

Q1: original

=>Without indices (only seqscans)

The screenshot shows the pgAdmin interface with the following details:

- Servers:** PostgreSQL 14
- Databases:** postgres, schema1, schema2, schema3, schema4
- Schema:** schema1
- Query Editor:** The query is:1 explain analyze
2 select *
3 from (select *
4 from student
5 where
6 department = 'CSEN') as CS1_student
7 full outer join
8 (select *
9 from takes t inner join section s
10 on t.section_id = s.section_id
11 where semester = 1
12 and
13 year = 2019) as semi_student
14 on CS1_student.id = semi_student.student_id;
- Messages:** No messages.
- Notifications:** No notifications.
- Explain:** The Explain plan shows the following steps:
 - Hash Full Join (cost=358.74..2263.74 rows=250 width=66) (actual time=14.561..14.666 rows=250 loops=1)
 - [...] Hash Cond: (student.id = t.student_id)
 - [...] > Seq Scan on student (cost=0.00..1904.99 rows=1 width=26) (actual time=10.726..10.726 rows=0 loops=1)
 - [...] Filter: ((department)=> 'CSEN')
 - [...] Rows Removed by Filter: 69999
 - [...] > Hash (cost=355.61..355.61 rows=250 width=40) (actual time=3.814..3.818 rows=250 loops=1)
 - [...] Buckets: 1024 Batches: 1 Memory Usage: 26kB
 - [...] > Hash Join (cost=39.56..355.61 rows=250 width=40) (actual time=0.344..3.717 rows=250 loops=1)
 - [...] Hash Cond: (t.section_id = s.section_id)
 - [...] > Seq Scan on takes t (cost=0.00..270.00 rows=17500 width=12) (actual time=0.020..1.348 rows=17500 loops=1)
 - [...] > Hash (cost=39.25..39.25 rows=25 width=28) (actual time=0.284..0.286 rows=25 loops=1)
 - [...] Buckets: 1024 Batches: 1 Memory Usage: 10kB
 - [...] > Seq Scan on section s (cost=0.00..39.25 rows=25 width=28) (actual time=0.019..0.274 rows=25 loops=1)
 - [...] Filter: ((semester = 1) AND (year = 2019))
 - [...] Rows Removed by Filter: 1725
 - Planning Time: 0.858 ms
 - Execution Time: 14.736 ms
- Data Output:** No data output shown.

Cost=2263, time=14.7ms

Notes:

All relation access are seqscans only, no indeces created

⇒ Using Btrees: on section(semester,year), student(department)

```

drop index idx_sec_sem
explain analyze
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 semi_student
  on CS1_student.id = semi_student.student_id;
create index idx_sec_sem on section using btree(semester,year );
--create index idx_sec_id on section using btree( section_id );
--create index idx_tak_scid on takes using btree(section_id );
--create index idx_tak_stid on takes using btree(student_id );
create index idx_stu_dpt on student using btree(department );
--create index idx_stu_id on student using btree(id );

```

QUERY PLAN

- 1 Hash Full Join (cost=26.54..343.25 rows=250 width=66) (actual time=0.373..4.951 rows=250 loops=1)
 - 2 [...] Hash Cond: (tstudent_id = student.id)
 - 3 [...] > Hash Join (cost=18.22..334.27 rows=250 width=40) (actual time=0.274..4.743 rows=250 loops=1)
 - 4 [...] Hash Cond: (tsection_id = s.section_id)
 - 5 [...] > Seq Scan on takes t (cost=0.00..270.00 rows=17500 width=12) (actual time=0.042..1.806 rows=17500 loops=1)
 - 6 [...] > Hash (cost=17.91..17.91 rows=25 width=28) (actual time=0.159..0.160 rows=25 loops=1)
 - 7 [...] Buckets: 1024 Batches: 1 Memory Usage: 10kB
 - 8 [...] > Bitmap Heap Scan on section s (cost=4.53..17.91 rows=25 width=28) (actual time=0.057..0.130 rows=25 loops=1)
 - 9 [...] Recheck Cond: ((semester = 1) AND (year = 2019))
 - 10 [...] Heap Blocks: exact=13
 - 11 [...] > Bitmap Index Scan on idx_sec_sem (cost=0.00..4.53 rows=25 width=0) (actual time=0.040..0.040 rows=25 loops=1)
 - 12 [...] Index Cond: ((semester = 1) AND (year = 2019))
 - 13 [...] > Hash (cost=8.31..8.31 rows=1 width=26) (actual time=0.064..0.065 rows=0 loops=1)
 - 14 [...] Buckets: 1024 Batches: 1 Memory Usage: 8kB
 - 15 [...] > Index Scan using idx_stu_dpt on student (cost=0.29..8.31 rows=1 width=26) (actual time=0.061..0.061 rows=0)
 - 16 [...] Index Cond: (department::text = 'CSEN':text)
 - 17 Planning Time: 0.762 ms
 - 18 Execution Time: 5.165 ms

Cost=343.25, time=5.16ms

Notes:

The btree indices reduced the cost to 15.16% of original cost, time to 35% of original time.

The indeces made the search for semester=1,year=2019,department='CSEN' much faster, so instead of doing seqscan, we do index scan.

⇒ Using Hash: on section(semester),section(year), student(department)

The screenshot shows the PgAdmin interface with a query editor containing the following SQL code:

```

1 drop index idx_sec_sem;
2 drop index idx_stu_dpt;
3 drop index idx_sec_sem;
4
5 explain analyze
6 select *
7 from (select *
8   from student
9   where
10      department = 'CSEN') as CS1_student
11 full outer join
12 (select *
13   from takes t inner join section s
14   on t.section_id = s.section_id
15   where semester = 1
16   and
17      year = 2019) as sem1_student
18   on CS1_student.id = sem1_student.student_id;
19
20 create index idx_sec_sem on section using hash(semester );
21 create index idx_sec_year on section using hash(year );
22 --create index idx_sec_id on section using btree( section_id );
23 --create index idx_tak_scid on takes using btree(section_id );
24 --create index idx_tak_stdid on takes using btree(student_id );
25 --create index idx_stu_dpt on student using hash(department );
26 create index idx_stu_dpt on student using hash(department );
27 --create index idx_stu_id on student using btree(id );

```

The right side of the interface displays the "QUERY PLAN" for this query, showing various stages of execution including Hash Full Join, Hash Cond, Seq Scan, Bitmap Scan, and Index Scan.

Cost=349.99, time=6.1ms

Notes:

The hash is usually alittle faster than btrees, but here it's not the case, because btree can have multicolumns, so it can check both semester=1 and check year =2019, but hash has to chose one and filter the other which takes some more time.

⇒ Using BRIN: on section(year), student(department), takes(section_id)

```

1 drop index idx_sec_sem;
2 drop index idx_stu_dpt;
3 drop index idx_sec_year;
4 drop index idx_sec_id;
5
6 explain analyze
7 select *
8 from (select *
9 from student
10 where
11 department = 'CSEN') as CS1_student
12 full outer join
13 (select *
14 from takes t inner join section s
15 on t.section_id = s.section_id
16 where semester = 1
17 and
18 year = 2019) as sem1_student
19 on CS1_student.id = sem1_student.student_id;
20
21 create index idx_sec_sem on section using brin(semester );
22 create index idx_sec_year on section using brin(year );
23 create index idx_sec_id on section using brin( section_id );
24 --create index idx_tak_scid on takes using btree(section_id );
25 --create index idx_tak_scid on takes using btree(student_id );
26 create index idx_stu_dpt on student using brin(department );
27 --create index idx_stu_id on student using btree(id );
28
29 set enable_seqscan=off

```

The screenshot shows the pgAdmin interface with a query editor containing the above SQL code. The 'Query Plan' tab is selected, displaying a detailed execution plan with steps like Hash Full Join, Bitmap Scan, and Index Cond. The plan spans from line 1 to line 29 of the query.

Cost=2392, time=2ms

24 Execution Time: 2.035 ms

Go to Settings to activate Windows.

Notes:

Since the brin is not good in exact values, the engine wasn't willing to use any of those indeces, but we turned seqscan off so it used the indeces mention above the screenshot, it increased the cost. The reason why brin is not good at this type of queries is explained in another query analysis.

- ⇒ Using mixed indeces: btree on section(semester,year), hash on student(department)

```

drop index idx_sec_sem;
drop index idx_stu_dpt;
drop index idx_sec_year;
drop index idx_sec_id;

explain analyze
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 semi_student
       on CS1_student.id = semi_student.student_id;

create index idx_sec_sem on section using btree(semester,year );
--create index idx_sec_year on section using brin(year );
--create index idx_sec_id on section using brin( section_id );
--create index idx_tak_stid on takes using btree(section_id );
--create index idx_tak_stid on takes using htree(student_id );

```

The screenshot shows the PgAdmin 4 interface. The left sidebar displays the database structure with several schemas (schema1, schema2, schema3, schema4) and their contents. The main area has two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab contains the SQL code above. The 'Data Output' tab is active and shows the 'EXPLAIN ANALYZE' results, which include the query plan, execution time (Cost=342.96, time=5.15ms), and memory usage details.

Cost=342.96, time=5.15ms

Notes:

In this mixed types, we make use of ability to have multicolm of btree to check both year and semester, hash is alittle bit faster so we used it to search about department (single column) so the cost decreased a little compared to the btree section above as the difference in performance between btree and hash index

Q2:Original

=>Without indices (only seqscans)

The screenshot shows the pgAdmin interface with a query in the Query Editor and its corresponding EXPLAIN PLAN results in the Data Output tab. The query is:

```
1 explain analyze
2 select distinct pnumber
3 from project
4 where pnumber in
5 (select pnumber
6 from project, department d, employee e
7 where e.dno=d.dnumber
8 and
9 d.mgr_snn=ssn
10 and
11 e.lname='employee1' )
12 or
13 pnumber in
14 (select pno
15 from works_on, employee
16 where essn=ssn and lname='employee1' );
17
18 --update pg_index set indisvalid = false where indexrelid = 'wor
```

The EXPLAIN PLAN output shows a complex nested loop join with multiple sequential scans (Seq Scan) on tables project, department, employee, and works_on. The cost for the query is 2040.07, and the actual time is 15.8ms.

cost=2040,time=15.8ms-----

Flags:

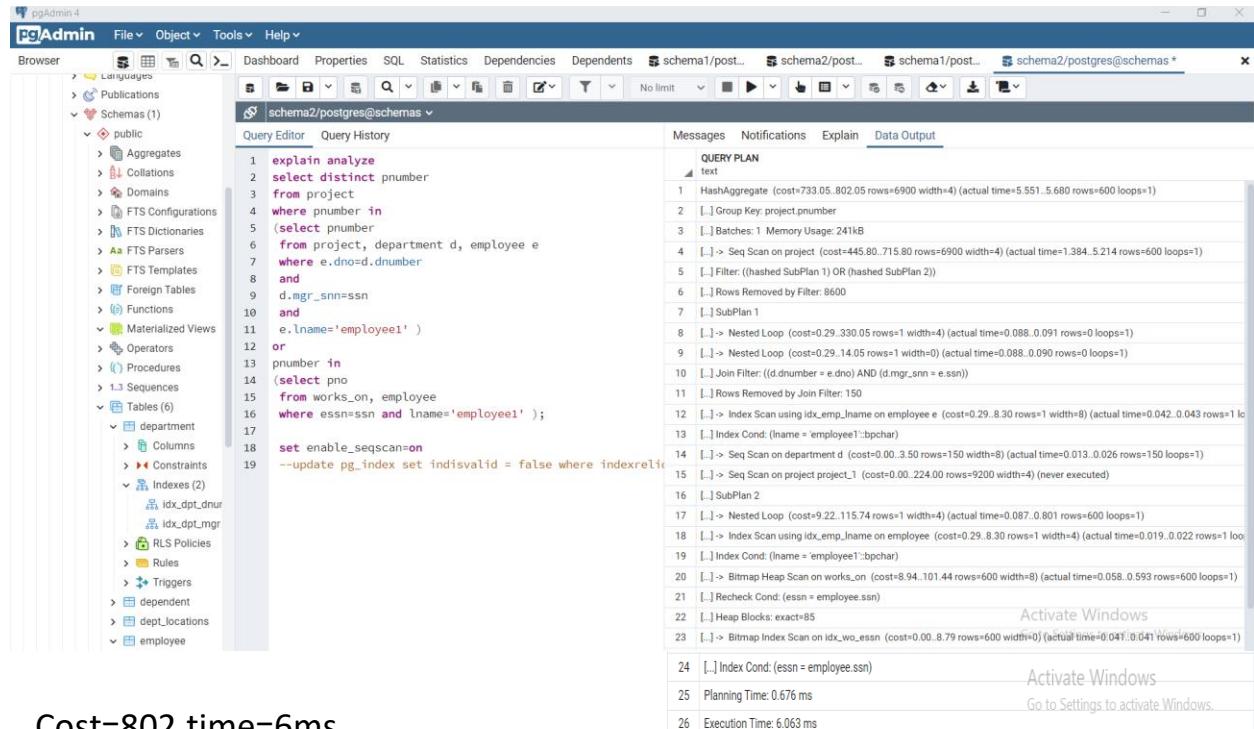
Turn off all index scan flags and turn on seqscan only

