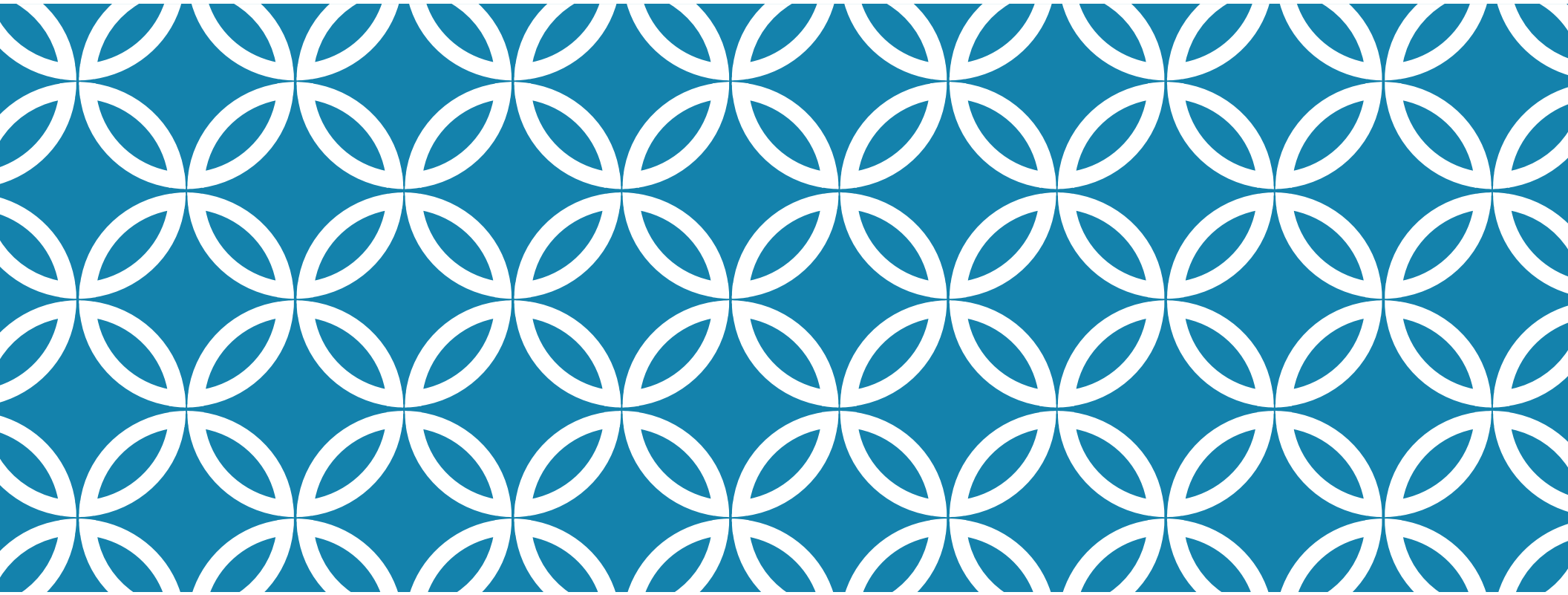


SCHEMA#1

QUERY#1

```
select *
from (select *
from student
where
department = 'CS1') as CS1_student
natural 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;
```

WITHOUT INDEX

Query History | Query Editor

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO OFF;
3  SET enable_indexonlyscan  TO OFF;
4  SET enable_seqscan  TO ON;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO On;
7  explain analyze select *
8  from (select *
9  from student
10 where
11 department = 'CS1') as CS1_student
12 natural full outer join
13 (select *
14 from takes t inner join section s
```

Messages | Explain | Notifications | Data Output

	QUERY PLAN text
1	Merge Full Join (cost=0.00..137.30 rows=435 width=64) (actual time=0.053..1.738 rows=435 loops=1)
2	-> Seq Scan on student (cost=0.00..129.53 rows=435 width=24) (actual time=0.024..1.512 rows=435 loops=1)
3	Filter: ((department)::text = 'CS1'::text)
4	Rows Removed by Filter: 6087
5	-> Materialize (cost=0.00..2.34 rows=1 width=40) (actual time=0.026..0.055 rows=1 loops=1)
6	-> Nested Loop (cost=0.00..2.34 rows=1 width=40) (actual time=0.024..0.027 rows=1 loops=1)
7	Join Filter: (t.section_id = s.section_id)
8	Rows Removed by Join Filter: 8
9	-> Seq Scan on section s (cost=0.00..1.14 rows=1 width=28) (actual time=0.011..0.012 rows=1 loops=1)
10	Filter: ((semester = 1) AND (year = 2019))
11	Rows Removed by Filter: 8
12	-> Seq Scan on takes t (cost=0.00..1.09 rows=9 width=12) (actual time=0.011..0.012 rows=9 loops=1)
13	Planning time: 0.404 ms
14	Execution time: 1.807 ms

WITH BTREE INDEX

For the btree we used 3 indices for tables student,section,takes all where used in the indexscan join an a merge full join was used where Each relation is sorted on the join attributes before the join starts Then the two relations are scanned in parallel, and matching rows are combined to form join rows. the execution time is reduced 30% from scenario1.

Query History | Query Editor

```
1  set enable_seqscan = off;
2  set enable_gathermerge= off;
3  set enable_hashagg = off;
4  set enable_hashjoin = off;
5  set enable_indexscan = on;
6  set enable_indexonlyscan = on;
7  set enable_material = off;
8  set enable_mergejoin = on;
9  set enable_nestloop = on;
10 set enable_bitmapscan= off;
11 set enable_sort = off;
12 set enable_tidscan = off;
13 create index Sbtree on student using btree(department);
14 create index SSbtree on section using btree(semester);
15  create index tbtree on takes using btree(section_id);
16 explain analyze select *
17 from (select *
```

Messages | Explain | Notifications | Data Output

	QUERY PLAN text
1	Merge Full Join (cost=0.55..227.74 rows=435 width=64) (actual time=0.193..0.536 rows=435 loops=1)
2	-> Nested Loop (cost=0.27..16.32 rows=1 width=40) (actual time=0.095..0.097 rows=1 loops=1)
3	-> Index Scan using ssbtree on section s (cost=0.14..8.15 rows=1 width=28) (actual time=0.082..0.083 rows=1 loops=1)
4	Index Cond: (semester = 1)
5	Filter: (year = 2019)
6	-> Index Scan using tbtree on takes t (cost=0.14..8.15 rows=1 width=12) (actual time=0.009..0.010 rows=1 loops=1)
7	Index Cond: (section_id = s.section_id)
8	-> Index Scan using sbtree on student (cost=0.28..207.07 rows=435 width=24) (actual time=0.095..0.306 rows=435 loops=1)
9	Index Cond: ((department)::text = 'CS1'::text)
10	Planning time: 0.475 ms
11	Execution time: 0.609 ms

WITH HASH INDEX

For hash index to work we had to keep the bitmap scan on due to nature of column department in table student. A full merge join strategy was used and a bitmap index scan which used the 3 indices I created.The hash was the least execution time since the the final result is built up by a tree of join steps merge and nestloop and in the merge each relation has to be scanned only once.

Query History | Query Editor

