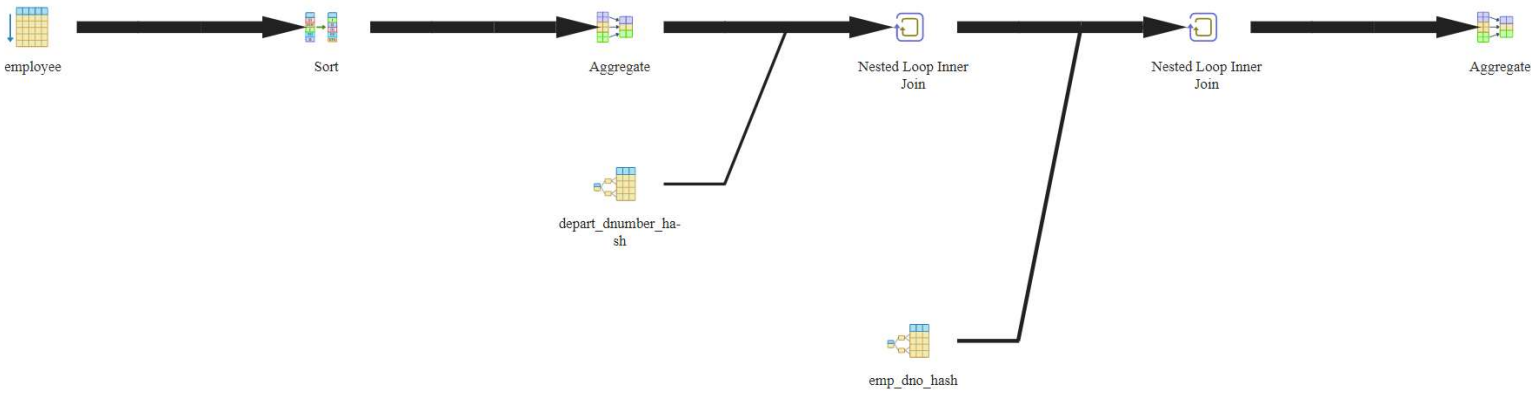
"Planning Time: 0.597 ms"

"Execution Time: 3.813 ms"

In this query we used two indexes a Btree on employee salary and a Btree on employee dno it gave us a cost of 690.16 which is better than the raw query since the btree speeds up the query with range search and the exact value queries and we have this in dno= dnumber=dno and salary > 40000

6. Hash:

Plan:



Analyze:

"GroupAggregate (cost=10000001540.26..10000002251.83 rows=132 width=12) (actual time=10.373..19.539 rows=100 loops=1)"

" Group Key: department.dnumber"

" -> Nested Loop (cost=10000001540.26..10000002249.85 rows=132 width=4) (actual time=10.275..19.421 rows=400 loops=1)"

"   -> Nested Loop (cost=10000001540.26..10000001688.01 rows=50 width=8) (actual time=5.788..9.364 rows=150 loops=1)"

"   -> GroupAggregate (cost=10000001540.26..10000001662.14 rows=50 width=4) (actual time=5.772..9.040 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

" -> Sort (cost=10000001540.26..10000001580.26 rows=16000 width=4) (actual time=5.733..7.036 rows=16000 loops=1)"

" Sort Key: employee_1.dno"

" Sort Method: quicksort Memory: 1135kB"

" -> Seq Scan on employee employee_1 (cost=10000000000.00..10000000423.00 rows=16000 width=4) (actual time=0.019..2.944 rows=16000 loops=1)"

" -> Index Scan using depart_dnumber_hash on department (cost=0.00..0.50 rows=1 width=4) (actual time=0.001..0.002 rows=1 loops=150)"

" Index Cond: (dnumber = employee_1.dno)"

" -> Index Scan using emp_dno_hash on employee (cost=0.00..11.21 rows=3 width=4) (actual time=0.062..0.066 rows=3 loops=150)"

" Index Cond: (dno = department.dnumber)"

" Filter: (salary > 40000)"

" Rows Removed by Filter: 104"

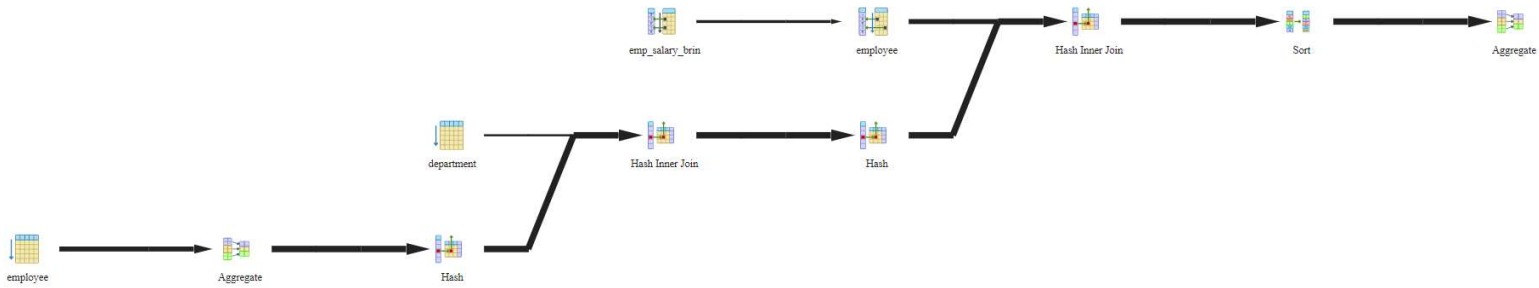
"Planning Time: 0.377 ms"

"Execution Time: 19.598 ms"

The cost of this query is 10000002251.83 because we closed seqscan since the engine always preferred the seqscan over hash index and the only way to force it to use hash index is to disable seqscan, the cost here is not the actual cost since the engine adds 10 billion to the cost because seqscan is closed the real value is 2251.83 which worse than raw query that uses no index and only seqscan

6. Brin:

Plan:



Analyze:

"GroupAggregate (cost=878.14..880.45 rows=132 width=12) (actual time=8.197..8.294 rows=100 loops=1)"

" Group Key: department.dnumber"

" -> Sort (cost=878.14..878.47 rows=132 width=4) (actual time=8.190..8.213 rows=400 loops=1)"

" Sort Key: department.dnumber"

" Sort Method: quicksort Memory: 43kB"

" -> Hash Join (cost=522.67..873.49 rows=132 width=4) (actual time=6.619..8.106 rows=400 loops=1)"

" Hash Cond: (employee.dno = department.dnumber)"

"   -> Bitmap Heap Scan on employee (cost=12.14..360.15 rows=397 width=4) (actual time=0.043..1.436 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Rows Removed by Index Recheck: 7408"

"   Heap Blocks: lossy=128"

"   -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=0.024..0.024 rows=1280 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=509.90..509.90 rows=50 width=8) (actual time=6.571..6.573 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> Hash Join (cost=506.00..509.90 rows=50 width=8) (actual time=6.481..6.544 rows=150 loops=1)"

"   Hash Cond: (department.dnumber = employee_1.dno)"

"   -> Seq Scan on department (cost=0.00..3.50 rows=150 width=4) (actual time=0.016..0.030 rows=150 loops=1)"

"   -> Hash (cost=505.38..505.38 rows=50 width=4) (actual time=6.459..6.460 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> HashAggregate (cost=503.00..504.88 rows=50 width=4) (actual time=6.377..6.417 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   Batches: 1 Memory Usage: 48kB"

"   -> Seq Scan on employee employee_1 (cost=0.00..423.00 rows=16000 width=4) (actual time=0.008..1.473 rows=16000 loops=1)"

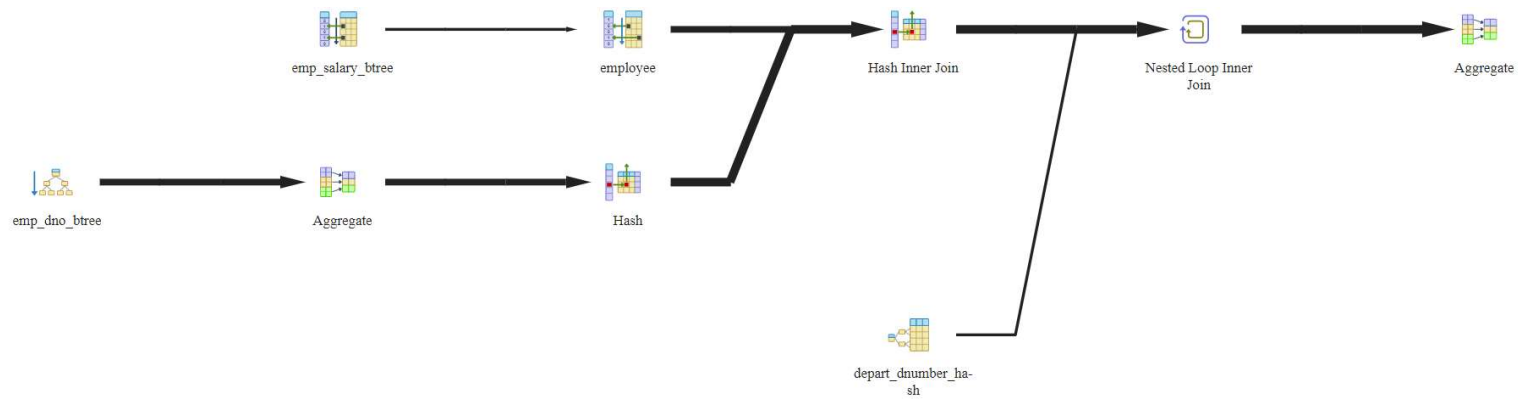
"Planning Time: 0.471 ms"

"Execution Time: 8.379 ms"

The cost of this query was 880.45 which is faster than the row query but slower than the btree index this is because brin is fast with range queries

6. Mix:

Plan:



Analyze:

"HashAggregate (cost=688.28..689.60 rows=132 width=12) (actual time=6.707..6.744 rows=100 loops=1)"

" Group Key: department.dnumber"

" Batches: 1 Memory Usage: 48kB"

" -> Nested Loop (cost=394.65..687.62 rows=132 width=4) (actual time=5.408..6.551 rows=400 loops=1)"

"   -> Hash Join (cost=394.65..676.01 rows=132 width=8) (actual time=5.390..5.715 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Bitmap Heap Scan on employee (cost=7.36..287.65 rows=397 width=4) (actual time=0.143..0.244 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Heap Blocks: exact=10"

"   -> Bitmap Index Scan on emp_salary_btree (cost=0.00..7.26 rows=397 width=0) (actual time=0.132..0.132 rows=400 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=386.66..386.66 rows=50 width=4) (actual time=5.234..5.235 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> GroupAggregate (cost=0.29..386.16 rows=50 width=4) (actual time=0.054..5.190 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   -> Index Only Scan using emp_dno_btree on employee employee_1 (cost=0.29..304.29 rows=16000 width=4) (actual time=0.013..2.619 rows=16000 loops=1)"

"   Heap Fetches: 0"

"   -> Index Scan using depart_dnumber_hash on department (cost=0.00..0.08 rows=1 width=4) (actual time=0.001..0.002 rows=1 loops=400)"

"   Index Cond: (dnumber = employee.dno)"

"Planning Time: 1.082 ms"

"Execution Time: 6.867 ms"

In this query we used three indexes btree on employee salary, btree on employee dno, and hash index on department number the mix of these queries gave us a cost equal to 689.60 which is slightly less than the cost of Btree indexes alone and also better than all other indexes, this because btree helps in the range and exact value queries and the hash helps in the exact value queries.

6. Optimised Query (without indexes):

select X.dno as dnumber, count (*)

from (

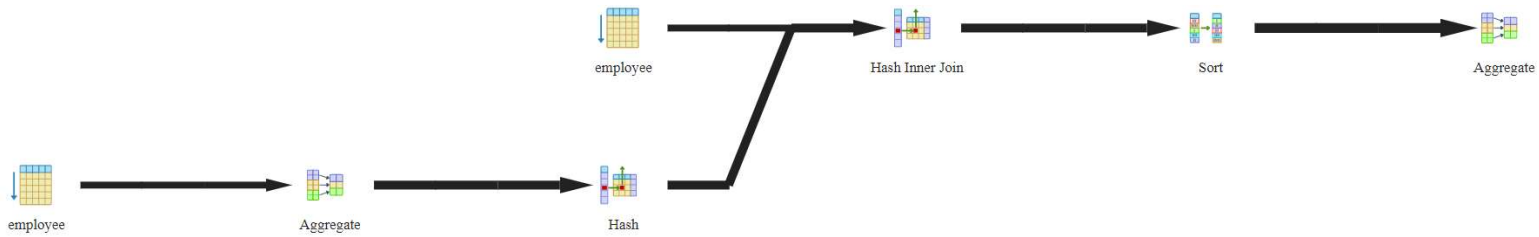
(**select** dno,salary **from** employee **where** salary>40000) **as** X

Inner join

(**select** dno from employee **group by** dno **having** count (*) > 5) **as** Y **on** X.dno=Y.dno

) **group by** X.dno

Plan:



Analyze:

"GroupAggregate (cost=974.72..977.03 rows=132 width=12) (actual time=5.516..5.572 rows=100 loops=1)"

" Group Key: employee.dno"

" -> Sort (cost=974.72..975.05 rows=132 width=4) (actual time=5.509..5.523 rows=400 loops=1)"

"   Sort Key: employee.dno"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=506.00..970.08 rows=132 width=4) (actual time=3.694..5.456 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Seq Scan on employee (cost=0.00..463.00 rows=397 width=4) (actual time=0.015..1.720 rows=400 loops=1)"

"   Filter: (salary > 40000)"

"   Rows Removed by Filter: 15600"

"   -> Hash (cost=505.38..505.38 rows=50 width=4) (actual time=3.674..3.675 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> HashAggregate (cost=503.00..504.88 rows=50 width=4) (actual time=3.639..3.656 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   Batches: 1 Memory Usage: 48kB"

"   -> Seq Scan on employee employee_1 (cost=0.00..423.00 rows=16000 width=4) (actual time=0.005..0.860 rows=16000 loops=1)"

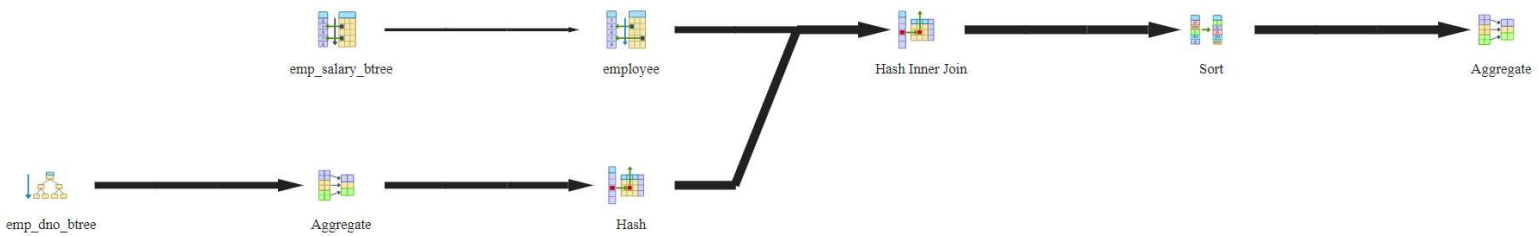
"Planning Time: 0.114 ms"

"Execution Time: 5.616 ms"

As we can see this query cost is 977.03 which is slightly better than the initial query we obtained this query by using inner joins instead of where-in in the initial query

6. Optimised + BTree:

Plan:



Analyze:

"GroupAggregate (cost=680.66..682.97 rows=132 width=12) (actual time=2.432..2.498 rows=100 loops=1)"

" Group Key: employee.dno"

" -> Sort (cost=680.66..680.99 rows=132 width=4) (actual time=2.429..2.445 rows=400 loops=1)"

"   Sort Key: employee.dno"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=394.65..676.01 rows=132 width=4) (actual time=2.232..2.362 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Bitmap Heap Scan on employee (cost=7.36..287.65 rows=397 width=4) (actual time=0.027..0.074 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Heap Blocks: exact=10"

"   -> Bitmap Index Scan on emp_salary_btree (cost=0.00..7.26 rows=397 width=0) (actual time=0.021..0.022 rows=400 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=386.66..386.66 rows=50 width=4) (actual time=2.199..2.200 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> GroupAggregate (cost=0.29..386.16 rows=50 width=4) (actual time=0.025..2.178 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   -> Index Only Scan using emp_dno_btree on employee employee_1 (cost=0.29..304.29 rows=16000 width=4) (actual time=0.007..1.144 rows=16000 loops=1)"

"   Heap Fetches: 0"

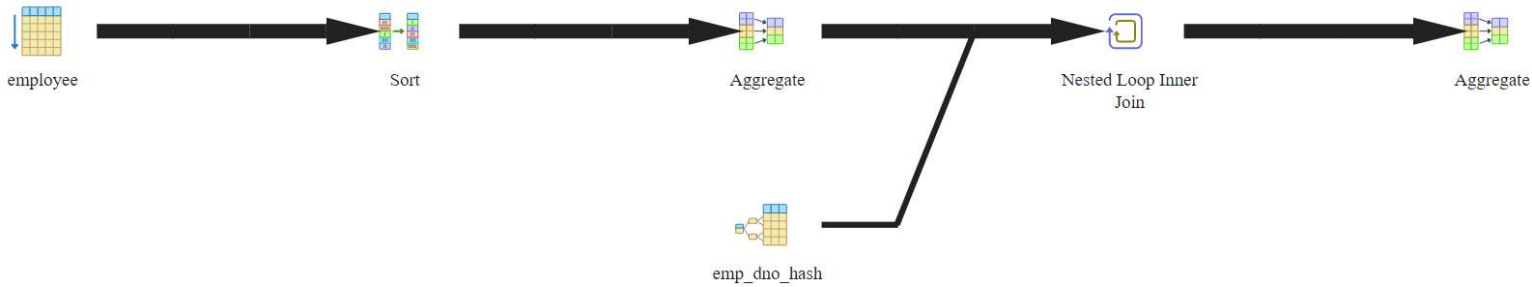
"Planning Time: 0.215 ms"

"Execution Time: 2.559 ms"

In this query we used two indexes Btree on employee salary and Btree on employee dno it gave us a cost of 682.97 which is better than the row optimized query because Btree is efficient in range queries and exact value queries.

6. Optimised + Hash:

Plan:



Analyze:

"GroupAggregate (cost=10000001540.26..10000002977.12 rows=132 width=12) (actual time=14.412..26.374 rows=100 loops=1)"

" Group Key: employee.dno"

" -> Nested Loop (cost=10000001540.26..10000002975.14 rows=132 width=4) (actual time=14.242..26.212 rows=400 loops=1)"

"   -> GroupAggregate (cost=10000001540.26..10000001662.14 rows=50 width=4) (actual time=7.215..12.034 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   -> Sort (cost=10000001540.26..10000001580.26 rows=16000 width=4) (actual time=7.172..8.573 rows=16000 loops=1)"

"   Sort Key: employee_1.dno"

"   Sort Method: quicksort Memory: 1135kB"

"   -> Seq Scan on employee employee_1 (cost=10000000000.00..10000000423.00 rows=16000 width=4) (actual time=0.027..3.630 rows=16000 loops=1)"

"   -> Index Scan using emp_dno_hash on employee (cost=0.00..26.22 rows=3 width=4) (actual time=0.088..0.093 rows=3 loops=150)"

"   Index Cond: (dno = employee_1.dno)"

"   Filter: (salary > 40000)"

"   Rows Removed by Filter: 104"

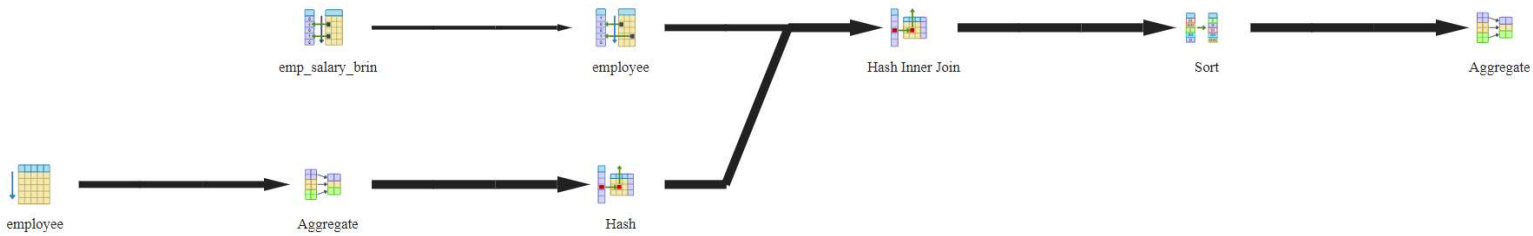
"Planning Time: 0.324 ms"

"Execution Time: 26.455 ms"

In this query we have a very high cost because seqscan was disabled to force the engine to use hash index, the cost is 10000001580.26 but it is not the real cost the engine adds 10 billion because the seqscan is closed the real cost is 1580.26 and as we can it is slower than the raw optimized query that's why the engine prefers seqscan over it.

6. Optimised + Brin:

Plan:



Analyze:

"GroupAggregate (cost=871.88..874.19 rows=132 width=12) (actual time=5.650..5.715 rows=100 loops=1)"

" Group Key: employee.dno"

" -> Sort (cost=871.88..872.21 rows=132 width=4) (actual time=5.641..5.658 rows=400 loops=1)"

"   Sort Key: employee.dno"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=518.14..867.23 rows=132 width=4) (actual time=4.492..5.581 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Bitmap Heap Scan on employee (cost=12.14..360.15 rows=397 width=4) (actual time=0.060..1.086 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Rows Removed by Index Recheck: 7408"

"   Heap Blocks: lossy=128"

"   -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=0.049..0.049 rows=1280 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=505.38..505.38 rows=50 width=4) (actual time=4.427..4.428 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> HashAggregate (cost=503.00..504.88 rows=50 width=4) (actual time=4.385..4.404 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   Batches: 1 Memory Usage: 48kB"

"   -> Seq Scan on employee employee_1 (cost=0.00..423.00 rows=16000 width=4) (actual time=0.010..1.045 rows=16000 loops=1)"

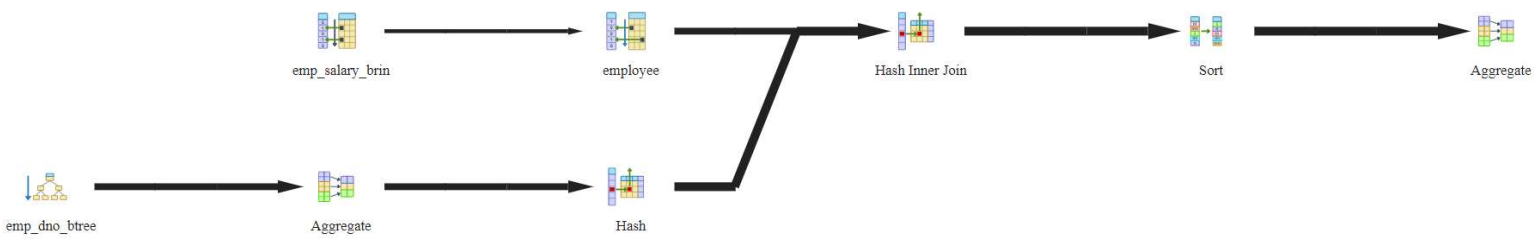
"Planning Time: 0.182 ms"

"Execution Time: 5.792 ms"

The cost of this query is 874.19 which better than the raw optimized query but worse than the optimized query with btree that's because brin is useful with range queries but not as powerful as btree.

6. Optimised + Mix:

Plan:



Analyze:

"GroupAggregate (cost=753.16..755.47 rows=132 width=12) (actual time=7.661..7.786 rows=100 loops=1)"

" Group Key: employee.dno"

" -> Sort (cost=753.16..753.49 rows=132 width=4) (actual time=7.652..7.685 rows=400 loops=1)"

"   Sort Key: employee.dno"

"   Sort Method: quicksort Memory: 43kB"

"   -> Hash Join (cost=399.43..748.51 rows=132 width=4) (actual time=5.341..7.478 rows=400 loops=1)"

"   Hash Cond: (employee.dno = employee_1.dno)"

"   -> Bitmap Heap Scan on employee (cost=12.14..360.15 rows=397 width=4) (actual time=0.135..2.136 rows=400 loops=1)"

"   Recheck Cond: (salary > 40000)"

"   Rows Removed by Index Recheck: 7408"

"   Heap Blocks: lossy=128"

"   -> Bitmap Index Scan on emp_salary_brin (cost=0.00..12.04 rows=6801 width=0) (actual time=0.085..0.085 rows=1280 loops=1)"

"   Index Cond: (salary > 40000)"

"   -> Hash (cost=386.66..386.66 rows=50 width=4) (actual time=5.176..5.177 rows=150 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 14kB"

"   -> GroupAggregate (cost=0.29..386.16 rows=50 width=4) (actual time=0.099..5.105 rows=150 loops=1)"

"   Group Key: employee_1.dno"

"   Filter: (count(*) > 5)"

"   -> Index Only Scan using emp_dno_btree on employee employee_1 (cost=0.29..304.29 rows=16000 width=4) (actual time=0.054..2.659 rows=16000 loops=1)"

"   Heap Fetches: 0"

"Planning Time: 0.920 ms"

"Execution Time: 7.959 ms"

cost in this query is 755.47 which is better than all other indexes except btree because in this query it is always better to use btree on both columns employee dno and employee salary rather than using a mix of two different indexes

Conclusion: in this query btree is the best index to use due to its powerful speed with range queries and exact value queries.

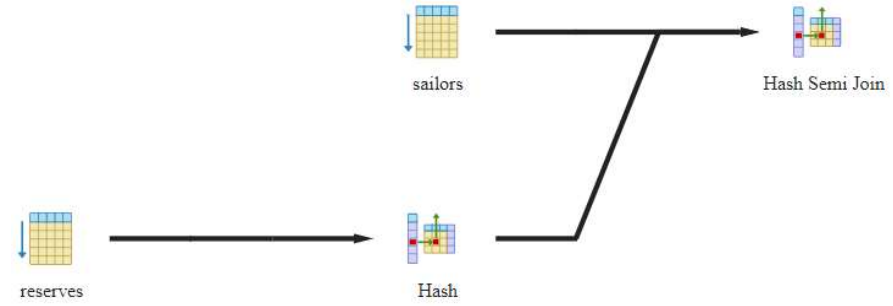
Schema 3

Query 7

Find the names of sailors who have reserved boat 103.

```
select s.sname
from sailors s
where
s.sid in( select r.sid
        from reserves r
        where r.bid = 103 );
```

when we analyzed this raw query without any indexes, we got the following plan:



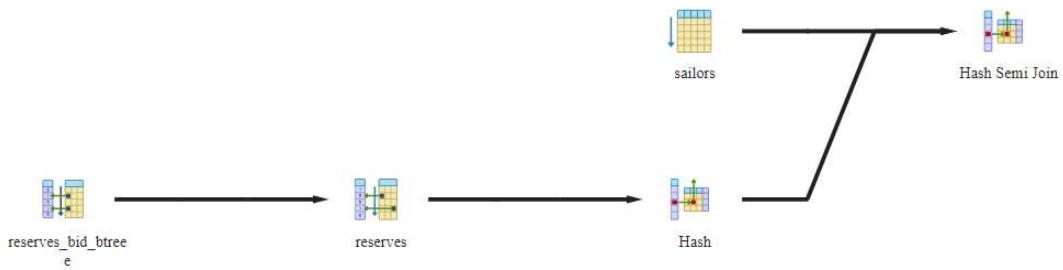
And the analysis for the plan was following:

"Hash Semi Join ***(cost=630.33..1032.00 rows=226 width=21)*** *(actual time=1.792..4.189 rows=225 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.011..1.015 rows=19000 loops=1)"
" -> Hash (cost=627.50..627.50 rows=226 width=4) (actual time=1.773..1.775 rows=225 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Seq Scan on reserves r (cost=0.00..627.50 rows=226 width=4) (actual time=0.008..1.753 rows=225 loops=1)"
" Filter: (bid = 103)"
" Rows Removed by Filter: 34775"
"Planning Time: 0.095 ms"
"Execution Time: 4.217 ms"

Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

7.Btree Index

When we add the Btree index for the *bid* column in both the *sailors* and *reserves* tables, the engine decides to use only the index on the *reserves* table and uses a normal seqscan on the *sailors* table, giving the following plan:



Giving a much better cost than the raw query of 613 and about half the execution time.

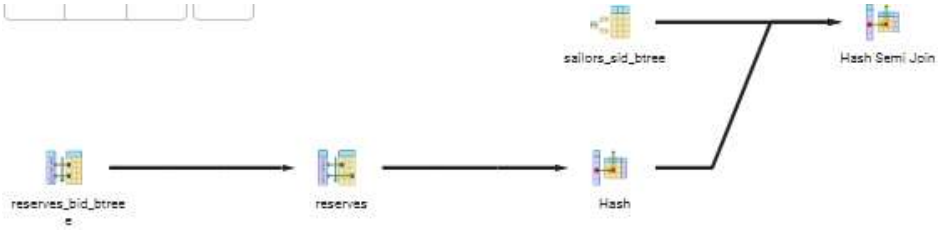
"Hash Semi Join ***(cost=211.41..613.08 rows=226 width=21)*** *(actual time=0.067..2.070 rows=225 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.007..0.844 rows=19000 loops=1)"
" -> Hash (cost=208.59..208.59 rows=226 width=4) (actual time=0.055..0.056 rows=225 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Bitmap Heap Scan on reserves r (cost=6.04..208.59 rows=226 width=4) (actual time=0.017..0.037 rows=225 loops=1)"

" Recheck Cond: (bid = 103)"
" Heap Blocks: exact=13"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..5.99 rows=226 width=0) (actual time=0.008..0.008 rows=225 loops=1)"
" Index Cond: (bid = 103)"
"Planning Time: 0.185 ms"
"Execution Time: 2.099 ms"

In order to force the Btree index on both of the tables, we disabled the seqscan flag by executing

Set enable_seqscan = off;

And re-executing the query to see the change in the plan and analysis



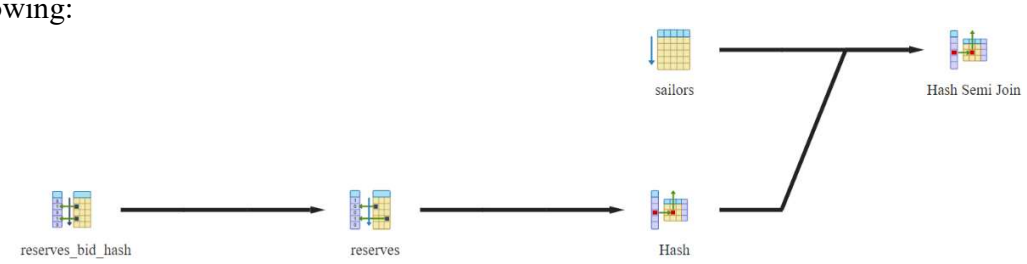
And the plan is the analysis for this following:

"Hash Semi Join
(cost=211.70..927.37 rows=226 width=21) *(actual time=0.069..3.158 rows=225 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.006..1.961 rows=19000 loops=1)"
" -> Hash (cost=208.59..208.59 rows=226 width=4) (actual time=0.058..0.060 rows=225 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Bitmap Heap Scan on reserves r (cost=6.04..208.59 rows=226 width=4) (actual time=0.015..0.040 rows=225 loops=1)"
" Recheck Cond: (bid = 103)"
" Heap Blocks: exact=13"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..5.99 rows=226 width=0) (actual time=0.008..0.008 rows=225 loops=1)"
" Index Cond: (bid = 103)"
"Planning Time: 0.172 ms"
"Execution Time: 3.189 ms"

It can be noticed here that when we tried to enforce the Btree index on the sailors table by disabling the seqscan, an increase in the cost (927.37) and the execution time (3.189ms) of the query occurs, however, it still gives better performance than the raw query itself.