Notes:

All scans over all tables are done using seqscan

=>Using B+Trees:

-Btrees indices are used on employee(lname), works_on(ssn)



```
1 explain analyze
2 select distinct pnumber
3 from project
4 where pnumber in
5 (select pnumber
6 from project, department d, employee e
7 where e.dno=d.dnumber
8 and
9 d.mgr_ssn=ssn
10 and
11 e.lname='employee1' )
12 or
13 pnumber in
14 (select pno
15 from works_on, employee
16 where essn=ssn and lname='employee1' );
17
18 set enable_seqscan=on
19 --update pg_index set indisvalid = false where indexrelid = 10
```

Cost=802,time=6ms

Notes:

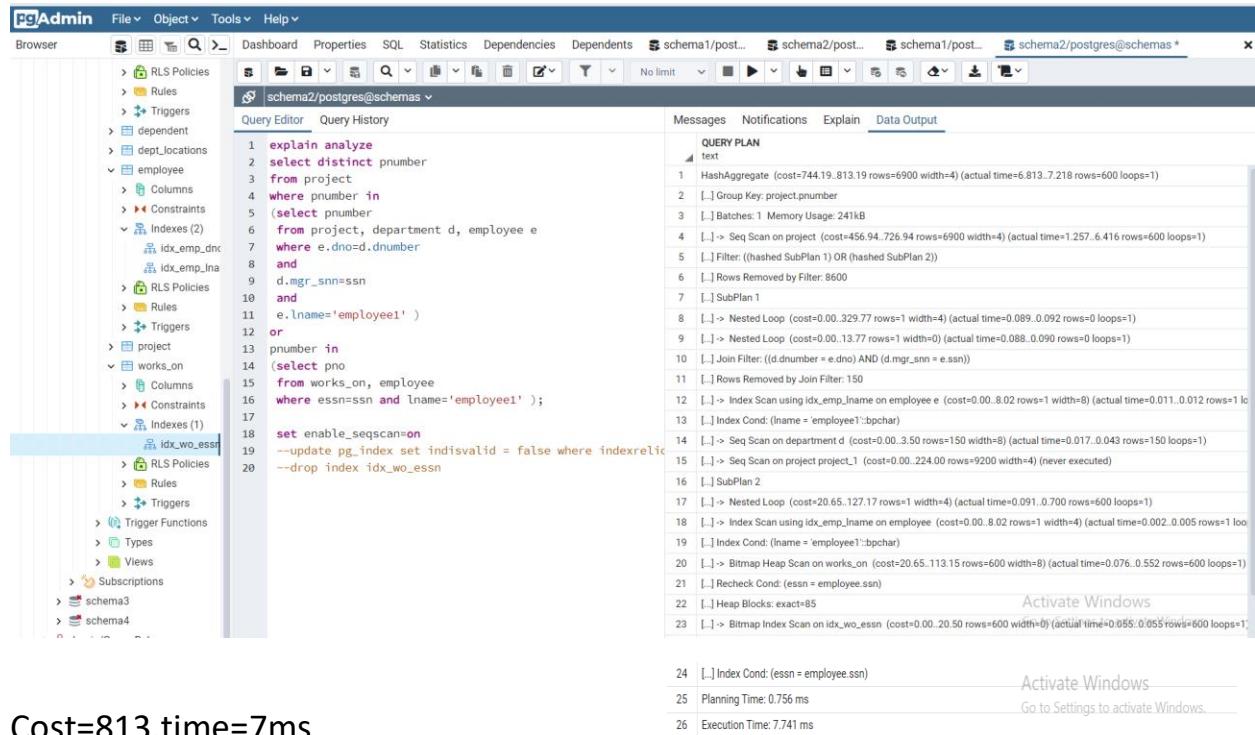
Using Btree indices decreased the cost to 39.3% of initial cost (W/O indices),

Time to 40% of initial time(W/O indices)

Difference in cost is referred to the difference in performance between seqscan to search for lname='employee1', search for matching ssn from works_on, and the index scan over those relations

=>Using Hash indeces:

-Hash indices are used on employee(lname), works_on(ssn)



The screenshot shows the pgAdmin interface with the following details:

- Browser:** schema2/postgres@schemas
- Query Editor:** Query History
- Text:** The query being analyzed is:

```
explain analyze
select distinct pnumber
from project
where pnumber in
(select pnumber
from project, department d, employee e
where e.dno=d.dnumber
and
d.mgr_ssn=ssn
and
e.lname='employee1' )
or
pnumber in
(select pno
from works_on, employee
where ssn=ssn and lname='employee1' );
set enable_seqscan=on
--update pg_index set indisvalid = false where indexrelid=drop index idx_wo_essn
```
- Messages:** No messages.
- Notifications:** No notifications.
- Explain:** The query plan is displayed in the Explain tab, showing the following steps:
 - HashAggregate (cost=744.19..813.19 rows=6900 width=4) (actual time=6.813..7.218 rows=600 loops=1)
 - [...] Group Key: project.pnumber
 - [...] Batches: 1 Memory Usage: 241kB
 - [...] > Seq Scan on project (cost=456.94..726.94 rows=6900 width=4) (actual time=1.257..6.416 rows=600 loops=1)
 - [...] Filter: (hashed SubPlan 1) OR (hashed SubPlan 2)
 - [...] Rows Removed by Filter: 8600
 - [...] SubPlan 1
 - [...] Nested Loop (cost=0.0..329.77 rows=1 width=4) (actual time=0.089..0.092 rows=0 loops=1)
 - [...] > Nested Loop (cost=0.0..13.77 rows=1 width=0) (actual time=0.088..0.090 rows=0 loops=1)
 - [...] Join Filter: ((d.dnumber = e.dno) AND (d.mgr_ssn = e.ssn))
 - [...] Rows Removed by Join Filter: 150
 - [...] Index Scan using idx_emp_lname on employee e (cost=0.0..8.02 rows=1 width=8) (actual time=0.011..0.012 rows=1 loops=1)
 - [...] Index Cond: (lname = 'employee1')::bpchar
 - [...] > Seq Scan on department d (cost=0.0..3.50 rows=150 width=8) (actual time=0.017..0.043 rows=150 loops=1)
 - [...] > Seq Scan on project project_1 (cost=0.0..224.00 rows=9200 width=4) (never executed)
 - [...] SubPlan 2
 - [...] Nested Loop (cost=20.65..127.17 rows=1 width=4) (actual time=0.091..0.700 rows=600 loops=1)
 - [...] > Index Scan using idx_emp_lname on employee e (cost=0.0..8.02 rows=1 width=4) (actual time=0.002..0.005 rows=1 loops=1)
 - [...] Index Cond: (lname = 'employee1')::bpchar
 - [...] > Bitmap Heap Scan on works_on (cost=20.65..113.15 rows=600 width=8) (actual time=0.076..0.552 rows=600 loops=1)
 - [...] Recheck Cond: (ssn = employee.ssn)
 - [...] > Bitmap Index Scan on idx_wo_essn (cost=0.0..20.50 rows=600 width=8) (actual time=0.055..0.055 rows=600 loops=1)
 - [...] Index Cond: (essn = employee.ssn)
 - [...] Planning Time: 0.756 ms
 - [...] Execution Time: 7.741 ms

- Data Output:** The Data Output tab is active, showing the results of the query execution.

Cost=813,time=7ms

Notes:

Using Hash indices decreased the cost to 39.8% of initial cost (W/O indices),

Time to 50% of initial time(W/O indices)

Difference in cost is referred to the difference in performance between seqscan to search for lname='employee1', search for matching ssn from works_on, and the index scan over those relations

=>Using BRIN indeces:

-BRIN indices are used on works_on(essn)

The screenshot shows the PgAdmin interface with a query editor containing the following SQL code:

```
1 explain analyze
2 select distinct pnumber
3 from project
4 where pnumber in
5 (select pnumber
6 from project, department d, employee e
7 where e.dno=d.dnumber
8 and
9 d.mgr_ssn=ssn
10 and
11 e.lname='employee1'
12 or
13 pnumber in
14 (select pno
15 from works_on, employee
16 where essn=ssn and lname='employee1');
17
18 set enable_nestloop=on
19 --update pg_index set indisvalid = false where indexrelid=idx_emp_dno
20 --drop index idx_emp_dno
```

The "Data Output" tab shows the query plan (QUERYP PLAN) with the following steps:

- HashAggregate (cost=1833.22..1902.22 rows=6900 width=4) (actual time=30.766..31.123 rows=600 loops=1)
 - [...] Group Key: project.pnumber
- [...] Batches: 1 Memory Usage: 241kB
- [...] Seq Scan on project (cost=1545.97..1815.97 rows=6900 width=4) (actual time=24.117..30.294 rows=600 loops=1)
 - [...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
- [...] Rows Removed by Filter: 8600
- [...] SubPlan 1
 - [...] Nested Loop (cost=0.00..784.75 rows=1 width=4) (actual time=8.099..8.103 rows=0 loops=1)
 - [...] Nested Loop (cost=0.00..468.75 rows=1 width=0) (actual time=8.098..8.100 rows=0 loops=1)
 - [...] Join Filter: ((d.dnumber = e.dno) AND (d.mgr_ssn = e.ssn))
 - [...] Rows Removed by Join Filter: 150
 - [...] Seq Scan on employee e (cost=0.00..463.00 rows=1 width=8) (actual time=0.030..7.955 rows=1 loops=1)
 - [...] Filter: (lname = 'employee1')::bpchar
 - [...] Rows Removed by Filter: 15999
 - [...] Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.049..0.083 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=12.18..761.22 rows=1 width=4) (actual time=0.226..15.093 rows=600 loops=1)
 - [...] Seq Scan on employee e (cost=0.00..463.00 rows=1 width=4) (actual time=0.061..7.009 rows=1 loops=1)
 - [...] Filter: (lname = 'employee1')::bpchar
 - [...] Rows Removed by Filter: 15999
 - [...] Bitmap Heap Scan on works_on (cost=12.18..292.22 rows=600 width=8) (actual time=0.138..2.837 rows=600 loops=1)
 - [...] Index Cond: (essn = employee.ssn)
 - [...] Recheck Cond: (essn = employee.ssn)

At the bottom right, there are two messages: "Activate Windows" and "Go to Settings to activate Windows".