```
1  set enable_seqscan = off;
2  set enable_gathermerge= off;
3  set enable_hashagg = on;
4  set enable_hashjoin = on;
5  set enable_indexscan = on;
6  set enable_indexonlyscan = on;
7  set enable_material = off;
8  set enable_mergejoin = on;
9  set enable_nestloop = on;
10 set enable_bitmapscan= on;
11 set enable_sort = off;
12 set enable_tidscan = off;
13 create index Shash on student using hash(department);
14  create index SShash on section using hash(semester);
15  create index thash on takes using hash(section_id);
```

Data Output | Notifications | Explain | Messages

	QUERY PLAN text
1	Merge Full Join (cost=19.37..94.29 rows=435 width=64) (actual time=0.080..0.400 rows=435 loops=1)
2	-> Bitmap Heap Scan on student (cost=19.37..72.81 rows=435 width=24) (actual time=0.067..0.249 rows=435 loops=1)
3	Recheck Cond: ((department)::text = 'CS1'::text)
4	Heap Blocks: exact=48
5	-> Bitmap Index Scan on shash (cost=0.00..19.26 rows=435 width=0) (actual time=0.052..0.052 rows=435 loops=1)
6	Index Cond: ((department)::text = 'CS1'::text)
7	-> Materialize (cost=0.00..16.05 rows=1 width=40) (actual time=0.011..0.034 rows=1 loops=1)
8	-> Nested Loop (cost=0.00..16.05 rows=1 width=40) (actual time=0.009..0.011 rows=1 loops=1)
9	-> Index Scan using sshash on section s (cost=0.00..8.02 rows=1 width=28) (actual time=0.004..0.005 rows=1 loops=1)
10	Index Cond: (semester = 1)
11	Filter: (year = 2019)
12	-> Index Scan using thash on takes t (cost=0.00..8.02 rows=1 width=12) (actual time=0.003..0.004 rows=1 loops=1)
13	Index Cond: (section_id = s.section_id)
14	Planning time: 0.407 ms
15	Execution time: 0.483 ms

WITH BITMAP INDEX

```
set enable_seqscan = off;
set enable_gathermerge= off;
set enable_hashagg = on;
set enable_hashjoin = on;
set enable_indexscan = on;
set enable_indexonlyscan = on;
set enable_material = off;
set enable_mergejoin = on;
set enable_nestloop = on;
set enable_bitmapscan= on;
set enable_sort = off;
set enable_tidscan = off;
create extension btree_gin;
create index Sgin on student using gin(department);
 create index Ssgin on section using gin(semester);
 create index tgin on takes using gin(section_id);
```

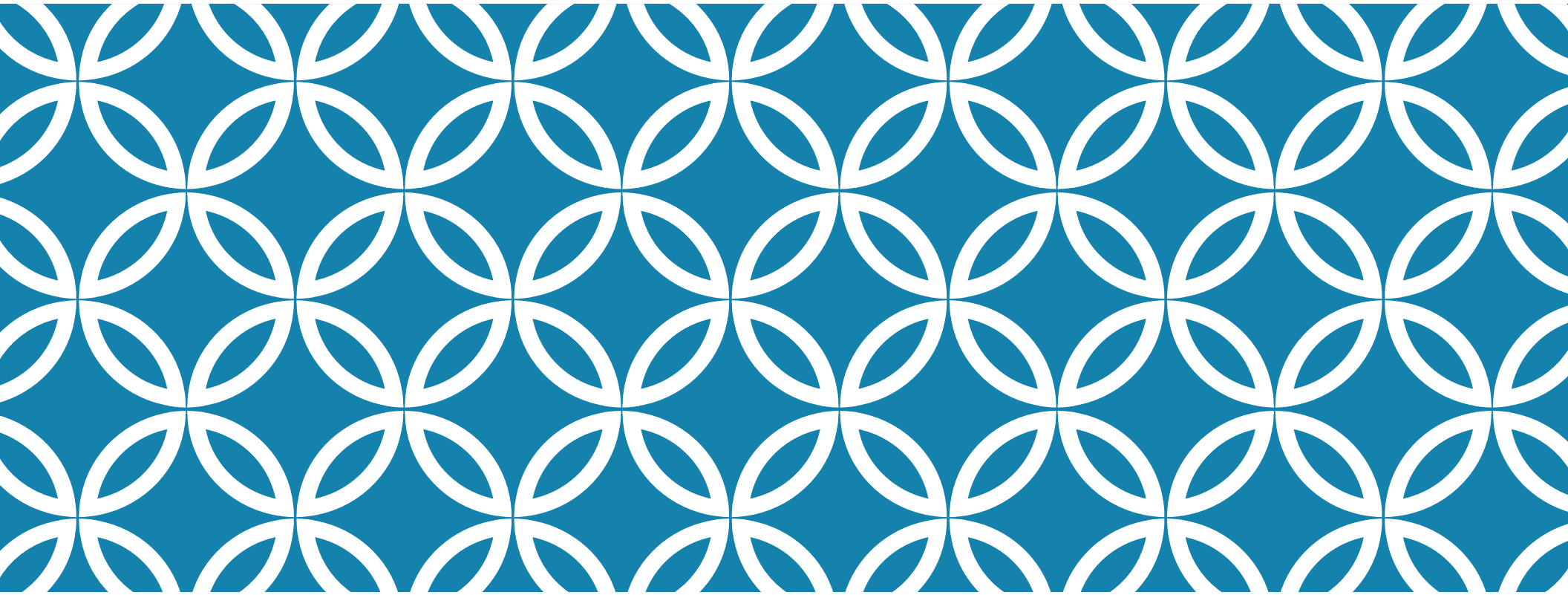
Data Output | Messages | Notifications | Explain

	QUERY PLAN text
1	Merge Full Join (cost=27.39..94.30 rows=435 width=64) (actual time=0.067..0.196 rows=435 loops=1)
2	-> Bitmap Heap Scan on student (cost=11.37..64.81 rows=435 width=24) (actual time=0.047..0.106 rows=435 loops=1)
3	Recheck Cond: ((department)::text = 'CS1'::text)
4	Heap Blocks: exact=48
5	-> Bitmap Index Scan on sgin (cost=0.00..11.26 rows=435 width=0) (actual time=0.040..0.040 rows=435 loops=1)
6	Index Cond: ((department)::text = 'CS1'::text)
7	-> Materialize (cost=16.02..24.06 rows=1 width=40) (actual time=0.018..0.025 rows=1 loops=1)
8	-> Nested Loop (cost=16.02..24.05 rows=1 width=40) (actual time=0.016..0.016 rows=1 loops=1)
9	-> Bitmap Heap Scan on section s (cost=8.01..12.02 rows=1 width=28) (actual time=0.003..0.003 rows=1 loops=1)
10	Recheck Cond: (semester = 1)
11	Filter: (year = 2019)
12	Heap Blocks: exact=1
13	-> Bitmap Index Scan on ssgin (cost=0.00..8.01 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1)
14	Index Cond: (semester = 1)
15	-> Bitmap Heap Scan on takes t (cost=8.01..12.02 rows=1 width=12) (actual time=0.004..0.004 rows=1 loops=1)
16	Recheck Cond: (section_id = s.section_id)
17	Heap Blocks: exact=1
18	-> Bitmap Index Scan on tgin (cost=0.00..8.01 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1)
19	Index Cond: (section_id = s.section_id)
20	Planning time: 1.190 ms
21	Execution time: 0.289 ms

The query planner used the three of the indices I created giving the best and least execution and planning time by using a full merge join strategy where Each relation is sorted on the join attributes before the join starts. Then the two relations are scanned in parallel, and matching rows are combined to form join rows. First relation used bitmap heap while the other used nestloop divided into to loops both bitmapped.

EXPLAIN

The bitmap index was the best choice with optimal and least execution /planning and cost as well with using a full merge join where Each relation is sorted on the join attributes before the join starts. Then the two relations are scanned in parallel, and matching rows are combined to form join rows.The first was on student using bitmap index scan and the second relation was on takes using bitmap index on takes section_id column and bitmap scan on section table semester column.



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' );
```

