Schema 2:

Description of the data inserted:

1- Employee table: Total of 5000 employees , 99 of them with lname = ‘emplyee1’, the first 2450 employees with salary 60000 and around 510 of the others with salary 50000 and the 5000 employees are distributed equally among the 30 department (each department has around 166 employees working in it).

2- Department table: Total of 30 department, the manager of department 1 is employee with ssn 1 and for other departments the manager is chosen randomly.

3- Department_locations table: 20 department locations

4- Projects and works_on tables: 600 projects distributed among the 30 departments (20 project for each department) and each project has exactly two employees works on it so we have 1200 works_on records

5- Dependent table: 1000 dependent (1000 employees only have dependents and each one has exactly one dependant)

Description of changes to the SQL and justification for that:

Dropping all primary key constraints after populating the data set (to guarantee data integrity) to remove any automatically created index on the primary to be able to analyze the no index scenario

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' );
```

First scenario: without any index (only seq scans)

Changes in flags: No flags are changed (The default flag setting)

The query plan and the actual measurements (The output of explain analyze):

```
"HashAggregate  (cost=366.33..370.83 rows=450 width=4) (actual time=287.927..288.038
rows=600 loops=1)"

"  Group Key: project.pnumber"

"  ->  Seq Scan on project  (cost=347.20..365.20 rows=450 width=4) (actual
time=287.296..287.617 rows=600 loops=1)"

"        Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"

"        SubPlan 1"

"          ->  Nested Loop  (cost=1.75..168.00 rows=12 width=4) (actual time=114.014..272.268
rows=600 loops=1)"

"                ->  Hash Join  (cost=1.75..147.00 rows=1 width=0) (actual time=113.943..271.786
rows=1 loops=1)"

"                      Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"

"                      ->  Seq Scan on employee e  (cost=0.00..144.50 rows=99 width=8) (actual
time=51.602..209.406 rows=99 loops=1)"

"                            Filter: (lname = 'employee1'::bpchar)"

"                            Rows Removed by Filter: 4901"

"                      ->  Hash  (cost=1.30..1.30 rows=30 width=8) (actual time=62.290..62.297 rows=30
loops=1)"

"                            Buckets: 1024  Batches: 1  Memory Usage: 10kB"

"                            ->  Seq Scan on department d  (cost=0.00..1.30 rows=30 width=8) (actual
time=62.243..62.252 rows=30 loops=1)"

"                ->  Seq Scan on project project_1  (cost=0.00..15.00 rows=600 width=4) (actual
time=0.067..0.363 rows=600 loops=1)"

"        SubPlan 2"

"          ->  Hash Join  (cost=34.00..179.11 rows=24 width=4) (never executed)"
"              Hash Cond: (employee.ssn = works_on.essn)"
"              ->  Seq Scan on employee  (cost=0.00..144.50 rows=99 width=4) (never executed)"
"                    Filter: (lname = 'employee1'::bpchar)"
"              ->  Hash  (cost=19.00..19.00 rows=1200 width=8) (never executed)"
"                    ->  Seq Scan on works_on  (cost=0.00..19.00 rows=1200 width=8) (never
executed)"
"Planning Time: 202.015 ms"
"Execution Time: 288.450 ms"
```

Second scenario: with BTree index: Creating a B+ tree on column employee.lname so the filter condtition is now faster as the planner uses the index to get all employees with lname = ‘employee1’ instead of seq scan on employee table and then filter

Changes in flags: No change

The query plan and actual measurements(The output of explain analyze):