Cost=1902,time=31ms

Notes:

Using BRIN indices decreased the cost to 93% of initial cost (W/O indices),

Time to 200% of initial time(W/O indices)

The reason of this bad cost and time reduction is that brin uses several sparse levels to reach data, but still better than seqscans only specially when working on works_on(essn)as it helps the join to find the match to each tuple of employee

=>Using Mixed indices: hash on employee(lname), btree on works_on(ssn)

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```

1 --drop index idx_emp_lname
2 explain analyze
3 select distinct pnumber
4 from project
5 where pnumber in
6 (select pnumber
7 from project, department d, employee e
8 where e.dno=d.dnumber
9 and
10 d.mgr_snn=ssn
11 and
12 e.lname='employee1' )
13 or
14 pnumber in
15 (select pno
16 from works_on, employee
17 where ssn=ssn and lname='employee1' );
18
19 --set enable_seqscan=on
20 set enable_nestloop:on
21 create index idx_dpt_mgr_dnum on department USING btree(dnumber,mgr_snn)
22 CREATE INDEX idx_wo_ssn ON works_on USING btree(ssn);
23 CREATE INDEX idx_emp_lname ON employee USING hash(lname);
24 select department.mgr_snn from department where dnumber=5
25

```

The query plan (shown in the bottom right) details the execution steps:

- HashAggregate (cost=732.48..801.48 rows=6900 width=4) (actual time=4.398..4.575 rows=600 loops=1)
- [...] Group Key: project.pnumber
- [...] Batches: 1 Memory Usage: 241kB
- [...] > Seq Scan on project (cost=445.23..715.23 rows=6900 width=4) (actual time=0.736..4.148 rows=600 loops=1)
- [...] Filter: (hashed SubPlan 1) OR (hashed SubPlan 2)
- [...] Rows Removed by Filter: 8600
- [...] SubPlan 1
- [...] > Nested Loop (cost=0.00..329.77 rows=1 width=4) (actual time=0.054..0.057 rows=0 loops=1)
- [...] > Nested Loop (cost=0.00..13.77 rows=1 width=0) (actual time=0.054..0.055 rows=0 loops=1)
- [...] Join Filter: (d.dnumber = e.dno) AND (d.mgr_snn = e.ssn)
- [...] Rows Removed by Join Filter: 150
- [...] > Index Scan using idx_emp_lname on employee e (cost=0.00..8.02 rows=1 width=8) (actual time=0.000..0.000 rows=1)
- [...] Index Cond: (lname = 'employee1':bpchar)
- [...] > Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.013..0.026 rows=1)
- [...] > Seq Scan on project project_1 (cost=0.00..224.00 rows=9200 width=4) (never executed)
- [...] SubPlan 2
- [...] > Nested Loop (cost=8.94..115.45 rows=1 width=4) (actual time=0.063..0.458 rows=600 loops=1)
- [...] > Index Scan on idx_emp_lname on employee (cost=0.00..8.02 rows=1 width=4) (actual time=0.000..0.000 rows=1)
- [...] Index Cond: (lname = 'employee1':bpchar)
- [...] > Bitmap Heap Scan on works_on (cost=8.94..101.44 rows=600 width=8) (actual time=0.044..0.34C rows=600)
- [...] Recheck Cond: (ssn = employee.ssn)
- [...] Heap Blocks: exact=85
- [...] > Bitmap Index Scan on idx_wo_ssn (cost=0.00..8.79 rows=600 width=0) (actual time=0.029..0.029 rows=600)
- [...] Index Cond: (ssn = employee.ssn)
- Planning Time: 0.523 ms
- Execution Time: 3.321 ms

cost = 801, time=3.3ms

Notes:

The mixed indices is the optimal solution as it gives the lowest cost among all previous, it's not far away from btree indices, but it gives less cost anyway.

Mixed indices reduces the cost to be 39.2% of the one w/o indices and time to 20.88% of initial time.

Both plans of mixed and btree indices runs the same, as we get the filtered version of employee relation where lname='employee1' and try to match it with the relation department. The other way around is not accepted as it takes the whole relation of department and search for each tuple in employee then check that the lname ='employee1', so the first does much fewer searches, the difference between the mixed indices version and the btree version is that the hash

Q2: Optimized

=>Without indices (seqscans only):

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
12 from works_on, employee
13 where essn=ssn and lname='employee1' ;
14
15
16
17
18 explain analyze
19 (select pnumber
20 from project, department d, employee e
21 where e.dno=d.dnumber
22 and
23 d.mgr_snn:ssn
24 and
25 e.lname='employee1' )
26 union
27 (select pno
28 from works_on, employee
29 where essn=ssn and lname='employee1' );
30
31
32
33
34
35 --set enable_seqscan=on
36 set enable_nestloop=on
37 create index idx_dpt_mgr_dnum on department USING btree(dnumber,mgr_snn)
38 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
39 CREATE INDEX idx_emp_lname ON employee USING hash(lname);
40 select department.mgr_snn from department where dnumber=5
```

The "Data Output" tab shows the query plan:

```
1 Unique (cost=1547.35..1547.36 rows=2 width=4) (actual time=11.908..12.085 rows=600 loops=1)
  2 [...] > Sort (cost=1547.35..1547.36 rows=2 width=4) (actual time=11.906..11.944 rows=600 loops=1)
  3 [...] Sort Key: project.pnumber
  4 [...] Sort Method: quicksort Memory: 53kB
  5 [...] > Append (cost=0.00..1547.34 rows=2 width=4) (actual time=6.258..11.631 rows=600 loops=1)
  6 [...] > Nested Loop (cost=0.00..784.75 rows=1 width=4) (actual time=3.593..3.595 rows=0 loops=1)
  7 [...] > Nested Loop (cost=0.00..468.75 rows=1 width=4) (actual time=3.592..3.593 rows=0 loops=1)
  8 [...] Join Filter: ((d.dnumber = e.dno) AND (d.mgr_snn = e.ssn))
  9 [...] Rows Removed by Join Filter: 150
 10 [...] > Seq Scan on employee e (cost=0.00..463.00 rows=1 width=8) (actual time=0.014..3.558 rows=1 loops=1)
 11 [...] Filter: (lname = 'employee1':bpchar)
 12 [...] Rows Removed by Filter: 15999
 13 [...] > Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.009..0.020 rows=150)
 14 [...] > Seq Scan on project (cost=0.00..224.00 rows=9200 width=4) (never executed)
 15 [...] > Hash Join (cost=463.01..762.56 rows=1 width=4) (actual time=2.663..7.968 rows=600 loops=1)
 16 [...] Hash Cond: (works_on.essn = employee.ssn)
 17 [...] > Seq Scan on works_on (cost=0.00..241.03 rows=15603 width=8) (actual time=0.029..2.167 rows=15603)
 18 [...] > Hash (cost=463.00..463.00 rows=1 width=4) (actual time=2.603..2.603 rows=1 loops=1)
 19 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
 20 [...] > Seq Scan on employee (cost=0.00..463.00 rows=1 width=4) (actual time=0.014..2.598 rows=1 loops=1)
 21 [...] Filter: (lname = 'employee1':bpchar)
 22 [...] Rows Removed by Filter: 15999
 23 Planning Time: 0.455 ms
```

Execution Time: 12.170 ms

Cost=1547,time=12ms

Notes:

Optimization reduced cost 75.8% of original, the reason is the removal of “or” operator that requires “distinct” operator to ensure no-duplicates and used union that remove duplicates in more efficient way

Time hasn’t been affected that much (became 75% of original time)

=>Using B+Tree: on employee(lname), works_on(essn)

The screenshot shows the PgAdmin interface with a query editor containing the following SQL code:

```

SELECT dname, dnumber, mgr_ssn, mgr_start_dt
FROM department
WHERE e.dno=d.number
AND d.mgr_ssn=ssn
AND e.lname='employee1'
UNION
(SELECT pno
FROM works_on, employee
WHERE essn=ssn AND lname='employee1');

```

The right side of the interface displays the "QUERY PLAN" for this query. The plan details the execution steps, including index scans, nested loops, and hash joins, along with their respective costs and execution times.

Cost=444.74, time=1.2ms

24	Execution Time: 1.201 ms	Go to Settings to activate Windows.
----	--------------------------	-------------------------------------

Notes:

Using Btree indices decreased the cost to 28.75% of initial cost (W/O indices),

Time to 10% of initial time (W/O indices)

The reason behind this is instead of seqscan on relation to do nested loop join(to find matching tuples) the indices made it much faster

Performance improvement here is good as btree is good regardless there is duplicates (like works_on(essn)) or being with not much duplicates (like employee(lname))

=>Using Hash: on employee(lname), works_on(essn)

```

13   from works_on, employee
14   where essn=ssn and lname='employee1' );
15
16
17
18
19 explain analyze
20 (select pnumbr
21   from project, department d, employee e
22   where e.dno=d.dnumber
23   and
24   d.mgr_snn=ssn
25   and
26   e.lname='employee1' )
27   union
28 (select pno
29   from works_on, employee
30   where essn=ssn and lname='employee1' );
31
32
33
34
35
36 --set enable_seqscan:on
37 set enable_nestloop:on
38 create index idx_dpt_mgr_dnum on department USING btree(dnumber,mgr_s
39 CREATE INDEX idx_wo_essn ON works_on USING hash(essn);
40 CREATE INDEX idx_emp_lname ON employee USING hash(lname);
41 select department.mgr_snn from department where dnumber=5

```

Execution Time: 1.074 ms

Cost=455, time=1ms

[Go to Settings to activate Windows.](#)

Notes:

Using hash indices decreased the cost to 29.4% of initial cost (W/O indices),

Time to 8.33% of initial time (W/O indices)

Performance improvement here is not good nor bad as hash is bad if there is duplicates (like works_on(essn)) but works well if column has not much duplicates (like employee(lname))

⇒ Using BRIN : on employee(lname), works_on(essn)

```

14   from works_on, employee
15   where essn=ssn and lname='employee1' );
16
17
18
19
20 explain analyze
21 (select pnumber
22   from project, department d, employee e
23   where e.dno=d.dnumber
24   and
25   d.mgr_ssn=ssn
26   and
27   e.lname='employee1' )
28 union
29 (select pno
30   from works_on, employee
31   where essn=ssn and lname='employee1' );
32
33
34
35
36
37 --set enable_seqscan=on
38 set enable_nestloop=on
39 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_ssn)
40 CREATE INDEX idx_wo_essn ON works_on USING brin(essn);
41 CREATE INDEX idx_emp_lname ON employee USING brin(lname);
42 select department.mgr_ssn from department where dnumber=5

```

QUERY PLAN

```

text
1 Unique (cost=1526.38..1526.39 rows=2 width=4) (actual time=7.740..7.875 rows=600 loops=1)
2 [...] > Sort (cost=1526.38..1526.39 rows=2 width=4) (actual time=7.737..7.772 rows=600 loops=1)
3 [...] Sort Key: project.pnumber
4 [...] Sort Method: quicksort Memory: 53kB
5 [...] > Append (cost=12.09..1526.37 rows=2 width=4) (actual time=2.262..7.534 rows=600 loops=1)
6 [...] > Nested Loop (cost=12.09..774.94 rows=1 width=4) (actual time=2.159..2.162 rows=0 loops=1)
7 [...] > Nested Loop (cost=12.09..458.94 rows=1 width=0) (actual time=2.158..2.160 rows=0 loops=1)
8 [...] Join Filter: ((d.dnumber = e.dno) AND (d.mgr_ssn = e.ssn))
9 [...] Rows Removed by Join Filter: 150
10 [...] > Bitmap Heap Scan on employee e (cost=12.09..453.19 rows=1 width=8) (actual time=0.076..2.10)
11 [...] Recheck Cond: (lname = 'employee1':bpchar)
12 [...] Rows Removed by Index Recheck: 7807
13 [...] Heap Blocks: lossy=128
14 [...] > Bitmap Index Scan on idx_emp_lname (cost=0.00..12.09 rows=14248 width=0) (actual time=0.06)
15 [...] Index Cond: (lname = 'employee1':bpchar)
16 [...] > Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.017..0.033 rows=1)
17 [...] > Seq Scan on project (cost=0.00..224.00 rows=9200 width=4) (never executed)
18 [...] > Nested Loop (cost=24.27..751.41 rows=1 width=4) (actual time=0.102..5.301 rows=600 loops=1)
19 [...] > Bitmap Heap Scan on employee (cost=12.09..453.19 rows=1 width=4) (actual time=0.042..1.718)
20 [...] Recheck Cond: (lname = 'employee1':bpchar)
21 [...] Rows Removed by Index Recheck: 7807
22 [...] Heap Blocks: lossy=128
23 [...] > Bitmap Index Scan on idx_emp_lname (cost=0.00..12.09 rows=14248 width=0) (actual time=0.03)

```

Activate Windows

cost =1526, time=8ms

Notes:

Using BRIN indices decreased the cost to 98.6% of initial cost (W/O indices),

Time to 66.6% of initial time (W/O indices)

BRIN as other indeces in this query are aiming to serve the joins, because the brin are not the best to get exact values, it's not the best index to use here, however it does better than nothing.