WITHOUT INDEX

Query History | Query Editor

```
1  explain analyze
2  select distinct pnumber
3  from project
4  where pnumber in
```

Notifications | Explain | Messages | Data Output

	QUERY PLAN text
1	HashAggregate (cost=328.64..332.80 rows=416 width=4) (actual time=2.178..2.178 rows=0 loops=1)
2	Group Key: project.pnumber
3	-> Seq Scan on project (cost=311.29..327.60 rows=416 width=4) (actual time=2.173..2.173 rows=0 loops=1)
4	Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
5	Rows Removed by Filter: 554
6	SubPlan 1
7	-> Nested Loop (cost=0.00..165.34 rows=1 width=4) (actual time=1.189..1.189 rows=0 loops=1)
8	-> Nested Loop (cost=0.00..146.26 rows=1 width=0) (actual time=1.189..1.189 rows=0 loops=1)
9	Join Filter: ((d.dnumber = e.dno) AND (d.mgr_snn = e.ssn))
10	-> Seq Scan on employee e (cost=0.00..144.51 rows=1 width=8) (actual time=1.186..1.186 rows=0 loops=1)
11	Filter: (lname = 'employee1'::bpchar)
12	Rows Removed by Filter: 5001
13	-> Seq Scan on department d (cost=0.00..1.30 rows=30 width=8) (never executed)
14	-> Seq Scan on project project_1 (cost=0.00..13.54 rows=554 width=4) (never executed)
15	SubPlan 2
16	-> Nested Loop (cost=0.00..145.94 rows=1 width=4) (actual time=0.838..0.838 rows=0 loops=1)
17	Join Filter: (works_on.essn = employee.ssn)
18	-> Seq Scan on employee (cost=0.00..144.51 rows=1 width=4) (actual time=0.838..0.838 rows=0 loops=1)
19	Filter: (lname = 'employee1'::bpchar)
20	Rows Removed by Filter: 5001
21	-> Seq Scan on works_on (cost=0.00..1.19 rows=19 width=8) (never executed)
22	Planning time: 0.453 ms
23	Execution time: 2.291 ms

WITH BTREE INDEX

```
SET enable_bitmapscan TO OFF;
SET enable_indexscan TO ON;
SET enable_indexonlyscan  TO ON;
SET enable_seqscan  TO OFF;
SET enable_hashjoin   TO ON;
SET enable_mergejoin    TO OFF;
set enable_nestloop TO on;
set enable_mergejoin to on;
 create index Pbtree on Project using btree(pnumber);
create index Ebtree on employee using btree(lname);
create index Dbtree on department using btree(dnumber);
create index Wbtree on works_on using btree(essn);
```

Explain | Messages | Data Output | Notifications

	QUERY PLAN text
1	Unique (cost=102.34..169.46 rows=416 width=4) (actual time=0.347..0.582 rows=554 loops=1)
2	-> Index Only Scan using pbtree on project (cost=102.34..168.42 rows=416 width=4) (actual time=0.347..0.504 rows=554 loops=1)
3	Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
4	Heap Fetches: 554
5	SubPlan 1
6	-> Nested Loop (cost=0.70..85.59 rows=1 width=4) (actual time=0.048..0.183 rows=554 loops=1)
7	-> Nested Loop (cost=0.42..16.47 rows=1 width=0) (actual time=0.023..0.024 rows=1 loops=1)
8	-> Index Scan using ebtree on employee e (cost=0.28..8.30 rows=1 width=8) (actual time=0.015..0.015 rows=1 loops=1)
9	Index Cond: (lname = 'Employee1'::bpchar)
10	-> Index Scan using dbtree on department d (cost=0.14..8.16 rows=1 width=8) (actual time=0.006..0.007 rows=1 loops=1)
11	Index Cond: (dnumber = e.dno)
12	Filter: (e.ssn = mgr_snn)
13	-> Index Only Scan using pbtree on project project_1 (cost=0.28..63.58 rows=554 width=4) (actual time=0.024..0.120 rows=554 loops=1)
14	Heap Fetches: 554
15	SubPlan 2
16	-> Nested Loop (cost=0.42..16.47 rows=1 width=4) (never executed)
17	-> Index Scan using ebtree on employee (cost=0.28..8.30 rows=1 width=4) (never executed)
18	Index Cond: (lname = 'Employee1'::bpchar)
19	-> Index Scan using wbtree on works_on (cost=0.14..8.15 rows=1 width=8) (never executed)
20	Index Cond: (essn = employee.ssn)
21	Planning time: 0.457 ms
22	Execution time: 0.683 ms