7.Hash Index

Now we will build Hash indices on the bid columns and record the new performance. The resulting plan looks as following:

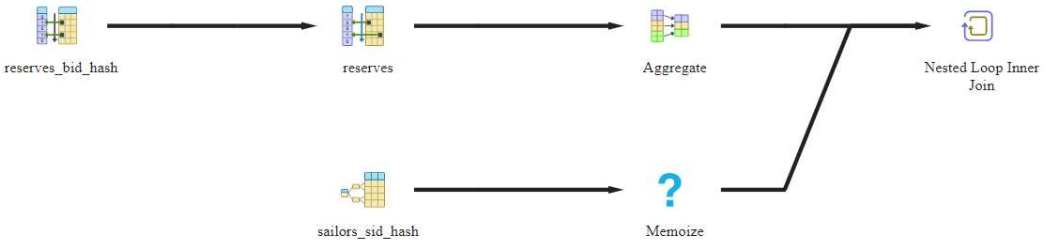


And the corresponding analysis for this plan is the following:

"Hash Semi Join ***(cost=211.12..612.79 rows=226 width=21)*** *(actual time=0.076..2.006 rows=225 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.008..0.823 rows=19000 loops=1)"
" -> Hash (cost=208.30..208.30 rows=226 width=4) (actual time=0.064..0.064 rows=225 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Bitmap Heap Scan on reserves r (cost=5.75..208.30 rows=226 width=4) (actual time=0.021..0.045 rows=225 loops=1)"
" Recheck Cond: (bid = 103)"
" Heap Blocks: exact=13"
" -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..5.70 rows=226 width=0) (actual time=0.012..0.012 rows=225 loops=1)"
" Index Cond: (bid = 103)"
"Planning Time: 0.121 ms"
"Execution Time: 2.032 ms"

The cost and execution time here are a little bit better than using Btree with Seqscan and both are much better than the raw query.
Next we will build Hash indices on the bid columns and record the new performance, however this time we will enforce using the hash index by disabling the seqscan. The resulting plan looks as following:

And the



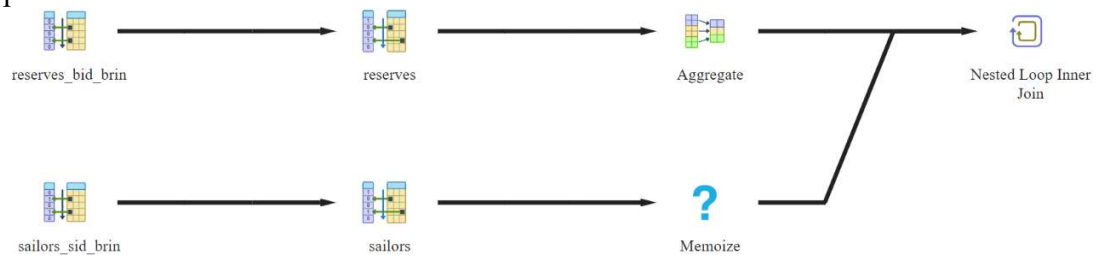
corresponding analysis for this plan is as follows:

*"Nested Loop **(cost=208.87..1009.16 rows=226 width=21)** (actual time=0.125..0.568 rows=225 loops=1)"*
" -> HashAggregate (cost=208.86..211.11 rows=225 width=4) (actual time=0.115..0.134 rows=225 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Bitmap Heap Scan on reserves r (cost=5.75..208.30 rows=226 width=4) (actual time=0.029..0.068 rows=225 loops=1)"
" Recheck Cond: (bid = 103)"
" Heap Blocks: exact=13"
" -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..5.70 rows=226 width=0) (actual time=0.019..0.019 rows=225 loops=1)"
" Index Cond: (bid = 103)"
" -> Memoize (cost=0.01..3.55 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=225)"
" Cache Key: r.sid"
" Cache Mode: logical"
" Hits: 0 Misses: 225 Evictions: 0 Overflows: 0 Memory Usage: 29kB"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..3.54 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=225)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.132 ms"
"Execution Time: 0.630 ms"

The previous analysis shows that the Hash index has a cost of 1009.16 which is a little less than the cost of the raw query, however it has a much lower execution time (0.63ms) than of the raw query (4.217ms).

7.BRIN Index

Now using a BRIN index on the specified columns, we get a plan where only seqscans are used on both tables, so in order to force the engine to use the BRIN index we disabled the seqscan and we got the following plan:



And the analysis for this plan is the following:

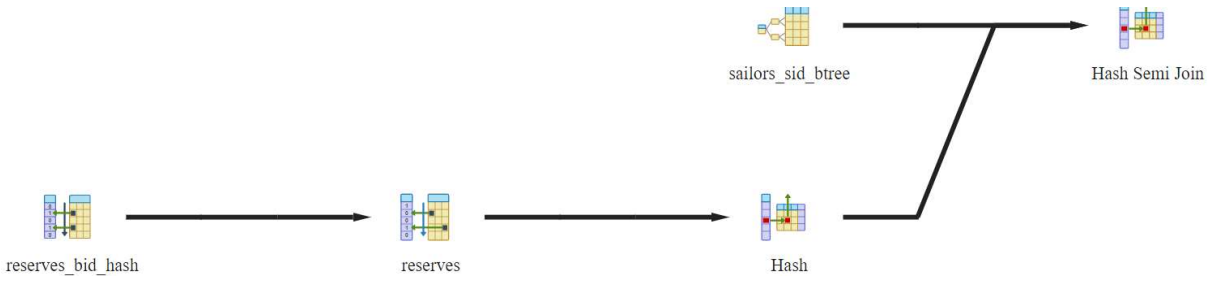
*"Nested Loop **(cost=3340.23..632976.26 rows=226 width=21)** (actual time=2.640..259.257 rows=225 loops=1)"*
" -> HashAggregate (cost=640.19..642.44 rows=225 width=4) (actual time=2.561..2.710 rows=225 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Bitmap Heap Scan on reserves r (cost=12.12..639.62 rows=226 width=4) (actual time=0.053..2.522 rows=225 loops=1)"
" Recheck Cond: (bid = 103)"
" Rows Removed by Index Recheck: 34775"
" Heap Blocks: lossy=190"
" -> Bitmap Index Scan on reserves_bid_brin (cost=0.00..12.06 rows=35000 width=0) (actual time=0.047..0.047 rows=1900 loops=1)"
" Index Cond: (bid = 103)"

" -> Memoize (cost=2700.04..2822.79 rows=1 width=25) (actual time=0.059..1.139 rows=1 loops=225)"
" Cache Key: r.sid"
" Cache Mode: logical"
" Hits: 0 Misses: 225 Evictions: 0 Overflows: 0 Memory Usage: 29kB"
" -> Bitmap Heap Scan on sailors s (cost=2700.03..2822.78 rows=1 width=25) (actual time=0.051..1.131 rows=1 loops=225)"
" Recheck Cond: (sid = r.sid)"
" Rows Removed by Index Recheck: 15307"
" Heap Blocks: lossy=28703"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..2700.03 rows=9500 width=0) (actual time=0.010..0.010 rows=1276 loops=225)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.146 ms"
"Execution Time: 259.455 ms"

It's obvious here that the cost and the execution time are worse than all the previous plans with and without indices, the reason behind the bad performance here is that the query doesn't involve any analytical data or calculations, where the BRIN is at the strongest, so here there was no reason to use a BRIN index in the first place, and that's why the engine decided to go with the seqscan from the beginning before we forced shutting it.

7.Mixed Indices

Finally, we will allow all the 3 types of indices on both tables and let the engine decide which combination of indices works best. Running the query with the seqscan enabled would result in choosing seqscan for the sailors table and using hash index for the reserves table, which basically the same plan we recorded before when we were testing the hash index alone, that means the engine finds it the most optimal one.
In order to test for other indices that seqscan, we would disable it and let the engine decide which index to use among the remaining 3 indices, and it resulted in the following plan:



And the analysis of the plan is the following:
"Hash Semi Join ***(cost=211.41..927.08 rows=226 width=21)*** *(actual time=0.122..4.658 rows=225 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.009..2.594 rows=19000 loops=1)"
" -> Hash (cost=208.30..208.30 rows=226 width=4) (actual time=0.107..0.108 rows=225 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Bitmap Heap Scan on reserves r (cost=5.75..208.30 rows=226 width=4) (actual time=0.028..0.077 rows=225 loops=1)"
" Recheck Cond: (bid = 103)"
" Heap Blocks: exact=13"
" -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..5.70 rows=226 width=0) (actual time=0.015..0.015 rows=225 loops=1)"
" Index Cond: (bid = 103)"
"Planning Time: 0.340 ms"
"Execution Time: 4.698 ms"

Conclusion:

Using seqscan on the sailors table and a hash index on the reserves table would give the least cost and fastest execution time, which makes sense since we used the hash index on the column that has an exact value query "where r.bid = 103", and seqscan was used on the sailors table that is used for joining as we will go through all the rows in it to do the joining.

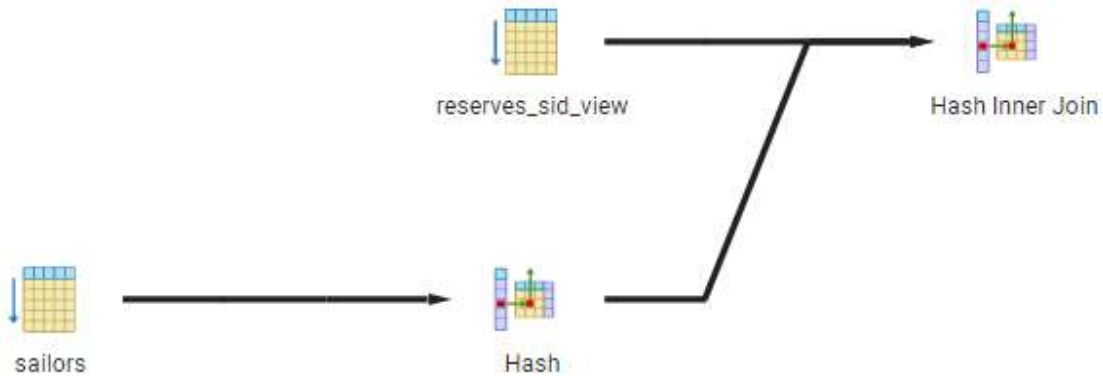
Query 7 optimization

Query 7 is too simple to be optimized by writing another equivalent query in terms of SQL, so in order to optimize it without changing the query itself we will create a materialized view on the inner query that will be used later. The idea behind using a materialized view is that the inner query will be executed and stored to disk only once the view is created. so each time Query 7 is executed, the engine would do the join on the view without having to re-calculate the inner query every single time, which saves a lot of time and computational power and reduces the cost of the query.

The new optimized query would be:

```
create materialized view reserves_sid_view as
( select r.sid
from reserves r
where r.bid = 103 );
select s.sname
from sailors s inner join reserves_sid_view on s.sid = reserves_sid_view.sid
```

The plan for the raw optimized query is the following:



And the analysis for this plan is the following:

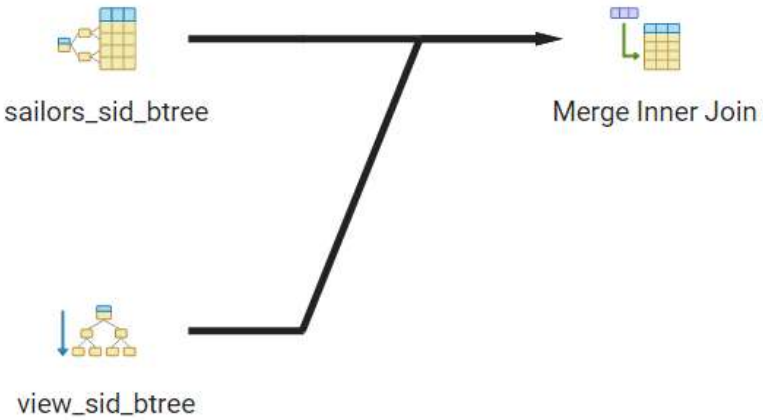
```
"Hash Join  (cost=6.06..428.56 rows=225 width=21) (actual time=0.048..1.997 rows=225 loops=1)"
"  Hash Cond: (s.sid = reserves_sid_view.sid)"
"  ->  Seq Scan on sailors s  (cost=0.00..349.00 rows=19000 width=25) (actual time=0.007..0.805 rows=19000 loops=1)"
"  ->  Hash  (cost=3.25..3.25 rows=225 width=4) (actual time=0.037..0.037 rows=225 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"        ->  Seq Scan on reserves_sid_view  (cost=0.00..3.25 rows=225 width=4) (actual time=0.008..0.016 rows=225 loops=1)"
"Planning Time: 0.108 ms"
"Execution Time: 2.013 ms"
```

This optimized version of query 7 greatly reduced cost and execution time, as the cost was 1032 and dropped to 428.56, and the execution time was 4.217ms and dropped down to 2.013ms.

Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

7opt.Btree Index

We will construct Btree indices on the columns that we are using while executing this query, namely sailors(sid), view(sid). The engine decided to use seqscans instead of the Btree indices, which means that the indices will not be helpful for this query. However, we disabled the seqscan and run the query again in order to record the performance of the Btree indices. The following plan is returned when executing:



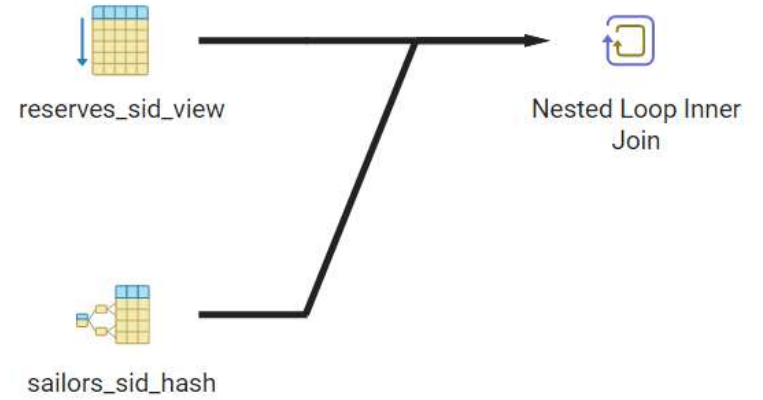
And the analysis of the plan is the following:

"Merge Join ***(cost=0.47..695.58*** *rows=225 width=21)* *(actual time=0.033..2.881 rows=225 loops=1)"*
" Merge Cond: (s.sid = reserves_sid_view.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.015..1.921 rows=18104 loops=1)"
" -> Index Only Scan using view_sid_btree on reserves_sid_view (cost=0.14..15.52 rows=225 width=4) (actual time=0.009..0.045 rows=225 loops=1)"
" Heap Fetches: 225"
"Planning Time: 0.365 ms"
"Execution Time: 2.943 ms"

As expected, the cost is higher than the raw optimized query, because the engine knows that the seqscan is already faster than the Btree indices.

7opt.hash Index

Now we will build Hash indices on the columns and record the new performance. The resulting plan looks as following:



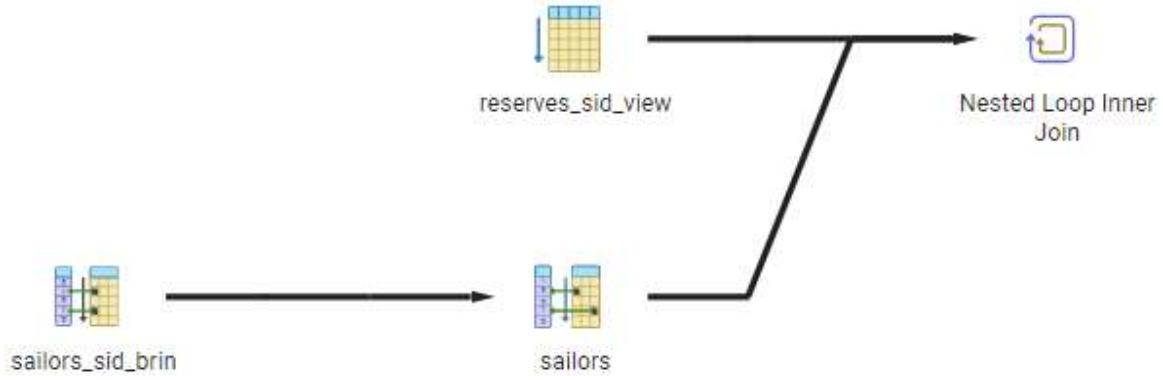
And the analysis for this plan is the following:

"Nested Loop
(cost=10000000000.00..10000000801.44 rows=225 width=21) *(actual time=0.020..0.312 rows=225 loops=1)"*
" -> Seq Scan on reserves_sid_view (cost=10000000000.00..10000000003.25 rows=225 width=4) (actual time=0.013..0.021 rows=225 loops=1)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..3.54 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=225)"
" Index Cond: (sid = reserves_sid_view.sid)"
"Planning Time: 0.092 ms"
"Execution Time: 0.329 ms"

This non-logical number of the cost is not actually a “real” number, it’s jut an indication that this query cannot be executed without a seqscan, so it just increases the cost dramatically as it’s gonna use seqscan anyway, so obviously this query can’t be executed with hash indices only.

7opt.BRIN Index

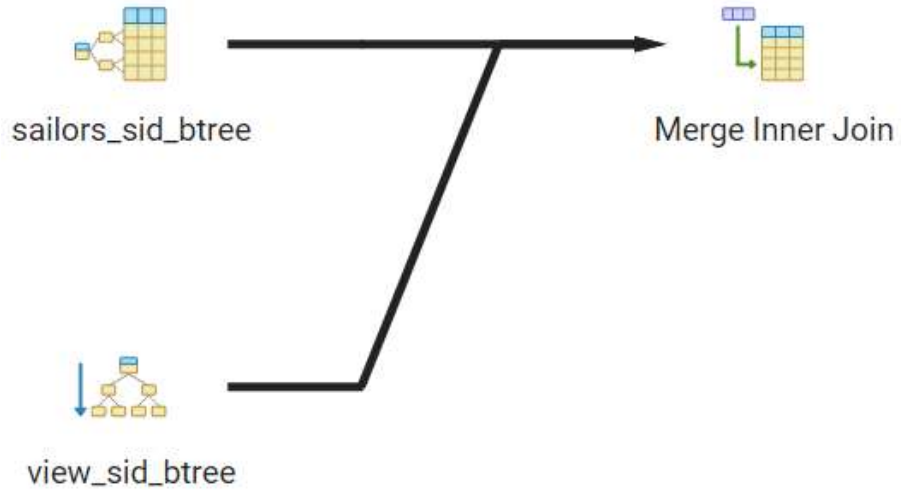
The case with the BRIN index is exactly the same as the hash index, it doesn't work as well without having to use the seqscan, so the plan has to include the seqscan and the cost would be stupidly high as well.



"Nested Loop ***(cost=10000002700.03..10000635131.51 rows=225 width=21)*** *(actual time=0.031..272.000 rows=225 loops=1)"*
" -> Seq Scan on reserves_sid_view (cost=10000000000.00..10000000003.25 rows=225 width=4) (actual time=0.009..0.108 rows=225 loops=1)"
" -> Bitmap Heap Scan on sailors s (cost=2700.03..2822.78 rows=1 width=25) (actual time=0.050..1.200 rows=1 loops=225)"
" Recheck Cond: (sid = reserves_sid_view.sid)"
" Rows Removed by Index Recheck: 15307"
" Heap Blocks: lossy=28703"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..2700.03 rows=9500 width=0) (actual time=0.010..0.010 rows=1276 loops=225)"
" Index Cond: (sid = reserves_sid_view.sid)"
"Planning Time: 0.100 ms"
"Execution Time: 272.195 ms"

7opt.Mix Index

Now we will start mixing indices together and let the engine decide the most optimal combination of the indices, accordingly, the engine generated the following plan:



And the above plan is analysis of the the following:

"Merge Join ***(cost=0.47..695.58 rows=225 width=21)*** *(actual time=0.011..2.622 rows=225 loops=1)"*
" Merge Cond: (s.sid = reserves_sid_view.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.005..1.790 rows=18104 loops=1)"
" -> Index Only Scan using view_sid_btree on reserves_sid_view (cost=0.14..15.52 rows=225 width=4) (actual time=0.003..0.039 rows=225 loops=1)"
" Heap Fetches: 225"
"Planning Time: 0.161 ms"
"Execution Time: 2.639 ms"

Conclusion:

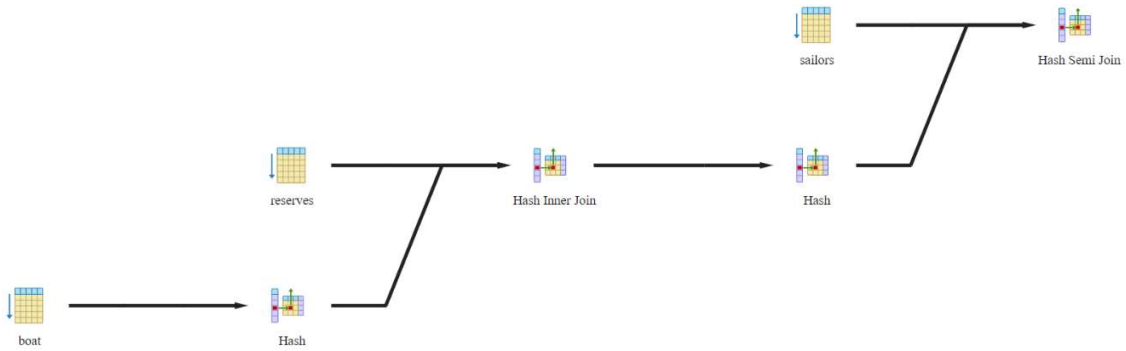
creating the materialized view was enough to optimize the query without adding more indices, because by comparing the cost and execution time of the raw query "using only seqscans", the numbers are much better than using any other combination of indices.

Query 8

Find the names of sailors who have reserved a red boat.

select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));

executing this query without adding any indices and with removing any automated primary key indices, we would get the following plan:



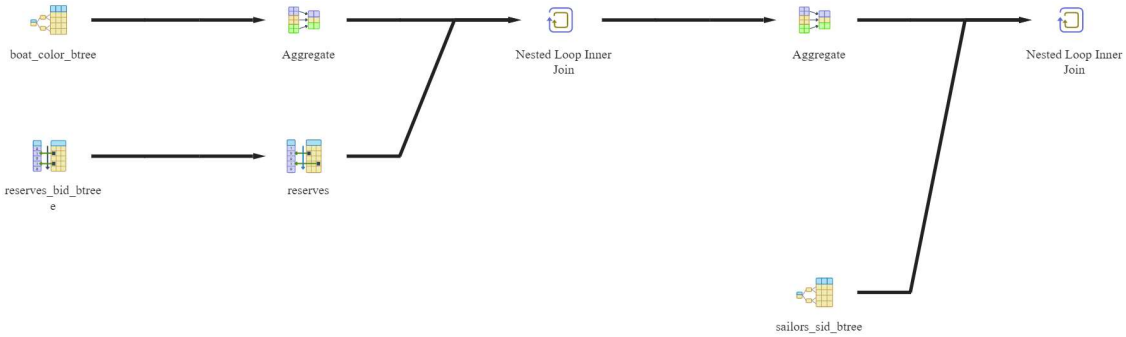
and the corresponding analysis of the plan is the following:

"Hash Semi Join ***(cost=699.86..1101.20 rows=222 width=21)*** *(actual time=4.439..6.234 rows=209 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.015..0.858 rows=19000 loops=1)"
" -> Hash (cost=697.08..697.08 rows=222 width=4) (actual time=4.165..4.167 rows=209 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Hash Semi Join (cost=62.74..697.08 rows=222 width=4) (actual time=0.797..4.137 rows=209 loops=1)"
" Hash Cond: (r.bid = b.bid)"
" -> Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.016..1.533 rows=35000 loops=1)"
" -> Hash (cost=62.50..62.50 rows=19 width=4) (actual time=0.462..0.463 rows=19 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 9kB"
" -> Seq Scan on boat b (cost=0.00..62.50 rows=19 width=4) (actual time=0.014..0.451 rows=19 loops=1)"
" Filter: (color = 'red'::bpchar)"
" Rows Removed by Filter: 2981"
"Planning Time: 0.229 ms"
"Execution Time: 6.269 ms"

Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

8.Btree Index

We will construct Btree indices on the columns that we are using while executing this query, namely sailors(sid), reserves(sid, bid), and boat(bid, color). The following plan is returned when executing:



And the corresponding analysis of the is plan is the following:

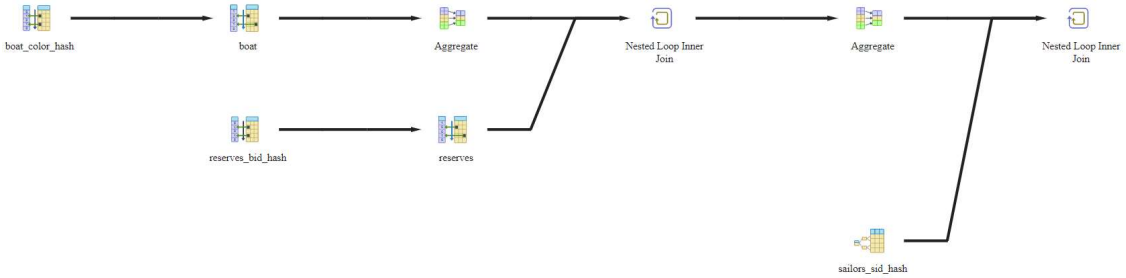
"Nested Loop ***(cost=604.38..686.30 rows=222 width=21)*** *(actual time=0.307..0.601 rows=209 loops=1)"*

" -> HashAggregate (cost=604.10..606.32 rows=222 width=4) (actual time=0.295..0.314 rows=209 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Nested Loop (cost=16.26..603.54 rows=222 width=4) (actual time=0.079..0.263 rows=209 loops=1)"
" -> HashAggregate (cost=12.51..12.70 rows=19 width=4) (actual time=0.064..0.067 rows=19 loops=1)"
" Group Key: b.bid"
" Batches: 1 Memory Usage: 24kB"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.054..0.057 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.75..30.98 rows=12 width=8) (actual time=0.003..0.006 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..3.75 rows=12 width=0) (actual time=0.002..0.002 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.270 ms"
"Execution Time: 0.665 ms"

Here we can notice a great increase in performance by adding the Btree indices, since the cost has been reduced to 686.30 and the execution time dropped from 6.269ms to 0.665ms.

8.Hash Index

Now we will build Hash indices on the same columns and record their performance. Running the query with the hash indices applied would execute the following plan:



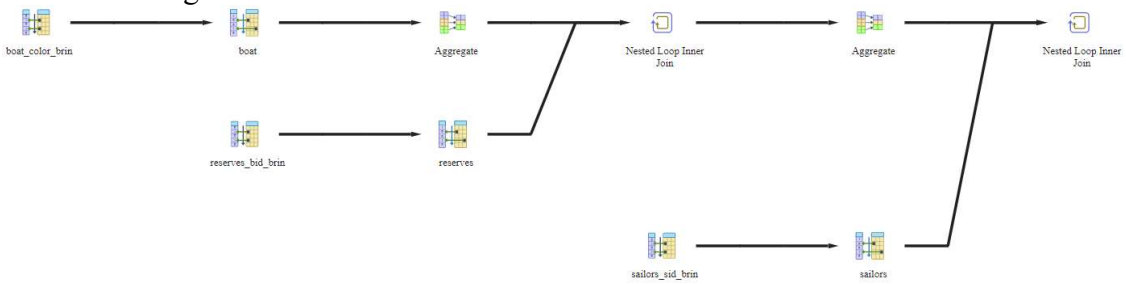
And the corresponding analysis for this plan is the following:

"Nested Loop ***(cost=623.08..642.02 rows=222 width=21)*** *(actual time=0.295..0.596 rows=209 loops=1)"*
" -> HashAggregate (cost=623.08..625.30 rows=222 width=4) (actual time=0.289..0.306 rows=209 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Nested Loop (cost=32.88..622.52 rows=222 width=4) (actual time=0.039..0.255 rows=209 loops=1)"
" -> HashAggregate (cost=29.00..29.19 rows=19 width=4) (actual time=0.028..0.031 rows=19 loops=1)"
" Group Key: b.bid"
" Batches: 1 Memory Usage: 24kB"
" -> Bitmap Heap Scan on boat b (cost=4.15..28.95 rows=19 width=4) (actual time=0.021..0.022 rows=19 loops=1)"
" Recheck Cond: (color = 'red'::bpchar)"
" Heap Blocks: exact=1"
" -> Bitmap Index Scan on boat_color_hash (cost=0.00..4.14 rows=19 width=0) (actual time=0.015..0.015 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.88..31.11 rows=12 width=8) (actual time=0.004..0.007 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..3.88 rows=12 width=0) (actual time=0.003..0.003 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.191 ms"
"Execution Time: 0.660 ms"

The hash indices performed even better that the Btree indices for reducing the cost, however the reduction in execution time was not as great, but still performed much better than the raw query.