⇒ Mixed indices: btree on works_on(essn), hash on employee(lname)

The screenshot shows the pgAdmin 4 interface with a query editor containing the following SQL code:

```

10 o.mgr_snn=ssn
11 ),project
12 union
13 (select pno
14   from works_on, employee
15   where essn=ssn and lname='employee1' );
16
17
18
19
20 explain analyze
21 select pnumber
22   from project, department d, employee e
23   where e.dno=d.dnumber
24   and
25   d.mgr_snn=ssn
26   and
27   e.lname='employee1' )
28 union
29 (select pno
30   from works_on, employee
31   where essn=ssn and lname='employee1' );
32
33
34
35
36
37 --set enable_seqscan=on
38 set enable_nestloop=on
39 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_si

```

The right pane displays the "QUERY PLAN" for this query, showing a detailed execution plan with various stages like Unique, Sort, Hash Join, and Bitmap Scan.

Cost=444.17, time=3.2ms

24 Execution Time: 3.204 ms

Go to Settings to activate Windows.

Activate Windows
Go to Settings to activate Windows.

Notes:

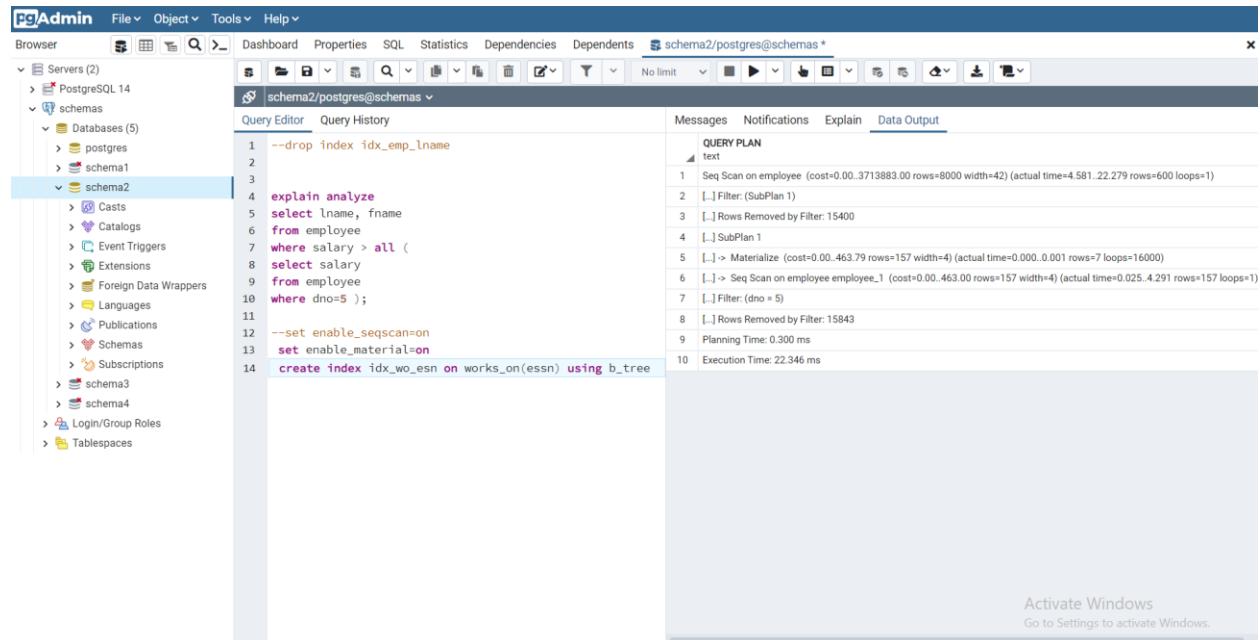
The mixed indices is the optimal solution as it gives the lowest cost among all previous, it's not far away from btree indices, but it gives less cost anyway.

Mixed indices reduces the cost to be 28.71% of the one w/o indices and time to 26.6% of initial time.

Both plans of mixed and btree indices runs the same, but when used btree in the one that has many duplicates(works_on(essn)) that hash is bad at, and used hash in employee(lname)that doesn't have many duplicates, the performance improved alittle (according to the little difference in access time between btree and hash indices)

Q3:

=>Without indices (seqscans only)(with enable_materialize=on):



The screenshot shows the PGAdmin interface. The left sidebar displays the database structure with 'Servers (2)' and 'Databases (5)'. The 'schema2' database is selected. The main area shows a query editor with the following SQL code:

```
--drop index idx_emp_lname
select lname, fname
from employee
where salary > all (
    select salary
    from employee
    where dno=5 );
--set enable_seqscan=on
set enable_materialize=on
create index idx_wo_esn on works_on(esnn) using b_tree
```

The 'Explain' tab is selected, showing the query plan:

Step	Operation	Cost	Rows	Width	Time
1	Seq Scan on employee	0.00	3713883.00	42	actual time=4.581..22.279 rows=600 loops=1
2	[...] Filter: (SubPlan 1)				
3	[...] Rows Removed by Filter: 15400				
4	[...] SubPlan 1				
5	[...] > Materialize	0.00	463.79	4	(actual time=0.000..0.001 rows=7 loops=16000)
6	[...] > Seq Scan on employee employee_1	0.00	463.00	4	(actual time=0.025..4.291 rows=157 loops=1)
7	[...] Filter: (dno = 5)				
8	[...] Rows Removed by Filter: 15843				
9	Planning Time: 0.300 ms				
10	Execution Time: 22.346 ms				

Activation Windows message: Go to Settings to activate Windows.

Cost=3713883, time=22ms

Without materialize: cost=3707603, time=4372ms

Notes:

One relation only used which is employee, materialization takes alittle more cost but reduces the time so much

=>Using BTree: on employee(dno)

The screenshot shows the pgAdmin interface. On the left, the Browser pane displays a tree view of database objects, including servers, databases, schemas, and tables. The 'schema2' schema is selected. In the center, the Query Editor contains the following SQL code:

```
1 --drop index idx_emp_lname
2
3
4 explain analyze
5 select lname, fname
6 from employee
7 where salary > all (
8 select salary
9 from employee
10 where dno=5 );
11
12 --set enable_seqscan=on
13 set enable_material=on
14 create index idx_emp_dno on employee(dno)
```

Below the Query Editor is the Explain tab, which shows the execution plan for the query:

Step	Operation	Cost	Rows	Width	Time
1	Seq Scan on employee	5.50	1927837	8000	0.378..21.351
2	[...] Filter: (SubPlan 1)				15400
3	[...] Rows Removed by Filter: 15400				
4	[...] SubPlan 1				
5	[...] > Materialize	5.50	246.03	157	0.000..0.000
6	[...] > Bitmap Heap Scan on employee employee_1	5.50	245.25	157	0.060..0.252
7	[...] Recheck Cond: (dno = 5)				
8	[...] Heap Blocks: exact=157				
9	[...] > Bitmap Index Scan on idx_emp_dno	0.00	5.46	0	0.034..0.034
10	[...] Index Cond: (dno = 5)				
11	Planning Time: 0.248 ms				
12	Execution Time: 21.446 ms				

On the right side of the pgAdmin window, there is a message: "Activate Windows Go to Settings to activate Windows."

Cost=1927837, time=21ms

Notes:

Index is used to select the exact value(dno=5) and reduced the cost to 51% but time hasn't been affected that much

=>Using Hash: on employee(dno)

The screenshot shows the PgAdmin 4 interface. The left sidebar displays the database schema, including tables like 'employee' and 'department'. The main area shows a query editor with the following SQL code:

```
1 --drop index idx_emp_lname
2 --drop index idx_wo_essn
3 --drop index idx_dpt_mgr_dnum
4
5
6
7
8 explain analyze
9 select lname, fname
10 from employee
11 where salary > all (
12     select salary
13     from employee
14     where dno=5 );
15
16
17
18
19
20 --set enable_seqscan=on
21 set enable_nestloop=on
22 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn)
23 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
24 CREATE INDEX idx_emp_dno ON employee USING hash(dno);
25 select department.mgr_snn from department where dnumber=5
```

The right side of the interface shows the 'Data Output' tab, which displays the query plan. The plan indicates a 'Seq Scan on employee' with a cost of 5.22 and 1927836 rows. It also shows the creation of indexes and the execution time of 19.45 ms.

Cost=1927836, time =19.45ms

Notes:

Index is used to select the exact value(dno=5) and reduced the cost to 51% but time hasn't been affected that much

=>Using BRIN: on employee(dno) enable_seqscan=off

The screenshot shows the pgAdmin interface. In the left sidebar, under 'Tables (6)', the 'department' table is selected. Under 'Columns (4)', the columns dname, dnumber, mgr_snn, and mgr_start_dn are listed. The main area contains a query in the 'Query Editor' tab:

```
1 --drop index idx_emp_dno
2 --drop index idx_wo_essn
3 --drop index idx_dpt_mgr_dnum
4
5
6
7
8
9 explain analyze
10 select lname, fname
11 from employee
12 where salary > all (
13     select salary
14     from employee
15     where dno=5 );
16
17
18
19 set enable_bitmapscan=on
20 --set enable_seqscan=off
21 set enable_nestloop=on
22 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn)
23 CREATE INDEX idx_wo_essn ON works_on USING btree(ssn);
24 CREATE INDEX idx_emp_dno ON employee USING brin(dno);
25 select department.mgr_snn from department where dnumber=5
```

The results pane shows the EXPLAIN ANALYZE output:

Step	Operation	Cost	Time	Details
1	Seq Scan on employee	10003713895.14	actual time=4.43s	(cost=10000000012.14..10003713895.14 rows=8000 width=42)
2	[...] Filter: (SubPlan 1)			
3	[...] Rows Removed by Filter: 15400			
4	[...] SubPlan 1			
5	[...] > Materialize	12.14..475.92	actual time=0.000..0.001	rows=7 loops=16000
6	[...] > Bitmap Heap Scan on employee employee_1	12.14..475.14	actual time=0.025..0.026	rows=157 width=4
7	[...] Recheck Cond: (dno = 5)			
8	[...] Rows Removed by Index Recheck: 15843			
9	[...] Heap Blocks: lossy=263			
10	[...] > Bitmap Index Scan on idx_emp_dno	0.00..12.10	actual time=0.025..0.026	rows=16000 width=0
11	[...] Index Cond: (dno = 5)			
12	Planning Time: 0.266 ms			
13	Execution Time: 24.032 ms			

Activation Windows message: Activate Windows
Go to Settings to activate Windows.