For the btree index we created 4 indidces on each of the 4 tables that where used in the seqscan search in scenario1.The query planner used the nestloop join strategy to where The right relation is scanned once for every row found in the left relation and inside the scan the indices are used to filter the rows according to the index condition stated in the query selection

WITH HASH INDEX

Query Editor | Query History

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO ON;
6  SET enable_mergejoin    TO On;
7  set enable_nestloop to on;
8  set enable_sort to off;
9     create index Phash on Project using hash(pnumber);
10  create index Ehash on employee using hash(lname);
11   create index Dhash on department using hash(dnumber);
12  create index Whash on works_on using hash(essn);
13 explain analyze select distinct pnumber
14 from project
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	HashAggregate (cost=20000000068.53..20000000072.69 rows=416 width=4) (actual time=0.767..0.847 rows=554 loops=1)
2	Group Key: project.pnumber
3	-> Seq Scan on project (cost=20000000051.18..20000000067.49 rows=416 width=4) (actual time=0.398..0.582 rows=554 loops=1)
4	Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
5	SubPlan 1
6	-> Nested Loop (cost=10000000000.00..10000000035.13 rows=1 width=4) (actual time=0.028..0.165 rows=554 loops=1)
7	-> Nested Loop (cost=0.00..16.05 rows=1 width=0) (actual time=0.014..0.017 rows=1 loops=1)
8	-> Index Scan using ehash on employee e (cost=0.00..8.02 rows=1 width=8) (actual time=0.009..0.009 rows=1 loops=1)
9	Index Cond: (lname = 'Employee1'::bpchar)
10	-> Index Scan using dhash on department d (cost=0.00..8.02 rows=1 width=8) (actual time=0.004..0.005 rows=1 loops=1)
11	Index Cond: (dnumber = e.dno)
12	Filter: (e.ssn = mgr_snn)
13	-> Seq Scan on project project_1 (cost=10000000000.00..10000000013.54 rows=554 width=4) (actual time=0.013..0.079 rows=554 loops=1)
14	SubPlan 2
15	-> Nested Loop (cost=0.00..16.05 rows=1 width=4) (never executed)
16	-> Index Scan using ehash on employee (cost=0.00..8.02 rows=1 width=4) (never executed)
17	Index Cond: (lname = 'Employee1'::bpchar)
18	-> Index Scan using whash on works_on (cost=0.00..8.02 rows=1 width=8) (never executed)
19	Index Cond: (essn = employee.ssn)
20	Planning time: 0.482 ms
21	Execution time: 1.047 ms

For the hash index the cost seem to be high since the seqscan is always used turned off or on it seems that the engine always uses it which is also the reason for the higher execution time than the Btree index.Also I needed to keep the sort turned off as sorting the data was increasing the cost without actually doing a difference in the execution and planning time while the other scenarios didn't need the sort so I don't care about its flag.

WITH BITMAP INDEX

For the bitmap index I used the same indices with the same table columns but to force the engine to use these indices as optimally the engine was using the primary key for each table as the index for the index scan so I had to drop theses primary key constraints in the tables used in the scan but since the seqscan was turned off the cost for the bitmap is extremely huge but to compensate that the exection time and planning time is almost half the scenario 1 values surely no the optimal choice as the hash/btree provided lower cost and values but the exection time was extremely high when I tried to keep the seqscan on since there is a nestloop join with two subplans

```
SET enable_bitmapscan TO On;
SET enable_indexscan TO On;
SET enable_indexonlyscan  TO Off;
SET enable_seqscan  TO Off;
SET enable_hashjoin   TO OFF;
SET enable_mergejoin    TO On;
set enable_nestloop to on;
set enable_sort to off;
create index Pgin on Project using gin(pnumber);
create index Egin on employee using gin(lname);
 create index Dgin on department using gin(dnumber);
create index Wgin on works_on using gin(essn);
 ALTER TABLE project DROP CONSTRAINT project_pkey cascade;
ALTER TABLE department DROP CONSTRAINT department_pkey cascade;
 ALTER TABLE employee DROP CONSTRAINT employee_pkey cascade;
 ALTER TABLE works_on DROP CONSTRAINT works_on_pkey cascade;
```

	QUERY PLAN text
1	HashAggregate (cost=20000000361.68..20000000365.84 rows=416 width=4) (actual time=0.835..0.928 rows=555 loops=1)
2	Group Key: project.pnumber
3	-> Seq Scan on project (cost=20000000344.31..20000000360.64 rows=416 width=4) (actual time=0.459..0.639 rows=555 loops=1)
4	Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
5	SubPlan 1
6	-> Nested Loop (cost=10000000012.53..10000000181.73 rows=3 width=4) (actual time=0.076..0.218 rows=555 loops=1)
7	-> Nested Loop (cost=12.53..162.63 rows=1 width=0) (actual time=0.062..0.064 rows=1 loops=1)
8	-> Bitmap Heap Scan on employee e (cost=12.19..57.85 rows=24 width=8) (actual time=0.041..0.041 rows=1 loops=1)
9	Recheck Cond: ((lname)::text = 'Employee1'::text)
10	Heap Blocks: exact=1
11	-> Bitmap Index Scan on egin (cost=0.00..12.18 rows=24 width=0) (actual time=0.037..0.037 rows=1 loops=1)
12	Index Cond: ((lname)::text = 'Employee1'::text)
13	-> Bitmap Heap Scan on department d (cost=0.34..4.36 rows=1 width=8) (actual time=0.008..0.009 rows=1 loops=1)
14	Recheck Cond: (dnumber = e.dno)
15	Filter: (e.ssn = mgr_snn)
16	Heap Blocks: exact=1
17	-> Bitmap Index Scan on dgin (cost=0.00..0.34 rows=1 width=0) (actual time=0.005..0.005 rows=1 loops=1)
18	Index Cond: (dnumber = e.dno)
19	-> Seq Scan on project project_1 (cost=10000000000.00..10000000013.55 rows=555 width=4) (actual time=0.013..0.084 rows=555 loops=1)
20	SubPlan 2
21	-> Nested Loop (cost=12.53..162.57 rows=1 width=4) (never executed)
22	-> Bitmap Heap Scan on employee (cost=12.19..57.85 rows=24 width=4) (never executed)
23	Recheck Cond: ((lname)::text = 'Employee1'::text)
24	-> Bitmap Index Scan on egin (cost=0.00..12.18 rows=24 width=0) (never executed)
25	Index Cond: ((lname)::text = 'Employee1'::text)
26	-> Bitmap Heap Scan on works_on (cost=0.34..4.35 rows=1 width=8) (never executed)
27	Recheck Cond: (essn = employee.ssn)
28	-> Bitmap Index Scan on wgin (cost=0.00..0.34 rows=1 width=0) (never executed)
29	Index Cond: (essn = employee.ssn)
30	Planning time: 0.561 ms
31	Execution time: 1.350 ms