8.BRIN Index

Now we will build BRIN indices on the same columns and record their performance. Running the query with the BRIN indices applied and with seqscan enabled would return the same plan used for the raw query, which indicates that using BRIN indices for this query is not efficient, however, we will disable the seqscan and record the analysis and see how bad would the BRIN indices perform. The plan that uses BRIN indices is the following:



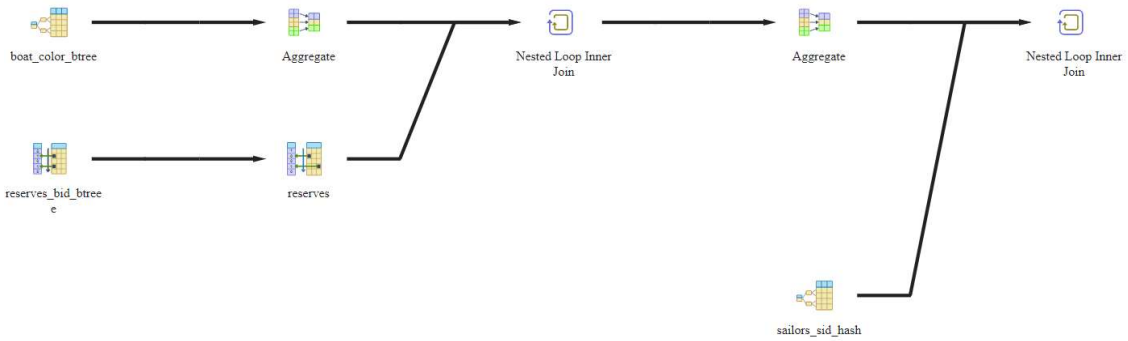
The analysis for this plan is the following:

"Nested Loop ***(cost=239036.65..50142462.71 rows=222 width=21)*** *(actual time=56.851..304.602 rows=209 loops=1)"*
" -> HashAggregate (cost=13352.62..13354.84 rows=222 width=4) (actual time=56.158..56.360 rows=209 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Nested Loop (cost=302.65..13352.06 rows=222 width=4) (actual time=0.600..56.027 rows=209 loops=1)"
" -> HashAggregate (cost=74.58..74.77 rows=19 width=4) (actual time=0.324..0.347 rows=19 loops=1)"
" Group Key: b.bid"
" Batches: 1 Memory Usage: 24kB"
" -> Bitmap Heap Scan on boat b (cost=12.04..74.54 rows=19 width=4) (actual time=0.023..0.320 rows=19 loops=1)"
" Recheck Cond: (color = 'red'::bpchar)"
" Rows Removed by Index Recheck: 2981"
" Heap Blocks: lossy=25"
" -> Bitmap Index Scan on boat_color_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.015..0.016 rows=250 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=228.07..698.68 rows=12 width=8) (actual time=0.255..2.917 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Rows Removed by Index Recheck: 34989"
" Heap Blocks: lossy=3610"
" -> Bitmap Index Scan on reserves_bid_brin (cost=0.00..228.06 rows=35000 width=0) (actual time=0.012..0.012 rows=1900 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Bitmap Heap Scan on sailors s (cost=225684.03..225806.78 rows=1 width=25) (actual time=0.653..1.179 rows=1 loops=209)"
" Recheck Cond: (sid = r.sid)"
" Rows Removed by Index Recheck: 14294"
" Heap Blocks: lossy=24909"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..225684.03 rows=9500 width=0) (actual time=0.011..0.011 rows=1192 loops=209)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.264 ms"
"Execution Time: 304.898 ms"

In the analysis, we can notice the dramatic increase in both the cost and the execution time of this query, which shows how bad the BRIN index performed, which is expected since the query doesn't involve aggregate or analytical functions which the BRIN is good at, so the engine decides that normal seqscan would be much better building useless BRIN indices for this query.

8.Mix Indices

Finally, we will create the 3 types of indices on all the columns used in this query and we will let the engine decide which combination of indices on which columns would generate the most optimal plan.
the generated plan is as follows:



And the analysis for this plan is the following:

*"Nested Loop **(cost=604.10..623.04 rows=222 width=21)** (actual time=0.352..0.619 rows=209 loops=1)"*
" -> HashAggregate (cost=604.10..606.32 rows=222 width=4) (actual time=0.347..0.363 rows=209 loops=1)"
" Group Key: r.sid"
" Batches: 1 Memory Usage: 48kB"
" -> Nested Loop (cost=16.26..603.54 rows=222 width=4) (actual time=0.051..0.306 rows=209 loops=1)"
" -> HashAggregate (cost=12.51..12.70 rows=19 width=4) (actual time=0.037..0.041 rows=19 loops=1)"
" Group Key: b.bid"
" Batches: 1 Memory Usage: 24kB"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.029..0.031 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.75..30.98 rows=12 width=8) (actual time=0.004..0.008 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..3.75 rows=12 width=0) (actual time=0.002..0.002 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.461 ms"
"Execution Time: 0.662 ms"

Conclusion:

Using Btree indices on boat(color) and reserves(bid) while using a hash index on sailors(sid) generates the most optimal plan with the lowest cost of 624 and execution time of 0.662ms. obviously, no BRIN indices can be spotted in this plan as it already performed the worst on its own. The Btree and Hash indices each were good on it's own with a little better performance for the Hash indices, which fits the nature of this query as it contains many joins and an exact value query "where b.color = 'red'".

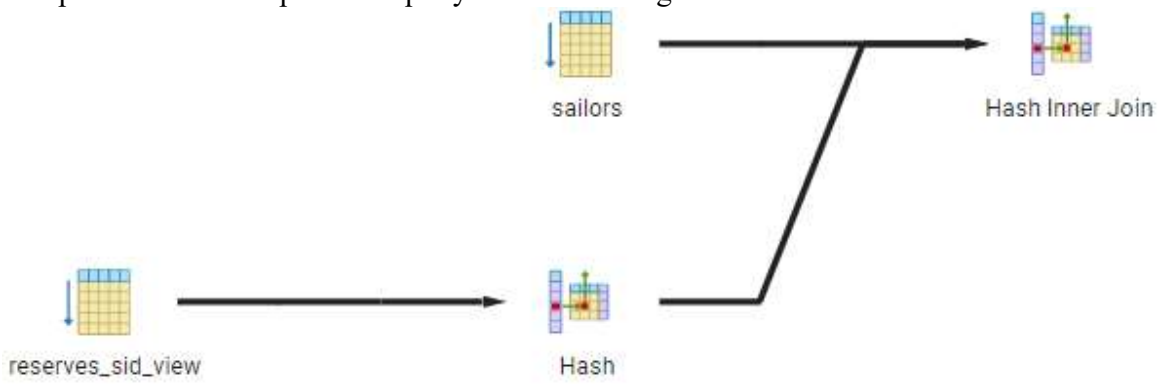
Query 8 optimization

Query 8 is too simple to be optimized by writing another equivalent query in terms of SQL, so in order to optimize it without changing the query itself we will create a materialized view on the inner query that will be used later. The idea behind using a materialized view is that the inner query will be executed and stored to disk only once the view is created. so each time Query 8 is executed, the engine would do the join on the view without having to re-calculate the inner query every single time, which saves a lot of time and computational power and reduces the cost of the query.

The new optimized query would be:

```
create materialized view reserves_sid_view as
select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red');
```

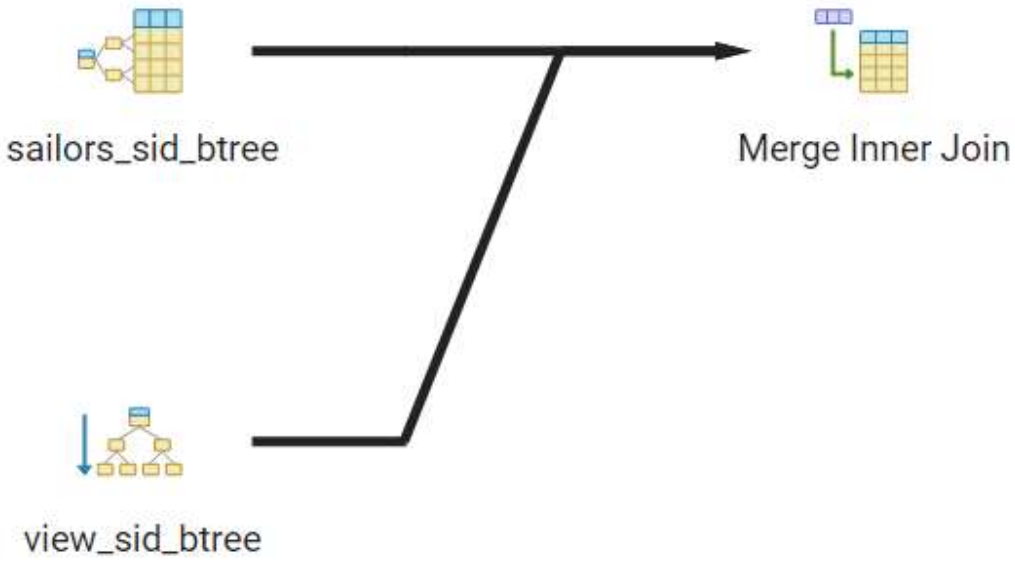
select s.sname
from sailors s inner join reserves_sid_view v on s.sid = v.sid;
The plan for the raw optimized query is the following:



And the analysis for this plan is the following:
"Hash Join ***(cost=5.70..428.04 rows=209 width=21)*** *(actual time=0.268..2.105 rows=209 loops=1)"*
" Hash Cond: (s.sid = v.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.009..0.884 rows=19000 loops=1)"
" -> Hash (cost=3.09..3.09 rows=209 width=4) (actual time=0.035..0.036 rows=209 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Seq Scan on reserves_sid_view v (cost=0.00..3.09 rows=209 width=4) (actual time=0.009..0.016 rows=209 loops=1)"
"Planning Time: 0.102 ms"
"Execution Time: 2.125 ms"

This optimized version of query 8 greatly reduced cost and execution time, as the cost was 1101.20 and dropped to 428.04, and the execution time was 6.269ms and dropped down to 2.125ms.
Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

8opt.Btree Index
We will construct Btree indices on the columns that we are using while executing this query, namely sailors(sid), view(sid). The engine decided to use seqscans instead of the Btree indices, which means that the indices will not be helpful for this query. However, we disabled the seqscan and run the query again in order to record the performance of the Btree indices. The following plan is returned when executing:



And the analysis of this plan is the following:
"Merge Join

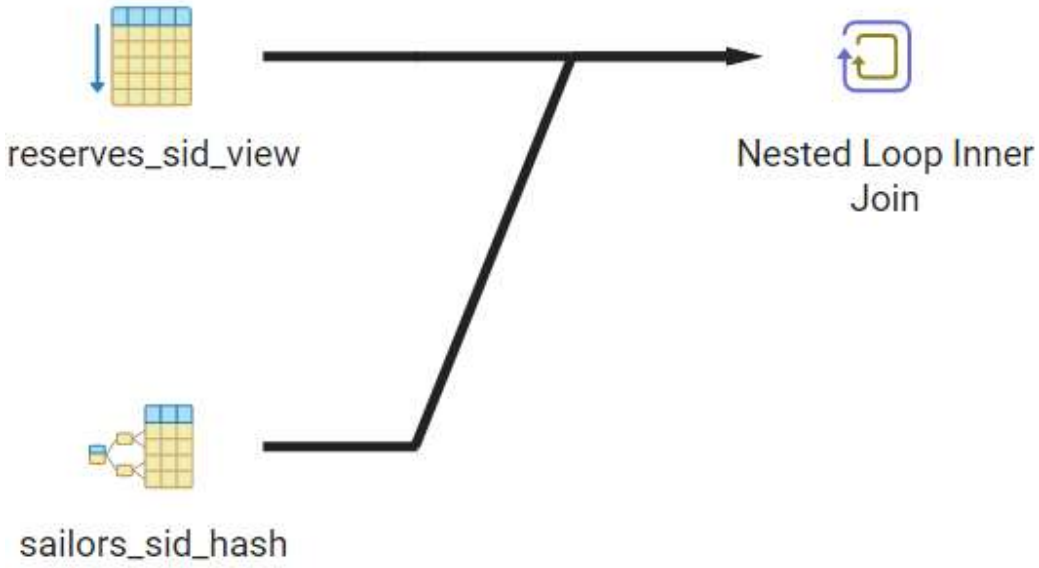
(cost=75.22..692.00 rows=209 width=21) *(actual time=0.306..2.590 rows=209 loops=1)"*

" Merge Cond: (s.sid = v.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.007..1.763 rows=18020 loops=1)"
" -> Index Only Scan using view_sid_btree on reserves_sid_view v (cost=0.14..15.28 rows=209 width=4) (actual time=0.009..0.043 rows=209 loops=1)"
" Heap Fetches: 209"
"Planning Time: 0.139 ms"
"Execution Time: 2.610 ms"

As expected, the cost is higher than the raw optimized query, because the engine knows that the seqscan is already faster than the Btree indices.

8opt.hash Index

Now we will build Hash indices on the columns and record the new performance. The resulting plan looks as following:



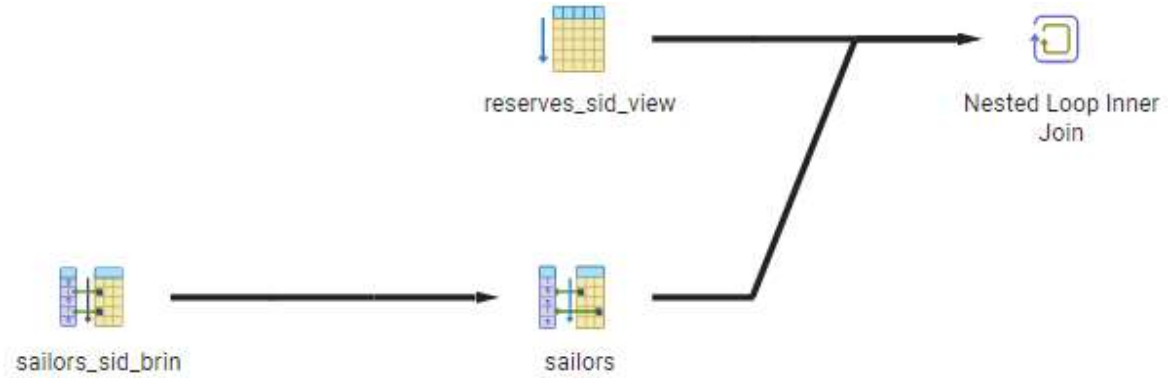
And the analysis for this plan is the following:

"Nested Loop ***(cost=10000000000.00..10000000780.84 rows=209 width=21)*** *(actual time=0.030..0.310 rows=209 loops=1)"*
" -> Seq Scan on reserves_sid_view v (cost=10000000000.00..10000000003.09 rows=209 width=4) (actual time=0.018..0.025 rows=209 loops=1)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..3.71 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = v.sid)"
"Planning Time: 0.116 ms"
"Execution Time: 0.329 ms"

This non-logical number of the cost is not actually a “real” number, it’s jut an indication that this query cannot be executed without a seqscan, so it just increases the cost dramatically as it’s gonna use seqscan anyway, so obviously this query can’t be executed with hash indices only.

8opt.BRIN Index

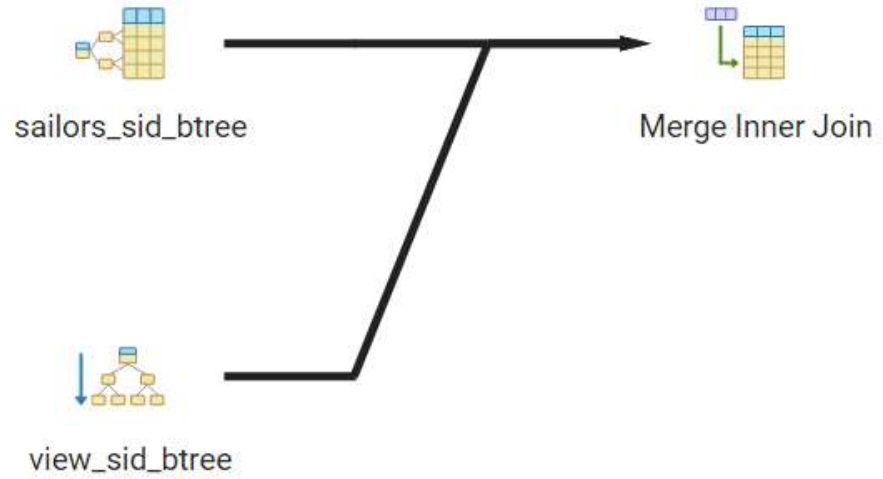
The case with the BRIN index is exactly the same as the hash index, it doesn’t work as well without having to use the seqscan, so the plan has to include the seqscan and the cost would be stupidly high as well.



*"Nested Loop **(cost=1000000250 8.03..1000054983 8.67 rows=209 width=21)** (actual time=0.249..229.5 70 rows=209 loops=1)"*
" -> Seq Scan on reserves_sid_view v (cost=10000000000.00..10000000003.09 rows=209 width=4) (actual time=0.010..0.082 rows=209 loops=1)"
" -> Bitmap Heap Scan on sailors s (cost=2508.03..2630.78 rows=1 width=25) (actual time=0.619..1.090 rows=1 loops=209)"
" Recheck Cond: (sid = v.sid)"
" Rows Removed by Index Recheck: 14294"
" Heap Blocks: lossy=24909"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..2508.03 rows=9500 width=0) (actual time=0.010..0.010 rows=1192 loops=209)"
" Index Cond: (sid = v.sid)"
"Planning Time: 0.098 ms"
"Execution Time: 229.803 ms"

8opt.Mix Index

Now we will start mixing indices together and let the engine decide the most optimal combination of the indices, accordingly, the engine generated the following plan:



And the analysis of this plan is the following:

*"Merge Join **(cost=75.22..692.00 rows=209 width=21)** (actual time=0.370..3.494 rows=209 loops=1)"*
" Merge Cond: (s.sid = v.sid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.022..2.425 rows=18020 loops=1)"
" -> Index Only Scan using view_sid_btree on reserves_sid_view v (cost=0.14..15.28 rows=209 width=4) (actual time=0.009..0.054 rows=209 loops=1)"
" Heap Fetches: 209"
"Planning Time: 0.147 ms"
"Execution Time: 3.516 ms"

Conclusion:

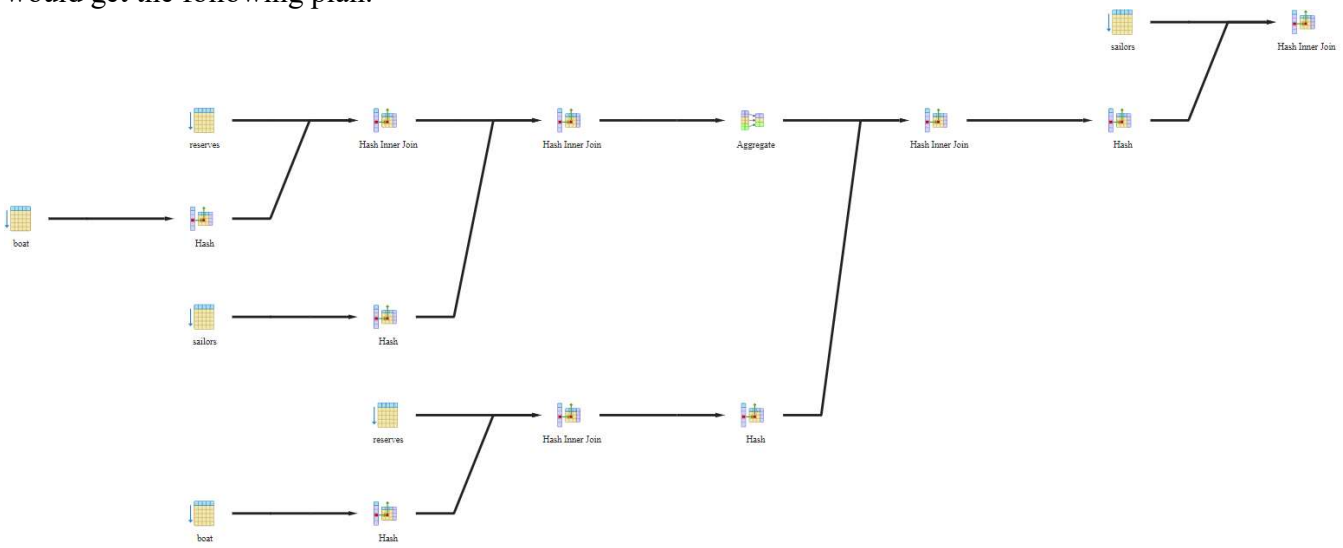
creating the materialized view was enough to optimize the query without adding more indices, because by comparing the cost and execution time of the raw query “using only seqscans”, the numbers are much better than using any other combination of indices.

Query 9

Find the names of sailors who have reserved both a red and a green boat.

```
select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid and r.bid = b.bid and b.color = 'red'
and s.sid in
        ( select s2.sid
        from sailors s2, boat b2, reserves r2
        where s2.sid = r2.sid
        and
        r2.bid = b2.bid
        and
        b2.color = 'green');
```

executing this query without adding any indices and with removing any automated primary key indices, we would get the following plan:



And the corresponding analysis of the plan is the following:

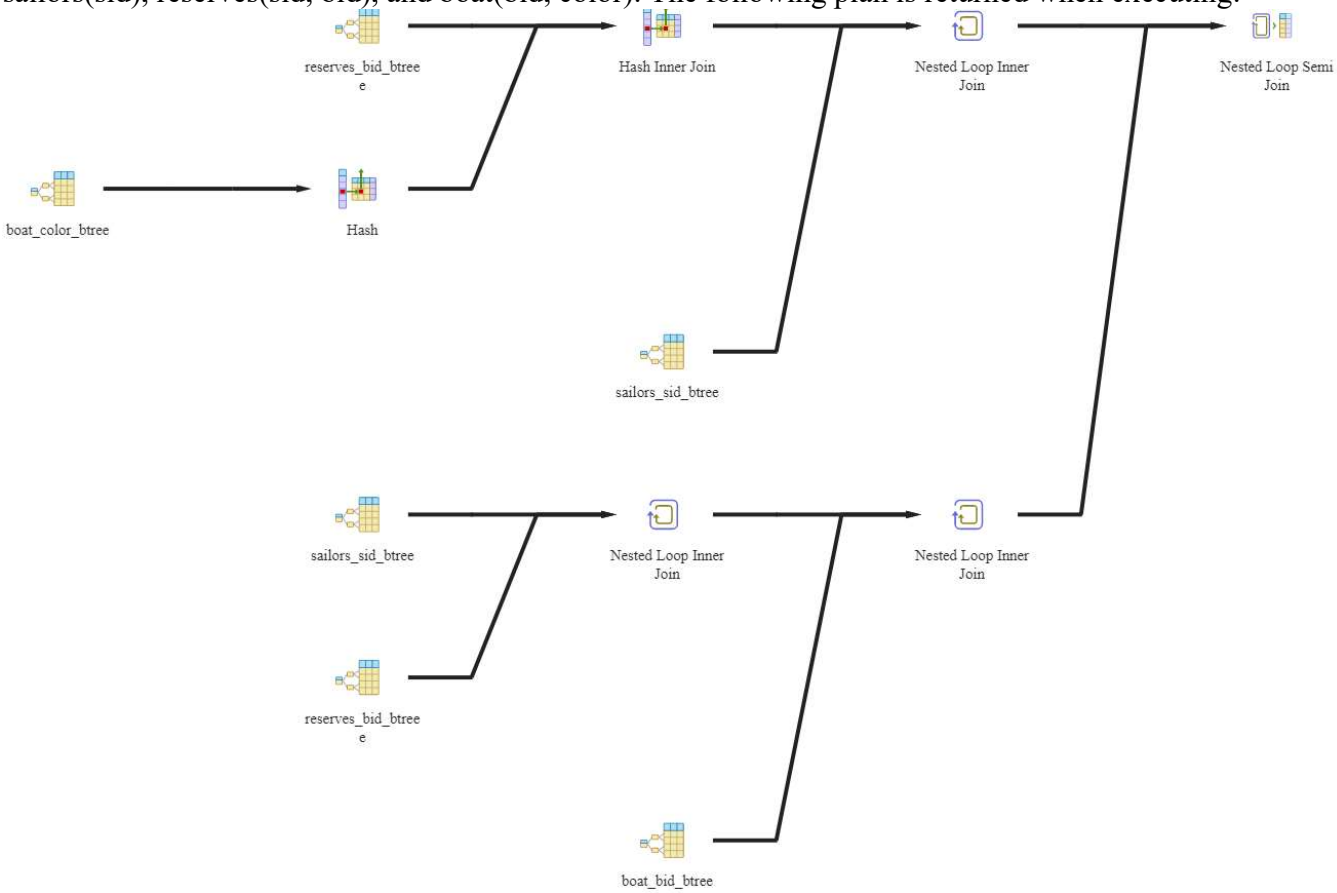
*"Hash Join **(cost=3275.66..3698.13 rows=222 width=21)** (actual time=41.438..43.367 rows=190 loops=1)"*
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.010..0.862 rows=19000 loops=1)"
" -> Hash (cost=3272.89..3272.89 rows=222 width=12) (actual time=41.209..41.213 rows=190 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 17kB"
" -> Hash Join (cost=3009.42..3272.89 rows=222 width=12) (actual time=37.872..41.175 rows=190 loops=1)"
" Hash Cond: (s2.sid = r.sid)"
" -> HashAggregate (cost=2270.43..2460.43 rows=19000 width=8) (actual time=33.406..35.706 rows=18981 loops=1)"
" Group Key: s2.sid"
" Batches: 1 Memory Usage: 1809kB"
" -> Hash Join (cost=686.26..2183.49 rows=34778 width=8) (actual time=4.709..25.338 rows=34791 loops=1)"
" Hash Cond: (r2.sid = s2.sid)"
" -> Hash Join (cost=99.76..1118.79 rows=34778 width=4) (actual time=1.285..11.994 rows=34791 loops=1)"
" Hash Cond: (r2.bid = b2.bid)"
" -> Seq Scan on reserves r2 (cost=0.00..540.00 rows=35000 width=8) (actual time=0.056..2.149 rows=35000 loops=1)"
" -> Hash (cost=62.50..62.50 rows=2981 width=4) (actual time=1.164..1.165 rows=2981 loops=1)"
" Buckets: 4096 Batches: 1 Memory Usage: 137kB"
" -> Seq Scan on boat b2 (cost=0.00..62.50 rows=2981 width=4) (actual time=0.050..0.700 rows=2981 loops=1)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 19"
" -> Hash (cost=349.00..349.00 rows=19000 width=4) (actual time=3.325..3.325 rows=19000 loops=1)"
" Buckets: 32768 Batches: 1 Memory Usage: 924kB"

"   -> Seq Scan on sailors s2 (cost=0.00..349.00 rows=19000 width=4) (actual time=0.015..1.346 rows=19000 loops=1)"
"   -> Hash (cost=736.21..736.21 rows=222 width=4) (actual time=4.398..4.399 rows=209 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 16kB"
"   -> Hash Join (cost=62.74..736.21 rows=222 width=4) (actual time=0.522..4.370 rows=209 loops=1)"
"   Hash Cond: (r.bid = b.bid)"
"   -> Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.006..1.619 rows=35000 loops=1)"
"   -> Hash (cost=62.50..62.50 rows=19 width=4) (actual time=0.214..0.214 rows=19 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 9kB"
"   -> Seq Scan on boat b (cost=0.00..62.50 rows=19 width=4) (actual time=0.006..0.210 rows=19 loops=1)"
"   Filter: (color = 'red'::bpchar)"
"   Rows Removed by Filter: 2981"
"Planning Time: 0.247 ms"
"Execution Time: 44.307 ms"

It can be noticed the cost (3698.13) and execution time (44.307ms) are really high.
Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

9.Btree Index

We will construct Btree indices on the columns that we are using while executing this query, namely sailors(sid), reserves(sid, bid), and boat(bid, color). The following plan is returned when executing:



And the corresponding analysis of this plan is the following:

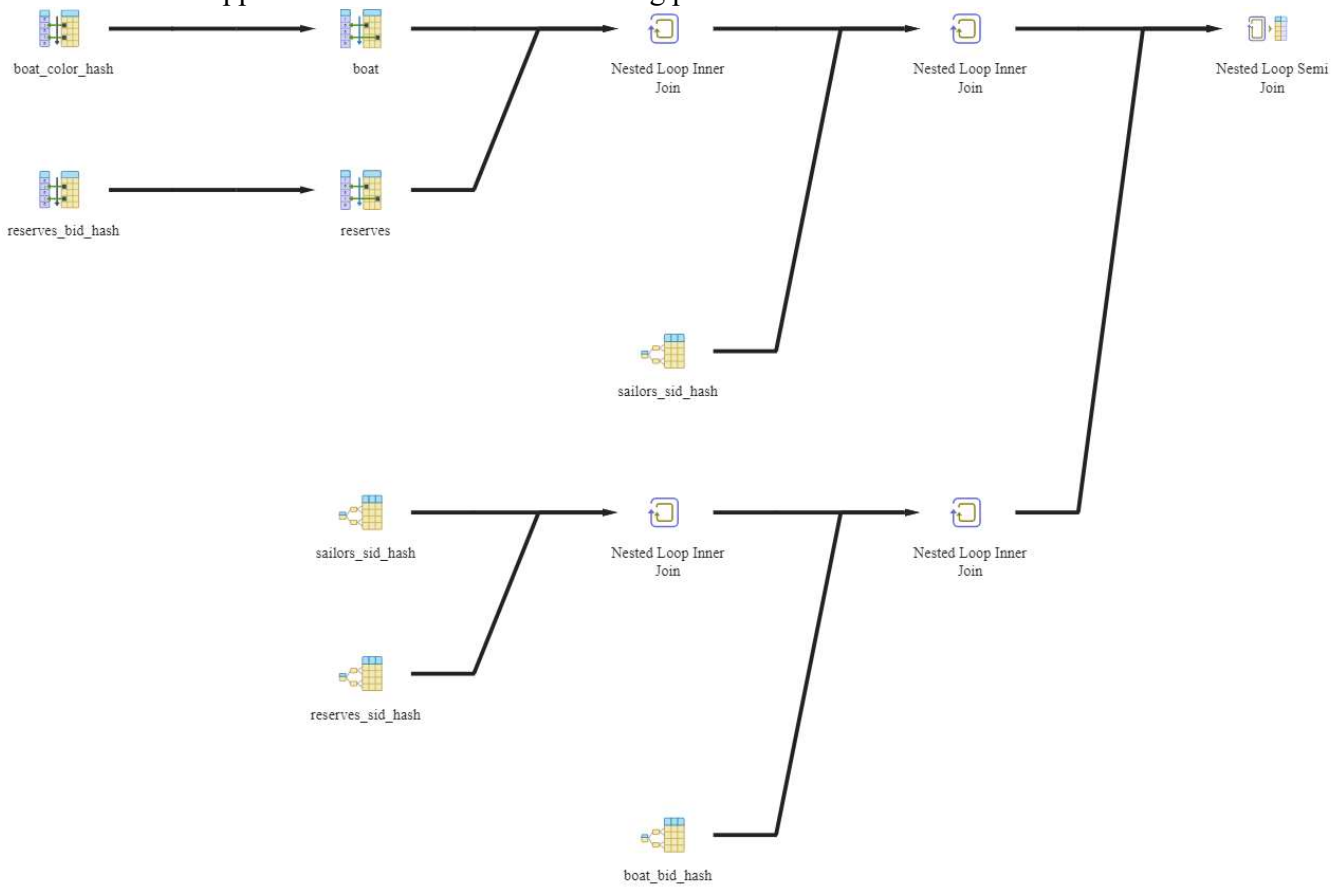
*"Nested Loop Semi Join **(cost=14.14..2004.52 rows=222 width=21)** (actual time=1.251..11.588 rows=190 loops=1)"*
" -> Nested Loop (cost=13.28..1782.97 rows=222 width=29) (actual time=1.227..10.609 rows=209 loops=1)"
"   -> Hash Join (cost=12.99..1707.63 rows=222 width=4) (actual time=1.211..10.185 rows=209 loops=1)"
"   Hash Cond: (r.bid = b.bid)"

" -> Index Scan using reserves_bid_btree on reserves r (cost=0.29..1561.46 rows=35000 width=8) (actual time=0.013..7.835 rows=35000 loops=1)"
" -> Hash (cost=12.47..12.47 rows=19 width=4) (actual time=0.081..0.082 rows=19 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 9kB"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.066..0.071 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..0.33 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Nested Loop (cost=0.86..1.05 rows=1 width=8) (actual time=0.004..0.004 rows=1 loops=209)"
" -> Nested Loop (cost=0.58..0.73 rows=1 width=12) (actual time=0.002..0.002 rows=1 loops=209)"
" Join Filter: (s2.sid = r2.sid)"
" -> Index Scan using sailors_sid_btree on sailors s2 (cost=0.29..0.35 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = s.sid)"
" -> Index Scan using reserves_bid_btree on reserves r2 (cost=0.29..0.36 rows=2 width=8) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using boat_bid_btree on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=304)"
" Index Cond: (bid = r2.bid)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 0"
"Planning Time: 0.690 ms"
"Execution Time: 11.676 ms"

For the above plan, the engine decided to use the Btree indices that we created instead of doing a sequential scan on all the tables as he did in the raw query, this have contributed in a dramatic decrease in the cost (2004.52) by almost 1700 and the execution time (11.676ms) have been cut down to almost the 4th.