Cost = 10003713895, time=24ms

Notes:

Index is used to select the exact value(dno=5) but the performance of brin is not good because of it passes by several layers of sparse and it requires a seqscan inside the target page

=>Using mixed:hash on employee(dno),btree on employee(salary)

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
1 explain analyze
2 select lname, fname
3 from employee
4 where salary > all (
5 select salary
6 from employee
7 where dno=5 );
8
9 create index idx1 on employee using btree(lname);
10 create index idx2 on works_on using btree(essn);
11 create index dn on employee using hash(dno);
12 create index sal on employee using btree(salary);
```

The "Explain" tab of the results pane displays the execution plan:

Step	Operation	Cost	Time
1	Seq Scan on employee	5.22	1927836.79 ms
2	[...] Filter: (SubPlan 1)		
3	[...] Rows Removed by Filter: 15400		
4	[...] SubPlan 1		
5	[...] -> Materialize	5.22	245.75 ms
6	[...] -> Bitmap Heap Scan on employee employee_1	5.22	244.96 ms
7	[...] Recheck Cond: (dno = 5)		
8	[...] Heap Blocks: exact=157		
9	[...] -> Bitmap Index Scan on dn	0.00	5.18 ms
10	[...] Index Cond: (dn = 5)		
11	Planning Time: 0.197 ms		
12	Execution Time: 21.786 ms		

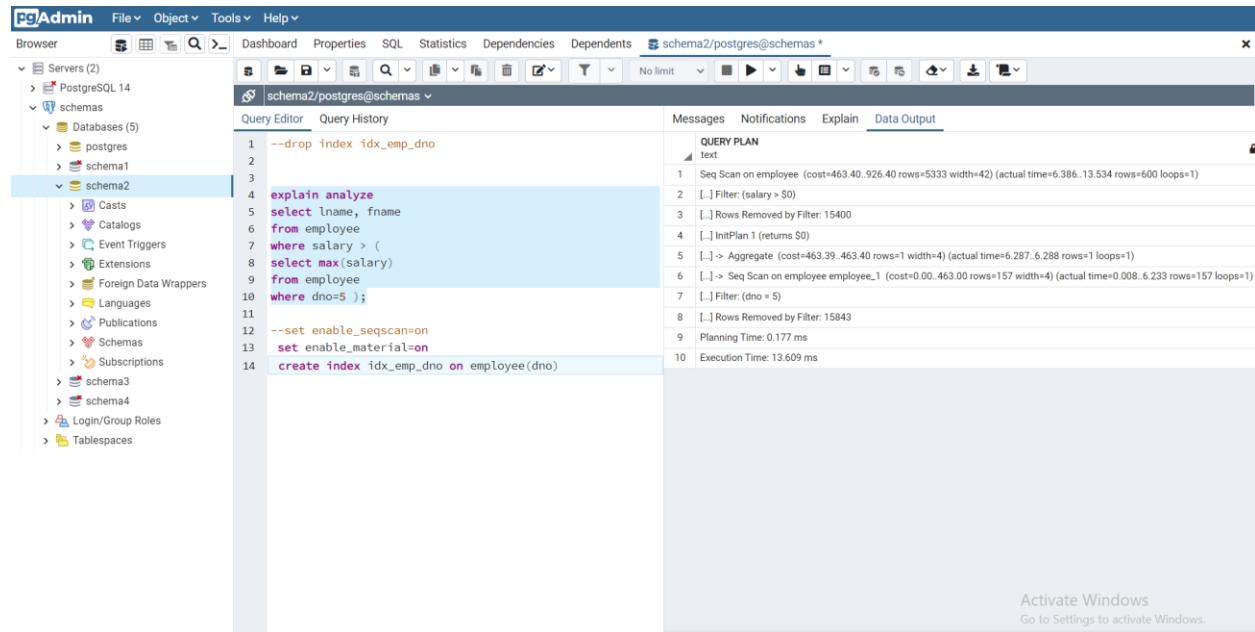
Cost=1927836, time=21ms

Notes:

Same as hash section since the engine could use one index only which is the hash index.

Q3: optimized

=>Without indices (seqscans only)(with enable_materialize=on):



The screenshot shows the PGAdmin interface with a query editor window. The left sidebar lists servers and databases, with 'schema2' selected. The query editor contains the following SQL code:

```
--drop index idx_emp_dno
explain analyze
select lname, fname
from employee
where salary > (
select max(salary)
from employee
where dno=5 );
--set enable_seqscan=on
set enable_materialize=on
create index idx_emp_dno on employee(dno)
```

The 'Explain' tab is active, showing the query plan:

Step	Operation	Cost	Rows	Width	Time
1	Seq Scan on employee	463.40	5333	42	actual time=6.386..13.534
2	[...] Filter: (salary > \$0)				
3	[...] Rows Removed by Filter: 15400				
4	[...] InitPlan 1 (returns \$0)				
5	[...] > Aggregate	463.39..463.40	1	4	actual time=6.287..6.288
6	[...] > Seq Scan on employee employee_1	0.00..463.00	157	4	actual time=0.008..6.233
7	[...] Filter: (dno = 5)				
8	[...] Rows Removed by Filter: 15843				
9	Planning Time: 0.177 ms				
10	Execution Time: 13.609 ms				

A message at the bottom right says "Activate Windows Go to Settings to activate Windows."

Cost=926, time=13ms

Notes:

Optimization is based on instead of going checking “all” salaries of employees of dno=5 to check if the current employee is greater than all of their salaries, it’s enough to check that the current employee gets more salary than the max of dno=5 employee salary

Optimization reduced cost to 0.025% of the original cost, time to 61.9% of original time

=>Using B+Tree: on employee(salary)

The screenshot shows the PGAdmin interface with a query editor window. The query being run is:

```
--drop index idx_emp_dno
--drop index idx_emp_sal
--drop index idx_wo_essn
--drop index idx_dpt_mgr_dnum
select lname, fname
from employee
where salary > (
    select max(salary)
    from employee
    where dno=5 );
set enable_bitmapscan=on
--set enable_seqscan=on
set enable_nestloop=on
CREATE INDEX idx_dpt_mgr_dnum ON department USING brin(dnumber,mgr_snn)
CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
CREATE INDEX idx_emp_dno ON employee USING btree(dno);
CREATE INDEX idx_emp_sal ON employee USING btree(salary);
select department.mgr_snn from department where dnumber=5
```

The right side of the window displays the "EXPLAIN PLAN" output, which includes the following details:

- Bitmap Heap Scan on employee (cost=74.87..404.53 rows=5333 width=42) (actual time=0.483..0.627 row=1)
- [...] Recheck Cond: (salary > \$1)
- [...] Heap Blocks: exact=164
- [...] InitPlan 2 (returns \$1)
- [...] > Result (cost=9.24..9.25 rows=1 width=4) (actual time=0.428..0.428 rows=1 loops=1)
- [...] InitPlan 1 (returns \$0)
- [...] > Limit (cost=0.29..9.24 rows=1 width=4) (actual time=0.426..0.426 rows=1 loops=1)
- [...] > Index Scan Backward using idx_emp_sal on employee employee_1 (cost=0.29..1406.88 rows=157 v)
- [...] Index Cond: (salary IS NOT NULL)
- [...] Filter: (dno = 5)
- [...] Rows Removed by Filter: 616
- [...] > Bitmap Index Scan on idx_emp_sal (cost=0.00..64.28 rows=5333 width=0) (actual time=0.462..0.462)
- [...] Index Cond: (salary > \$1)
- Planning Time: 0.370 ms
- Execution Time: 0.699 ms

Cost=404.5, time=0.7ms

Notes:

Using Btree indices decreased the cost to 43.68% of initial cost (W/O indices),

Time to 5.38% of initial time(W/O indices)

⇒ Using Hash: on employee(dno)

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects under 'schema2/postgres@schemas'. The main area has tabs for 'Query Editor' and 'Query History', with 'Query Editor' selected. A code editor window contains the following SQL code:

```
--drop index idx_emp_dno
--drop index idx_emp_sal
--drop index idx_wo_essn
--drop index idx_dpt_mgr_dnum
explain analyze
select lname, fname
from employee
where salary > (
    select max(salary)
    from employee
    where dno=5 );
set enable_bitmapscan=on
--set enable_seqscan=on
set enable_nestloop=on
create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn)
CREATE INDEX idx_wo_essn ON works USING btree(essn);
CREATE INDEX idx_emp_dno ON employee USING hash(dno);
CREATE INDEX idx_emp_sal ON employee USING btree(salary);
select department.mgr_snn from department where dnumber=5
```

To the right of the code editor is a 'Messages' panel showing the execution of the code. Below the code editor are tabs for 'Messages', 'Notifications', 'Explain', and 'Data Output', with 'Data Output' selected. The 'Data Output' tab displays the 'QUERY PLAN' for the query, listing the following steps:

- Seq Scan on employee (cost=245.36..708.36 rows=5333 width=42) (actual time=0.306..2.972 rows=60)
- [...] Filter: (salary > \$0)
- [...] Rows Removed by Filter: 15400
- [...] InitPlan 1 returns \$0
- [...] Aggregate (cost=245.35..245.36 rows=1 width=4) (actual time=0.257..0.259 rows=1 loops=1)
- [...] Bitmap Heap Scan on employee employee_1 (cost=5.22..244.96 rows=157 width=4) (actual time=
- [...] Recheck Cond: (dno = 5)
- [...] Heap Blocks: exact=157
- [...] Bitmap Index Scan on idx_emp_dno (cost=0.00..0.18 rows=157 width=0) (actual time=0.018..0.019 ms)
- [...] Index Cond: (dno = 5)
- Planning Time: 0.327 ms
- Execution Time: 3.039 ms

Cost=708, time=3ms

Notes:

Using Hash indices decreased the cost to 76.5% of initial cost (W/O indices),

Time to 23% of initial time (W/O indices)

The hash is not that efficient as btree in this query because btree support ranges so it can scan the part of employee table with salaries > max of dno=5 salaries, unlike the hash that is going to check every single tuple

⇒ Using BRIN: on employee(dno),employee(salary)

The screenshot shows the PgAdmin 4 interface. The left sidebar is the 'Browser' pane, listing various database objects like Schemas, Tables, and Columns. The main area is the 'Query Editor' containing the following SQL code:

```

1 --drop index idx_emp_dno
2 --drop index idx_emp_sal
3 --drop index idx_wo_essn
4 --drop index idx_dpt_mgr_dnum
5
6
7
8
9
10 explain analyze
11 select lname, fname
12 from employee
13 where salary > (
14     select max(salary)
15     from employee
16     where dno=5 );
17
18
19
20 set enable_bitmapscan=on
21 --set enable_seqscan=off
22 set enable_nestloop=on
23 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn)
24 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
25 CREATE INDEX idx_emp_dno ON employee USING brin(dno);
26 CREATE INDEX idx_emp_sal ON employee USING brin(salary);
27 select department.mgr_snn from department where dnumber=5

```

The 'Messages' tab shows the query plan and execution details:

- QUERY PLAN
- text
- 1 Bitmap Heap Scan on employee (cost=488.97..951.97 rows=5333 width=42) (actual time=4,418..9,508 ms)
 - [...] Rows Removed by Index Recheck: Cond: (salary > \$0)
 - [...] Rows Removed by Index Recheck: 15016
 - [...] Heap Blocks: lossy=256
 - [...] InitPlan 1 (returns \$0)
 - [...] > Aggregate (cost=475.53..475.54 rows=1 width=4) (actual time=4,299..4,301 rows=1 loops=1)
 - [...] > Bitmap Heap Scan on employee employee_1 (cost=12.14..475.14 rows=157 width=4) (actual time=12.14..475.14 rows=157 loops=1)
 - [...] Index Cond: (dno = 5)
 - [...] Rows Removed by Index Recheck: 15843
 - [...] Heap Blocks: lossy=263
 - [...] > Bitmap Index Scan on idx_emp_dno (cost=0.00..12.10 rows=16000 width=0) (actual time=0.053..0.053 rows=16000 loops=1)
 - [...] Index Cond: (dno = 5)
 - [...] > Bitmap Index Scan on idx_emp_sal (cost=0.00..12.10 rows=16000 width=0) (actual time=4,378..4,378.43 rows=16000 loops=1)
 - [...] Index Cond: (salary > \$0)
 - [...] Planning Time: 0.444 ms
 - [...] Execution Time: 9,660 ms

Activation Windows message: Go to Settings to activate Windows.

Cost =951, time =9.6ms

Notes:

Using BRIN indices increased the cost to 102.7% of initial cost (W/O indices),

Decreased Time to 73% of initial time (W/O indices)

The brin do support ranges, but the high cost operation was after detecting the entry in idx, we needed to go retrieve it from employee relation to continue executing the query, even brin doesn't support included columns that gives the chance to carry the attributes that you may need not to go back to relation.

⇒ Mixed Indices: btree on employee(salary), hash on employee(dno)

The screenshot shows the PgAdmin 4 interface. In the top navigation bar, 'File', 'Object', 'Tools', and 'Help' are visible. Below the bar, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. The current tab is 'SQL'. The title bar shows 'schema2/postgres@schemas *'. The main area has two panes: 'Query Editor' on the left containing a PostgreSQL script, and 'Messages', 'Notifications', 'Explain', and 'Data Output' on the right. The 'Explain' tab is selected, displaying a detailed query plan. The plan starts with a 'Bitmap Heap Scan on employee' and includes various index scans and joins. At the bottom right of the interface, there is an 'Activate Windows' watermark.