EXPLAIN

Btree Index Seems to be the fastest index of them all with the least execution time and planning time and cos because the hash index using hashaggregate join which is slower in execution time and bitmap index also used the hash aggregate join as the query had two subplans both using nestloop join and both hash and bitmap had higher costs since the need of using seqscan that was turned off in all indexed scenarios to be able to test the indices created only with no other index or no other linear scan.

Execution time:0.683

Cost =102.34…169.46

QUERY#3

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

WITHOUT INDEX

Query History | Query Editor

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO OFF;
3  SET enable_indexonlyscan  TO OFF;
4  SET enable_seqscan  TO ON;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO OFF;
7  explain analyze
8  select lname, fname
9  from employee
10 where salary > all (
11 select salary
12 from employee
13 where dno=5 );
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..1181996.27 rows=2500 width=42) (actual time=1.283..5105.240 rows=4996 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 5
4	SubPlan 1
5	-> Seq Scan on employee employee_1 (cost=0.00..472.51 rows=1 width=4) (actual time=0.003..1.006 rows=1 loops=5001)
6	Filter: (dno = 5)
7	Rows Removed by Filter: 4995
8	Planning time: 0.108 ms
9	Execution time: 5106.006 ms

WITH HASH INDEX

For the hash index I created one index on table employee using column department number and after the flags and tuning the engine was forced to use index scan using the index Ehash but also used seqscan to select and retrieve all the data from the table even though the seqscan was turned off but it didn't affect the cost of the plan and lowered the exection time and planning time.

Query History | Query Editor

```
1  set enable_seqscan = off;
2  set enable_gathermerge= off;
3  set enable_hashagg = off;
4  set enable_hashjoin = off;
5  set enable_indexscan = on;
6  set enable_indexonlyscan = on;
7  set enable_material = off;
8  set enable_mergejoin = off;
```

```
13 create index Ehash ON employee Using hash(dno);
14 explain analyze select lname, fname
15 from employee
16 where salary > all (
17 select salary
```

Messages | Explain | Notifications | Data Output

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..20198.52 rows=2500 width=42) (actual time=0.038..5.347 rows=4996 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 5
4	SubPlan 1
5	-> Index Scan using ehash on employee employee_1 (cost=0.00..8.02 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=5001)
6	Index Cond: (dno = 5)
7	Planning time: 0.107 ms
8	Execution time: 5.481 ms

WITH BTREE INDEX

For the btree index I used the same table and same column just like the rest,but the exection time was higher as btree index is know to be slower than hash having o(log n).

Query History | Query Editor

```
1  set enable_seqscan = off;
2  set enable_gathermerge= off;
3  set enable_hashagg = off;
4  set enable_hashjoin = off;
5  set enable_indexscan = on;
6  set enable_indexonlyscan = on;
7  create index btree on employee using btree(dno);
8
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..21939.31 rows=2500 width=42) (actual time=0.053..11.383 rows=4996 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 5
4	SubPlan 1
5	-> Index Scan using btree on employee employee_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=5001)
6	Index Cond: (dno = 5)
7	Planning time: 0.163 ms
8	Execution time: 11.622 ms

WITH BITMAP INDEX

For the bitmap index we used one index on table employee column department number.The bitmap is the highest in execution time since it had to use seqscan to retrieve all the data in table employees and the bitmap index has a time complexity of O(n) which is slower than btree and hash.

```
1  SET enable_bitmapscan TO On;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO On;
5  SET enable_hashjoin   TO Off;
6  SET enable_mergejoin    TO OFF;
7
8
9  create extension btree_gin;
10 create index Ebitmap on employee using gin(dno);
11 explain analyze
12 select lname, fname
13 from employee
14 where salary > all (
15 select salary
```

Data Output | Messages | Explain | Notifications

	QUERY PLAN text
1	Seq Scan on employee (cost=9.26..201548.92 rows=2420 width=42) (actual time=63.009..63.058 rows=3 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 4836
4	SubPlan 1
5	-> Materialize (cost=9.26..92.09 rows=162 width=4) (actual time=0.000..0.005 rows=82 loops=4839)
6	-> Bitmap Heap Scan on employee employee_1 (cost=9.26..91.28 rows=162 width=4) (actual time=0.048..0.307 rows=162 loops=1)
7	Recheck Cond: (dno = 5)
8	Heap Blocks: exact=80
9	-> Bitmap Index Scan on ebitmap (cost=0.00..9.22 rows=162 width=0) (actual time=0.032..0.032 rows=162 loops=1)
10	Index Cond: (dno = 5)
11	Planning time: 0.247 ms
12	Execution time: 63.124 ms

EXPLAIN

The hash index was the optimal and best execution plan with least execution and planning time as well as cost because the engine used first seqscan in all cases and the only measurement in testing is index only scan and the actual execution time for the hash was the lowest as hashing does search in O(1) in searching for employees with department number equal to *5*.

QUERY#4

```
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 );
```

WITHOUT INDEX

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO OFF;
3  SET enable_indexonlyscan  TO OFF;
4  SET enable_seqscan  TO ON;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO OFF;
7
8  explain analyze
9  select e.fname, e.lname
10 from employee as e
11 where e.ssn in (
12 select essn
```

Notifications | Explain | Messages | Data Output

	QUERY PLAN text
1	Seq Scan on employee e (cost=0.00..40400.97 rows=2420 width=42) (actual time=0.042..79.882 rows=20 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 4819
4	SubPlan 1
5	-> Seq Scan on dependent d (cost=0.00..15.70 rows=376 width=4) (actual time=0.003..0.005 rows=10 loops=4839)
6	Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))
7	Rows Removed by Filter: 10
8	Planning time: 0.153 ms
9	Execution time: 79.911 ms