9.Hash Index

Now we will build Hash indices on the same columns and record their performance. Running the query with the hash indices applied would execute the following plan:



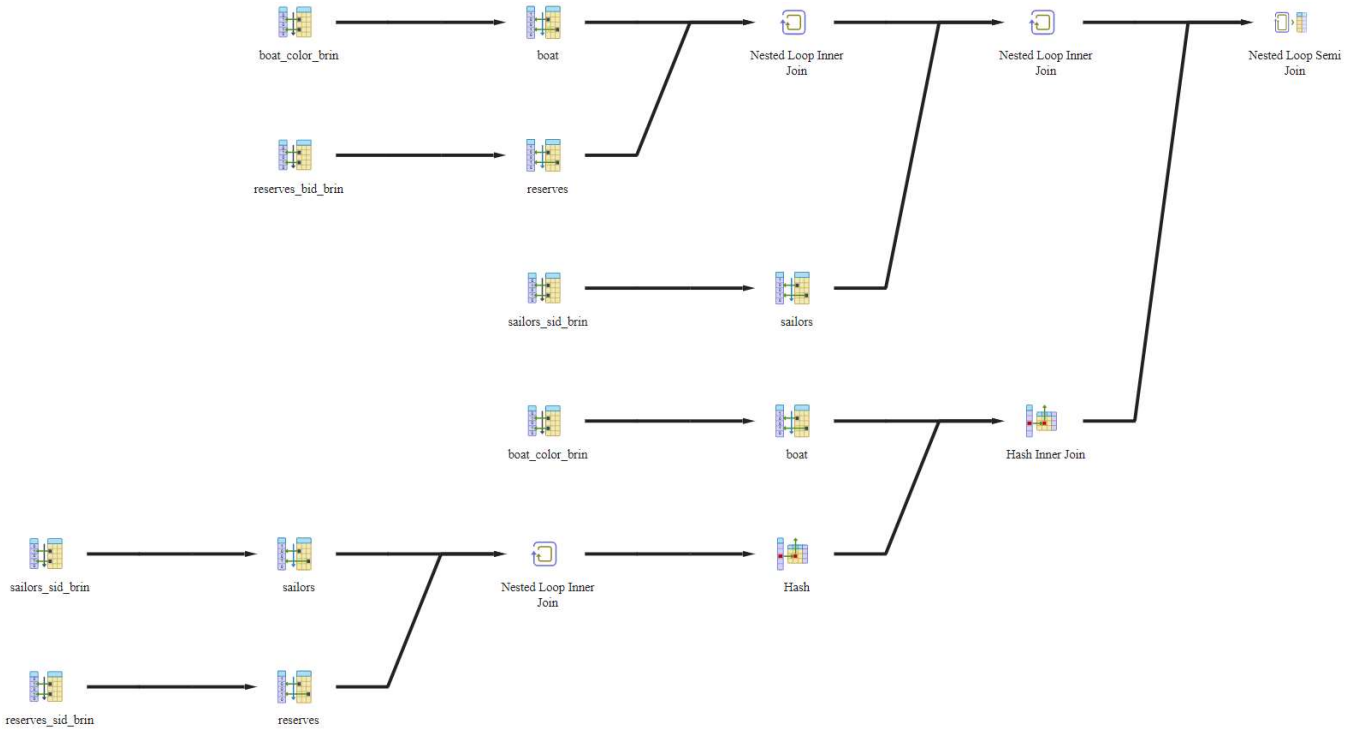
And the analysis of this plan is the following:

*"Nested Loop Semi Join **(cost=8.03..666.52 rows=222 width=21)** (actual time=0.055..1.403 rows=190 loops=1)"*
" -> Nested Loop (cost=8.03..634.10 rows=222 width=29) (actual time=0.038..0.609 rows=209 loops=1)"
" -> Nested Loop (cost=8.03..622.28 rows=222 width=4) (actual time=0.030..0.278 rows=209 loops=1)"
" -> Bitmap Heap Scan on boat b (cost=4.15..28.95 rows=19 width=4) (actual time=0.013..0.015 rows=19 loops=1)"
" Recheck Cond: (color = 'red'::bpchar)"
" Heap Blocks: exact=1"
" -> Bitmap Index Scan on boat_color_hash (cost=0.00..4.14 rows=19 width=0) (actual time=0.008..0.008 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.88..31.11 rows=12 width=8) (actual time=0.004..0.009 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..3.88 rows=12 width=0) (actual time=0.003..0.003 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Nested Loop (cost=0.00..0.20 rows=1 width=8) (actual time=0.004..0.004 rows=1 loops=209)"
" -> Nested Loop (cost=0.00..0.16 rows=1 width=12) (actual time=0.002..0.002 rows=1 loops=209)"
" Join Filter: (s2.sid = r2.sid)"
" -> Index Scan using sailors_sid_hash on sailors s2 (cost=0.00..0.06 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=209)"
" Index Cond: (sid = s.sid)"
" -> Index Scan using reserves_sid_hash on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using boat_bid_hash on boat b2 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=304)"
" Index Cond: (bid = r2.bid)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 0"
"Planning Time: 0.479 ms"
"Execution Time: 1.454 ms"

For the above plan, the query performed much better when the engine used the hash indices on the tables than using Btree indices. The cost has been reduced to 666.52 and the execution time has been reduced to 1.454ms which is a huge reduction.

9.BRIN Index

Now we will build BRIN indices on the same columns and record their performance. Running the query with the BRIN indices applied and with seqscan enabled would return the same plan used for the raw query, which indicates that using BRIN indices for this query is not efficient, however, we will disable the seqscan and record the analysis and see how bad would the BRIN indices perform. The plan that uses BRIN indices is the following:



The analysis of the above plan is the following:

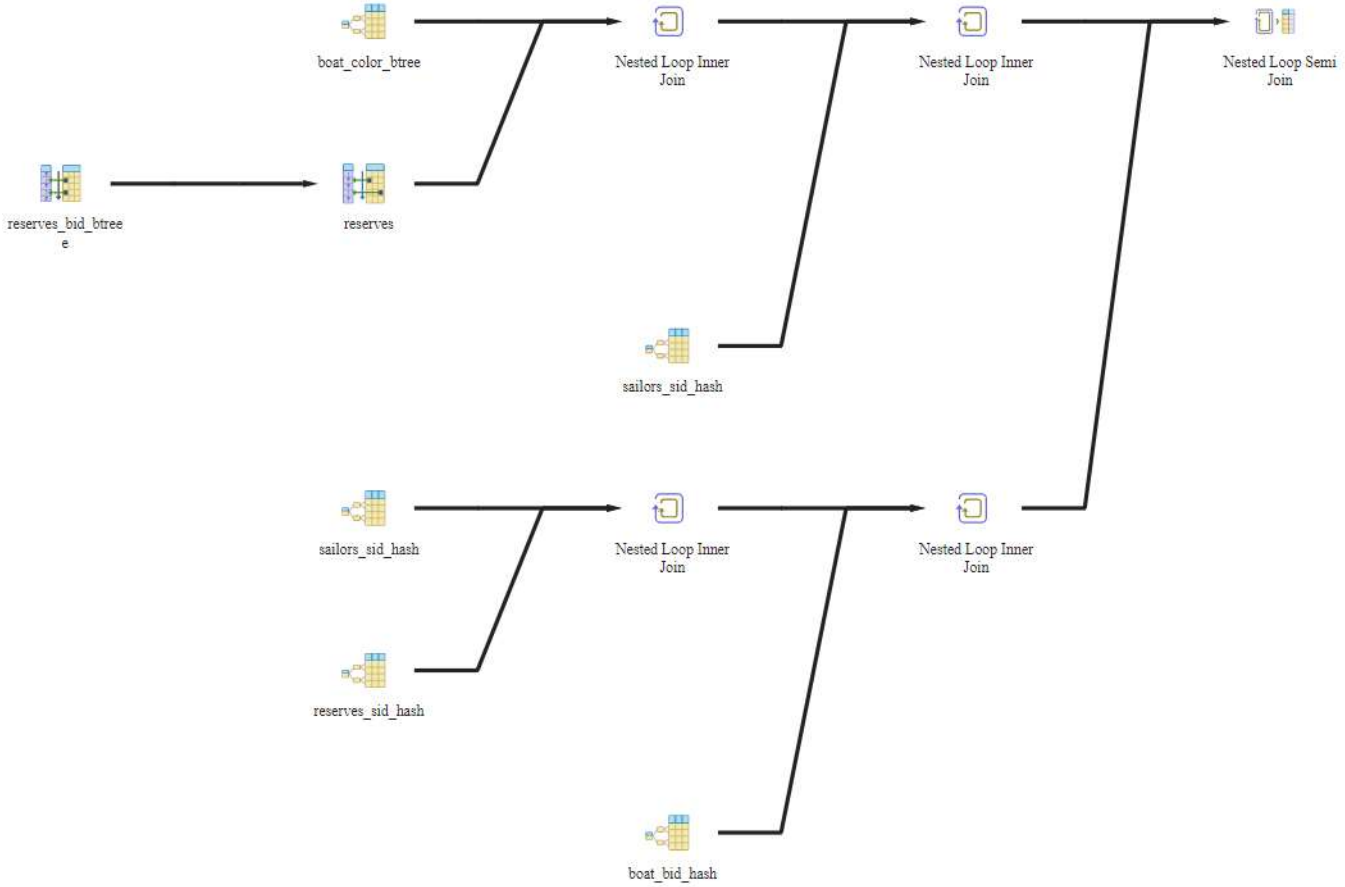
*"Nested Loop Semi Join **(cost=876817.31..93748774.54 rows=222 width=21)** (actual time=4.461..1104.777 rows=190 loops=1)"*
" -> Nested Loop (cost=420240.14..93280611.70 rows=222 width=29) (actual time=0.621..292.541 rows=209 loops=1)"
" -> Nested Loop (cost=240.10..13351.82 rows=222 width=4) (actual time=0.320..51.503 rows=209 loops=1)"
" -> Bitmap Heap Scan on boat b (cost=12.04..74.54 rows=19 width=4) (actual time=0.045..0.355 rows=19 loops=1)"
" Recheck Cond: (color = 'red'::bpchar)"
" Rows Removed by Index Recheck: 2981"
" Heap Blocks: lossy=25"
" -> Bitmap Index Scan on boat_color_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.040..0.040 rows=250 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=228.07..698.68 rows=12 width=8) (actual time=0.223..2.681 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Rows Removed by Index Recheck: 34989"
" Heap Blocks: lossy=3610"
" -> Bitmap Index Scan on reserves_bid_brin (cost=0.00..228.06 rows=35000 width=0) (actual time=0.011..0.011 rows=1900 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Bitmap Heap Scan on sailors s (cost=420000.03..420122.78 rows=1 width=25) (actual time=0.652..1.145 rows=1 loops=209)"
" Recheck Cond: (sid = r.sid)"
" Rows Removed by Index Recheck: 14294"
" Heap Blocks: lossy=24909"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..420000.03 rows=9500 width=0) (actual time=0.010..0.010 rows=1192 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Hash Join (cost=456577.17..456650.87 rows=2 width=8) (actual time=3.878..3.878 rows=1 loops=209)"
" Hash Cond: (b2.bid = r2.bid)"
" -> Bitmap Heap Scan on boat b2 (cost=12.78..75.28 rows=2981 width=4) (actual time=0.023..0.205 rows=1626 loops=209)"
" Recheck Cond: (color = 'green'::bpchar)"
" Rows Removed by Index Recheck: 19"
" Heap Blocks: lossy=2945"
" -> Bitmap Index Scan on boat_color_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.018..0.018 rows=250 loops=209)"
" Index Cond: (color = 'green'::bpchar)"
" -> Hash (cost=456564.37..456564.37 rows=2 width=12) (actual time=3.594..3.594 rows=2 loops=209)"
" Buckets: 1024 Batches: 1 Memory Usage: 9kB"
" -> Nested Loop (cost=456000.10..456564.37 rows=2 width=12) (actual time=1.330..3.591 rows=2 loops=209)"
" -> Bitmap Heap Scan on sailors s2 (cost=228000.03..228122.78 rows=1 width=4) (actual time=0.614..1.137 rows=1 loops=209)"

" Recheck Cond: (sid = s.sid)"
" Rows Removed by Index Recheck: 14294"
" Heap Blocks: lossy=24909"
" -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..228000.03 rows=9500 width=0) (actual time=0.009..0.009 rows=1192 loops=209)"
" Index Cond: (sid = s.sid)"
" -> Bitmap Heap Scan on reserves r2 (cost=228000.06..228441.56 rows=2 width=8) (actual time=0.705..2.443 rows=2 loops=209)"
" Recheck Cond: (sid = s2.sid)"
" Rows Removed by Index Recheck: 31911"
" Heap Blocks: lossy=36176"
" -> Bitmap Index Scan on reserves_sid_brin (cost=0.00..228000.06 rows=35000 width=0) (actual time=0.011..0.011 rows=1731 loops=209)"
" Index Cond: (sid = s2.sid)"
"Planning Time: 0.520 ms"
"Execution Time: 1105.675 ms"

In the analysis, we can notice the dramatic increase in both the cost and the execution time of this query, which shows how bad the BRIN index performed, which is expected since the query doesn't involve aggregate or analytical functions which the BRIN is good at, so the engine decides that normal seqscan would be much better building useless BRIN indices for this query.

9.Mix Indices

Finally, we will create the 3 types of indices on all the columns used in this query and we will let the engine decide which combination of indices on which columns would generate the most optimal plan.
the generated plan is as follows:



And the analysis to this plan is as follows:

*"Nested Loop Semi Join **(cost=4.03..647.54 rows=222 width=21)** (actual time=0.084..1.359 rows=190 loops=1)"*
" -> Nested Loop (cost=4.03..615.12 rows=222 width=29) (actual time=0.068..0.574 rows=209 loops=1)"
" -> Nested Loop (cost=4.03..603.30 rows=222 width=4) (actual time=0.058..0.274 rows=209 loops=1)"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.037..0.040 rows=19 loops=1)"

" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.75..30.98 rows=12 width=8) (actual time=0.003..0.007 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..3.75 rows=12 width=0) (actual time=0.002..0.002 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Nested Loop (cost=0.00..0.20 rows=1 width=8) (actual time=0.004..0.004 rows=1 loops=209)"
" -> Nested Loop (cost=0.00..0.16 rows=1 width=12) (actual time=0.002..0.002 rows=1 loops=209)"
" Join Filter: (s2.sid = r2.sid)"
" -> Index Scan using sailors_sid_hash on sailors s2 (cost=0.00..0.06 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=209)"
" Index Cond: (sid = s.sid)"
" -> Index Scan using reserves_sid_hash on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using boat_bid_hash on boat b2 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=304)"
" Index Cond: (bid = r2.bid)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 0"
"Planning Time: 0.950 ms"
"Execution Time: 1.413 ms"

Conclusion:

Using Btree indices on boat(color) and reserves(bid) while using a hash index on sailors(sid), reserves(sid), and boat(bid) generates the most optimal plan with the lowest cost of 647.54 and execution time of 1.413ms. obviously, no BRIN indices can be spotted in this plan as it already performed the worst on its own. The Btree and Hash indices each were good on their own with a little better performance for the Hash indices, which fits the nature of this query as it contains many joins and an exact value query “where b.color = 'red'” and “where b.color = 'green'”.

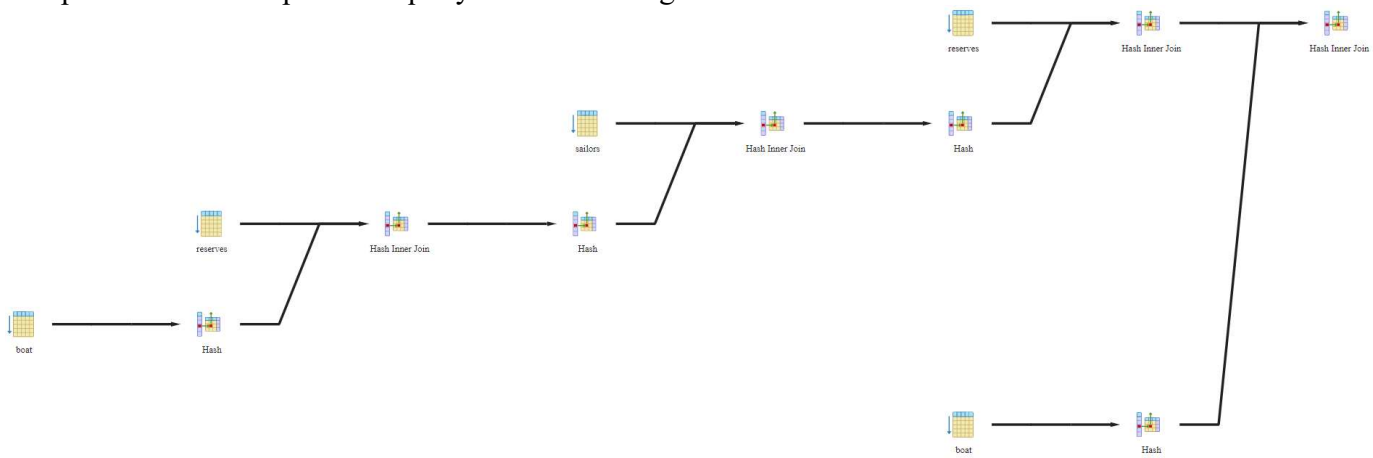
Query 9 optimized

The optimized query is the following:

```
select s.sname
from sailors s inner join reserves r on s.sid = r.sid inner join boat b on r.bid = b.bid
        inner join reserves r2 on r.sid = r2.sid inner join boat b2 on r2.bid = b2.bid
where b.color = 'red' and b2.color = 'green'
```

the technique that was used in optimizing query 9 is using inner joins instead of querying on all of the tables at the same time and putting the join condition in the where clause, and replacing the subquery with inner joins as well.

The plan for the raw optimized query is the following:



And the analysis for this plan is the following:

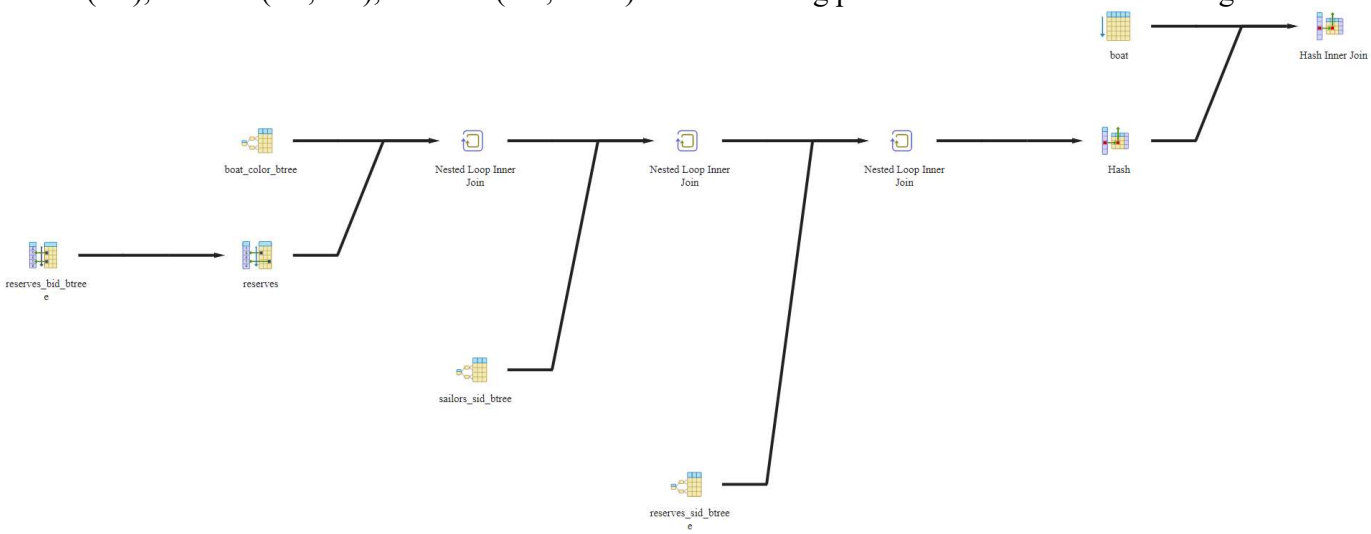
*"Hash Join **(cost=1263.99..1944.92 rows=406 width=21)** (actual time=7.100..10.307 rows=190 loops=1)"*
" Hash Cond: (r2.bid = b2.bid)"
" -> Hash Join (cost=1164.23..1839.57 rows=409 width=25) (actual time=6.387..9.534 rows=399 loops=1)"
" Hash Cond: (r2.sid = s.sid)"
" -> Seq Scan on reserves r2 (cost=0.00..540.00 rows=35000 width=8) (actual time=0.011..1.486 rows=35000 loops=1)"
" -> Hash (cost=1161.45..1161.45 rows=222 width=29) (actual time=6.070..6.071 rows=209 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 22kB"
" -> Hash Join (cost=738.98..1161.45 rows=222 width=29) (actual time=4.189..6.021 rows=209 loops=1)"
" Hash Cond: (s.sid = r.sid)"
" -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.005..0.887 rows=19000 loops=1)"
" -> Hash (cost=736.21..736.21 rows=222 width=4) (actual time=3.958..3.960 rows=209 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 16kB"
" -> Hash Join (cost=62.74..736.21 rows=222 width=4) (actual time=0.578..3.930 rows=209 loops=1)"
" Hash Cond: (r.bid = b.bid)"
" -> Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.004..1.516 rows=35000 loops=1)"
" -> Hash (cost=62.50..62.50 rows=19 width=4) (actual time=0.281..0.282 rows=19 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 9kB"
" -> Seq Scan on boat b (cost=0.00..62.50 rows=19 width=4) (actual time=0.005..0.278 rows=19 loops=1)"
" Filter: (color = 'red'::bpchar)"
" Rows Removed by Filter: 2981"
" -> Hash (cost=62.50..62.50 rows=2981 width=4) (actual time=0.651..0.651 rows=2981 loops=1)"
" Buckets: 4096 Batches: 1 Memory Usage: 137kB"
" -> Seq Scan on boat b2 (cost=0.00..62.50 rows=2981 width=4) (actual time=0.026..0.333 rows=2981 loops=1)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 19"
"Planning Time: 0.275 ms"
"Execution Time: 10.422 ms"

This optimized version of query 9 greatly reduced cost and execution time, as the cost was 3698.13 and dropped to 1944.92, and the execution time was 44.307ms and dropped down to 10.422ms.

Now we will start applying different indices in order to reduce the cost and execution time of the plan, hence improving query performance.

9opt.Btree Index

We will construct Btree indices on the columns that we are using while executing this query, namely sailors(sid), reserves(sid, bid), and boat(bid, color). The following plan is returned when executing:



And the analysis of the above query is the following:

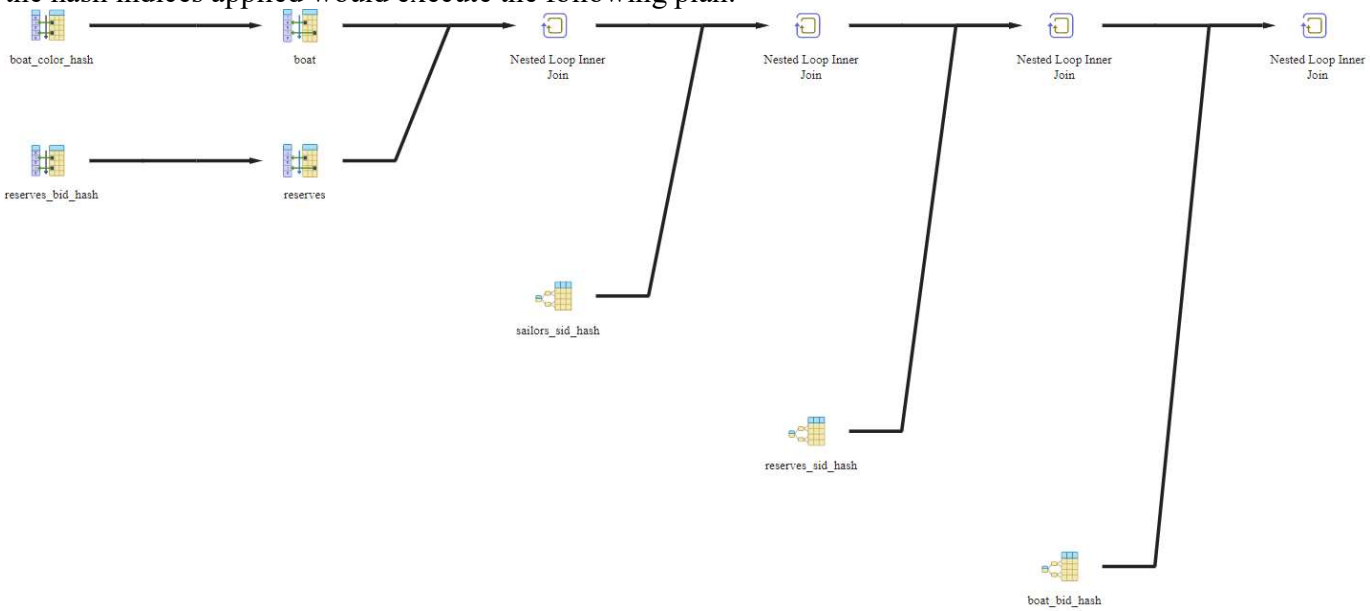
*"Hash Join **(cost=768.55..853.74 rows=406 width=21)** (actual time=1.187..1.477 rows=190 loops=1)"*
" Hash Cond: (b2.bid = r2.bid)"
" -> Seq Scan on boat b2 (cost=0.00..62.50 rows=2981 width=4) (actual time=0.012..0.284 rows=2981 loops=1)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 19"
" -> Hash (cost=763.44..763.44 rows=409 width=25) (actual time=1.032..1.033 rows=399 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 32kB"
" -> Nested Loop (cost=4.61..763.44 rows=409 width=25) (actual time=0.073..0.990 rows=399 loops=1)"
" Join Filter: (s.sid = r2.sid)"
" -> Nested Loop (cost=4.32..678.64 rows=222 width=29) (actual time=0.063..0.600 rows=209 loops=1)"
" -> Nested Loop (cost=4.03..603.30 rows=222 width=4) (actual time=0.047..0.256 rows=209 loops=1)"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.026..0.029 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.75..30.98 rows=12 width=8) (actual time=0.004..0.007 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..3.75 rows=12 width=0) (actual time=0.002..0.002 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_btree on sailors s (cost=0.29..0.33 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using reserves_sid_btree on reserves r2 (cost=0.29..0.36 rows=2 width=8) (actual time=0.001..0.002 rows=2 loops=209)"
" Index Cond: (sid = r.sid)"
"Planning Time: 0.669 ms"
"Execution Time: 1.554 ms"

Using the Btree indices instead of seqscans lowered the cost from 1944.92 to 853.74 and the execution time was cut down to 1.554ms instead of 10.422ms.