```

1 --drop index idx_emp_dno
2 --drop index idx_emp_sal
3 --drop index idx_wo_essn
4 --drop index idx_dpt_mgr_dnum
5
6
7
8
9
10 explain analyze
11 select lname, fname
12 from employee
13 where salary > (
14 select max(salary)
15 from employee
16 where dno=5 );
17
18
19
20 set enable_bitmapscan=on
21 --set enable_seqscan=on
22 set enable_nestloop=on
23 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn);
24 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
25 CREATE INDEX idx_emp_dno ON employee USING hash(dno);
26 CREATE INDEX idx_emp_sal ON employee USING brin(salary);
27 select department.mgr_snn from department where dnumber=5

```

Cost=404.5, time= 0.9ms

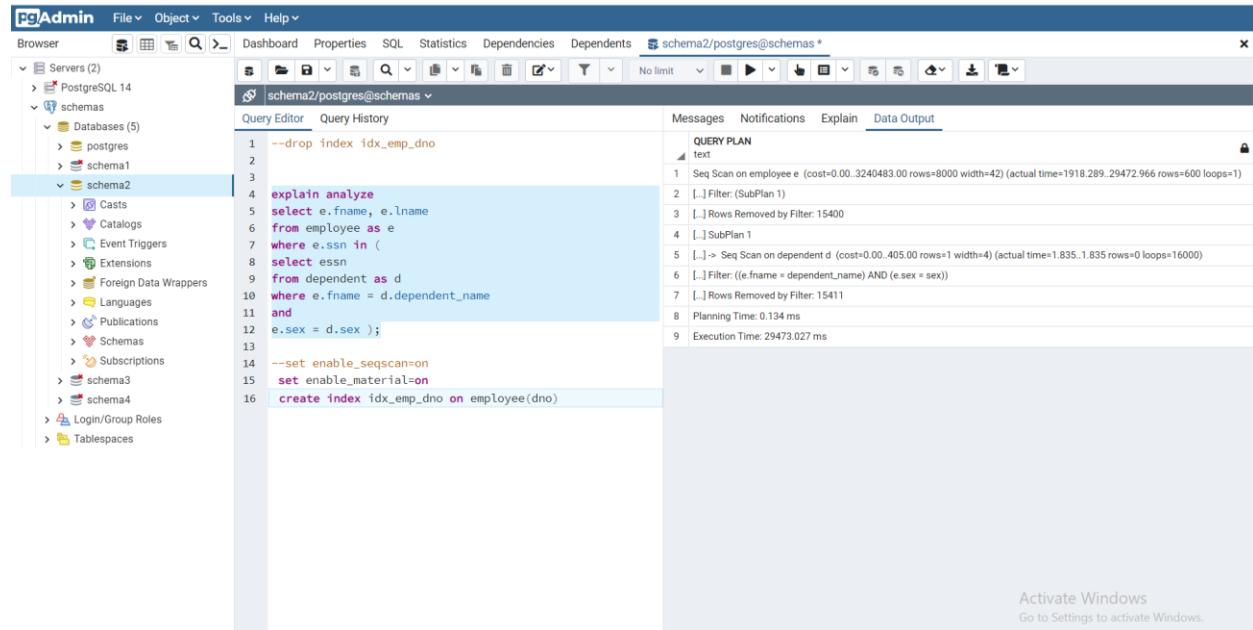
*only one index was used which is employee(salary) as the query can use only one idx, so the chosen one is the one giving least cost

Notes:

The results are exactly the same as btree section, because the query plan uses only one idx which is btree on employee(salary)

Q4:

=>Without indices (seqscans only):



The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
--drop index idx_emp_dno
explain analyze
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 );
--set enable_seqscan=on
set enable_material=on
create index idx_emp_dno on employee(dno)
```

The 'Explain' tab of the results pane displays the query plan:

Step	Operation	Cost	Time
1	Seq Scan on employee e	0.00..3240483.00	actual time=1918.289..29472.966 rows=8000 width=42
2	[...] Filter: (SubPlan 1)		[...] Rows Removed by Filter: 15400
3	[...] SubPlan 1		[...] Filter: ((e.fname = dependent_name) AND (e.sex = sex))
4	[...] Rows Removed by Filter: 15411		
5	[...] -> Seq Scan on dependent d	0.00..405.00	actual time=1.835..1.835 rows=0 loops=16000
6	[...] Filter: ((e.fname = dependent_name) AND (e.sex = sex))		[...] Rows Removed by Filter: 15411
7	Planning Time:	0.134 ms	
8	Execution Time:	29473.027 ms	

A message at the bottom right of the interface says: "Activate Windows Go to Settings to activate Windows."

Cost=3240483, time=29473ms

Notes:

The cost is too high, the time is high as well.

=>Using BTree: on dependent(dependent_name,sex)

The screenshot shows the pgAdmin interface with a query editor containing SQL code and an explain plan window. The code includes dropping existing indexes, setting configuration parameters, and creating new Btree and Brin indices. The explain plan details a sequential scan on the employee table followed by an index scan on the dependent table.

```
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12 explain analyze
13 select e.fname, e.lname
14 from employee as e
15 where e.ssn in (
16 select essn
17 from dependent as d
18 where e.fname = d.dependent_name
19 and
20 e.sex = d.sex );
21
22
23
24 set enable_bitmapscan=on
25 --set enable_seqscan=on
26 set enable_nestloop=on
27 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn);
28 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
29 CREATE INDEX idx_emp_fname ON employee USING btree(fname);
30 CREATE INDEX idx_emp_sex ON employee USING btree(sex);
31 CREATE INDEX idx_dep_sex ON dependent USING btree(sex);
32 CREATE INDEX idx_dep_dependentname ON dependent USING btree(dependent_name,se
```

Activate Windows
Go to Settings to activate Windows.

Cost=69203, time=408ms

Notes:

Using Btree indices decreased the cost to 2.1% of initial cost (W/O indices),

Time to 1.38% of initial time (W/O indices)

The reason behind this is instead of seqscan on relation to find matching tuples, the indices made it much faster

⇒ Using Hash:

The screenshot shows the PGAdmin interface. In the top navigation bar, 'File', 'Object', 'Tools', and 'Help' are visible. Below the bar, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', 'Dependents', and 'schema2/postgres@schemas*'. The main area has tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying the following SQL code:

```
3 --drop index idx_dep_dependentname
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12 explain analyze
13 select e.fname, e.lname
14 from employee as e
15 where e.ssn in (
16 select essn
17 from dependent as d
18 where e.fname = d.dependent_name
19 and
20 e.sex = d.sex );
21
22
23
24 set enable_bitmapscan=on
25 --set enable_seqscan=on
26 set enable_nestloop=on
27 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn);
28 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
29 CREATE INDEX idx_emp_fname ON employee USING btree(fname);
30 CREATE INDEX idx_emp_sex ON employee USING btree(sex);
31 CREATE INDEX idx_dep_sex ON dependent USING btree(sex);
```

To the right of the query editor, there is a 'Messages' window showing a single message: 'Activate Windows Go to Settings to activate Windows.' Below the messages is an 'Explain' window titled 'QUERY PLAN' with the following details:

text
1 Seq Scan on employee e (cost=0.00..64643.00 rows=8000 width=42) (actual time=2.000..23.335 rows=600 loops=1)
2 [...] Filter: (SubPlan 1)
3 [...] Rows Removed by Filter: 15400
4 [...] SubPlan 1
5 [...] -> Index Scan using idx_dep_dependentname on dependent d (cost=0.00..8.02 rows=1 width=4) (actual time=0.001..0.001 ms)
6 [...] Index Cond: (dependent_name = e.fname)
7 [...] Filter: (e.sex = sex)
8 Planning Time: 0.232 ms
9 Execution Time: 23.451 ms

Cost =64643, time=23ms

Notes:

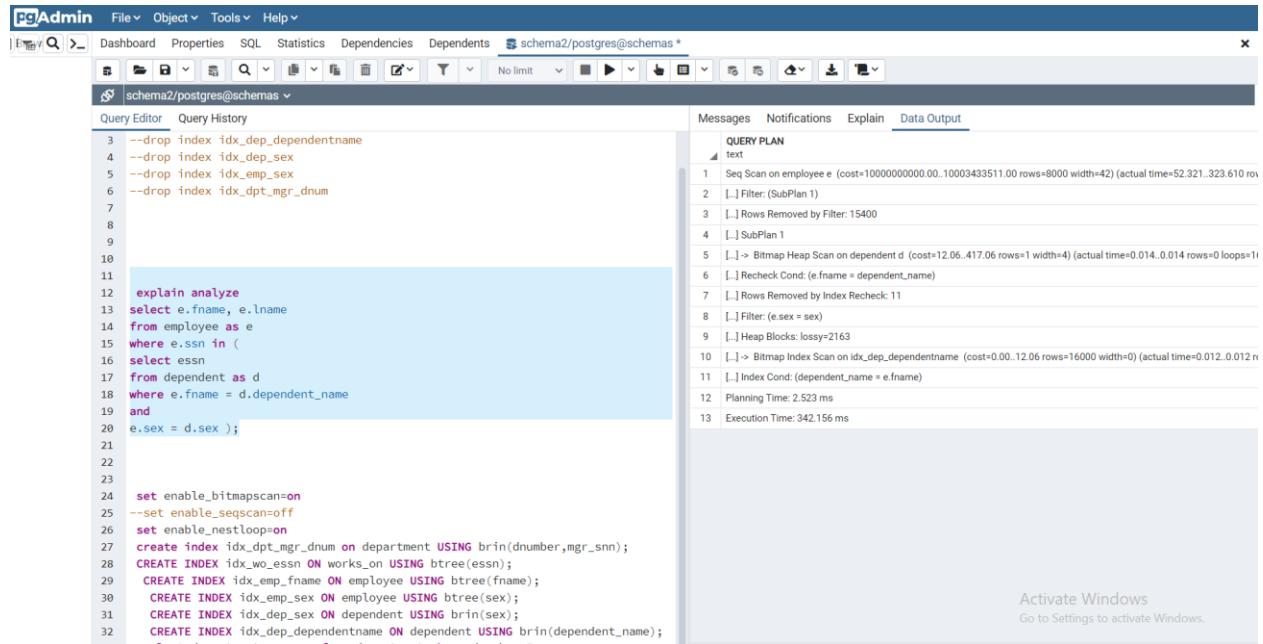
Using Hash indices decreased the cost to 1.9% of initial cost (W/O indices),

Time to 0.078% of initial time (W/O indices)

The reason behind this is instead of seqscan on relation to find matching tuples, the indices made it much faster.