WITH BTREE INDEX

For the Btree index I tried to apply an index on all the columns that used seqscan in the without index scenario and try to force them to use the indices I created but whatever I do the planner seem to always use seqscan even when trying to turn it off it only increased the cost and execution time but didn't stop the planner from using it. As you see in the screenshot I had 3 indices for employee table and 2 for dependent table and they weren't used by the planner , leaving the cost and execution time almost the same as the 1st scenario with a slight decrease due to the engine efficiency.

```
1  SET enable_bitmapscan TO Off;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO On;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO OFF;
7
8  create index ebtree on employee using btree(ssn);
9   create index ebtree2 on employee using btree(fname);
10  create index ebtree3 on employee using btree(sex);
11 create index dbtree on dependent using btree(dependent_name);
12  create index dbtree2 on dependent using btree(sex);
13 explain analyze
14 select e.fname, e.lname
15 from employee as e
16 where e.ssn in (
17 select essn
18 from dependent as d
19 where e.fname != d.dependent_name
20 and
21 e.sex!=d.sex );
```

Notifications | Explain | Messages | Data Output

	QUERY PLAN text
1	Seq Scan on employee e (cost=0.00..3394.71 rows=2420 width=42) (actual time=0.072..76.074 rows=20 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 4819
4	SubPlan 1
5	-> Seq Scan on dependent d (cost=0.00..1.30 rows=18 width=4) (actual time=0.003..0.005 rows=10 loops=4839)
6	Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))
7	Rows Removed by Filter: 10
8	Planning time: 0.293 ms
9	Execution time: 76.130 ms

WITH HASH INDEX

The hash index was facing the same problem like the btree with just a slight change in the exection time but still using seqscan which happened because of the use of != as the planner has to retrieve the whole table (employee/dependent) to apply the not equal condition on the sex and firstname and turning the seqscan off only increased the cost but still the planner found it to be the optimal choice for the query

```
1  SET enable_bitmapscan TO Off;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO On;
5  SET enable_hashjoin   TO Off;
6  SET enable_mergejoin    TO ON;
7  --  create index ehash on employee using hash(ssn);
8  --  create index ehash2 on employee using hash(fname);
9  --  create index ehash3 on employee using hash(sex);
10 --  create index dhash on dependent using hash(dependent_name);
11 --  create index dhash2 on dependent using hash(sex);
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
```

Notifications | Explain | Messages | Data Output

	QUERY PLAN text
1	Seq Scan on employee e (cost=0.00..3394.71 rows=2420 width=42) (actual time=0.032..62.209 rows=20 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 4819
4	SubPlan 1
5	-> Seq Scan on dependent d (cost=0.00..1.30 rows=18 width=4) (actual time=0.002..0.004 rows=10 loops=4839)
6	Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))
7	Rows Removed by Filter: 10
8	Planning time: 0.169 ms
9	Execution time: 62.235 ms

WITH BITMAP INDEX

The same problem is faced with the bitmap index it was aways optimal to use the seqscan due to the behaviour of the != which demands a full scan of all the tuples no matter what index was used and turing it off will only increase the cost and execution time with no effect on the query planner output strategy.

```
1  SET enable_bitmapscan TO On;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO OFF;
4  SET enable_seqscan  TO On;
5  SET enable_hashjoin   TO ON;
6  SET enable_mergejoin    TO OFF;
7  create extension btree_gin;
8   create index ebitmap1 on employee using gin(ssn);
9  create index ebitmap2 on employee using gin(fname);
10 create index ebitmap3 on employee using gin(sex);
11 create index dbitmap on dependent using gin(sex);
12 create index dbitmap2 on dependent using gin(dependent_name);
13  explain analyze
14  select e.fname, e.lname
15 from employee as e
16 where e.ssn in (
```

Data Output | Messages | Explain | Notifications

	QUERY PLAN text
1	Seq Scan on employee e (cost=0.00..3374.71 rows=2420 width=116) (actual time=0.032..39.213 rows=20 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 4819
4	SubPlan 1
5	-> Seq Scan on dependent d (cost=0.00..1.30 rows=18 width=4) (actual time=0.001..0.002 rows=10 loops=4839)
6	Filter: (((e.fname)::text <> (dependent_name)::text) AND ((e.sex)::text <> (sex)::text))
7	Rows Removed by Filter: 10
8	Planning time: 1.034 ms
9	Execution time: 39.241 ms

EXPLAIN

Although that invoking multiple indices on all scenarios the optimal choice for the engine appears to always be seqscan since the query is joining on != so the engine had to go through both tables employee and dependent linearly to retrieve the employees with different first names and dependent names and different sex.It yet appears that the Bitmap gave the least execution time which maybe resulted from the engine or the processor itself.

QUERY#5

```
select fname, lname
from employee
where exists ( select *
from dependent
where ssn=essn );
```

WITHOUT INDEX

Query History | Query Editor

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO OFF;
3  SET enable_indexonlyscan  TO OFF;
4  SET enable_seqscan  TO ON;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO OFF;
7
8  explain analyze
9  select fname, lname
10 from employee
11 where exists ( select *
12                          from dependent
13                         where ssn=essn );
14
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	Nested Loop Semi Join (cost=0.00..27602.94 rows=19 width=42) (actual time=0.073..72.556 rows=19 loops=1)
2	Join Filter: (employee.ssn = dependent.essn)
3	Rows Removed by Join Filter: 94829
4	-> Seq Scan on employee (cost=0.00..460.01 rows=5001 width=46) (actual time=0.028..0.757 rows=5001 loops=1)
5	-> Seq Scan on dependent (cost=0.00..5.19 rows=19 width=4) (actual time=0.001..0.002 rows=19 loops=5001)
6	Planning time: 0.995 ms
7	Execution time: 72.594 ms

WITH BTREE INDEX

For the Btree index, I created 2 indices one on table employee ***ssn*** column and another on dependent ***essn*** column which the planner used along with a nestloop join strategy using index scan on the employee table and on dependent table which caused the drastic drop in execution time with a slight increase in cost.