9opt.Hash Index

Now we will build Hash indices on the same columns and record their performance. Running the query with the hash indices applied would execute the following plan:



And the analysis for this plan is as follows:

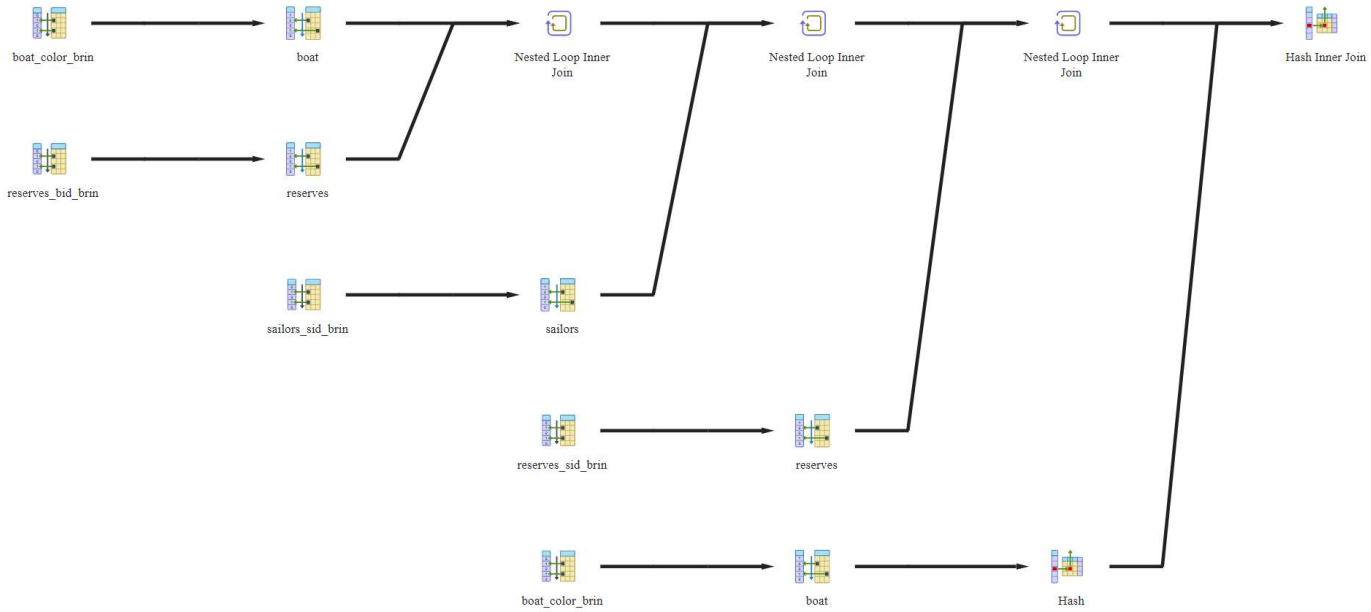
*"Nested Loop **(cost=8.03..669.81 rows=406 width=21)** (actual time=0.054..1.318 rows=190 loops=1)"*
" -> Nested Loop (cost=8.03..655.54 rows=409 width=25) (actual time=0.049..0.997 rows=399 loops=1)"
" Join Filter: (s.sid = r2.sid)"
" -> Nested Loop (cost=8.03..634.10 rows=222 width=29) (actual time=0.040..0.592 rows=209 loops=1)"
" -> Nested Loop (cost=8.03..622.28 rows=222 width=4) (actual time=0.035..0.278 rows=209 loops=1)"
" -> Bitmap Heap Scan on boat b (cost=4.15..28.95 rows=19 width=4) (actual time=0.012..0.021 rows=19 loops=1)"
" Recheck Cond: (color = 'red'::bpchar)"
" Heap Blocks: exact=1"
" -> Bitmap Index Scan on boat_color_hash (cost=0.00..4.14 rows=19 width=0) (actual time=0.008..0.008 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.88..31.11 rows=12 width=8) (actual time=0.005..0.009 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"

"   -> Bitmap Index Scan on reserves_bid_hash (cost=0.00..3.88 rows=12 width=0) (actual time=0.004..0.004 rows=11 loops=19)"
"   Index Cond: (bid = b.bid)"
"   -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
"   Index Cond: (sid = r.sid)"
"   -> Index Scan using reserves_sid_hash on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.002 rows=2 loops=209)"
"   Index Cond: (sid = r.sid)"
"   -> Index Scan using boat_bid_hash on boat b2 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=399)"
"   Index Cond: (bid = r2.bid)"
"   Filter: (color = 'green'::bpchar)"
"   Rows Removed by Filter: 1"
"Planning Time: 0.402 ms"
"Execution Time: 1.371 ms"

Using the hash indices saved even more execution time (1.371ms) than the Btree indices and had a total cost of 669.81 which also less than the cost of the Btree indices.

9opt.BRIN Index

Now we will build BRIN indices on the same columns and record their performance. Running the query with the BRIN indices applied and with seqscan enabled would return the same plan used for the raw query, which indicates that using BRIN indices for this query is not efficient, however, we will disable the seqscan and record the analysis and see how bad would the BRIN indices perform. The plan that uses BRIN indices is the following:



The analysis of the above plan is the following:

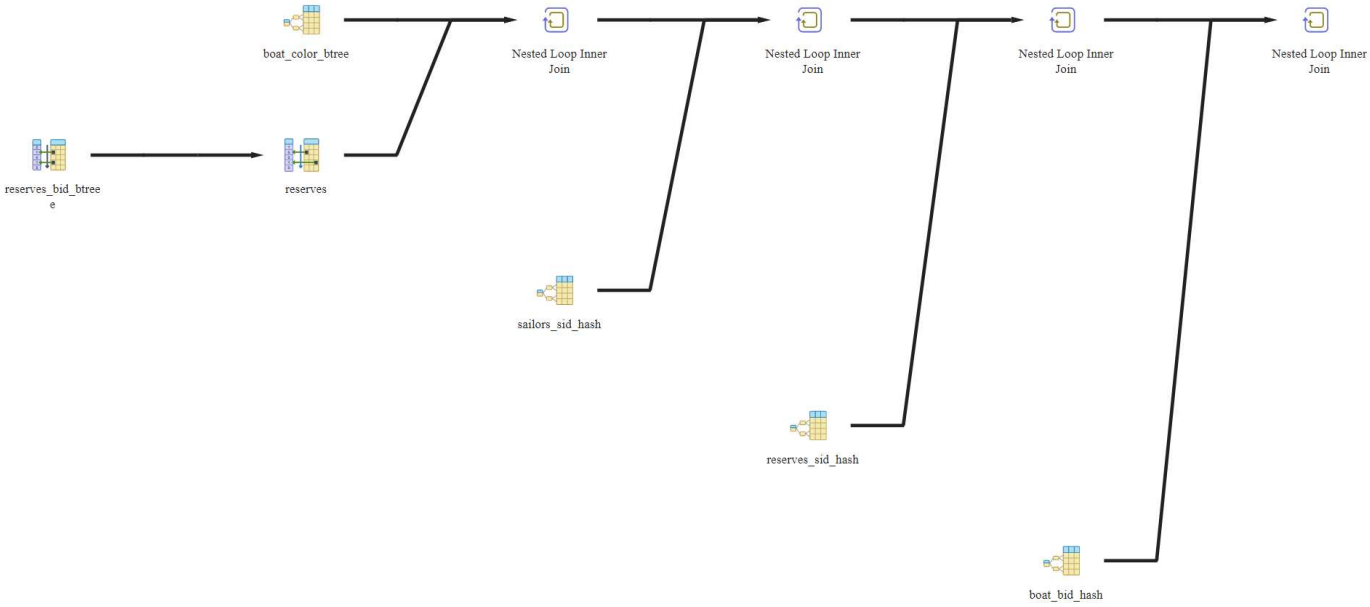
*"Hash Join **(cost=648352.74..143994761.59 rows=406 width=21)** (actual time=2.671..824.890 rows=190 loops=1)"*
" Hash Cond: (r2.bid = b2.bid)"
" -> Nested Loop (cost=648240.20..143994643.46 rows=409 width=25) (actual time=0.756..823.774 rows=399 loops=1)"
"   -> Nested Loop (cost=420240.14..93280611.70 rows=222 width=29) (actual time=0.540..311.195 rows=209 loops=1)"
"   -> Nested Loop (cost=240.10..13351.82 rows=222 width=4) (actual time=0.258..54.050 rows=209 loops=1)"
"   -> Bitmap Heap Scan on boat b (cost=12.04..74.54 rows=19 width=4) (actual time=0.023..0.318 rows=19 loops=1)"
"   Recheck Cond: (color = 'red'::bpchar)"
"   Rows Removed by Index Recheck: 2981"
"   Heap Blocks: lossy=25"
"   -> Bitmap Index Scan on boat_color_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.019..0.019 rows=250 loops=1)"
"   Index Cond: (color = 'red'::bpchar)"
"   -> Bitmap Heap Scan on reserves r (cost=228.07..698.68 rows=12 width=8) (actual time=0.232..2.814 rows=11 loops=19)"
"   Recheck Cond: (bid = b.bid)"
"   Rows Removed by Index Recheck: 34989"
"   Heap Blocks: lossy=3610"

"   -> Bitmap Index Scan on reserves_bid_brin (cost=0.00..228.06 rows=35000 width=0) (actual time=0.015..0.015 rows=1900 loops=19)"
"   Index Cond: (bid = b.bid)"
"   -> Bitmap Heap Scan on sailors s (cost=420000.03..420122.78 rows=1 width=25) (actual time=0.660..1.220 rows=1 loops=209)"
"   Recheck Cond: (sid = r.sid)"
"   Rows Removed by Index Recheck: 14294"
"   Heap Blocks: lossy=24909"
"   -> Bitmap Index Scan on sailors_sid_brin (cost=0.00..420000.03 rows=9500 width=0) (actual time=0.012..0.012 rows=1192 loops=209)"
"   Index Cond: (sid = r.sid)"
"   -> Bitmap Heap Scan on reserves r2 (cost=228000.06..228441.56 rows=2 width=8) (actual time=0.711..2.445 rows=2 loops=209)"
"   Recheck Cond: (sid = s.sid)"
"   Rows Removed by Index Recheck: 31911"
"   Heap Blocks: lossy=36176"
"   -> Bitmap Index Scan on reserves_sid_brin (cost=0.00..228000.06 rows=35000 width=0) (actual time=0.010..0.010 rows=1731 loops=209)"
"   Index Cond: (sid = s.sid)"
" -> Hash (cost=75.28..75.28 rows=2981 width=4) (actual time=0.661..0.662 rows=2981 loops=1)"
"   Buckets: 4096 Batches: 1 Memory Usage: 137kB"
"   -> Bitmap Heap Scan on boat b2 (cost=12.78..75.28 rows=2981 width=4) (actual time=0.024..0.379 rows=2981 loops=1)"
"   Recheck Cond: (color = 'green'::bpchar)"
"   Rows Removed by Index Recheck: 19"
"   Heap Blocks: lossy=25"
"   -> Bitmap Index Scan on boat_color_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.015..0.015 rows=250 loops=1)"
"   Index Cond: (color = 'green'::bpchar)"
"Planning Time: 0.355 ms"
"Execution Time: 825.339 ms"

In the analysis, we can notice the dramatic increase in both the cost and the execution time of this query, which shows how bad the BRIN index performed, which is expected since the query doesn't involve aggregate or analytical functions which the BRIN is good at, so the engine decides that normal seqscan would be much better building useless BRIN indices for this query.

9opt.Mix Indices

Finally, we will create the 3 types of indices on all the columns used in this query and we will let the engine decide which combination of indices on which columns would generate the most optimal plan.
the generated plan is as follows:



The analysis of the above plan is the following:

"Nested Loop ***(cost=4.03..650.84 rows=406 width=21)*** *(actual time=0.048..1.225 rows=190 loops=1)"*
" -> Nested Loop (cost=4.03..636.56 rows=409 width=25) (actual time=0.043..0.916 rows=399 loops=1)"
"   Join Filter: (s.sid = r2.sid)"

" -> Nested Loop (cost=4.03..615.12 rows=222 width=29) (actual time=0.039..0.542 rows=209 loops=1)"
" -> Nested Loop (cost=4.03..603.30 rows=222 width=4) (actual time=0.034..0.238 rows=209 loops=1)"
" -> Index Scan using boat_color_btree on boat b (cost=0.28..12.47 rows=19 width=4) (actual time=0.015..0.018 rows=19 loops=1)"
" Index Cond: (color = 'red'::bpchar)"
" -> Bitmap Heap Scan on reserves r (cost=3.75..30.98 rows=12 width=8) (actual time=0.003..0.007 rows=11 loops=19)"
" Recheck Cond: (bid = b.bid)"
" Heap Blocks: exact=209"
" -> Bitmap Index Scan on reserves_bid_btree (cost=0.00..3.75 rows=12 width=0) (actual time=0.002..0.002 rows=11 loops=19)"
" Index Cond: (bid = b.bid)"
" -> Index Scan using sailors_sid_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using reserves_sid_hash on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.001 rows=2 loops=209)"
" Index Cond: (sid = r.sid)"
" -> Index Scan using boat_bid_hash on boat b2 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=399)"
" Index Cond: (bid = r2.bid)"
" Filter: (color = 'green'::bpchar)"
" Rows Removed by Filter: 1"
"Planning Time: 0.772 ms"
"Execution Time: 1.267 ms"

Conclusion:

1- Using Btree indices on boat(color) and reserves(bid) while using a hash index on sailors(sid), reserves(sid), and boat(bid) generates the most optimal plan with the lowest cost of 650.84 and execution time of 1267ms. obviously, no BRIN indices can be spotted in this plan as it already performed the worst on its own. The Btree and Hash indices each were good on their own with a little better performance for the Hash indices, which fits the nature of this query as it contains many joins and an exact value query “where b.color = 'red'” and “where b.color = 'green'”.

2- The optimized query had much lower cost than the original query when it was tested without any indices and with Btree indices. However, when it was tested with Hash and BRIN indices, it performed almost the same in both the cost and execution time.

Schema 4

Query 10:

List all the information of the actors who played a role in the movie 'Annie Hall'.

```
select *
from actor
where act_id in(
select act_id
from movie_cast
where mov_id in(
select mov_id
from movie
where mov_title ='Annie Hall'));
```

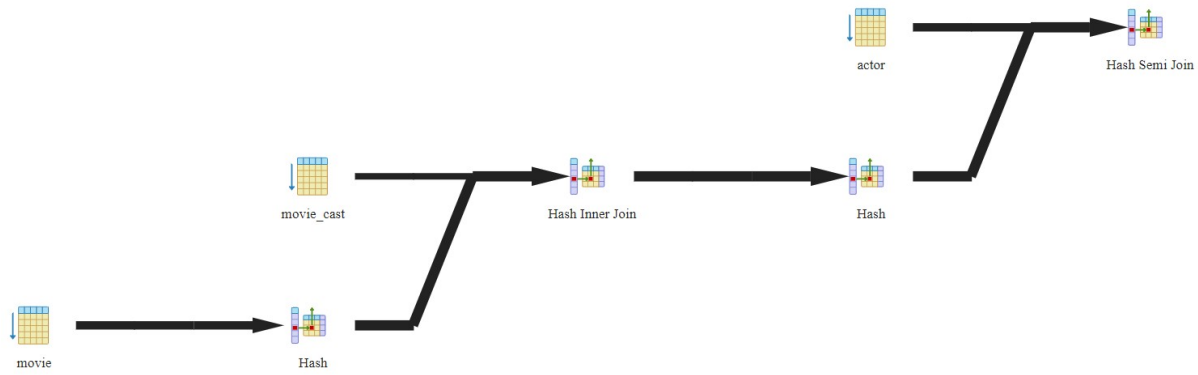
10. Raw Query

Initial configuration:

Disable all auto generated indices on movie_pkey, movie_cast_pkey and actor_pkey to force the query planner to do seqscan on the tables using the command

update pg_index set indisvalid = false where indexrelid = ‘index_name’::regclass;

Query plan:



Analyze output:

"Hash Semi Join (cost=3384.26..6021.27 rows=1 width=48) (actual time=10.461..21.888 rows=222 loops=1)"

" Hash Cond: (actor.act_id = movie_cast.act_id)"

" -> Seq Scan on actor (cost=0.00..2322.00 rows=120000 width=48) (actual time=0.011..5.063 rows=120000 loops=1)"

" -> Hash (cost=3384.25..3384.25 rows=1 width=4) (actual time=10.440..10.442 rows=222 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 16kB"

" -> Hash Join (cost=3174.01..3384.25 rows=1 width=4) (actual time=9.345..10.403 rows=222 loops=1)"

" Hash Cond: (movie_cast.mov_id = movie.mov_id)"

" -> Seq Scan on movie_cast (cost=0.00..183.99 rows=9999 width=8) (actual time=0.007..0.462 rows=9999 loops=1)"

" -> Hash (cost=3174.00..3174.00 rows=1 width=4) (actual time=9.328..9.329 rows=1 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 9kB"

" -> Seq Scan on movie (cost=0.00..3174.00 rows=1 width=4) (actual time=0.008..9.325 rows=1 loops=1)"

" Filter: (mov_title = 'Annie Hall'::bpchar)"

" Rows Removed by Filter: 99999"

"Planning Time: 0.266 ms"

"Execution Time: 21.917 ms"

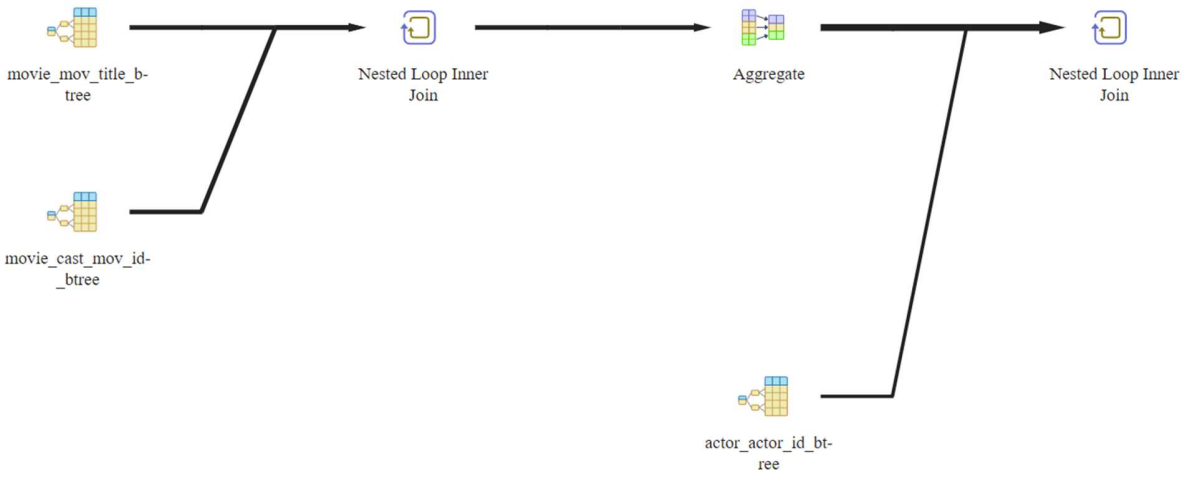
Total cost = 6021.27

10. B+tree

Initial configuration:

Create b+tree indices on movie(mov_title), move_cast(mov_id) and actor(act_id).

Query plan:



Analyze output:

"Nested Loop (cost=17.17..17.79 rows=1 width=48) (actual time=0.122..0.437 rows=222 loops=1)"

" -> HashAggregate (cost=16.75..16.76 rows=1 width=4) (actual time=0.115..0.130 rows=222 loops=1)"

"   Group Key: movie_cast.act_id"

"   Batches: 1 Memory Usage: 56kB"

"   -> Nested Loop (cost=0.70..16.75 rows=1 width=4) (actual time=0.031..0.066 rows=222 loops=1)"

"   -> Index Scan using movie_mov_title_btree on movie (cost=0.42..8.44 rows=1 width=4) (actual time=0.018..0.018 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_btree on movie_cast (cost=0.29..8.30 rows=1 width=8) (actual time=0.011..0.033 rows=222 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

" -> Index Scan using actor_actor_id_btree on actor (cost=0.42..1.02 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=222)"

"   Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.273 ms"

"Execution Time: 0.471 ms"

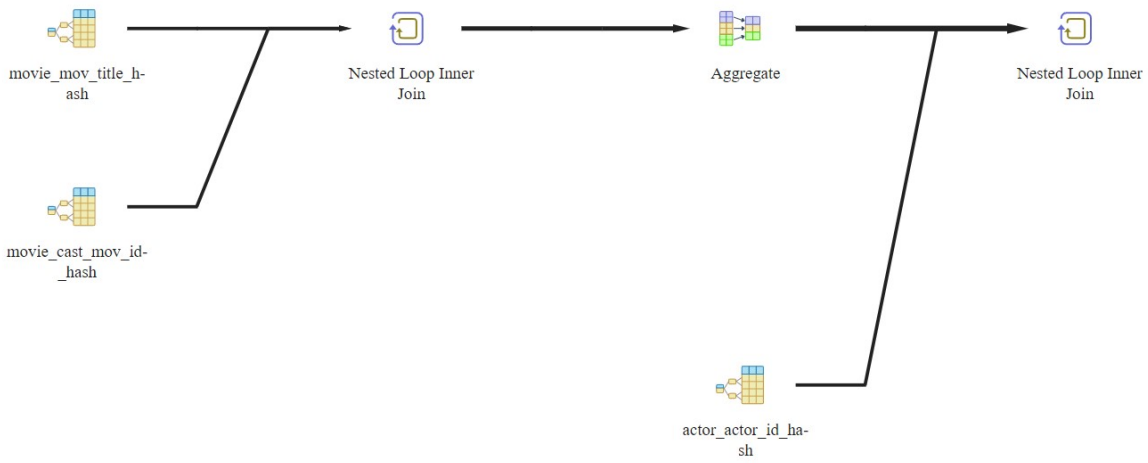
Total cost = 17.79

10. Hash

Initial configuration:

Create hash indices on movie(mov_title), move_cast(mov_id) and actor(act_id).

Query plan:



Analyze output:

"Nested Loop (cost=16.05..16.74 rows=1 width=48) (actual time=0.097..0.449 rows=222 loops=1)"

" -> HashAggregate (cost=16.05..16.06 rows=1 width=4) (actual time=0.094..0.111 rows=222 loops=1)"

"   Group Key: movie_cast.act_id"

"   Batches: 1 Memory Usage: 56kB"

"   -> Nested Loop (cost=0.00..16.05 rows=1 width=4) (actual time=0.013..0.051 rows=222 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_hash on movie_cast (cost=0.00..8.02 rows=1 width=8) (actual time=0.004..0.034 rows=222 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

" -> Index Scan using actor_actor_id_hash on actor (cost=0.00..0.67 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=222)"

" Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.318 ms"

"Execution Time: 0.477 ms"

Total cost = 16.74

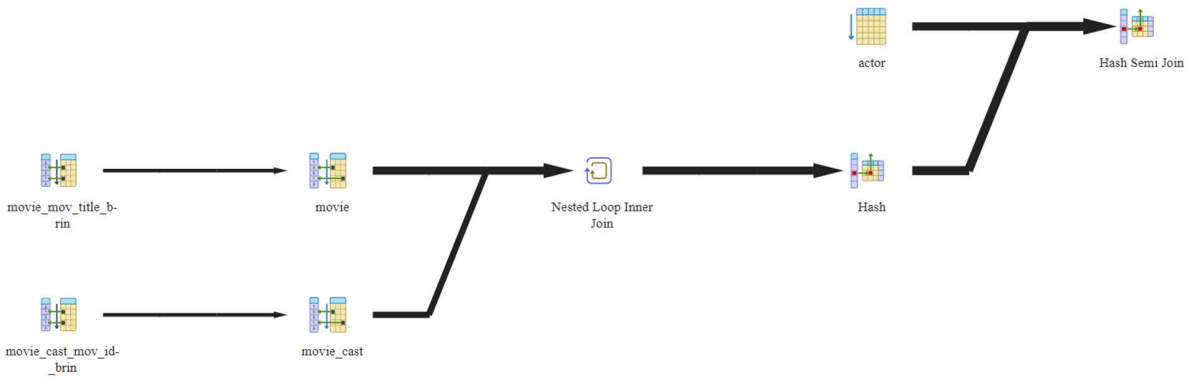
10. BRIN

Initial configuration:

Create BRIN indices on movie(mov_title), move_cast(mov_id) and actor(act_id).

The query planner decides to neglect the actor index because it will produce higher cost.

Query plan:



Analyze output:

"Hash Semi Join (cost=2251.96..4888.97 rows=1 width=48) (actual time=1.806..17.275 rows=222 loops=1)"

" Hash Cond: (actor.act_id = movie_cast.act_id)"

" -> Seq Scan on actor (cost=0.00..2322.00 rows=120000 width=48) (actual time=0.009..6.926 rows=120000 loops=1)"

" -> Hash (cost=2251.94..2251.94 rows=1 width=4) (actual time=1.787..1.790 rows=222 loops=1)"

```
"       Buckets: 1024  Batches: 1  Memory Usage: 16kB"

"       ->  Nested Loop  (cost=24.07..2251.94 rows=1 width=4) (actual time=0.056..1.768
rows=222 loops=1)"

"             ->  Bitmap Heap Scan on movie  (cost=12.04..2030.91 rows=1 width=4) (actual
time=0.034..0.916 rows=1 loops=1)"

"                   Recheck Cond: (mov_title = 'Annie Hall'::bpchar)"

"                   Rows Removed by Index Recheck: 6655"

"                   Heap Blocks: lossy=128"

"                   ->  Bitmap Index Scan on movie_mov_title_brin  (cost=0.00..12.04 rows=7590
width=0) (actual time=0.028..0.029 rows=1280 loops=1)"

"                         Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"             ->  Bitmap Heap Scan on movie_cast  (cost=12.03..221.02 rows=1 width=8) (actual
time=0.016..0.836 rows=222 loops=1)"

"                   Recheck Cond: (mov_id = movie.mov_id)"

"                   Rows Removed by Index Recheck: 9777"

"                   Heap Blocks: lossy=84"

"                   ->  Bitmap Index Scan on movie_cast_mov_id_brin  (cost=0.00..12.03 rows=9999
width=0) (actual time=0.011..0.011 rows=840 loops=1)"

"                         Index Cond: (mov_id = movie.mov_id)"

"Planning Time: 0.226 ms"

"Execution Time: 17.325 ms"
```

Total cost = 4888.97

10. Mixed indices

Initial configuration:

Create b+tree indices on movie (mov_title) and move_cast (mov_id).

Create hash indices on movie (mov_title) and actor (act_id).

Create BRIN indices on movie (mov_title), move_cast (mov_id) and actor (act_id).

The query planner neglects BRIN indices and uses a mix of hash and b+tree indices as follows.

Query plan:

Analyze output:

"Nested Loop (cost=16.33..17.02 rows=1 width=48) (actual time=0.084..0.416 rows=222 loops=1)"

" -> HashAggregate (cost=16.33..16.34 rows=1 width=4) (actual time=0.080..0.096 rows=222 loops=1)"

"   Group Key: movie_cast.act_id"

"   Batches: 1 Memory Usage: 56kB"

"   -> Nested Loop (cost=0.29..16.33 rows=1 width=4) (actual time=0.011..0.043 rows=222 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_btree on movie_cast (cost=0.29..8.30 rows=1 width=8) (actual time=0.004..0.024 rows=222 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

" -> Index Scan using actor_actor_id_hash on actor (cost=0.00..0.67 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=222)"

"   Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.221 ms"

"Execution Time: 0.442 ms"

Total cost = 17.02

10. Conclusion

When we use indices the cost and execution time are reduced even if use BRIN index which is not suitable for this query as it doesn't contain ranges.

The hash index is the most suitable index for this query as it contains exact value searches and we can see this from its cost.

The hash index has the smallest cost. The B+tree has similar cost and slightly better execution time so in the mix we get better execution time than the b+tree but higher cost than the hash index.

Query 10 equivalent query:

We can't make a huge optimization for this query because the in clause translates to an inner join in the query planner and it uses hash join so it's the most optimal.

But we can give an equivalent query for it which has slightly better cost and better performance with indices (because it makes the query planner use multi-dimensional indices more) which is:

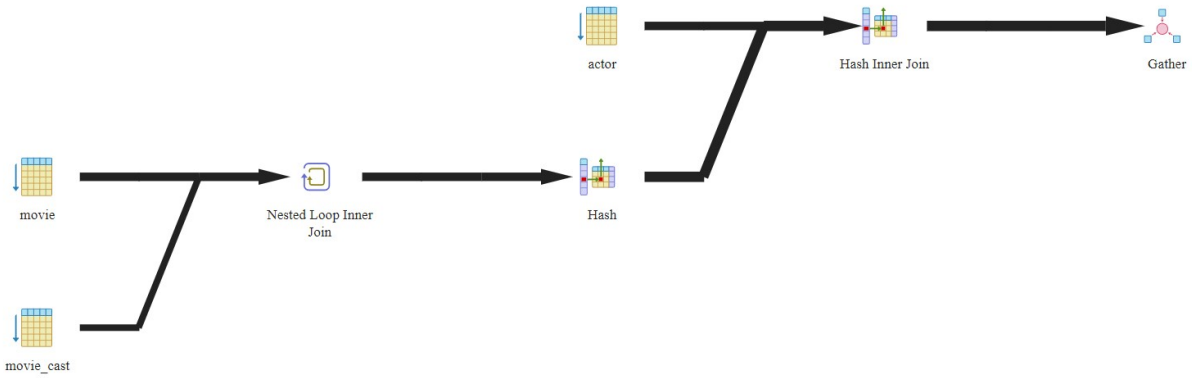
```
select * from (select mov_id from movie where mov_title ='Annie Hall') as m
inner join movie_cast on movie_cast.mov_id = m.mov_id
inner join actor on actor.act_id = movie_cast.act_id
```

10. Equivalent Raw Query

Initial configuration:

The same initial configuration as the original raw query

Query plan:



The difference here is the nested loop join instead of the hash join which produces almost the same cost and the gather node to collect the parallel sub queries.