```
"HashAggregate  (cost=262.43..266.93 rows=450 width=4) (actual time=52.125..52.262
rows=600 loops=1)"
"  Group Key: project.pnumber"
"  ->  Seq Scan on project  (cost=243.31..261.31 rows=450 width=4) (actual time=50.777..51.212
rows=600 loops=1)"
"        Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"        SubPlan 1"
"          ->  Nested Loop  (cost=6.80..116.06 rows=12 width=4) (actual time=27.459..28.241
rows=600 loops=1)"
"              ->  Hash Join  (cost=6.80..95.06 rows=1 width=0) (actual time=27.428..27.496 rows=1
loops=1)"
"                    Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
"                    ->  Bitmap Heap Scan on employee e  (cost=5.05..92.55 rows=99 width=8) (actual
time=27.259..27.283 rows=99 loops=1)"
"                          Recheck Cond: (lname = 'employee1'::bpchar)"
"                          Heap Blocks: exact=2"
"                    -> Bitmap Index Scan on employee_lname_btree  (cost=0.00..5.02 rows=99 width=0) (actual time=27.239..27.239 rows=99 loops=1)"
"                          Index Cond: (lname = 'employee1'::bpchar)"
"                -> Hash  (cost=1.30..1.30 rows=30 width=8) (actual time=0.078..0.083 rows=30 loops=1)"
"                      Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                      -> Seq Scan on department d  (cost=0.00..1.30 rows=30 width=8) (actual time=0.042..0.049 rows=30 loops=1)"
"            -> Seq Scan on project project_1  (cost=0.00..15.00 rows=600 width=4) (actual time=0.025..0.620 rows=600 loops=1)"
"      SubPlan 2"
"        -> Hash Join  (cost=39.05..127.16 rows=24 width=4) (never executed)"
"            Hash Cond: (employee.ssn = works_on.essn)"
"            -> Bitmap Heap Scan on employee  (cost=5.05..92.55 rows=99 width=4) (never executed)"
"                Recheck Cond: (lname = 'employee1'::bpchar)"
"                -> Bitmap Index Scan on employee_lname_btree  (cost=0.00..5.02 rows=99 width=0) (never executed)"
"                    Index Cond: (lname = 'employee1'::bpchar)"
"            -> Hash  (cost=19.00..19.00 rows=1200 width=8) (never executed)"
"                -> Seq Scan on works_on  (cost=0.00..19.00 rows=1200 width=8) (never executed)"
"Planning Time: 111.297 ms"
"Execution Time: 127.517 ms"
```

Third scenario: with bitmap index (GIN): on employee.lname to retrieve emoloyees with lname = ‘employee1’ faster

Changes in flags: No changes

The query plane and actual measurements(The explain analyze output):

```
"HashAggregate  (cost=269.87..274.37 rows=450 width=4) (actual time=101.173..101.281
rows=600 loops=1)"
"  Group Key: project.pnumber"
"  ->  Seq Scan on project  (cost=250.74..268.74 rows=450 width=4) (actual
time=100.467..100.816 rows=600 loops=1)"
"        Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"        SubPlan 1"
"          ->  Nested Loop  (cost=10.52..119.77 rows=12 width=4) (actual time=0.159..0.485
rows=600 loops=1)"
"                ->  Hash Join  (cost=10.52..98.77 rows=1 width=0) (actual time=0.141..0.215 rows=1
loops=1)"
"                      Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
"                      ->  Bitmap Heap Scan on employee e  (cost=8.77..96.27 rows=99 width=8) (actual
time=0.080..0.098 rows=99 loops=1)"
"                            Recheck Cond: (lname = 'employee1'::bpchar)"
"                            Heap Blocks: exact=2"
"                            ->  Bitmap Index Scan on employee_lname_gin  (cost=0.00..8.74 rows=99
width=0) (actual time=0.068..0.068 rows=99 loops=1)"
"                                  Index Cond: (lname = 'employee1'::bpchar)"
"                      ->  Hash  (cost=1.30..1.30 rows=30 width=8) (actual time=0.041..0.042 rows=30
loops=1)"
"                            Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                            ->  Seq Scan on department d  (cost=0.00..1.30 rows=30 width=8) (actual
time=0.014..0.021 rows=30 loops=1)"
"                ->  Seq Scan on project project_1  (cost=0.00..15.00 rows=600 width=4) (actual
time=0.015..0.162 rows=600 loops=1)"
"        SubPlan 2"
"          ->  Hash Join  (cost=42.77..130.88 rows=24 width=4) (never executed)"
"                Hash Cond: (employee.ssn = works_on.essn)"
"               ->  Bitmap Heap Scan on employee  (cost=8.77..96.27 rows=99 width=4) (never
executed)"
"                     Recheck Cond: (lname = 'employee1'::bpchar)"
"                     ->  Bitmap Index Scan on employee_lname_gin  (cost=0.00..8.74 rows=99 width=0)
(never executed)"
"                           Index Cond: (lname = 'employee1'::bpchar)"
"               ->  Hash  (cost=19.00..19.00 rows=1200 width=8) (never executed)"
"                     ->  Seq Scan on works_on  (cost=0.00..19.00 rows=1200 width=8) (never
executed)"
"Planning Time: 4.292 ms"
"Execution Time: 101.591 ms"
```

With hash index:

```
Seq Scan on employee  (cost=0.00..364507.00 rows=2500 width=42) (actual time=1.799..68.552 rows=2450 loops=1)
```

Fourth scenario: with hash: on employee.lname to retrieve emoloyees with lname = ‘employee1’ faster

Changes in flags: No changes

The query plane and actual measurements(The explain analyze output):