Query History | Query Editor

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO On;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO OFF;
7  create index Ebtree on employee using btree(ssn);
8  create index Dbtree on dependent using btree(essn);
9  explain analyze
10 select fname, lname
11 from employee
12 where exists ( select *
13                          from dependent
14                         where ssn=essn );
15
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	Nested Loop Semi Join (cost=0.42..1335.64 rows=19 width=42) (actual time=0.174..9.555 rows=19 loops=1)
2	-> Index Scan using ebtree on employee (cost=0.28..548.30 rows=5001 width=46) (actual time=0.049..2.345 rows=5001 loops=1)
3	-> Index Only Scan using dbtree on dependent (cost=0.14..0.16 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=5001)
4	Index Cond: (essn = employee.ssn)
5	Heap Fetches: 19
6	Planning time: 1.273 ms
7	Execution time: 9.632 ms

WITH HASH INDEX

With hash index the cost rose high since it needs seqscan search and I disabled it along with a nestloop join strategy and index scan on dependent using the hash index I created ,if I tried to enable the seqscan the indices weren't used since it's the most efficient search that the planner optimally decides to choose.

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO Off;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO ON;
6  set enable_hashagg to off;
7  SET enable_mergejoin    TO OFF;
8  set enable_nestloop to on;
9
10  ALTER TABLE dependent DROP CONSTRAINT dependent_pkey CASCADE;
11  create index Ehash on employee using hash(ssn);
12  create index Dhash on dependent using hash(essn);
13 explain analyze
14 select fname, lname
15 from employee
16 where exists ( select *
17                          from dependent
18                         where ssn=essn );
19
```

Notifications | Messages | Explain | Data Output

	QUERY PLAN text
1	Nested Loop Semi Join (cost=10000000000.00..10000000567.72 rows=19 width=42) (actual time=0.036..3.044 rows=19 loops=1)
2	-> Seq Scan on employee (cost=10000000000.00..10000000460.01 rows=5001 width=46) (actual time=0.023..0.560 rows=5001 loops=1)
3	-> Index Scan using dhash on dependent (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=5001)
4	Index Cond: (essn = employee.ssn)
5	Planning time: 0.168 ms
6	Execution time: 3.070 ms

WITH BITMAP INDEX

The bitmap index was having the same problem like the hash of closing the seqscan so I kept it turned off to conceive consistency across all scenarios and also the nestloop join was using sort on dependent column to use the seqscan.

```
1  SET enable_bitmapscan TO On;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO Off;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO Off;
6  set enable_hashagg to off;
7  SET enable_mergejoin    TO OFF;
8  set enable_nestloop to on;
9  create index Egin on employee using gin(ssn);
10 create index Dgin on dependent using gin(essn);
11
```

Data Output | Messages | Notifications | Explain

	QUERY PLAN text
1	Nested Loop (cost=10000000006.84..10000000186.34 rows=20 width=116) (actual time=0.080..0.344 rows=20 loops=1)
2	-> Unique (cost=10000000001.63..10000000001.73 rows=20 width=4) (actual time=0.041..0.048 rows=20 loops=1)
3	-> Sort (cost=10000000001.63..10000000001.68 rows=20 width=4) (actual time=0.040..0.042 rows=20 loops=1)
4	Sort Key: dependent.essn
5	Sort Method: quicksort Memory: 25kB
6	-> Seq Scan on dependent (cost=10000000000.00..10000000001.20 rows=20 width=4) (actual time=0.023..0.026 rows=20 loops=1)
7	-> Bitmap Heap Scan on employee (cost=5.21..9.22 rows=1 width=120) (actual time=0.004..0.004 rows=1 loops=20)
8	Recheck Cond: (ssn = dependent.essn)
9	Heap Blocks: exact=20
10	-> Bitmap Index Scan on egin (cost=0.00..5.21 rows=1 width=0) (actual time=0.003..0.003 rows=1 loops=20)
11	Index Cond: (ssn = dependent.essn)
12	Planning time: 0.336 ms
13	Execution time: 0.413 ms

EXPLAIN

The bitmap was the most optimal with least execution /planning time while the cost was high it could be lowered by keeping the seqscan on it would still not affect the usage of the indices already created.The join strategy was a full nestloop where The right relation is scanned once using unique sort on employee ssn column for every row found in the left relation using a bitmap scan once using employee ssn and another bitmap scan using dependent essn column.

12	Planning time: 0.336 ms
13	Execution time: 0.413 ms

QUERY#6

```
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40
and
dno IN (
  select dno
  from employee
  group by dno
  having count (*) > 2)
group by dnumber;
```

WITHOUT INDEX

```
SET enable_bitmapscan TO OFF;
SET enable_indexscan TO OFF;
SET enable_indexonlyscan  TO OFF;
SET enable_seqscan  TO ON;
SET enable_hashjoin   TO OFF;
SET enable_mergejoin    TO OFF;
```

Messages | Explain | Data Output

	QUERY PLAN text
1	GroupAggregate (cost=3096.44..3098.35 rows=109 width=12) (actual time=14.616..15.238 rows=30 loops=1)
2	Group Key: department.dnumber
3	-> Sort (cost=3096.44..3096.71 rows=109 width=4) (actual time=14.584..14.839 rows=4839 loops=1)
4	Sort Key: department.dnumber
5	Sort Method: quicksort Memory: 419kB
6	-> Nested Loop (cost=152.58..3092.75 rows=109 width=4) (actual time=1.333..13.860 rows=4839 loops=1)
7	Join Filter: (department.dnumber = employee.dno)
8	Rows Removed by Join Filter: 140331
9	-> Seq Scan on employee (cost=0.00..140.49 rows=4839 width=4) (actual time=0.015..0.784 rows=4839 loops=1)
10	Filter: (salary > 40)
11	-> Materialize (cost=152.58..774.79 rows=30 width=8) (actual time=0.000..0.001 rows=30 loops=4839)
12	-> Nested Loop (cost=152.58..774.64 rows=30 width=8) (actual time=1.315..1.355 rows=30 loops=1)
13	Join Filter: (department.dnumber = employee_1.dno)
14	Rows Removed by Join Filter: 435
15	-> Seq Scan on department (cost=0.00..23.30 rows=1330 width=4) (actual time=0.009..0.010 rows=30 loops=1)
16	-> Materialize (cost=152.58..153.34 rows=30 width=4) (actual time=0.043..0.044 rows=16 loops=30)
17	-> HashAggregate (cost=152.58..152.88 rows=30 width=4) (actual time=1.297..1.300 rows=30 loops=1)
18	Group Key: employee_1.dno
19	Filter: (count(*) > 2)
20	-> Seq Scan on employee employee_1 (cost=0.00..128.39 rows=4839 width=4) (actual time=0.008..0.417 rows=4839 loops=1)
21	Planning time: 0.270 ms
22	Execution time: 15.384 ms

WITH BTREE INDEX

For the btree index we created 2 indices one on table employee and the other on table department and since the seqscan is turned off so thee engine used only index scan and group aggregate strategy so that gives the least execution and planning time as well as cost.

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO OFF;
6  SET enable_mergejoin    TO Off;
7  set enable_nestloop to on;
8  create index Ebtree on employee using btree(dno);
9   create index Dbtree on department using btree(dnumber);
10 explain analyze
11 select dnumber, count(*)
```