Analyze output:

"Gather (cost=3968.28..6060.98 rows=1 width=211) (actual time=22.760..50.073 rows=222 loops=1)"

" Workers Planned: 1"

" Workers Launched: 1"

" -> Parallel Hash Join (cost=2968.28..5060.88 rows=1 width=211) (actual time=11.255..21.291 rows=111 loops=2)"

"   Hash Cond: (actor.act_id = movie_cast.act_id)"

"   -> Parallel Seq Scan on actor (cost=0.00..1827.88 rows=70588 width=48) (actual time=0.004..8.079 rows=120000 loops=1)"

"   -> Parallel Hash (cost=2968.27..2968.27 rows=1 width=163) (actual time=11.108..11.109 rows=111 loops=2)"

"   Buckets: 1024 Batches: 1 Memory Usage: 72kB"

"   -> Nested Loop (cost=0.00..2968.27 rows=1 width=163) (actual time=0.032..22.056 rows=222 loops=1)"

"   Join Filter: (movie.mov_id = movie_cast.mov_id)"

"   Rows Removed by Join Filter: 9777"

"   -> Parallel Seq Scan on movie (cost=0.00..2659.29 rows=1 width=124) (actual time=0.006..18.688 rows=1 loops=1)"

"   Filter: (mov_title = 'Annie Hall'::bpchar)"

"   Rows Removed by Filter: 99999"

" -> Seq Scan on movie_cast (cost=0.00..183.99 rows=9999 width=39) (actual time=0.022..1.191 rows=9999 loops=1)"

"Planning Time: 0.301 ms"

"Execution Time: 50.113 ms"

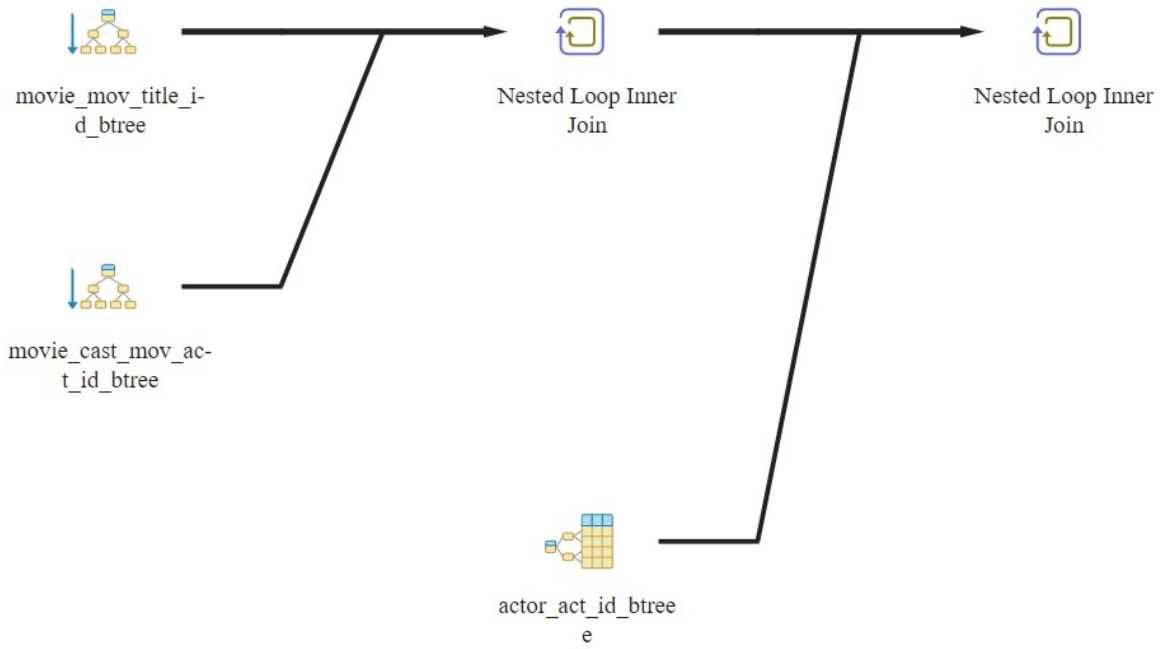
Total cost =6060.98

10. Equivalent B+tree

Initial configuration:

Create b+tree indices on actor table (act_id column), movie table (multi-dimensional on mov_id and mov_title) and on movie_cast (multi-dimensional on mov_id and act_id).

Query plan:



Analyze output:

"Nested Loop (cost=1.12..13.77 rows=1 width=48) (actual time=0.027..0.339 rows=222 loops=1)"

" -> Nested Loop (cost=0.70..12.75 rows=1 width=4) (actual time=0.022..0.049 rows=222 loops=1)"

" -> Index Only Scan using movie_mov_title_id_btree on movie (cost=0.42..8.44 rows=1 width=4) (actual time=0.015..0.015 rows=1 loops=1)"

" Index Cond: (mov_title = 'Annie Hall'::bpchar)"

" Heap Fetches: 0"

" -> Index Only Scan using movie_cast_mov_act_id_btree on movie_cast (cost=0.29..4.30 rows=1 width=8) (actual time=0.006..0.021 rows=222 loops=1)"

" Index Cond: (mov_id = movie.mov_id)"

" Heap Fetches: 0"

" -> Index Scan using actor_act_id_btree on actor (cost=0.42..1.02 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=222)"

" Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.264 ms"

"Execution Time: 0.360 ms"

Total cost = 13.77

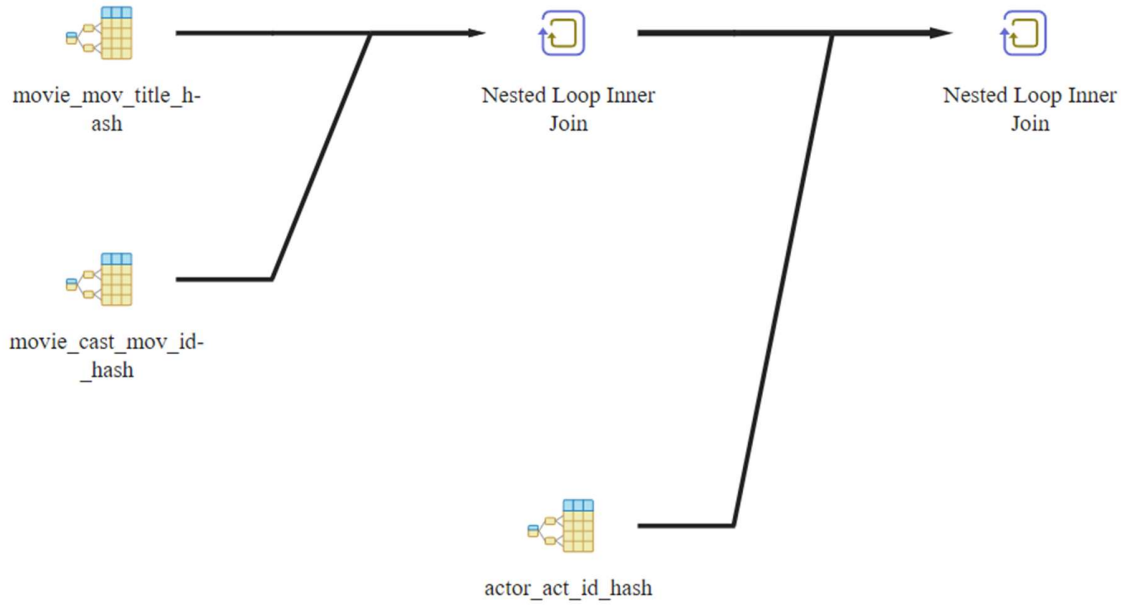
10. Equivalent hash

Initial configuration:

Create hash indices on actor table (act_id column), movie table (mov_id and mov_title) and on movie_cast (mov_id and act_id).

The query planner decides to use movie_cast (mov_id) and movie (mov_title) as more searches are on these two.

Query plan:



Analyze output:

"Nested Loop (cost=0.00..16.73 rows=1 width=48) (actual time=0.017..0.421 rows=222 loops=1)"

" -> Nested Loop (cost=0.00..16.05 rows=1 width=4) (actual time=0.015..0.065 rows=222 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.009..0.009 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_hash on movie_cast (cost=0.00..8.02 rows=1 width=8) (actual time=0.005..0.040 rows=222 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

" -> Index Scan using actor_act_id_hash on actor (cost=0.00..0.67 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=222)"

"   Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.203 ms"

"Execution Time: 0.445 ms"

Total cost = 16.73

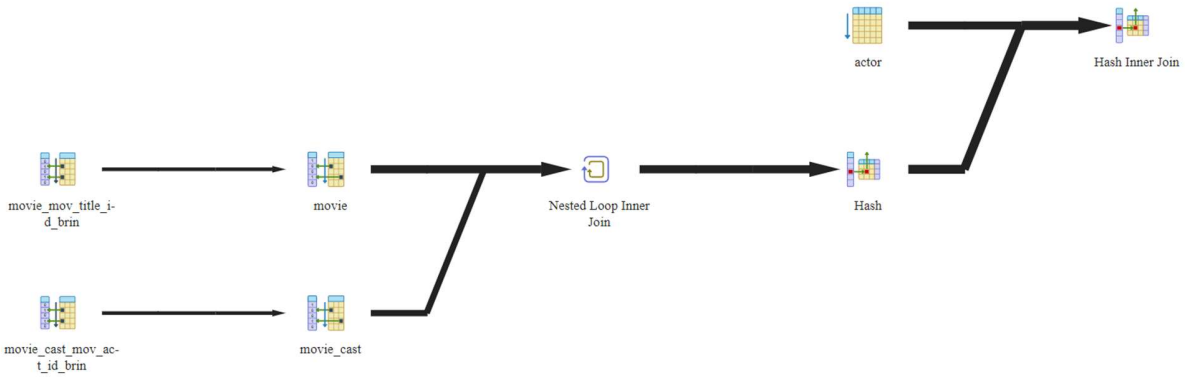
10. Equivalent BRIN

Initial configuration:

Create BRIN indices on actor table (act_id column), movie table (multi-dimensional on mov_id and mov_title) and on movie_cast (multi-dimensional on mov_id and act_id).

The query planner decides to use only the multi-dimensional indices and neglects the actor index because it will produce higher cost as almost all the table is being accessed so seqscan is better.

Query plan:



Analyze output:

"Hash Join (cost=2252.48..5024.49 rows=1 width=48) (actual time=2.797..18.653 rows=222 loops=1)"

" Hash Cond: (actor.act_id = movie_cast.act_id)"

" -> Seq Scan on actor (cost=0.00..2322.00 rows=120000 width=48) (actual time=0.010..6.821 rows=120000 loops=1)"

" -> Hash (cost=2252.47..2252.47 rows=1 width=4) (actual time=2.776..2.778 rows=222 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 16kB"

"   -> Nested Loop (cost=24.07..2252.47 rows=1 width=4) (actual time=0.059..2.755 rows=222 loops=1)"

"   -> Bitmap Heap Scan on movie (cost=12.04..2031.44 rows=1 width=4) (actual time=0.033..1.305 rows=1 loops=1)"

"   Recheck Cond: (mov_title = 'Annie Hall'::bpchar)"

```
"                Rows Removed by Index Recheck: 6655"
"                Heap Blocks: lossy=128"
"                ->  Bitmap Index Scan on movie_mov_title_id_brin  (cost=0.00..12.04 rows=7632 width=0) (actual time=0.027..0.028 rows=1280 loops=1)"
"                      Index Cond: (mov_title = 'Annie Hall'::bpchar)"
"           ->  Bitmap Heap Scan on movie_cast  (cost=12.03..221.02 rows=1 width=8) (actual time=0.016..1.432 rows=222 loops=1)"
"                Recheck Cond: (mov_id = movie.mov_id)"
"                Rows Removed by Index Recheck: 9777"
"                Heap Blocks: lossy=84"
"                ->  Bitmap Index Scan on movie_cast_mov_act_id_brin  (cost=0.00..12.03 rows=9999 width=0) (actual time=0.011..0.011 rows=840 loops=1)"
"                      Index Cond: (mov_id = movie.mov_id)"
"Planning Time: 0.197 ms"
"Execution Time: 18.710 ms"
```

Total cost = 5024.49

10. Equivalent mixed indices

Initial configuration:

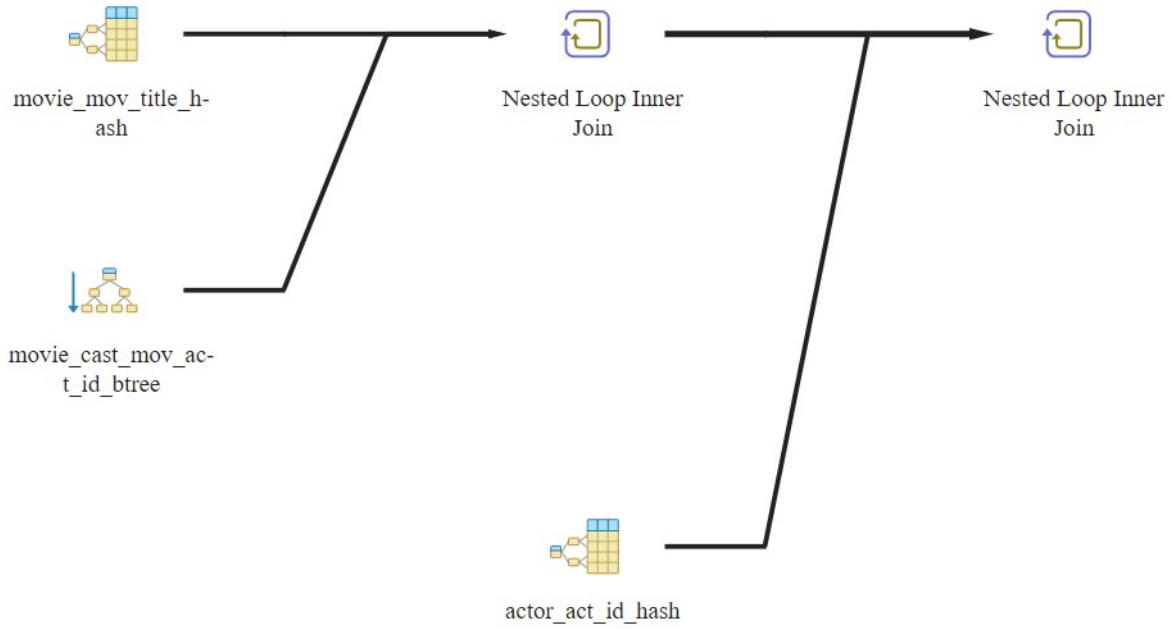
Create b+tree indices on actor table (act_id column), movie table (multi-dimensional on mov_id and mov_title) and on movie_cast (multi-dimensional on mov_id and act_id).

Create hash indices on actor table (act_id column), movie table (mov_id and mov_title) and on movie_cast (mov_id and act_id).

Create BRIN indices on actor table (act_id column), movie table (multi-dimensional on mov_id and mov_title) and on movie_cast (multi-dimensional on mov_id and act_id).

The query planner neglects BRIN indices and uses a mix of hash and b+tree indices as follows.

Query plan:



Analyze output:

"Nested Loop (cost=0.29..13.01 rows=1 width=48) (actual time=0.017..0.443 rows=222 loops=1)"

" -> Nested Loop (cost=0.29..12.33 rows=1 width=4) (actual time=0.014..0.040 rows=222 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.007..0.007 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Annie Hall'::bpchar)"

"   -> Index Only Scan using movie_cast_mov_act_id_btree on movie_cast (cost=0.29..4.30 rows=1 width=8) (actual time=0.006..0.022 rows=222 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   Heap Fetches: 0"

" -> Index Scan using actor_act_id_hash on actor (cost=0.00..0.67 rows=1 width=48) (actual time=0.001..0.002 rows=1 loops=222)"

"   Index Cond: (act_id = movie_cast.act_id)"

"Planning Time: 0.297 ms"

"Execution Time: 0.483 ms"

Total cost = 13.01

10. Equivalent Conclusion

When we use indices the cost and execution time are reduced even if use BRIN index which is not suitable for this query as it doesn’t contain ranges.

The B+tree index is the most suitable index for this query as it contains exact value searches on multiple columns from the same table so we can use multi-dimensional indices and we can see that it has the best cost.

We combine the multi-dimensional B+trees with the single-dimensional hash index to get the best cost of the query as the hash is faster in single-dimensional searches and multi-dimensional B+trees allows use to perform index-only scan which is faster as we retrieve all needed data from the index and we don’t go to the relation.

The equivalent query is better than the original because it makes use of more multi-dimensional indices.

Query 11:

Find the name of the director (first and last names) who directed a movie that casted a role for 'Eyes Wide Shut'.

```
select dir_fname, dir_lname

from director where dir_id in(select dir_id from movie_direction

where mov_id in (select mov_id from movie_cast

where role =any(select rolefrom movie_cast

where mov_id in(select mov_id from movie where mov_title='Eyes Wide Shut'))));
```

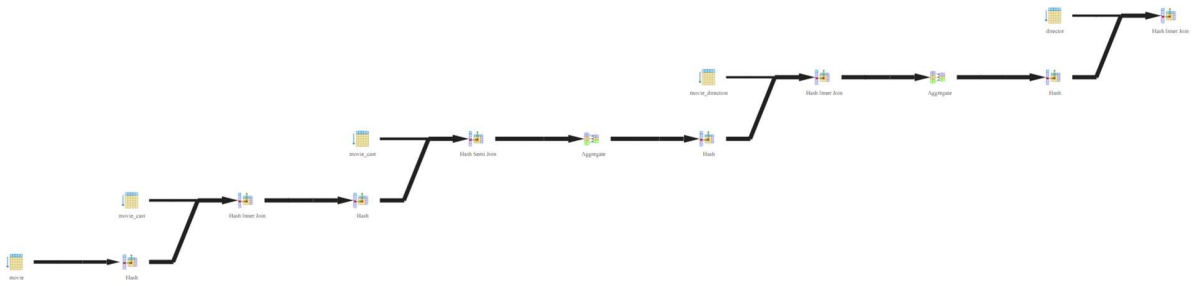
11. Raw Query

Initial configuration:

Disable all auto generated indices on movie_pkey, movie_cast_pkey, movie_direction_pkey and director_pkey to force the query planner to do seqscan on the tables using the command

update pg_index set indisvalid = false where indexrelid = ‘index_name’::regclass;

Query plan:



Analyze output:

"Hash Join (cost=3801.86..3940.17 rows=500 width=42) (actual time=19.681..20.817 rows=2101 loops=1)"

" Hash Cond: (director.dir_id = movie_direction.dir_id)"

" -> Seq Scan on director (cost=0.00..117.00 rows=6000 width=46) (actual time=0.010..0.322 rows=6000 loops=1)"

" -> Hash (cost=3795.61..3795.61 rows=500 width=4) (actual time=19.663..19.666 rows=2101 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 106kB"

"   -> HashAggregate (cost=3790.61..3795.61 rows=500 width=4) (actual time=19.312..19.470 rows=2101 loops=1)"

"   Group Key: movie_direction.dir_id"

"   Batches: 1 Memory Usage: 241kB"

"   -> Hash Join (cost=3612.56..3789.36 rows=500 width=4) (actual time=17.405..18.762 rows=3418 loops=1)"

"   Hash Cond: (movie_direction.mov_id = movie_cast.mov_id)"

"   -> Seq Scan on movie_direction (cost=0.00..144.99 rows=9999 width=8) (actual time=0.007..0.436 rows=9999 loops=1)"

"   -> Hash (cost=3606.31..3606.31 rows=500 width=4) (actual time=17.391..17.393 rows=3418 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 153kB"

"   -> HashAggregate (cost=3601.31..3606.31 rows=500 width=4) (actual time=16.813..17.067 rows=3418 loops=1)"

"   Group Key: movie_cast.mov_id"

```
"                        Batches: 1  Memory Usage: 369kB"
"                        ->  Hash Semi Join  (cost=3384.26..3600.06 rows=500 width=4) (actual time=14.442..16.164 rows=3500 loops=1)"
"                              Hash Cond: (movie_cast.role = movie_cast_1.role)"
"                              ->  Seq Scan on movie_cast  (cost=0.00..183.99 rows=9999 width=35) (actual time=0.007..0.448 rows=9999 loops=1)"
"                              ->  Hash  (cost=3384.25..3384.25 rows=1 width=31) (actual time=14.425..14.427 rows=7 loops=1)"
"                                    Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                                    ->  Hash Join  (cost=3174.01..3384.25 rows=1 width=31) (actual time=13.415..14.418 rows=7 loops=1)"
"                                          Hash Cond: (movie_cast_1.mov_id = movie.mov_id)"
"                                          ->  Seq Scan on movie_cast movie_cast_1  (cost=0.00..183.99 rows=9999 width=35) (actual time=0.006..0.445 rows=9999 loops=1)"
"                                          ->  Hash  (cost=3174.00..3174.00 rows=1 width=4) (actual time=13.375..13.376 rows=1 loops=1)"
"                                                Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                                                ->  Seq Scan on movie  (cost=0.00..3174.00 rows=1 width=4) (actual time=0.008..13.372 rows=1 loops=1)"
"                                                      Filter: (mov_title = 'Eyes Wide Shut'::bpchar)"
"                                                      Rows Removed by Filter: 99999"
"Planning Time: 0.373 ms"
"Execution Time: 21.085 ms"
```

Total cost = 3940.17

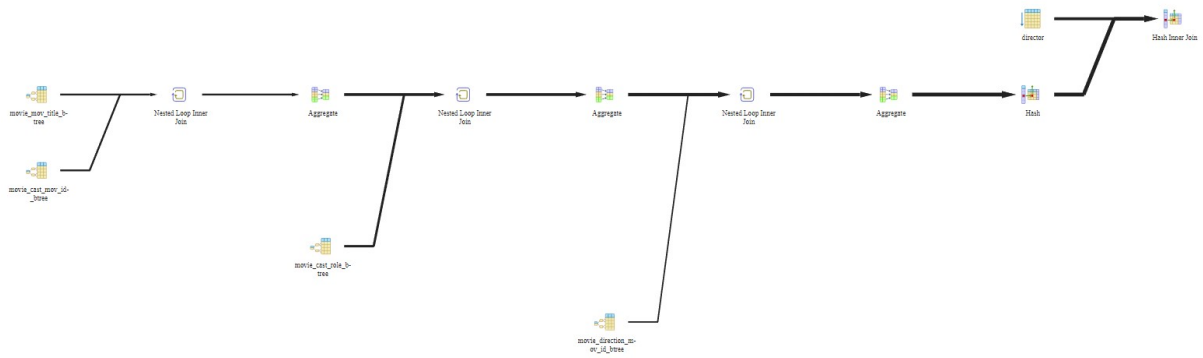
11. B+tree

Initial configuration:

Create b+tree indices on movie(mov_title), movie_cast(mov_id), director (dir_id), movie_cast(role) and movie_direction(mov_id).

The query planner doesn't use the director index as it will make the cost higher as we will go through almost the whole table.

Query plan:



Analyze output:

"Hash Join (cost=240.13..378.45 rows=500 width=42) (actual time=8.031..8.859 rows=2101 loops=1)"

" Hash Cond: (director.dir_id = movie_direction.dir_id)"

" -> Seq Scan on director (cost=0.00..117.00 rows=6000 width=46) (actual time=0.012..0.280 rows=6000 loops=1)"

" -> Hash (cost=233.88..233.88 rows=500 width=4) (actual time=8.012..8.014 rows=2101 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 106kB"

"   -> HashAggregate (cost=228.88..233.88 rows=500 width=4) (actual time=7.676..7.824 rows=2101 loops=1)"

"   Group Key: movie_direction.dir_id"

"   Batches: 1 Memory Usage: 241kB"

"   -> Nested Loop (cost=51.32..227.63 rows=500 width=4) (actual time=1.914..6.891 rows=3418 loops=1)"

"   -> HashAggregate (cost=51.03..56.03 rows=500 width=4) (actual time=1.905..2.393 rows=3418 loops=1)"

"   Group Key: movie_cast.mov_id"

"   Batches: 1 Memory Usage: 465kB"

"   -> Nested Loop (cost=17.04..49.78 rows=500 width=4) (actual time=0.040..1.002 rows=3500 loops=1)"

"   -> HashAggregate (cost=16.75..16.76 rows=1 width=31) (actual time=0.029..0.032 rows=7 loops=1)"

"   Group Key: movie_cast_1.role"

"   Batches: 1 Memory Usage: 24kB"

"   -> Nested Loop (cost=0.70..16.75 rows=1 width=31) (actual time=0.023..0.026 rows=7 loops=1)"

"   -> Index Scan using movie_mov_title_btree on movie (cost=0.42..8.44 rows=1 width=4) (actual time=0.014..0.015 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_btree on movie_cast movie_cast_1 (cost=0.29..8.30 rows=1 width=35) (actual time=0.007..0.008 rows=7 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   -> Index Scan using movie_cast_role_btree on movie_cast (cost=0.29..28.02 rows=500 width=35) (actual time=0.006..0.102 rows=500 loops=7)"

"   Index Cond: (role = movie_cast_1.role)"

"   -> Index Scan using movie_direction_mov_id_btree on movie_direction (cost=0.29..0.33 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3418)"

"   Index Cond: (mov_id = movie_cast.mov_id)"

"Planning Time: 0.523 ms"

"Execution Time: 9.096 ms"

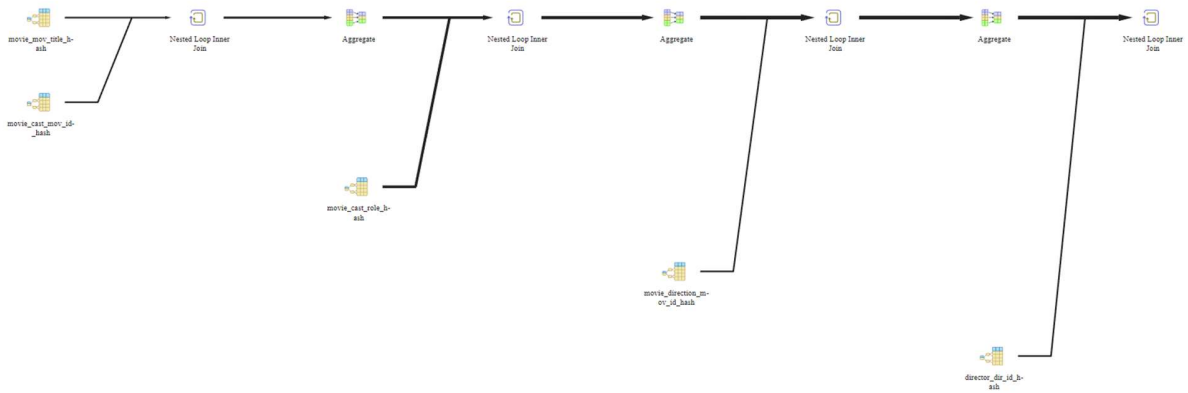
Total cost = 378.45

11. Hash

Initial configuration:

Create hash indices on movie(mov_title), movie_cast(mov_id), director (dir_id), movie_cast(role) and movie_direction(mov_id).

Query plan:



Analyze output:

"Nested Loop (cost=103.38..144.75 rows=500 width=42) (actual time=5.857..7.729 rows=2101 loops=1)"

" -> HashAggregate (cost=103.38..108.38 rows=500 width=4) (actual time=5.852..6.051 rows=2101 loops=1)"

"   Group Key: movie_direction.dir_id"

"   Batches: 1 Memory Usage: 241kB"

"   -> Nested Loop (cost=60.66..102.13 rows=500 width=4) (actual time=2.208..5.285 rows=3418 loops=1)"

"   -> HashAggregate (cost=60.66..65.66 rows=500 width=4) (actual time=2.202..2.558 rows=3418 loops=1)"

"   Group Key: movie_cast.mov_id"

"   Batches: 1 Memory Usage: 465kB"

"   -> Nested Loop (cost=16.05..59.41 rows=500 width=4) (actual time=0.026..1.281 rows=3500 loops=1)"

"   -> HashAggregate (cost=16.05..16.06 rows=1 width=31) (actual time=0.017..0.021 rows=7 loops=1)"

"   Group Key: movie_cast_1.role"

"   Batches: 1 Memory Usage: 24kB"

"   -> Nested Loop (cost=0.00..16.05 rows=1 width=31) (actual time=0.011..0.014 rows=7 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_hash on movie_cast movie_cast_1 (cost=0.00..8.02 rows=1 width=35) (actual time=0.004..0.006 rows=7 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   -> Index Scan using movie_cast_role_hash on movie_cast (cost=0.00..38.35 rows=500 width=35) (actual time=0.004..0.148 rows=500 loops=7)"

"   Index Cond: (role = movie_cast_1.role)"

"   -> Index Scan using movie_direction_mov_id_hash on movie_direction (cost=0.00..0.06 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3418)"

"   Index Cond: (mov_id = movie_cast.mov_id)"

" -> Index Scan using director_dir_id_hash on director (cost=0.00..0.08 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=2101)"

" Index Cond: (dir_id = movie_direction.dir_id)"

"Planning Time: 0.942 ms"

"Execution Time: 7.942 ms"

Total cost = 144.75

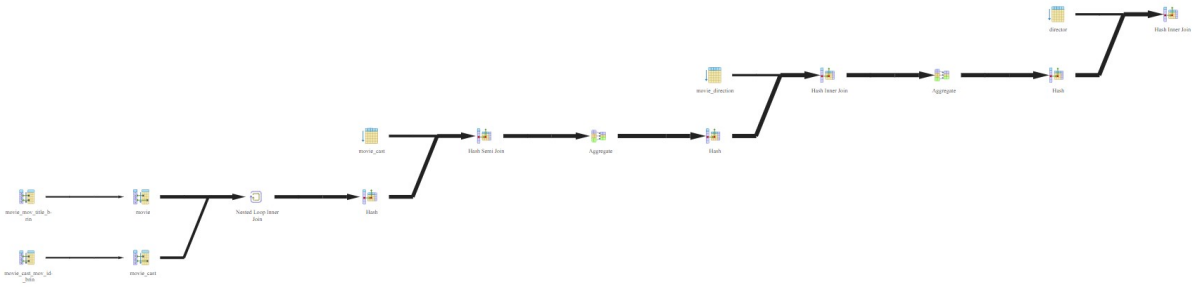
11. BRIN

Initial configuration:

Create BRIN indices on movie(mov_title), movie_cast(mov_id), director (dir_id), movie_cast(role) and movie_direction(mov_id).

The query planner decides to use only movie(mov_title) and movie_cast (mov_id) and neglect other indices to have the best cost.