```
"HashAggregate  (cost=261.87..266.37 rows=450 width=4) (actual time=1.437..1.561 rows=600
loops=1)"
"  Group Key: project.pnumber"
"  ->  Seq Scan on project  (cost=242.74..260.74 rows=450 width=4) (actual time=0.728..1.097
rows=600 loops=1)"
"        Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))"
"        SubPlan 1"
"          ->  Nested Loop  (cost=6.52..115.77 rows=12 width=4) (actual time=0.127..0.424
rows=600 loops=1)"
"               ->  Hash Join  (cost=6.52..94.77 rows=1 width=0) (actual time=0.111..0.185 rows=1
loops=1)"
"                     Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))"
"                ->  Bitmap Heap Scan on employee e  (cost=4.77..92.27 rows=99 width=8) (actual time=0.052..0.089 rows=99 loops=1)"
"                      Recheck Cond: (lname = 'employee1'::bpchar)"
"                      Heap Blocks: exact=2"
"                      ->  Bitmap Index Scan on employee_lname_hash  (cost=0.00..4.74 rows=99 width=0) (actual time=0.037..0.038 rows=99 loops=1)"
"                            Index Cond: (lname = 'employee1'::bpchar)"
"                ->  Hash  (cost=1.30..1.30 rows=30 width=8) (actual time=0.042..0.043 rows=30 loops=1)"
"                      Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                      ->  Seq Scan on department d  (cost=0.00..1.30 rows=30 width=8) (actual time=0.014..0.023 rows=30 loops=1)"
"          ->  Seq Scan on project project_1  (cost=0.00..15.00 rows=600 width=4) (actual time=0.013..0.139 rows=600 loops=1)"
"  SubPlan 2"
"    ->  Hash Join  (cost=38.77..126.88 rows=24 width=4) (never executed)"
"          Hash Cond: (employee.ssn = works_on.essn)"
"          ->  Bitmap Heap Scan on employee  (cost=4.77..92.27 rows=99 width=4) (never executed)"
"                Recheck Cond: (lname = 'employee1'::bpchar)"
"                ->  Bitmap Index Scan on employee_lname_hash  (cost=0.00..4.74 rows=99 width=0) (never executed)"
"                      Index Cond: (lname = 'employee1'::bpchar)"
"          ->  Hash  (cost=19.00..19.00 rows=1200 width=8) (never executed)"
"                ->  Seq Scan on works_on  (cost=0.00..19.00 rows=1200 width=8) (never executed)"
"Planning Time: 3.466 ms"
"Execution Time: 1.842 ms"
```

Illustrating what the query retrieves:

If there is an employee with last name "employee1" and works in a department as a manager then the first subquery will give all the projects in the database as output otherwise the first subquery will give an empty bag (0 rows) (In my opinion this behavior is the main reason for the cost over-estimation by the planner as when actually executing the query there are a lot of subqueries that are never executed)

The best scenario in performance and justification for that:

The hash index scenario is the one with the least total estimated cost.

That’s because creating hash index on column lname in employee table makes the retrieving of employees with lname = ‘employee1’ very fast O(1) and faster than seq scan on the table and then filter and also faster than any other index(either btree or bitmap)

____

Query 3:

```
select lname, fname

from employee

where salary > all (

select salary

from employee

where dno=5 );
```

First scenario: without any index (only seq scans)

Changes in flag : No flags are changed

The query plan and the actual measurements:

```
"Seq Scan on employee  (cost=0.00..364507.00 rows=2500 width=42) (actual time=1.799..68.552 rows=2450 loops=1)"

"  Filter: (SubPlan 1)"

"  Rows Removed by Filter: 2550"

"  SubPlan 1"
"    ->  Materialize  (cost=0.00..145.33 rows=166 width=4) (actual time=0.000..0.005 rows=82 loops=5000)"
"          ->  Seq Scan on employee employee_1  (cost=0.00..144.50 rows=166 width=4) (actual time=0.168..1.694 rows=166 loops=1)"
"                Filter: (dno = 5)"
"                Rows Removed by Filter: 4834"
"Planning Time: 0.135 ms"
"Execution Time: 68.716 ms"
```

Second scenario: with btree index: creating btree index on employee.dno to speed up the retrieval for employees in department number 5

Changes in flags: No flags are changed

The query plane and actual measurements(The explain analyze output):