Messages Explain Data Output

	QUERY PLAN text
1	GroupAggregate (cost=0.70..1059.39 rows=30 width=12) (actual time=0.386..4.432 rows=30 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=0.70..1034.89 rows=4839 width=4) (actual time=0.144..4.051 rows=4839 loops=1)
4	-> Nested Loop (cost=0.42..497.63 rows=30 width=8) (actual time=0.136..2.059 rows=30 loops=1)
5	-> GroupAggregate (cost=0.28..481.07 rows=30 width=4) (actual time=0.128..1.987 rows=30 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 2)
8	-> Index Only Scan using ebtree on employee employee_1 (cost=0.28..456.57 rows=4839 width=4) (actual time=0.017..1.576 rows=4839 loops=1)
9	Heap Fetches: 4839
10	-> Index Only Scan using dbtree on department (cost=0.14..0.55 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=30)
11	Index Cond: (dnumber = employee_1.dno)
12	Heap Fetches: 30
13	-> Index Scan using ebtree on employee (cost=0.28..16.30 rows=161 width=4) (actual time=0.004..0.056 rows=161 loops=30)
14	Index Cond: (dno = department.dnumber)
15	Filter: (salary > 40)
16	Planning time: 0.265 ms
17	Execution time: 4.491 ms

WITH HASH INDEX

For the hash index I created 2 indices one on employee and another on hash using the columns that where seqscanned in scenario1.The hash index seems to be higher in cost and execution time than the other scenarios due to using seqscan search on employee even though it was turned off but the query planner seems to use it as it's the optimal choice.

```
1  SET enable_bitmapscan TO OFF;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO Off;
5  SET enable_hashjoin   TO Off;
6  SET enable_mergejoin    TO Off;
7  create index Ehash on employee using hash(dno);
8  create index Dhash on department using hash(dnumber);
9  explain analyze
10 select dnumber, count(*)
11 from department, employee
12 where dnumber=dno
13 and
```

Data Output | Messages | Explain | Notifications

	QUERY PLAN text
1	HashAggregate (cost=10000000735.10..10000000735.40 rows=30 width=12) (actual time=6.965..6.970 rows=30 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=10000000152.58..10000000710.91 rows=4839 width=4) (actual time=2.233..5.874 rows=4839 loops=1)
4	-> Nested Loop (cost=10000000152.58..10000000174.01 rows=30 width=8) (actual time=2.228..2.296 rows=30 loops=1)
5	-> HashAggregate (cost=10000000152.58..10000000152.88 rows=30 width=4) (actual time=2.219..2.228 rows=30 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 2)
8	-> Seq Scan on employee employee_1 (cost=10000000000.00..10000000128.39 rows=4839 width=4) (actual time=0.025..0.629 rows=4839 loops=1)
9	-> Index Scan using dhash on department (cost=0.00..0.68 rows=1 width=4) (actual time=0.001..0.002 rows=1 loops=30)
10	Index Cond: (dnumber = employee_1.dno)
11	-> Index Scan using ehash on employee (cost=0.00..16.29 rows=161 width=4) (actual time=0.002..0.103 rows=161 loops=30)
12	Index Cond: (dno = department.dnumber)
13	Filter: (salary > 40)
14	Planning time: 0.359 ms
15	Execution time: 7.036 ms

WITH BITMAP INDEX

For the bitmap index it also needed seqscan as well but it was turned off for all scenarios so that caused the cost to have this drastic rise as in the nestloop after sorting the employee column by department number it used seqscan on employee but still the execution time is the least so far.

```
1  SET enable_bitmapscan TO On;
2  SET enable_indexscan TO ON;
3  SET enable_indexonlyscan  TO On;
4  SET enable_seqscan  TO off;
5  SET enable_hashjoin   TO Off;
6  SET enable_mergejoin    TO OFF;
7  set enable_nestloop to on;
8  create index egin on employee using gin(dno);
9  create index dgin on department using gin(dnumber);
```

Data Output | Messages | Notifications | Explain

	QUERY PLAN text
1	GroupAggregate (cost=10000000406.34..10000000968.23 rows=30 width=12) (actual time=1.417..4.235 rows=30 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=10000000406.34..10000000943.74 rows=4839 width=4) (actual time=1.297..3.920 rows=4839 loops=1)
4	-> Nested Loop (cost=10000000404.82..10000000570.35 rows=30 width=8) (actual time=1.266..1.960 rows=30 loops=1)
5	-> GroupAggregate (cost=10000000404.55..10000000441.14 rows=30 width=4) (actual time=1.250..1.729 rows=30 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 2)
8	-> Sort (cost=10000000404.55..10000000416.65 rows=4839 width=4) (actual time=1.224..1.412 rows=4839 loops=1)
9	Sort Key: employee_1.dno
10	Sort Method: quicksort Memory: 419kB
11	-> Seq Scan on employee employee_1 (cost=10000000000.00..10000000108.39 rows=4839 width=4) (actual time=0.011..0.611 rows=4839 loops=1)
12	-> Bitmap Heap Scan on department (cost=0.27..4.29 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=30)
13	Recheck Cond: (dnumber = employee_1.dno)
14	Heap Blocks: exact=30
15	-> Bitmap Index Scan on dgin (cost=0.00..0.27 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=30)
16	Index Cond: (dnumber = employee_1.dno)
17	-> Bitmap Heap Scan on employee (cost=1.52..10.84 rows=161 width=4) (actual time=0.015..0.052 rows=161 loops=30)
18	Recheck Cond: (dno = department.dnumber)
19	Filter: (salary > 40)
20	Heap Blocks: exact=1800
21	-> Bitmap Index Scan on egin (cost=0.00..1.48 rows=161 width=0) (actual time=0.012..0.012 rows=161 loops=30)
22	Index Cond: (dno = department.dnumber)
23	Planning time: 0.182 ms
24	Execution time: 4.407 ms

EXPLAIN

The most optimal scenario would be Bitmap index on table employee column dno and on department table on dnumber column having the least execution and planning time but a high cost which can be lowered by allowing seqscan on but in this case only 1 index will be used on employee and the other one will be seqscan but with a low cost so it's a trade off.