Query plan:



Analyze output:

"Hash Join (cost=2670.08..2808.39 rows=500 width=42) (actual time=7.510..8.361 rows=2101 loops=1)"

" Hash Cond: (director.dir_id = movie_direction.dir_id)"

" -> Seq Scan on director (cost=0.00..117.00 rows=6000 width=46) (actual time=0.008..0.261 rows=6000 loops=1)"

" -> Hash (cost=2663.83..2663.83 rows=500 width=4) (actual time=7.495..7.498 rows=2101 loops=1)"

" Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 106kB"

" -> HashAggregate (cost=2658.83..2663.83 rows=500 width=4) (actual time=7.147..7.304 rows=2101 loops=1)"

" Group Key: movie_direction.dir_id"

" Batches: 1 Memory Usage: 241kB"

" -> Hash Join (cost=2480.78..2657.58 rows=500 width=4) (actual time=4.913..6.509 rows=3418 loops=1)"

" Hash Cond: (movie_direction.mov_id = movie_cast.mov_id)"

" -> Seq Scan on movie_direction (cost=0.00..144.99 rows=9999 width=8) (actual time=0.007..0.478 rows=9999 loops=1)"

" -> Hash (cost=2474.53..2474.53 rows=500 width=4) (actual time=4.899..4.902 rows=3418 loops=1)"

" Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 153kB"

" -> HashAggregate (cost=2469.53..2474.53 rows=500 width=4) (actual time=4.305..4.563 rows=3418 loops=1)"

" Group Key: movie_cast.mov_id"

" Batches: 1 Memory Usage: 369kB"

" -> Hash Semi Join (cost=2252.48..2468.28 rows=500 width=4) (actual time=1.681..3.596 rows=3500 loops=1)"

" Hash Cond: (movie_cast.role = movie_cast_1.role)"

" -> Seq Scan on movie_cast (cost=0.00..183.99 rows=9999 width=35) (actual time=0.008..0.539 rows=9999 loops=1)"

" -> Hash (cost=2252.47..2252.47 rows=1 width=31) (actual time=1.665..1.667 rows=7 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 9kB"

" -> Nested Loop (cost=24.07..2252.47 rows=1 width=31) (actual time=0.070..1.662 rows=7 loops=1)"

" -> Bitmap Heap Scan on movie (cost=12.04..2031.44 rows=1 width=4) (actual time=0.031..0.904 rows=1 loops=1)"

" Recheck Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

" Rows Removed by Index Recheck: 6655"

" Heap Blocks: lossy=128"

" -> Bitmap Index Scan on movie_mov_title_brin (cost=0.00..12.04 rows=7632 width=0) (actual time=0.026..0.027 rows=1280 loops=1)"

" Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

" -> Bitmap Heap Scan on movie_cast movie_cast_1 (cost=12.03..221.02 rows=1 width=35) (actual time=0.032..0.751 rows=7 loops=1)"

" Recheck Cond: (mov_id = movie.mov_id)"

" Rows Removed by Index Recheck: 9992"

" Heap Blocks: lossy=84"

" -> Bitmap Index Scan on movie_cast_mov_id_brin (cost=0.00..12.03 rows=9999 width=0) (actual time=0.013..0.013 rows=840 loops=1)"

" Index Cond: (mov_id = movie.mov_id)"

"Planning Time: 0.450 ms"

"Execution Time: 8.618 ms"

Total cost = 2808.39

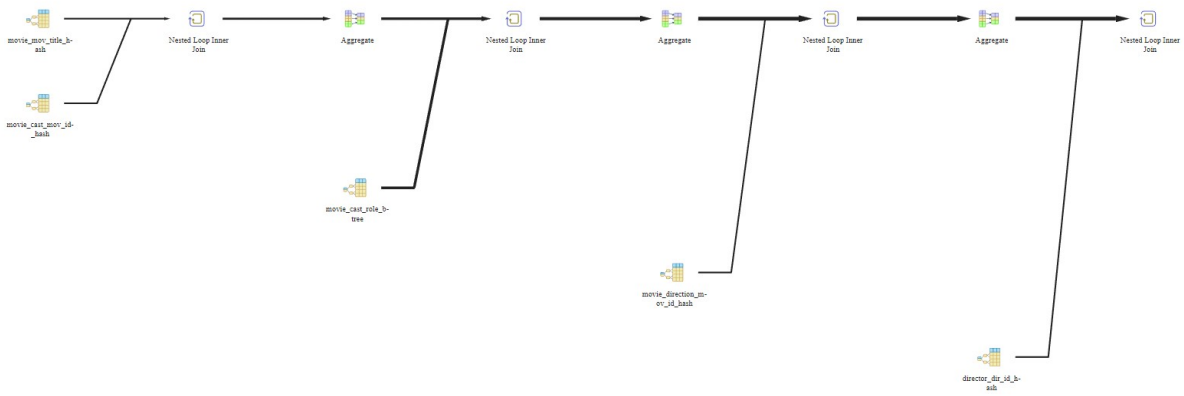
11. Mixed indices

Initial configuration:

Create all B+tree, hash and BRIN indices specified before.

The query planner decides to use one B+tree index and hash indices for the remaining columns.

Query plan:



Analyze output:

"Nested Loop (cost=93.05..134.42 rows=500 width=42) (actual time=5.583..7.125 rows=2101 loops=1)"

" -> HashAggregate (cost=93.05..98.05 rows=500 width=4) (actual time=5.579..5.738 rows=2101 loops=1)"

" Group Key: movie_direction.dir_id"

" Batches: 1 Memory Usage: 241kB"

" -> Nested Loop (cost=50.33..91.80 rows=500 width=4) (actual time=1.770..4.996 rows=3418 loops=1)"

" -> HashAggregate (cost=50.33..55.33 rows=500 width=4) (actual time=1.764..2.181 rows=3418 loops=1)"

" Group Key: movie_cast.mov_id"

" Batches: 1 Memory Usage: 465kB"

" -> Nested Loop (cost=16.33..49.08 rows=500 width=4) (actual time=0.024..0.965 rows=3500 loops=1)"

" -> HashAggregate (cost=16.05..16.06 rows=1 width=31) (actual time=0.016..0.019 rows=7 loops=1)"

" Group Key: movie_cast_1.role"

" Batches: 1 Memory Usage: 24kB"

" -> Nested Loop (cost=0.00..16.05 rows=1 width=31) (actual time=0.010..0.013 rows=7 loops=1)"

" -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)"

" Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

" -> Index Scan using movie_cast_mov_id_hash on movie_cast movie_cast_1 (cost=0.00..8.02 rows=1 width=35) (actual time=0.003..0.004 rows=7 loops=1)"

" Index Cond: (mov_id = movie.mov_id)"

" -> Index Scan using movie_cast_role_btree on movie_cast (cost=0.29..28.02 rows=500 width=35) (actual time=0.005..0.104 rows=500 loops=7)"

" Index Cond: (role = movie_cast_1.role)"

" -> Index Scan using movie_direction_mov_id_hash on movie_direction (cost=0.00..0.06 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3418)"

" Index Cond: (mov_id = movie_cast.mov_id)"

" -> Index Scan using director_dir_id_hash on director (cost=0.00..0.08 rows=1 width=46) (actual time=0.000..0.000 rows=1 loops=2101)"

" Index Cond: (dir_id = movie_direction.dir_id)"

"Planning Time: 0.535 ms"

"Execution Time: 7.312 ms"

Total cost = 134.42

11. Conclusion

When we use indices the cost and execution time are reduced even if use BRIN index which is not suitable for this query as it doesn't contain ranges.

The hash index is the most suitable index for this query as it contains exact value searches and we can see this from its cost.

The hash index has the smallest cost. But we can use one B+tree index which has less cost instead of the corresponding hash index to make the mix better.

Query 11 equivalent query:

We can't make a huge optimization for this query because the in clause translates to an inner join in the query planner and it uses hash join so it's the most optimal.

But we can give an equivalent query for it which has slightly worse cost for raw query and better performance with indices (because it makes the query planner use multi-dimensional indices more and perform aggregation only at the end) which is:

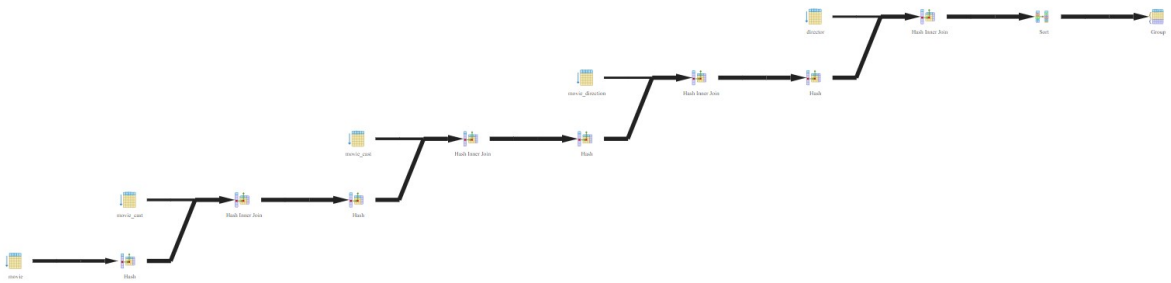
```
select dir_fname, dir_lname

from director d

inner join movie_direction md on d.dir_id = md.dir_id

inner join movie_cast mc on md.mov_id = mc.mov_id

inner join movie_cast mc2 on mc.role = mc2.role

inner join (select mov_id from movie where mov_title='Eyes Wide Shut') as m

on mc2.mov_id = m.mov_id

group by dir_fname,dir_lname
```

11. Equivalent Raw Query

Initial configuration:

The same initial configuration as the original raw query

Query plan:



Analyze output:

"Group (cost=4080.14..4080.52 rows=50 width=42) (actual time=28.038..28.630 rows=2101 loops=1)"

" Group Key: d.dir_fname, d.dir_lname"

" -> Sort (cost=4080.14..4080.27 rows=50 width=42) (actual time=28.035..28.132 rows=3500 loops=1)"

"   Sort Key: d.dir_fname, d.dir_lname"

"   Sort Method: quicksort Memory: 370kB"

"   -> Hash Join (cost=3931.23..4078.73 rows=50 width=42) (actual time=15.634..16.780 rows=3500 loops=1)"

"   Hash Cond: (d.dir_id = md.dir_id)"

"   -> Seq Scan on director d (cost=0.00..117.00 rows=6000 width=46) (actual time=0.010..0.294 rows=6000 loops=1)"

"   -> Hash (cost=3930.61..3930.61 rows=50 width=4) (actual time=15.616..15.619 rows=3500 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 156kB"

"   -> Hash Join (cost=3622.63..3930.61 rows=50 width=4) (actual time=13.663..15.236 rows=3500 loops=1)"

"   Hash Cond: (md.mov_id = mc.mov_id)"

"   -> Seq Scan on movie_direction md (cost=0.00..144.99 rows=9999 width=8) (actual time=0.008..0.481 rows=9999 loops=1)"

"   -> Hash (cost=3622.01..3622.01 rows=50 width=4) (actual time=13.650..13.652 rows=3500 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 156kB"

"   -> Hash Join (cost=3395.52..3622.01 rows=50 width=4) (actual time=11.368..13.229 rows=3500 loops=1)"

"   Hash Cond: (mc.role = mc2.role)"

" -> Seq Scan on movie_cast mc (cost=0.00..183.99 rows=9999 width=35) (actual time=0.007..0.461 rows=9999 loops=1)"

" -> Hash (cost=3395.51..3395.51 rows=1 width=31) (actual time=11.351..11.352 rows=7 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 9kB"

" -> Hash Join (cost=3174.01..3395.51 rows=1 width=31) (actual time=10.283..11.342 rows=7 loops=1)"

" Hash Cond: (mc2.mov_id = movie.mov_id)"

" -> Seq Scan on movie_cast mc2 (cost=0.00..183.99 rows=9999 width=35) (actual time=0.007..0.483 rows=9999 loops=1)"

" -> Hash (cost=3174.00..3174.00 rows=1 width=4) (actual time=10.243..10.244 rows=1 loops=1)"

" Buckets: 1024 Batches: 1 Memory Usage: 9kB"

" -> Seq Scan on movie (cost=0.00..3174.00 rows=1 width=4) (actual time=0.008..10.240 rows=1 loops=1)"

" Filter: (mov_title = 'Eyes Wide Shut'::bpchar)"

" Rows Removed by Filter: 99999"

"Planning Time: 0.343 ms"

"Execution Time: 28.816 ms"

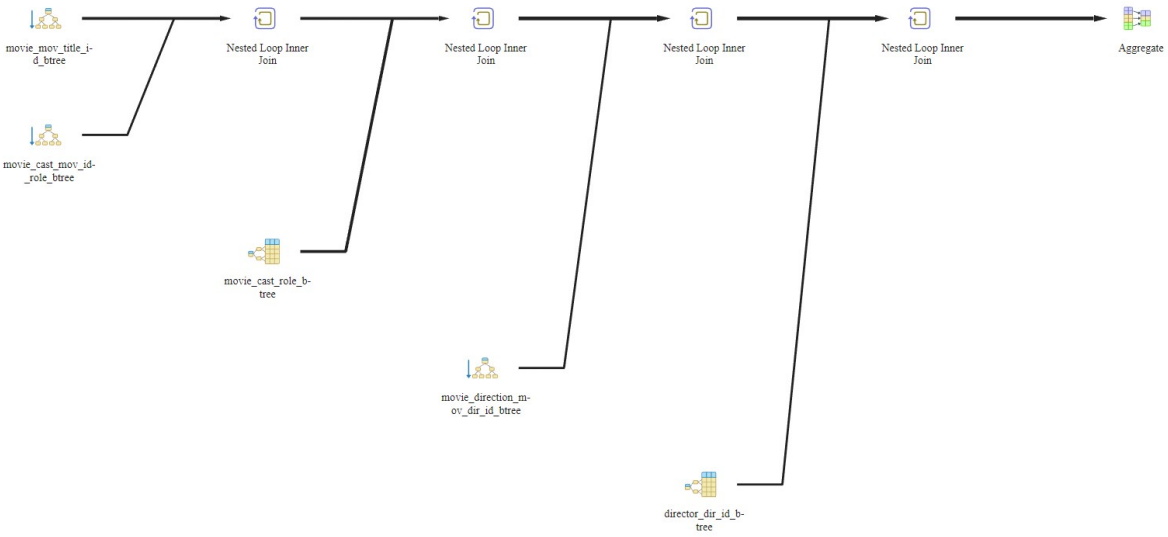
Total cost = 4080.27

11. Equivalent B+tree

Initial configuration:

Create b+tree indices on movie (multi-dimensional on mov_id and mov_title), movie_cast (multi-dimensional on mov_id and role), movie_cast (role), movie_direction(multi on dir_id and mov_id) and director(dir_id)

Query plan:



Analyze output:

"HashAggregate (cost=60.32..60.82 rows=50 width=42) (actual time=10.468..10.731 rows=2101 loops=1)"

" Group Key: d.dir_fname, d.dir_lname"

" Batches: 1 Memory Usage: 393kB"

" -> Nested Loop (cost=1.55..60.07 rows=50 width=42) (actual time=0.043..9.378 rows=3500 loops=1)"

"   -> Nested Loop (cost=1.27..43.05 rows=50 width=4) (actual time=0.038..4.978 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.99..26.82 rows=50 width=4) (actual time=0.033..1.076 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.70..12.75 rows=1 width=31) (actual time=0.023..0.026 rows=7 loops=1)"

"   -> Index Only Scan using movie_mov_title_id_btree on movie (cost=0.42..8.44 rows=1 width=4) (actual time=0.016..0.016 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   Heap Fetches: 0"

"   -> Index Only Scan using movie_cast_mov_id_role_btree on movie_cast mc2 (cost=0.29..4.30 rows=1 width=35) (actual time=0.005..0.007 rows=7 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   Heap Fetches: 0"

" -> Index Scan using movie_cast_role_btree on movie_cast mc (cost=0.29..9.07 rows=500 width=35) (actual time=0.006..0.116 rows=500 loops=7)"

" Index Cond: (role = mc2.role)"

" -> Index Only Scan using movie_direction_mov_dir_id_btree on movie_direction md (cost=0.29..0.31 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3500)"

" Index Cond: (mov_id = mc.mov_id)"

" Heap Fetches: 0"

" -> Index Scan using director_dir_id_btree on director d (cost=0.28..0.33 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=3500)"

" Index Cond: (dir_id = md.dir_id)"

"Planning Time: 0.806 ms"

"Execution Time: 10.862 ms"

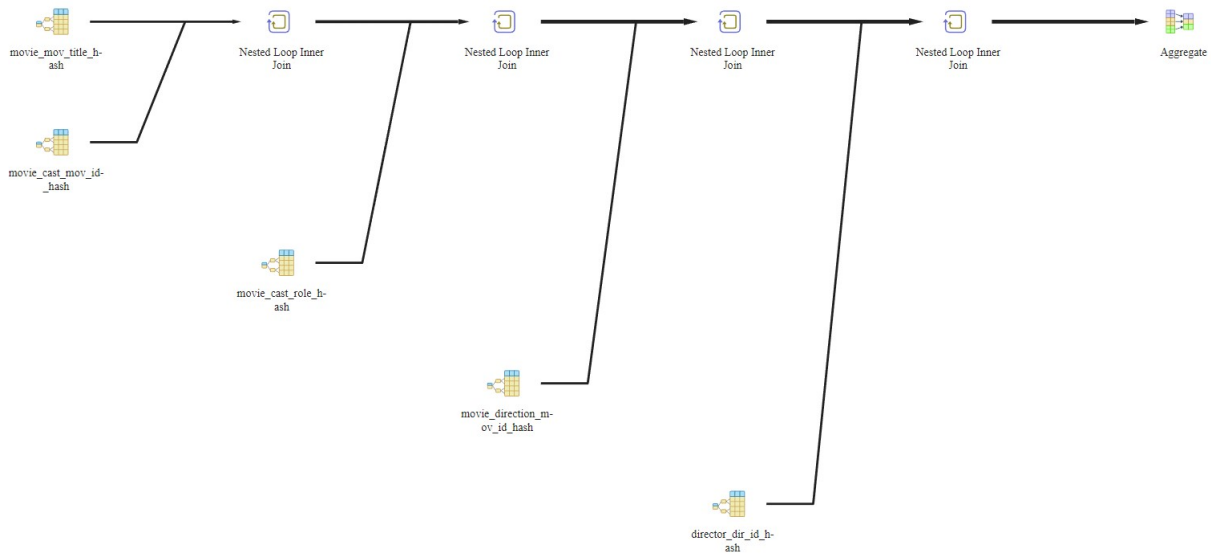
Total cost = 60.82

11. Equivalent hash

Initial configuration:

Create hash indices on movie (mov_title), movie_cast (mov_id), movie_cast (role), movie_direction (mov_id) and director (dir_id)

Query plan:



Analyze output:

"HashAggregate (cost=36.90..37.40 rows=50 width=42) (actual time=8.571..8.781 rows=2101 loops=1)"

" Group Key: d.dir_fname, d.dir_lname"

" Batches: 1 Memory Usage: 393kB"

" -> Nested Loop (cost=0.00..36.65 rows=50 width=42) (actual time=0.033..7.512 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.00..33.46 rows=50 width=4) (actual time=0.029..4.568 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.00..29.86 rows=50 width=4) (actual time=0.024..1.235 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.00..16.05 rows=1 width=31) (actual time=0.018..0.024 rows=7 loops=1)"

"   -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)"

"   Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   -> Index Scan using movie_cast_mov_id_hash on movie_cast mc2 (cost=0.00..8.02 rows=1 width=35) (actual time=0.009..0.012 rows=7 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   -> Index Scan using movie_cast_role_hash on movie_cast mc (cost=0.00..8.82 rows=500 width=35) (actual time=0.004..0.144 rows=500 loops=7)"

"   Index Cond: (role = mc2.role)"

"   -> Index Scan using movie_direction_mov_id_hash on movie_direction md (cost=0.00..0.06 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3500)"

"   Index Cond: (mov_id = mc.mov_id)"

"   -> Index Scan using director_dir_id_hash on director d (cost=0.00..0.05 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=3500)"

"   Index Cond: (dir_id = md.dir_id)"

"Planning Time: 0.746 ms"

"Execution Time: 8.925 ms"

Total cost = 37.40

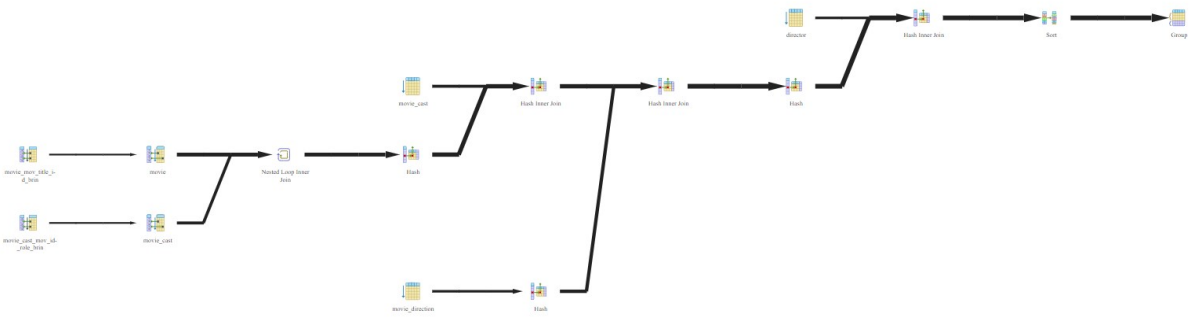
11. Equivalent BRIN

Initial configuration:

Create BRIN indices on movie (multi-dimensional on mov_id and mov_title), movie_cast (multi-dimensional on mov_id and role), movie_cast (role), movie_direction(multi on dir_id and mov_id) and director(dir_id)

The query planner decides to use only some of the multi-dimensional indices and neglects other indices because they will produce higher cost as other tables have a lot of rows being accessed so seqscan is better.

Query plan:



Analyze output:

"Group (cost=2899.17..2899.54 rows=50 width=42) (actual time=19.340..19.944 rows=2101 loops=1)"

" Group Key: d.dir_fname, d.dir_lname"

" -> Sort (cost=2899.17..2899.29 rows=50 width=42) (actual time=19.337..19.447 rows=3500 loops=1)"

"   Sort Key: d.dir_fname, d.dir_lname"

"   Sort Method: quicksort Memory: 370kB"

"   -> Hash Join (cost=2750.26..2897.76 rows=50 width=42) (actual time=7.369..8.408 rows=3500 loops=1)"

"   Hash Cond: (d.dir_id = md.dir_id)"

"   -> Seq Scan on director d (cost=0.00..117.00 rows=6000 width=46) (actual time=0.010..0.299 rows=6000 loops=1)"

"   -> Hash (cost=2749.63..2749.63 rows=50 width=4) (actual time=7.351..7.354 rows=3500 loops=1)"

"   Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 156kB"

"   -> Hash Join (cost=2522.46..2749.63 rows=50 width=4) (actual time=3.885..6.915 rows=3500 loops=1)"

"   Hash Cond: (mc.mov_id = md.mov_id)"

"   -> Hash Join (cost=2252.48..2478.97 rows=50 width=4) (actual time=2.457..4.705 rows=3500 loops=1)"

"   Hash Cond: (mc.role = mc2.role)"

"   -> Seq Scan on movie_cast mc (cost=0.00..183.99 rows=9999 width=35) (actual time=0.018..0.585 rows=9999 loops=1)"

"   -> Hash (cost=2252.47..2252.47 rows=1 width=31) (actual time=2.412..2.414 rows=7 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 9kB"

"   -> Nested Loop (cost=24.07..2252.47 rows=1 width=31) (actual time=0.171..2.407 rows=7 loops=1)"

"   -> Bitmap Heap Scan on movie (cost=12.04..2031.44 rows=1 width=4) (actual time=0.064..0.857 rows=1 loops=1)"

"   Recheck Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   Rows Removed by Index Recheck: 6655"

"   Heap Blocks: lossy=128"

"   -> Bitmap Index Scan on movie_mov_title_id_brin (cost=0.00..12.04 rows=7632 width=0) (actual time=0.056..0.056 rows=1280 loops=1)"

"   Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

"   -> Bitmap Heap Scan on movie_cast mc2 (cost=12.03..221.02 rows=1 width=35) (actual time=0.092..1.533 rows=7 loops=1)"

"   Recheck Cond: (mov_id = movie.mov_id)"

"   Rows Removed by Index Recheck: 9992"

"   Heap Blocks: lossy=84"

"   -> Bitmap Index Scan on movie_cast_mov_id_role_brin (cost=0.00..12.03 rows=9999 width=0) (actual time=0.034..0.034 rows=840 loops=1)"

"   Index Cond: (mov_id = movie.mov_id)"

"   -> Hash (cost=144.99..144.99 rows=9999 width=8) (actual time=1.377..1.378 rows=9999 loops=1)"

"   Buckets: 16384 Batches: 1 Memory Usage: 519kB"

"   -> Seq Scan on movie_direction md (cost=0.00..144.99 rows=9999 width=8) (actual time=0.008..0.484 rows=9999 loops=1)"

"Planning Time: 0.784 ms"

"Execution Time: 20.217 ms"

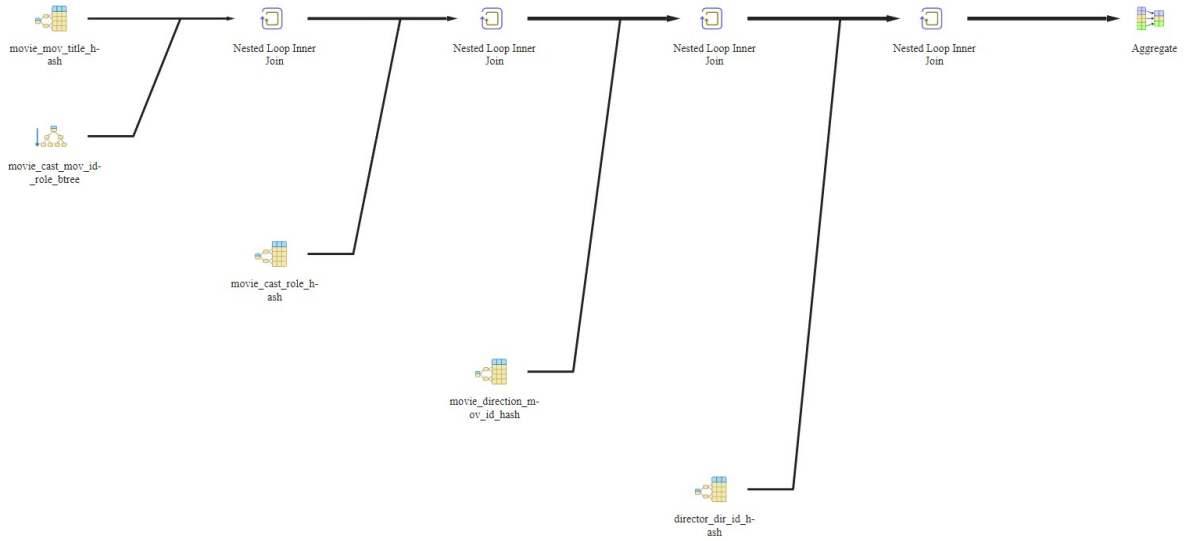
Total cost = 2899.54

11. Equivalent mixed indices

Create all B+tree, hash and BRIN indices specified before.

The query planner neglects BRIN indices and uses a mix of hash and b+tree indices as follows.

Query plan:



Analyze output:

"HashAggregate (cost=33.19..33.69 rows=50 width=42) (actual time=8.501..8.699 rows=2101 loops=1)"

" Group Key: d.dir_fname, d.dir_lname"

" Batches: 1 Memory Usage: 393kB"

" -> Nested Loop (cost=0.29..32.94 rows=50 width=42) (actual time=0.031..7.508 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.29..29.74 rows=50 width=4) (actual time=0.025..4.391 rows=3500 loops=1)"

"   -> Nested Loop (cost=0.29..26.15 rows=50 width=4) (actual time=0.022..1.211 rows=3500 loops=1)"

" -> Nested Loop (cost=0.29..12.33 rows=1 width=31) (actual time=0.016..0.022 rows=7 loops=1)"

" -> Index Scan using movie_mov_title_hash on movie (cost=0.00..8.02 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)"

" Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)"

" -> Index Only Scan using movie_cast_mov_id_role_btree on movie_cast mc2 (cost=0.29..4.30 rows=1 width=35) (actual time=0.009..0.013 rows=7 loops=1)"

" Index Cond: (mov_id = movie.mov_id)"

" Heap Fetches: 0"

" -> Index Scan using movie_cast_role_hash on movie_cast mc (cost=0.00..8.82 rows=500 width=35) (actual time=0.004..0.144 rows=500 loops=7)"

" Index Cond: (role = mc2.role)"

" -> Index Scan using movie_direction_mov_id_hash on movie_direction md (cost=0.00..0.06 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=3500)"

" Index Cond: (mov_id = mc.mov_id)"

" -> Index Scan using director_dir_id_hash on director d (cost=0.00..0.05 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=3500)"

" Index Cond: (dir_id = md.dir_id)"

"Planning Time: 1.062 ms"

"Execution Time: 8.817 ms"

Total cost = 33.69

11. Equivalent Conclusion

When we use indices the cost and execution time are reduced even if use BRIN index which is not suitable for this query as it doesn't contain ranges.

The hash index is the most suitable index for this query as it contains exact value searches and we can see that it has the best cost.

We combine hash indices with multi-dimensional B+tree to get the best cost of the query as the hash is faster in single-dimensional searches and multi-dimensional B+trees allows use to perform index-only scan which is faster as we retrieve all needed data from the index and we don't go to the relation.

The equivalent query is better than the original because it has less cost when we use the indices.

Query 12:

Find the titles of all movies directed by the director whose first and last name are Woddy Allen.

```
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
```

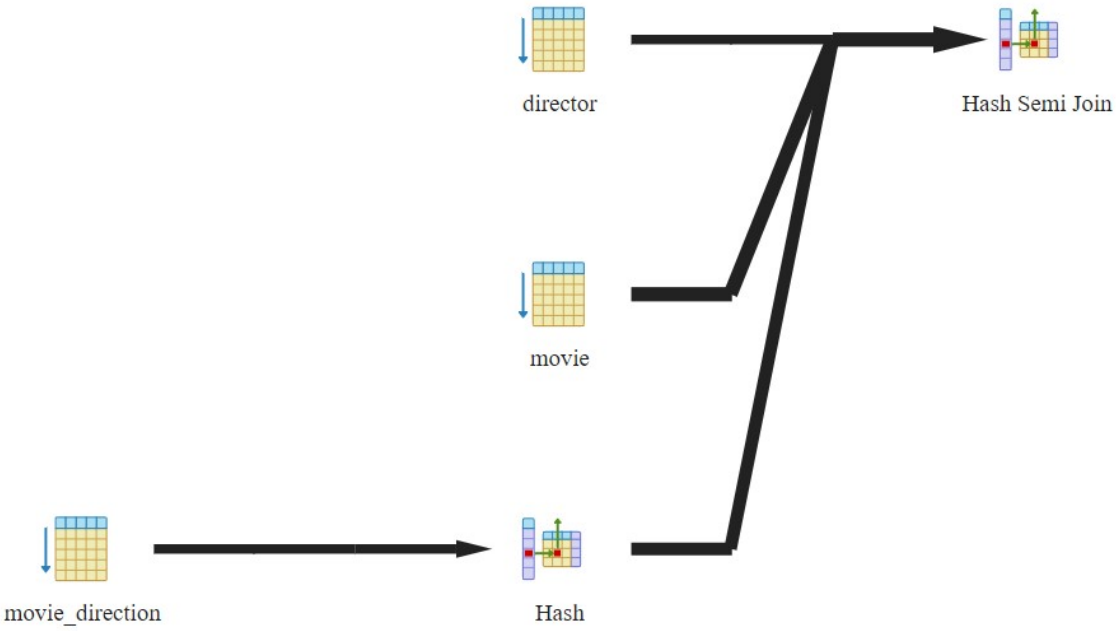
12. Raw Query

Initial configuration:

Disable all auto generated indices on movie_pkey, movie_direction_pkey and director_pkey to force the query planner to do seqscan on the tables using the command

update pg_index set indisvalid = false where indexrelid = ‘index_name’::regclass;

Query plan:



Analyze output:

"Hash Semi Join (cost=317.01..3503.53 rows=2 width=51) (actual time=1.122..12.889 rows=350 loops=1)"

" Hash Cond: (movie.mov_id = movie_direction.mov_id)"

" InitPlan 1 (returns $0)"

"   -> Seq Scan on director (cost=0.00..147.00 rows=1 width=4) (actual time=0.008..0.455 rows=1 loops=1)"

"   Filter: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   Rows Removed by Filter: 5999"

" -> Seq Scan on movie (cost=0.00..2924.00 rows=100000 width=55) (actual time=0.012..4.953 rows=100000 loops=1)"

" -> Hash (cost=169.99..169.99 rows=2 width=4) (actual time=1.105..1.106 rows=350 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 21kB"

"   -> Seq Scan on movie_direction (cost=0.00..169.99 rows=2 width=4) (actual time=0.464..1.078 rows=350 loops=1)"

" Filter: (dir_id = $0)"

" Rows Removed by Filter: 9649"

"Planning Time: 0.389 ms"

"Execution Time: 12.915 ms"

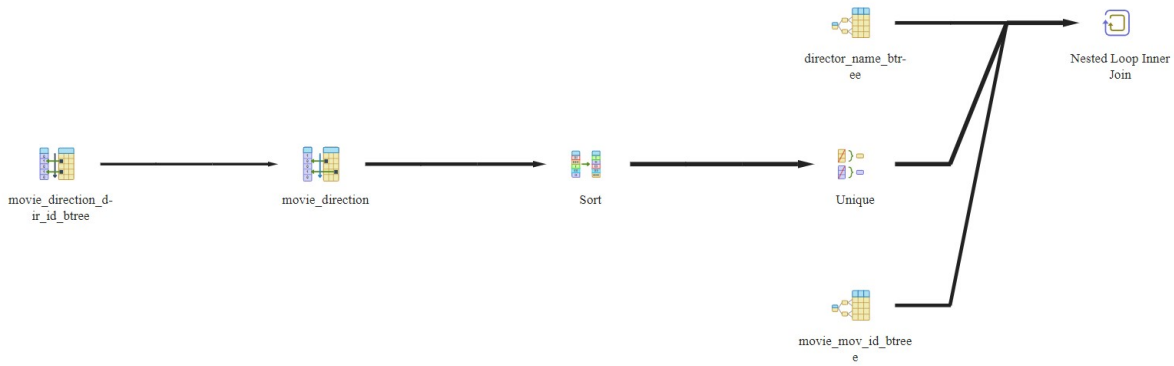
Total cost = 3503.53

12. B+tree

Initial configuration:

Create b+tree indices on movie (mov_id), director (multi on fname and lname) and movie_direction (dir_id).