```
"Seq Scan on employee  (cost=0.28..35519.78 rows=2500 width=42) (actual time=0.251..57.398 rows=2450 loops=1)"
"  Filter: (SubPlan 1)"
"  Rows Removed by Filter: 2550"
"  SubPlan 1"
"    ->  Materialize  (cost=0.28..14.02 rows=166 width=4) (actual time=0.000..0.004 rows=82 loops=5000)"
"          ->  Index Scan using employee_dno_btree on employee employee_1  (cost=0.28..13.19 rows=166 width=4) (actual time=0.074..0.143 rows=166 loops=1)"
"                Index Cond: (dno = 5)"
"Planning Time: 128.087 ms"
"Execution Time: 57.566 ms"
```

Third scenario: with bitmap index(GIN): creating bitmap index on employee.dno to speed up the retrieval for employees in department number 5

Changes in flags: No change

The query plane and actual measurements (The explain analyze output):

```
"Seq Scan on employee  (cost=9.29..213453.79 rows=2500 width=42) (actual time=5.684..69.503 rows=2450 loops=1)"
"  Filter: (SubPlan 1)"
"  Rows Removed by Filter: 2550"
"  SubPlan 1"
"    ->  Materialize  (cost=9.29..94.19 rows=166 width=4) (actual time=0.001..0.006 rows=82 loops=5000)"
"          ->  Bitmap Heap Scan on employee employee_1  (cost=9.29..93.36 rows=166 width=4) (actual time=5.560..5.602 rows=166 loops=1)"
"                Recheck Cond: (dno = 5)"
"                Heap Blocks: exact=4"
"                ->  Bitmap Index Scan on employee_dno_gin  (cost=0.00..9.24 rows=166 width=0) (actual time=5.550..5.550 rows=166 loops=1)"
"                      Index Cond: (dno = 5)"
"Planning Time: 53.786 ms"
"Execution Time: 103.473 ms"
```

Fourth scenario: with hash index: creating hash index on employee.dno to speed up the retrieval for employees in department number 5

Changes in flags: No changes in flags

The query plan and actual measurements (The output of explain analyze):

```
"Seq Scan on employee  (cost=9.29..213453.79 rows=2500 width=42) (actual time=0.255..79.846 rows=2450 loops=1)"
"  Filter: (SubPlan 1)"
"  Rows Removed by Filter: 2550"
"  SubPlan 1"
"    ->  Materialize  (cost=9.29..94.19 rows=166 width=4) (actual time=0.000..0.005 rows=82 loops=5000)"
"        ->  Bitmap Heap Scan on employee employee_1  (cost=9.29..93.36 rows=166 width=4) (actual time=0.053..0.134 rows=166 loops=1)"
"              Recheck Cond: (dno = 5)"
"              Heap Blocks: exact=4"
"              ->  Bitmap Index Scan on employee_dno_hash  (cost=0.00..9.24 rows=166 width=0) (actual time=0.037..0.038 rows=166 loops=1)"
"                    Index Cond: (dno = 5)"
"Planning Time: 0.285 ms"
"Execution Time: 85.888 ms"
```

The best scenario in performance and justification:

The bitmap index scenario is the one with the least total estimated cost

That's because creating bitmap index on column dno in table employee makes retrieving the employees in department 5 faster than the other scenarios like:seq scan on the table or creating either btree or bitmap indicies.

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 );
```

First scenario: without any index (only seq scans)

Changes in flags: No change

The query plan and actual measurements (The output of explain analyze):

```
"Seq Scan on employee e  (cost=0.00..68269.50 rows=2500 width=42) (actual time=2.251..2374.829 rows=1000 loops=1)"

"  Filter: (SubPlan 1)"

"  Rows Removed by Filter: 4000"

"  SubPlan 1"

"    ->  Seq Scan on dependent d  (cost=0.00..26.00 rows=500 width=4) (actual time=0.111..0.414 rows=450 loops=5000)"

"          Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))"

"          Rows Removed by Filter: 450"

"Planning Time: 0.182 ms"

"Execution Time: 2375.431 ms"
```

Second scenarion: with btree: on dependent_name column in dependant table

```
"Seq Scan on employee e  (cost=0.00..68269.50 rows=2500 width=42) (actual time=1.713..1919.204 rows=1000 loops=1)"

"  Filter: (SubPlan 1)"

"  Rows Removed by Filter: 4000"

"  SubPlan 1"

"    ->  Seq Scan on dependent d  (cost=0.00..26.00 rows=500 width=4) (actual time=0.090..0.336 rows=450 loops=5000)"

"          Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))"

"          Rows Removed by Filter: 450"

"Planning Time: 1.878 ms"

"Execution Time: 1919.812 ms"
```