Hash index here is even much more efficient than btree on this query because it's exact values query that is accessible in O(1) with hashing

=>Using BRIN:

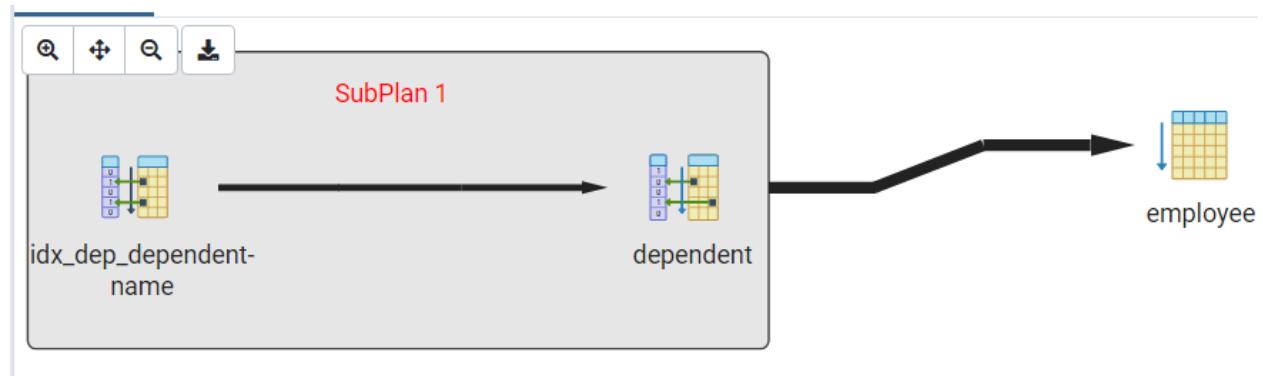


The screenshot shows the PGAdmin interface with a query editor containing SQL code and an explain analyze output. The code includes dropping existing indexes and creating new ones using the BRIN index type. The explain output details the execution plan, showing a seq scan on the employee table, a filter operation, and a bitmaps heap scan on the dependent table. The total execution time is 342.156 ms.

```
3 --drop index idx_dep_dependentname
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12 explain analyze
13 select e.fname, e.lname
14 from employee as e
15 where e.ssn in (
16     select ssn
17     from dependent as d
18     where e.fname = d.dependent_name
19     and
20     e.sex = d.sex );
21
22
23
24 set enable_bitmapscan=on
25 --set enable_seqscan=off
26 set enable_nestloop=on
27 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn);
28 CREATE INDEX idx_wo_ssn ON works_on USING btree(ssn);
29 CREATE INDEX idx_emp_fname ON employee USING btree(fname);
30 CREATE INDEX idx_emp_sex ON employee USING btree(sex);
31 CREATE INDEX idx_dep_sex ON dependent USING brin(sex);
32 CREATE INDEX idx_dep_dependentname ON dependent USING brin(dependent_name);
```

Activate Windows
Go to Settings to activate Windows.

Cost is too high, time=342ms



Notes:

The reason of this very high cost is that we are forcing to use brin index which is not much good in exact value queries, It takes time to reach the data of relation due to the layers of sparse, in addition, it doesn't support multicolumn indx, so it checks the first name and then goes to the relation to check the sex of dependent

=>Using mixed indices: hash on dependent(dependent_name),btree on dependent(sex)

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
3 --drop index idx_dep_dependentname
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12 explain analyze
13 select e.fname, e.lname
14 from employee as e
15 where e.ssn in (
16 select essn
17 from dependent as d
18 where e.fname = d.dependent_name
19 and
20 e.sex = d.sex );
21
22
23
24 set enable_bitmaps=on
25 --set enable_seqscan=on
26 set enable_nestloop=on
27 create index idx_dpt_mgr_dnum on department USING brin(dnumber,mgr_snn);
28 CREATE INDEX idx_wo_essn ON works_on USING btree(essn);
29 CREATE INDEX idx_emp_fname ON employee USING btree(fname);
30 CREATE INDEX idx_emp_sex ON employee USING btree(sex);
31 CREATE INDEX idx_dep_sex ON dependent USING btree(sex);
```

The right pane displays the execution plan:

Step	Operation	Cost	Time
1	Seq Scan on employee e	0.00	64643.00 ms
2	[...] Filter: (SubPlan 1)		
3	[...] Rows Removed by Filter: 15400		
4	[...] SubPlan 1		
5	[...] > Index Scan using idx_dep_dependentname on dependent d	0.00	8.02 ms
6	[...] Index Cond: (dependent_name = e.fname)		
7	[...] Filter: (e.sex = sex)		
8	Planning Time: 0.232 ms		
9	Execution Time: 23.451 ms		

Activation Windows message: Activate Windows
Go to Settings to activate Windows.

Cost=64643, time= 23.45ms

*only one index was used which is dependent(dependent_name) as the query can use only one idx, so the chosen one is the one giving least cost

Notes:

The results are exactly the same as Hash section, because the query plan uses only one idx which is btree on dependent(dependent_name)

Q4: optimized

=>Without indices (seqscans only):

The screenshot shows the PGAdmin interface. In the left sidebar, under 'Servers (2)', 'PostgreSQL 14' is selected. Under 'schemas', 'schema2' is selected. The 'Query Editor' tab is active, displaying the following SQL code:

```
--drop index idx_emp_sexfname
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 );
explain analyze select e.fname, e.lname
from employee as e inner join dependent as d
on ssn=essn
and e.fname = d.dependent_name
and e.sex = d.sex
--set enable_seqscan=on
set enable_material=on
create index idx_emp_sexName on employee(sex,fname)
```

The 'Results' pane shows the 'QUERY PLAN' for the first part of the query:

```
text
1 Hash Join (cost=605.00..1208.01 rows=1 width=42) (actual time=13.976..22.532 rows=600 loops=1)
2 [.] Hash Cond: ((e.ssn = d.essn) AND (e.fname = d.dependent_name) AND (e.sex = d.sex))
3 [.]> Seq Scan on employee e (cost=0.00..423.00 rows=16000 width=48) (actual time=0.044..1.923 rows=16000 loops=1)
4 [.]> Hash (cost=325.00..325.00 rows=16000 width=27) (actual time=13.194..13.196 rows=16000 loops=1)
5 [.] Buckets: 16384 Batches: 1 Memory Usage: 1050kB
6 [.]> Seq Scan on dependent d (cost=0.00..325.00 rows=16000 width=27) (actual time=0.036..4.341 rows=16000 loops=1)
7 Planning Time: 0.592 ms
8 Execution Time: 22.892 ms
```

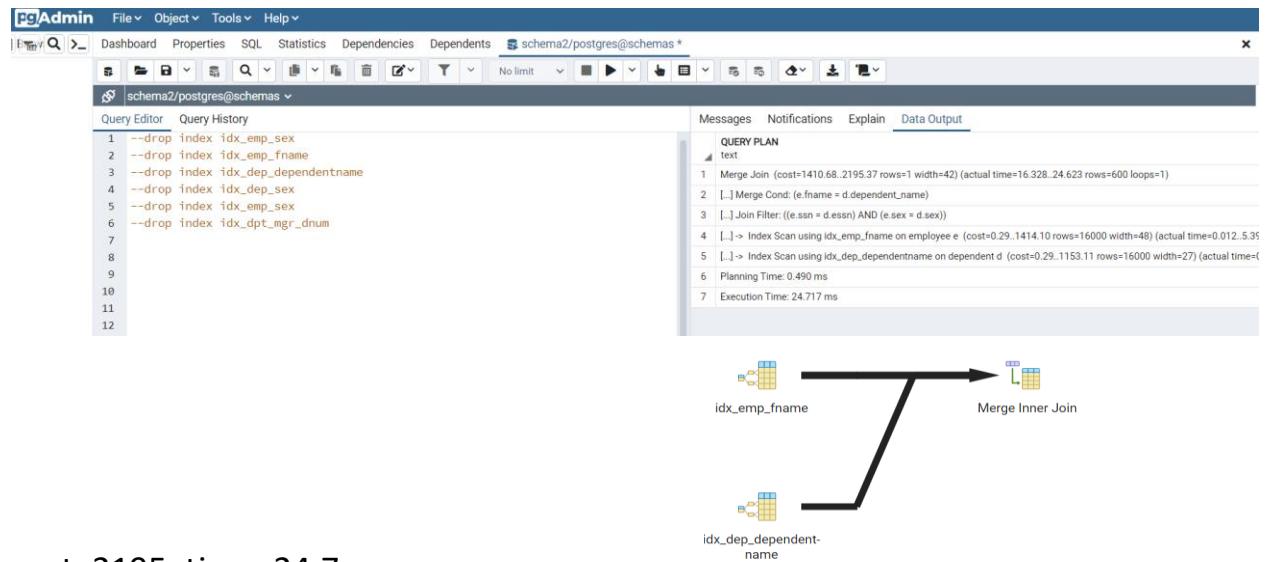
Cost=1208, time=22ms

Notes:

Optimization is based on instead of seqscan of the whole relation dependent for every tuple in the relation employee to check the where clause conditions, we can inner join(hash join) on the same conditions

Optimization reduced cost to 0.037% of the original cost, time to 0.075% of original time

=>Using Btree:



Notes:

The cost increased to 181.7% of the cost without indeces, the reason is that the hash join was the optimal plan, but it doesn't require indeces, so to force the engine to use my btrees, seqscan was turned off, so the type of join done is merge join.

=>Using Hash: on dependent(dependent_name)

The screenshot shows the PgAdmin interface with a query editor containing the following SQL code:

```
1 --drop index idx_emp_sex
2 --drop index idx_emp_fname
3 --drop index idx_dep_dependentname
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12
13 explain analyze
14 select e.fname, e.lname
15 from employee as e inner join dependent as d
16 on ssn=essn
17 and e.fname = d.dependent_name
18 and e.sex = d.sex
```

The "Messages" tab shows the query plan in text format:

```
1 Nested Loop (cost=1000000000.00..10000001867.00 rows=1 width=42) (actual time=1.339..23.975 rows=600 loc
2 [...] > Seq Scan on employee e (cost=1000000000.00..10000000423.00 rows=16000 width=48) (actual time=0.01
3 [...] > Index Scan using idx_dep_dependentname on dependent d (cost=0.00..0.08 rows=1 width=27) (actual time=
4 [...] Index Cond: (dependent_name = e.fname)
5 [...] Filter: ((e.ssn = essn) AND (e.sex = sex))
6 Planning Time: 0.361 ms
7 Execution Time: 24.022 ms
```

The "Data Output" tab shows the results of the query.

Below the interface, a query plan diagram is displayed:

```
graph TD
    employee[employee] --> NestedLoop[Nested Loop Inner Join]
    idxDependentName[idx_dep_dependent-name] --> NestedLoop
```

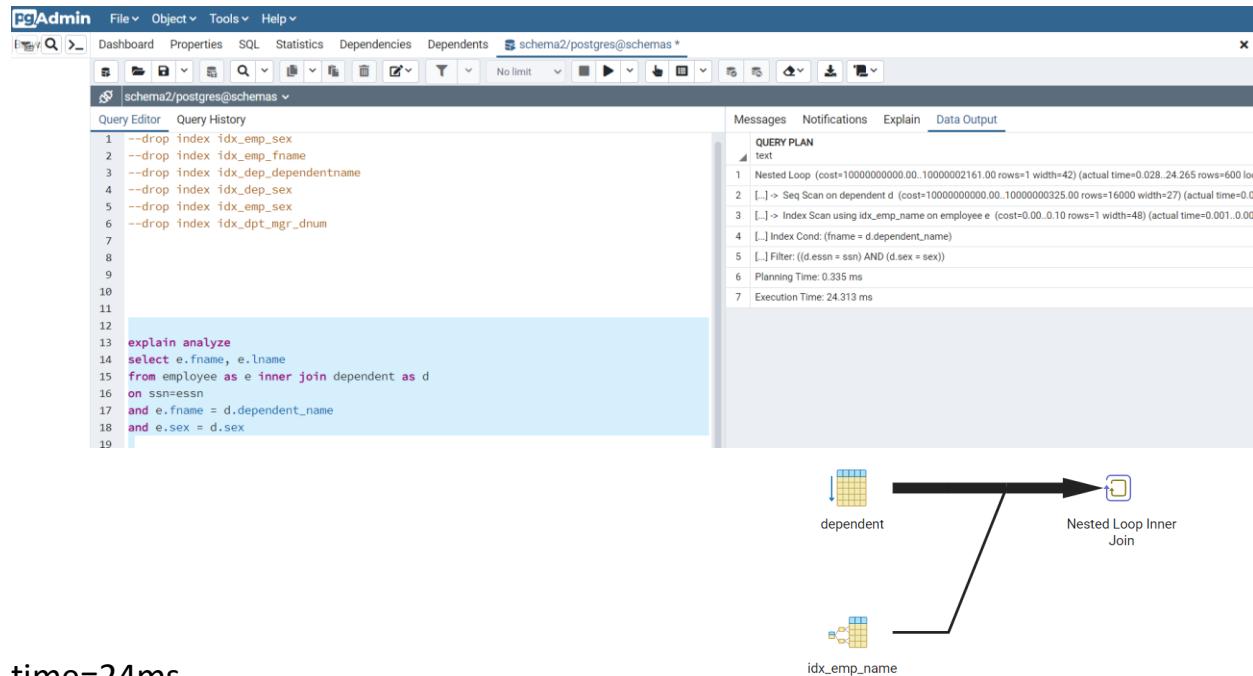
The diagram illustrates a nested loop join between the "employee" table and the "idx_dep_dependent-name" index. The "employee" table is scanned sequentially, and for each row, a search is performed on the "idx_dep_dependent-name" index to find matching tuples.

cost=10000001867, time=24ms

Notes:

The cost increased too much, the reason is that the hash join was the optimal plan, but it doesn't require indeces, so to force the engine to use my hash indeces, seqscan was turned off, so the type of join done is nested loop join, which gets one by one from employee and searches in dependent if there is matching tuple.

=>Using BRIN: on employee(name)



time=24ms

Notes:

The cost increased too much, the reason is that the hash join was the optimal plan, but it doesn't require indeces, so to force the engine to use my brin indeces, seqscan was turned off, so the type of join done is nested loop join, which gets one by one from employee and searches in dependent if there is matching tuple using brin index which is bad in exact values query.

=>Using Mixed: btree on employee(fname), hash on dependent(dependent_name)

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```

1 --drop index idx_emp_sex
2 --drop index idx_emp_fname
3 --drop index idx_dep_dependentname
4 --drop index idx_dep_sex
5 --drop index idx_emp_sex
6 --drop index idx_dpt_mgr_dnum
7
8
9
10
11
12
13 explain analyze
14 select e.fname, e.lname
15 from employee as e inner join dependent as d
16 on ssn=essn
17 and e.fname = d.dependent_name
18 and e.sex = d.sex
19