Query plan:



Analyze output:

"Nested Loop (cost=19.67..36.02 rows=2 width=51) (actual time=0.193..1.450 rows=350 loops=1)"

" InitPlan 1 (returns $0)"

" -> Index Scan using director_name_btree on director (cost=0.28..8.30 rows=1 width=4) (actual time=0.021..0.022 rows=1 loops=1)"

" Index Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

" -> Unique (cost=11.07..11.08 rows=2 width=4) (actual time=0.186..0.365 rows=350 loops=1)"

" -> Sort (cost=11.07..11.08 rows=2 width=4) (actual time=0.185..0.226 rows=350 loops=1)"

" Sort Key: movie_direction.mov_id"

" Sort Method: quicksort Memory: 41kB"

" -> Bitmap Heap Scan on movie_direction (cost=4.30..11.06 rows=2 width=4) (actual time=0.058..0.118 rows=350 loops=1)"

" Recheck Cond: (dir_id = $0)"

" Heap Blocks: exact=3"

" -> Bitmap Index Scan on movie_direction_dir_id_btree (cost=0.00..4.30 rows=2 width=0) (actual time=0.042..0.043 rows=350 loops=1)"

" Index Cond: (dir_id = $0)"

" -> Index Scan using movie_mov_id_btree on movie (cost=0.29..8.31 rows=1 width=55) (actual time=0.002..0.002 rows=1 loops=350)"

" Index Cond: (mov_id = movie_direction.mov_id)"

"Planning Time: 0.331 ms"

"Execution Time: 1.520 ms"

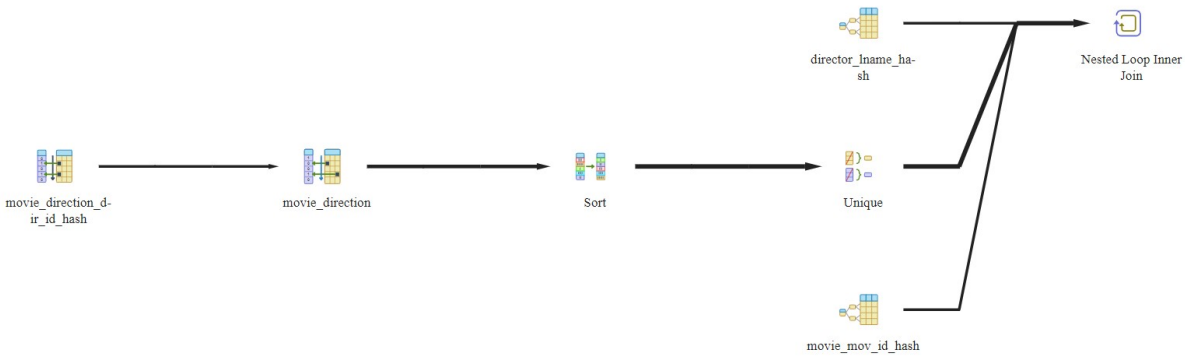
Total cost = 36.02

12. Hash

Initial configuration:

Create hash indices on movie (mov_id), director (fname), director (lname) and movie_direction (dir_id). The query planner prefers the lname index to the fname one.

Query plan:



Analyze output:

"Nested Loop (cost=18.81..34.87 rows=2 width=51) (actual time=0.088..0.648 rows=350 loops=1)"

```
"  InitPlan 1 (returns $0)"
"    ->  Index Scan using director_lname_hash on director  (cost=0.00..8.02 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1)"
"          Index Cond: (dir_lname = 'Allen'::bpchar)"
"          Filter: (dir_fname = 'Woddy'::bpchar)"
"  ->  Unique  (cost=10.79..10.80 rows=2 width=4) (actual time=0.084..0.130 rows=350 loops=1)"
"        ->  Sort  (cost=10.79..10.79 rows=2 width=4) (actual time=0.084..0.099 rows=350 loops=1)"
"              Sort Key: movie_direction.mov_id"
"              Sort Method: quicksort  Memory: 41kB"
"              ->  Bitmap Heap Scan on movie_direction  (cost=4.02..10.78 rows=2 width=4) (actual time=0.031..0.060 rows=350 loops=1)"
"                    Recheck Cond: (dir_id = $0)"
"                    Heap Blocks: exact=3"
"                    ->  Bitmap Index Scan on movie_direction_dir_id_hash  (cost=0.00..4.01 rows=2 width=0) (actual time=0.023..0.023 rows=350 loops=1)"
"                          Index Cond: (dir_id = $0)"
"  ->  Index Scan using movie_mov_id_hash on movie  (cost=0.00..8.02 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=350)"
"        Index Cond: (mov_id = movie_direction.mov_id)"
"Planning Time: 0.169 ms"
"Execution Time: 0.684 ms"
```

Total cost = 34.87

12. BRIN

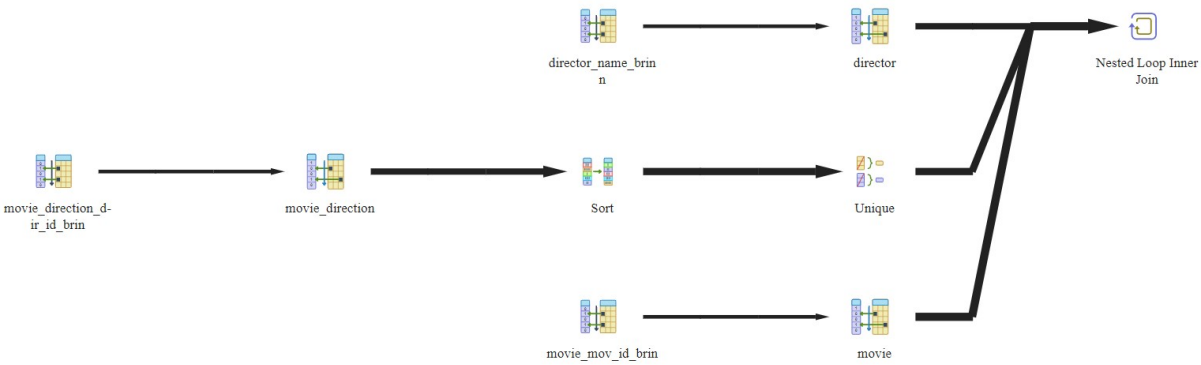
Initial configuration:

Create BRIN indices on movie (mov_id), director (multi on fname and lname) and movie_direction (dir_id).

The query planner decides not to use any of the indices because it will produce higher cost so we force it to use them by disabling seqscan with this command:

Set enable_seqscan = off;

Query plan:



Analyze output:

"Nested Loop (cost=365.09..4159.99 rows=2 width=51) (actual time=1.903..191.463 rows=350 loops=1)"

" InitPlan 1 (returns $0)"

"   -> Bitmap Heap Scan on director (cost=12.03..159.03 rows=1 width=4) (actual time=0.033..0.883 rows=1 loops=1)"

"   Recheck Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   Rows Removed by Index Recheck: 5999"

"   Heap Blocks: lossy=57"

"   -> Bitmap Index Scan on director_name_brin (cost=0.00..12.03 rows=6000 width=0) (actual time=0.025..0.025 rows=570 loops=1)"

"   Index Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

" -> Unique (cost=182.03..182.04 rows=2 width=4) (actual time=1.870..2.008 rows=350 loops=1)"

"   -> Sort (cost=182.03..182.03 rows=2 width=4) (actual time=1.869..1.911 rows=350 loops=1)"

"   Sort Key: movie_direction.mov_id"

"   Sort Method: quicksort Memory: 41kB"

"   -> Bitmap Heap Scan on movie_direction (cost=12.03..182.02 rows=2 width=4) (actual time=0.911..1.831 rows=350 loops=1)"

"   Recheck Cond: (dir_id = $0)"

"   Rows Removed by Index Recheck: 9649"

"   Heap Blocks: lossy=45"

"   -> Bitmap Index Scan on movie_direction_dir_id_brin (cost=0.00..12.03 rows=9999 width=0) (actual time=0.904..0.904 rows=450 loops=1)"

"   Index Cond: (dir_id = $0)"

" -> Bitmap Heap Scan on movie (cost=24.03..1909.45 rows=1 width=55) (actual time=0.029..0.534 rows=1 loops=350)"

"   Recheck Cond: (mov_id = movie_direction.mov_id)"

"   Rows Removed by Index Recheck: 6655"

"   Heap Blocks: lossy=44800"

"   -> Bitmap Index Scan on movie_mov_id_brin (cost=0.00..24.03 rows=6250 width=0) (actual time=0.013..0.013 rows=1280 loops=350)"

"   Index Cond: (mov_id = movie_direction.mov_id)"

"Planning Time: 0.200 ms"

"Execution Time: 191.785 ms"

Total cost = 4159.99

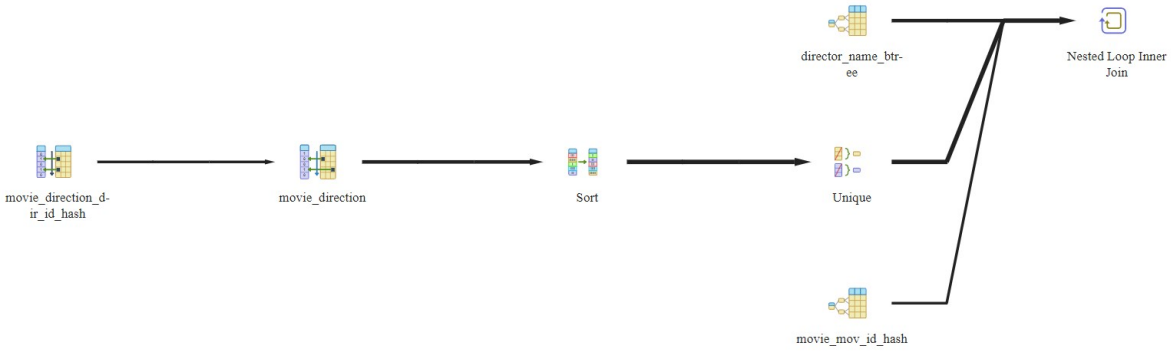
12. Mixed indices

Initial configuration:

Create all B+tree, hash and BRIN indices specified before.

The query planner decides to use one B+tree index (the multi-dimensional index on director name) and hash indices for the remaining columns.

Query plan:



Analyze output:

"Nested Loop (cost=19.09..35.15 rows=2 width=51) (actual time=0.103..0.615 rows=350 loops=1)"

" InitPlan 1 (returns $0)"

"   -> Index Scan using director_name_btree on director (cost=0.28..8.30 rows=1 width=4) (actual time=0.017..0.018 rows=1 loops=1)"

"   Index Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

" -> Unique (cost=10.79..10.80 rows=2 width=4) (actual time=0.099..0.141 rows=350 loops=1)"

"   -> Sort (cost=10.79..10.79 rows=2 width=4) (actual time=0.099..0.109 rows=350 loops=1)"

"   Sort Key: movie_direction.mov_id"

"   Sort Method: quicksort Memory: 41kB"

"   -> Bitmap Heap Scan on movie_direction (cost=4.02..10.78 rows=2 width=4) (actual time=0.044..0.074 rows=350 loops=1)"

"   Recheck Cond: (dir_id = $0)"

"   Heap Blocks: exact=3"

"   -> Bitmap Index Scan on movie_direction_dir_id_hash (cost=0.00..4.01 rows=2 width=0) (actual time=0.035..0.035 rows=350 loops=1)"

"   Index Cond: (dir_id = $0)"

" -> Index Scan using movie_mov_id_hash on movie (cost=0.00..8.02 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=350)"

"   Index Cond: (mov_id = movie_direction.mov_id)"

"Planning Time: 0.181 ms"

"Execution Time: 0.655 ms"

Total cost = 35.15

12. Conclusion

When we use b+tree and hash indices the cost and execution time are reduced but if we force BRIN index the cost is increased as the BRIN is more suitable for analytical queries and range ones not exact value ones.

The hash index is the most suitable index for this query as it contains exact value searches and we can see this from its cost.

We can use a mix of hash and b+tree indices to get an in-between cost and execution time of those of the hash only and b+tree only using the better index in each scenario.

Query 12 equivalent query:

We can't make a huge optimization for this query because the in clause translates to an inner join in the query planner and it uses hash join so it's the most optimal.

But we can give an equivalent query for it which has slightly worse cost for raw query and better performance with indices (because it makes the query planner use multi-dimensional indices more and perform aggregation only at the end) which is:

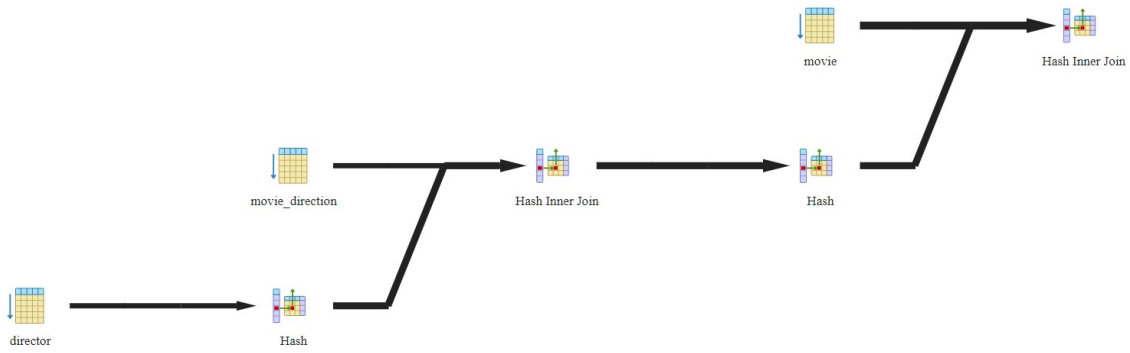
```
select mov_title
from (select dir_id from director where dir_fname='Woddy' and dir_lname='Allen') as d
inner join movie_direction md
on d.dir_id = md.dir_id
inner join movie m
on m.mov_id = md.mov_id
```

12. Equivalent Raw Query

Initial configuration:

The same initial configuration as the original raw query

Query plan:



Analyze output:

"Hash Join (cost=329.54..3628.56 rows=2 width=51) (actual time=1.583..13.448 rows=350 loops=1)"

" Hash Cond: (m.mov_id = md.mov_id)"

" -> Seq Scan on movie m (cost=0.00..2924.00 rows=100000 width=55) (actual time=0.010..5.212 rows=100000 loops=1)"

" -> Hash (cost=329.52..329.52 rows=2 width=4) (actual time=1.568..1.570 rows=350 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 21kB"

"   -> Hash Join (cost=147.01..329.52 rows=2 width=4) (actual time=0.469..1.535 rows=350 loops=1)"

"   Hash Cond: (md.dir_id = director.dir_id)"

"   -> Seq Scan on movie_direction md (cost=0.00..144.99 rows=9999 width=8) (actual time=0.008..0.415 rows=9999 loops=1)"

"   -> Hash (cost=147.00..147.00 rows=1 width=4) (actual time=0.453..0.454 rows=1 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 9kB"

"   -> Seq Scan on director (cost=0.00..147.00 rows=1 width=4) (actual time=0.009..0.446 rows=1 loops=1)"

"   Filter: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   Rows Removed by Filter: 5999"

"Planning Time: 0.324 ms"

"Execution Time: 13.477 ms"

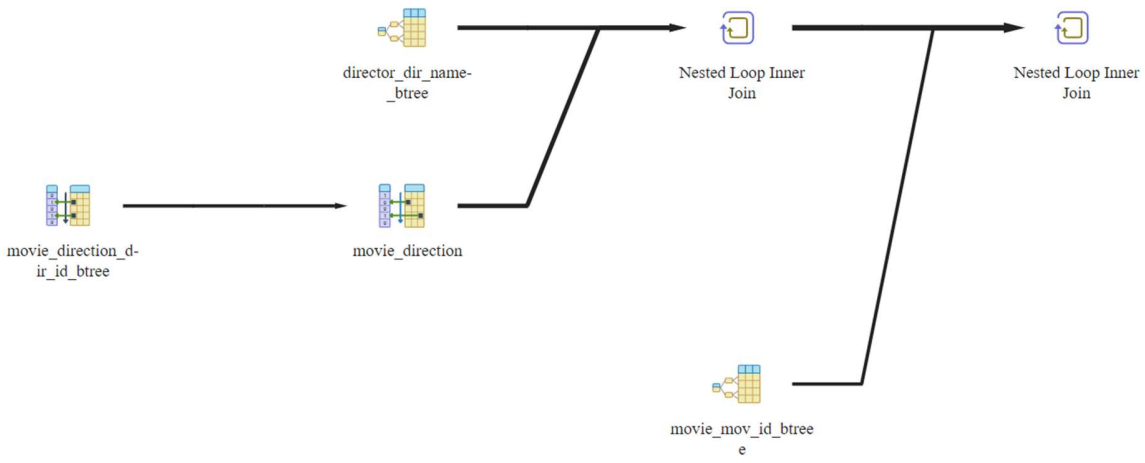
Total cost = 3628.56

12. Equivalent B+tree

Initial configuration:

Create b+tree indices on movie (mov_id), director (multi on fname and lname) and movie_direction (dir_id).

Query plan:



Analyze output:

"Nested Loop (cost=4.88..21.78 rows=2 width=51) (actual time=0.041..0.534 rows=350 loops=1)"

" -> Nested Loop (cost=4.58..19.38 rows=2 width=4) (actual time=0.038..0.082 rows=350 loops=1)"

"   -> Index Scan using director_dir_name_btree on director (cost=0.28..8.30 rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)"

"   Index Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   -> Bitmap Heap Scan on movie_direction md (cost=4.30..11.06 rows=2 width=8) (actual time=0.019..0.042 rows=350 loops=1)"

"   Recheck Cond: (dir_id = director.dir_id)"

"   Heap Blocks: exact=3"

"   -> Bitmap Index Scan on movie_direction_dir_id_btree (cost=0.00..4.30 rows=2 width=0) (actual time=0.008..0.009 rows=350 loops=1)"

"   Index Cond: (dir_id = director.dir_id)"

" -> Index Scan using movie_mov_id_btree on movie m (cost=0.29..1.19 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=350)"

"   Index Cond: (mov_id = md.mov_id)"

"Planning Time: 0.619 ms"

"Execution Time: 0.569 ms"

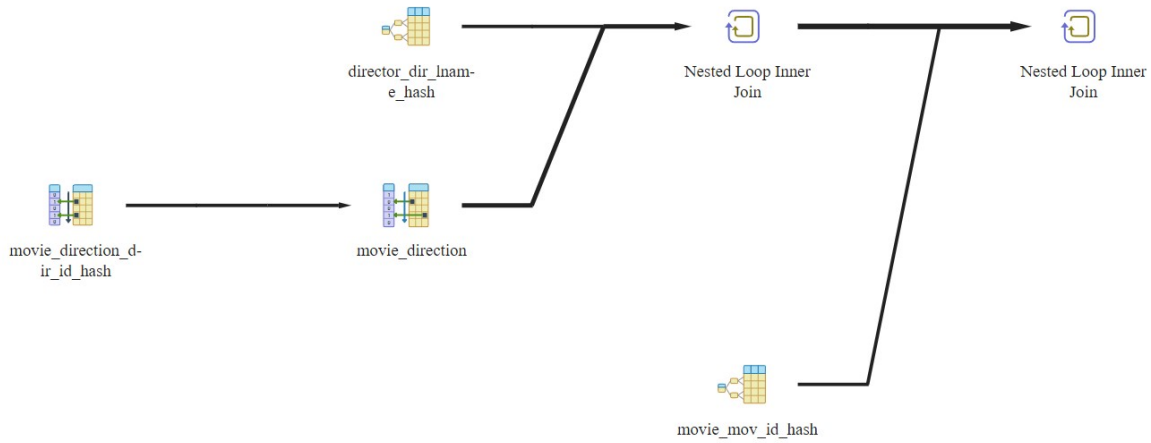
Total cost = 21.78

12. Equivalent hash

Initial configuration:

Create hash indices on movie (mov_id), director (fname), director (lname) and movie_direction (dir_id). The query planner prefers the lname index to the fname one.

Query plan:



Analyze output:

"Nested Loop (cost=4.02..20.82 rows=2 width=51) (actual time=0.037..0.661 rows=350 loops=1)"

" -> Nested Loop (cost=4.02..18.82 rows=2 width=4) (actual time=0.033..0.085 rows=350 loops=1)"

"   -> Index Scan using director_dir_lname_hash on director (cost=0.00..8.02 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)"

"   Index Cond: (dir_lname = 'Allen'::bpchar)"

"   Filter: (dir_fname = 'Woddy'::bpchar)"

" -> Bitmap Heap Scan on movie_direction md (cost=4.02..10.78 rows=2 width=8) (actual time=0.020..0.052 rows=350 loops=1)"

" Recheck Cond: (dir_id = director.dir_id)"

" Heap Blocks: exact=3"

" -> Bitmap Index Scan on movie_direction_dir_id_hash (cost=0.00..4.01 rows=2 width=0) (actual time=0.012..0.012 rows=350 loops=1)"

" Index Cond: (dir_id = director.dir_id)"

" -> Index Scan using movie_mov_id_hash on movie m (cost=0.00..0.99 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=350)"

" Index Cond: (mov_id = md.mov_id)"

"Planning Time: 0.222 ms"

"Execution Time: 0.696 ms"

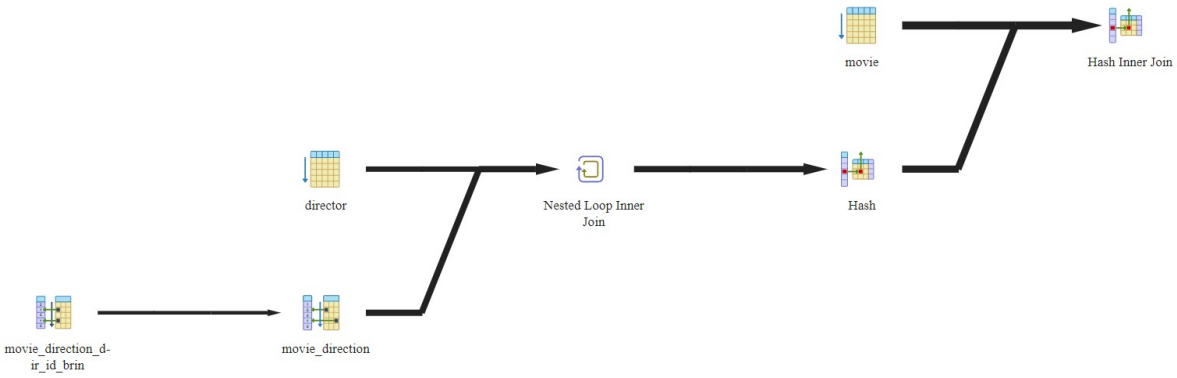
Total cost = 20.82

12. Equivalent BRIN

Initial configuration:

Create BRIN indices on movie (mov_id), director (multi on fname and lname) and movie_direction (dir_id). The query planner only uses the movie_direction one.

Query plan:



Analyze output:

"Hash Join (cost=329.06..3628.09 rows=2 width=51) (actual time=1.365..13.825 rows=350 loops=1)"

" Hash Cond: (m.mov_id = md.mov_id)"

" -> Seq Scan on movie m (cost=0.00..2924.00 rows=100000 width=55) (actual time=0.008..5.498 rows=100000 loops=1)"

" -> Hash (cost=329.04..329.04 rows=2 width=4) (actual time=1.350..1.352 rows=350 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 21kB"

"   -> Nested Loop (cost=12.03..329.04 rows=2 width=4) (actual time=0.035..1.318 rows=350 loops=1)"

"   -> Seq Scan on director (cost=0.00..147.00 rows=1 width=4) (actual time=0.009..0.493 rows=1 loops=1)"

"   Filter: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   Rows Removed by Filter: 5999"

"   -> Bitmap Heap Scan on movie_direction md (cost=12.03..182.02 rows=2 width=8) (actual time=0.020..0.801 rows=350 loops=1)"

"   Recheck Cond: (dir_id = director.dir_id)"

"   Rows Removed by Index Recheck: 9649"

"   Heap Blocks: lossy=45"

"   -> Bitmap Index Scan on movie_direction_dir_id_brin (cost=0.00..12.03 rows=9999 width=0) (actual time=0.013..0.013 rows=450 loops=1)"

"   Index Cond: (dir_id = director.dir_id)"

"Planning Time: 0.259 ms"

"Execution Time: 13.869 ms"

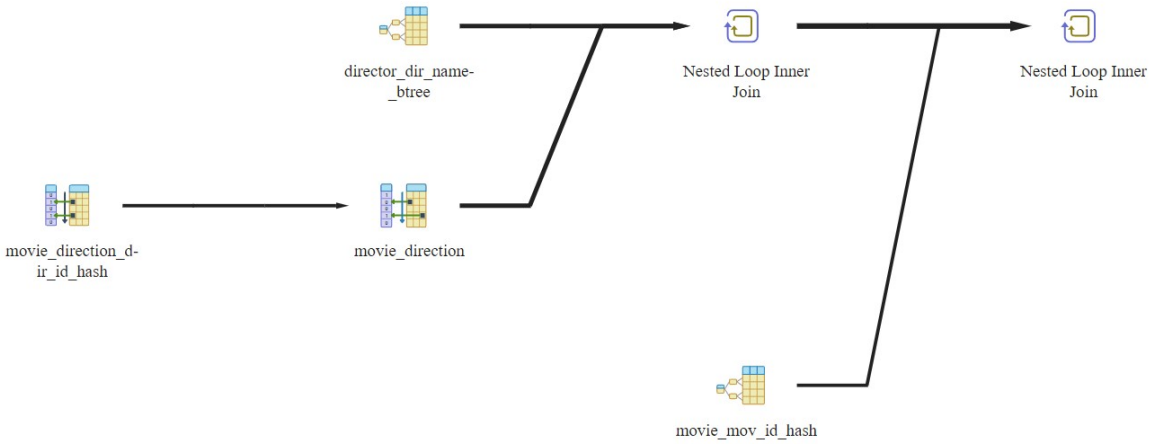
Total cost = 3628.09

12. Equivalent mixed indices

Create all B+tree, hash and BRIN indices specified before.

The query planner neglects BRIN indices and uses a mix of hash and b+tree indices as follows.

Query plan:



Analyze output:

"Nested Loop (cost=4.30..21.10 rows=2 width=51) (actual time=0.052..0.801 rows=350 loops=1)"

" -> Nested Loop (cost=4.30..19.10 rows=2 width=4) (actual time=0.049..0.151 rows=350 loops=1)"

"   -> Index Scan using director_dir_name_btree on director (cost=0.28..8.30 rows=1 width=4) (actual time=0.014..0.015 rows=1 loops=1)"

"   Index Cond: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))"

"   -> Bitmap Heap Scan on movie_direction md (cost=4.02..10.78 rows=2 width=8) (actual time=0.027..0.069 rows=350 loops=1)"

"   Recheck Cond: (dir_id = director.dir_id)"

"   Heap Blocks: exact=3"

"   -> Bitmap Index Scan on movie_direction_dir_id_hash (cost=0.00..4.01 rows=2 width=0) (actual time=0.016..0.016 rows=350 loops=1)"

"   Index Cond: (dir_id = director.dir_id)"

" -> Index Scan using movie_mov_id_hash on movie m (cost=0.00..0.99 rows=1 width=55) (actual time=0.002..0.002 rows=1 loops=350)"

"   Index Cond: (mov_id = md.mov_id)"

"Planning Time: 0.423 ms"

"Execution Time: 0.840 ms"

Total cost = 21.10

12. Equivalent Conclusion

When we use indices the cost and execution time are reduced even if use BRIN index which is not suitable for this query as it doesn’t contain ranges.

BRIN index is accepted by the query planner in this query and makes performance slightly better which is opposite to the original query.

The hash index is the most suitable index for this query as it contains exact value searches and we can see that it has the best cost.

We combine hash indices with multi-dimensional B+tree to get the best cost of the query as the hash is faster in single-dimensional searches and multi-dimensional B+trees allows use to perform search on multiple columns at the same time.

The equivalent query is better than the original because it has less cost when we use the indices.