```

The "Data Output" tab is selected, showing the query plan:

```

1 Nested Loop (cost=0.29..2858.10 rows=1 width=42) (actual time=51.121..52.212 rows=600 loops=1)
2 [...] > Index Scan using idx_emp_fname on employee e (cost=0.29..1414.10 rows=16000 width=48) (actual time=0.00..0.08 rows=1 width=27)
3 [...] > Index Scan using idx_dep_dependentname on dependent d (cost=0.00..0.08 rows=1 width=27) (actual time=0.00..0.00 rows=1 width=27)
4 [...] Index Cond: (dependent_name = e.fname)
5 [...] Filter: ((e.sex = essn) AND (e.sex = sex))
6 Planning Time: 3.062 ms
7 Execution Time: 52.260 ms

```

Below the query editor, a query plan diagram is displayed. It shows two index scans: "idx_emp_fname" and "idx_dep_dependentname". Arrows point from these indices to a central node labeled "Nested Loop Inner Join".

cost=2858,time=52ms

Notes:

The cost is not too high, but it's higher than the one w/o indexes. The engine was not willing to use any indexes to do a hash join, so we turned the seqscan off. The cost is not far away from btree version.

Q5:

=>Without indices (seqscans only):

The screenshot shows the pgAdmin interface with a query editor window. The query is:`--drop index idx_emp_sexname
explain analyze
select fname, lname from employee
where exists (select *
 from dependent
 where ssn=essn);
--set enable_seqscan=on
set enable_material=on
create index idx_emp_sexName on employee(sex,fname)
CREATE INDEX idx_emp_name ON employee USING hash(fname);`

The query plan (shown in the 'QUERY PLAN' tab) indicates a Hash Join operation with a cost of 378.50, taking 13.883 ms. It also shows a Hash Cond, Seq Scan on employee, HashAggregate, and Hash operations.

Cost=850, time=13.9ms

Notes:

One relation only used which is employee, materialization takes alittle more cost but reduces the time so much

=>Using BTree: on employee(ssn), dependent(essn)

The screenshot shows the pgAdmin interface with the following details:

- Servers:** PostgreSQL 14
- Schemas:** schema2 (selected)
- Query Editor:** Contains the following SQL code:

```
--drop index idx_emp_ssn
explain analyze
select fname, lname from employee
where exists ( select *
               from dependent
               where ssn=essn );
--set enable_seqscan=on
set enable_material=on
create index idx_emp_ssn on employee USING btree(ssn)
CREATE INDEX idx_dep_essn ON dependent USING btree(essn);
```
- Messages:** Nested Loop (cost=352.58..428.71 rows=600 width=42) (actual time=5.079..7.236 rows=600 loops=1)
- Notifications:** None
- Explain:** Nested Loop (cost=352.58..428.71 rows=600 width=42) (actual time=5.079..7.236 rows=600 loops=1)
 - [...] > HashAggregate (cost=352.29..358.29 rows=600 width=4) (actual time=4.928..5.074 rows=600 loops=1)
 - [...] Group Key: dependent.essn
 - [...] Batches: 1 Memory Usage: 73kB
 - [...] > Index Only Scan using idx_dep_essn on dependent (cost=0.29..312.29 rows=16000 width=4) (actual time=0.002..0.002 rows=1 loops=1)
 - [...] Cache Key: dependent.essn
 - [...] Cache Mode: logical
 - [...] Hits: 0 Misses: 600 Evictions: 0 Overflows: 0 Memory Usage: 87kB
 - [...] > Index Scan using idx_emp_ssn on employee (cost=0.29..2.36 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=1)
 - [...] Index Cond: (ssn = dependent.essn)
 - Data Output:** QUERY PLAN text
- Activate Windows:** Go to Settings to activate Windows.

Cost=428, time=7ms

Notes:

Btree indeces reduce the cost to 50.3% of same query W/O indeces, time to 50% of previous time.

Instead of hashing to do hash join, the btree indices provides fast way to search for the condition of nested loop join, so we are searching for ssn=essn, so we can reach both of them faster

=>Using Hash: on employee(ssn)

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a tree structure of databases, schemas, and objects. The current schema selected is 'schema2'. In the center, a query editor window contains the following SQL code:

```
1 --drop index idx_dep_essn
2 explain analyze
3 select fname,lname from employee
4 where exists (select *
   from dependent
   where ssn=essn );
7
8
9 --set enable_seqscan=on
10 set enable_nestloop=on
11 create index idx_emp_ssn on employee USING hash(ssn)
--CREATE INDEX idx_dep_essn ON dependent USING hash(essn);
```

To the right of the query editor is a 'QUERY PLAN' pane. The plan details the execution steps:

- 1 Nested Loop (cost=365.01..437.87 rows=600 width=42) (actual time=8.082..9.868 rows=600 loops=1)
 - 2 [...] -> HashAggregate (cost=365.00..371.00 rows=600 width=4) (actual time=8.056..8.149 rows=600 loops=1)
 - 3 [...] Group Key: dependent.essn
 - 4 [...] Batches: 1 Memory Usage: 73kB
 - 5 [...] -> Seq Scan on dependent (cost=0.00..325.00 rows=16000 width=4) (actual time=0.020..1.977 rows=16000)
 - 6 [...] Memoize (cost=0.01..2.22 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=600)
 - 7 [...] Cache Key: dependent.essn
 - 8 [...] Cache Mode: logical
 - 9 [...] Hits: 0 Misses: 600 Evictions: 0 Overflows: 0 Memory Usage: 87kB
 - 10 [...] -> Index Scan using idx_emp_ssn on employee (cost=0.00..2.21 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=600)
 - 11 [...] Index Cond: (san = dependent.essn)
 - 12 Planning Time: 0.337 ms
 - 13 Execution Time: 10.071 ms

Activate Windows
Go to Settings to activate Windows.

Cost=437, time=10ms

Notes:

Hash indeces reduce the cost to 51.4% of same query W/O indeces, time to 71.9% of previous time.

Since ssn is primary key (unique), but essn is not unique, so searching for unique key using hash is much easier so employee(ssn) is chosen to be the key which is being searched for, for each tuple of dependent relation.

=>with BRIN: on employee(ssn)

The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
--drop index idx_emp_ssn
explain analyze
select fname,Lname from employee
where exists ( select *
from dependent
where ssn=essn );
--set enable_seqscan=off
set enable_nestloop=on
create index idx_emp_ssn on employee USING brin(ssn)
CREATE INDEX idx_dep_essn ON dependent USING brin(essn);
```

The right pane displays the query plan:

Step	Operation	Cost	Rows	Width
1	Nested Loop	cost=10000007565.04..10000170974.67	rows=600	width=42
2	[...] HashAggregate	(cost=10000000365.00..10000000371.00)	rows=600	width=4
3	[...] Group Key: dependent.essn			
4	[...] Batches: 1 Memory Usage: 73kB			
5	[...] Seq Scan on dependent (cost=10000000000.00..10000000325.00)	rows=16000	width=4	
6	[...] Memoize (cost=7200.04..7270.70)	rows=1	width=46	
7	[...] Cache Key dependent.essn			
8	[...] Cache Mode: logical			
9	[...] Hits: 0 Misses: 600 Evictions: 0 Overflows: 0 Memory Usage: 87kB			
10	[...] Bitmap Heap Scan on employee (cost=7200.03..7270.69)	rows=1	width=46	
11	[...] Recheck Cond: (ssn = dependent.essn)			
12	[...] Rows Removed by Index Recheck: 7807			
13	[...] Heap Blocks: lossy=76800			
14	[...] Bitmap Index Scan on idx_emp_ssn (cost=0.00..7200.03)	rows=5333	width=0	
15	[...] Index Cond: (ssn = dependent.essn)			
16	Planning Time: 0.262 ms			
17	Execution Time: 614.504 ms			

Activation Windows message: Go to Settings to activate Windows.

Cost=10000170974, time=614ms

Notes:

The plan can use one brin index only to search for each key of the other relation, the options are employee(ssn) or dependent(essn), postgres has a feature of caching repetitive searches, so when choosing employee to have the seqscan to search in dependent, it's a bad choice because we won't repeat any search as ssn is unique, but the other side will have 600 unique search only instead of 16000.

It's obvious that brin performance is much worse than the one w/o index because the search of brin is not that easy, it passes by several sparse layers, however the efficiency of brin appears when searching in small range of big domain which is not the case here.

=>mixed indices: hash on employee(ssn),btree on dependent(essn)

The screenshot shows two Explain Analyze results side-by-side in pgAdmin. Both results are for the same query:

```

1 --drop index idx_dep_essn
2 explain analyze
3 select fname,lname from employee
4 where exists ( select *
      from dependent
      where ssn=essn );
6
7
8
9 --set enable_seqscan=on
10 set enable_nestloop=on
11 create index idx_emp_ssn on employee USING hash(ssn)
12 CREATE INDEX idx_dep_essn ON dependent USING btree(essn);

```

Graphical Explain Plan (Left):

- idx_dep_essn (BTree Index Scan) feeds into Aggregate.
- idx_emp_ssn (Index Only Scan) feeds into Memoize.
- Memoize feeds into Nested Loop Inner Join, which then feeds into Aggregate.
- Aggregate produces the final result.

Text Explain Plan (Right):

- Nested Loop (cost=352.30..425.15)
- HashAggregate (cost=352.29..358.29)
- Group Key: dependent.essn
- Batches: 1 Memory Usage: 73kB
- Index Only Scan using idx_dep_essn on dependent (cost=0.29..312.29)
- Heap Fetches: 0
- Memoize (cost=0.01..2.22)
- Cache Key: dependent.essn
- Cache Mode: logical
- Hits: 0 Misses: 600 Evictions: 0 Overflows: 0 Memory Usage: 87kB
- Index Scan using idx_emp_ssn on employee (cost=0.00..2.21)
- Index Cond: (ssn = dependent.essn)
- Planning Time: 0.465 ms
- Execution Time: 8.637 ms

Cost=425, time=8.6ms

Notes:

The mixed indices is the optimal solution as it gives the lowest cost among all previous, it's not far away from btree indices, but it gives less cost anyway.

Mixed indices reduces the cost to be 50.0% of the one w/o indices and time to 61% of initial time.

Both plans of mixed and btree indices runs the same, as we get tuple of dependent from its index and search for matching tuple in employee relation using the index on ssn, but hash index is a bit faster than the btree index in searching specially in exact value in a unique column.