Although there is a btree index on dpendent name in dependant table, the planner chooses to do seq scan on the table instead of using the index as the btree index will be worse with the (!=) operator as the whole table should be scanned so using the index will only add an overhead

Third scenario: with bitmap: dependent_name column in dependant table

```
"Seq Scan on employee e  (cost=0.00..68269.50 rows=2500 width=42) (actual time=2.083..1877.151 rows=1000 loops=1)"
"  Filter: (SubPlan 1)"
"  Rows Removed by Filter: 4000"
"  SubPlan 1"
"    ->  Seq Scan on dependent d  (cost=0.00..26.00 rows=500 width=4) (actual time=0.087..0.329 rows=450 loops=5000)"
"          Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))"
"          Rows Removed by Filter: 450"
"Planning Time: 1.811 ms"
"Execution Time: 1877.589 ms"
```

Although there is a bitmap index on dependent name in dependent table, the planner chooses to do seq scan on the table instead of using the index as the bitmap index will be worse with the (!=) operator as the whole table should be scanned so using the index will only add an overhead

Fourth scenario: with hash: dependent_name column in dependant table

```
"Seq Scan on employee e  (cost=0.00..68269.50 rows=2500 width=42) (actual time=2.386..2160.876 rows=1000 loops=1)"
"  Filter: (SubPlan 1)"
"  Rows Removed by Filter: 4000"
"  SubPlan 1"
"    ->  Seq Scan on dependent d  (cost=0.00..26.00 rows=500 width=4) (actual time=0.104..0.376 rows=450 loops=5000)"
"          Filter: ((e.fname <> dependent_name) AND (e.sex <> sex))"
"        Rows Removed by Filter: 450"
"Planning Time: 2.490 ms"
"Execution Time: 2161.415 ms"
```

Although there is a hash index on dependent name in dependent table, the planner chooses to do seq scan on the table instead of using the index as the hash index will be worse with the (!=) operator as the whole table should be scanned so using the index will only add an overhead

The best scenario in performance and justification:

The no index scenario is the one with the least total estimated cost

That's because filtering with the (!=) operator is best done with sequential scan and without any index

---

Query 5

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

First scenario: without any index (only seq scans)

Changes in flags : No flags are changed

The query plan and the actual measurements (The output of the explain analyze):

```
"Hash Semi Join  (cost=33.50..189.75 rows=1000 width=42) (actual time=0.345..1.816 rows=1000 loops=1)"
"  Hash Cond: (employee.ssn = dependent.essn)"
"  ->  Seq Scan on employee  (cost=0.00..132.00 rows=5000 width=46) (actual time=0.021..0.631 rows=5000 loops=1)"
```

" -> Hash (cost=21.00..21.00 rows=1000 width=4) (actual time=0.306..0.306 rows=1000 loops=1)"

"   Buckets: 1024 Batches: 1 Memory Usage: 44kB"

"   -> Seq Scan on dependent (cost=0.00..21.00 rows=1000 width=4) (actual time=0.011..0.158 rows=1000 loops=1)"

"Planning Time: 0.179 ms"

"Execution Time: 1.887 ms"

Second scenario: with btree index: on employee.ssn and dependent.essn

Changes in flags: No flags changed

The query plan and actual measurements (The output of explain analyze):

"Merge Semi Join (cost=0.75..298.56 rows=1000 width=42) (actual time=0.043..3.987 rows=1000 loops=1)"

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

" -> Index Scan using employee_ssn_btree on employee (cost=0.28..224.28 rows=5000 width=46) (actual time=0.015..2.021 rows=5000 loops=1)"

" -> Index Only Scan using dependent_essn_btree on dependent (cost=0.28..49.27 rows=1000 width=4) (actual time=0.016..0.692 rows=1000 loops=1)"

"   Heap Fetches: 1000"

"Planning Time: 0.502 ms"

"Execution Time: 4.093 ms"

Third scenario: with bitmap: on dependent.ssn

"Hash Semi Join (cost=33.50..189.75 rows=1000 width=42) (actual time=0.326..1.971 rows=1000 loops=1)"

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

" -> Seq Scan on employee (cost=0.00..132.00 rows=5000 width=46) (actual time=0.021..0.700 rows=5000 loops=1)"

```
"  ->  Hash  (cost=21.00..21.00 rows=1000 width=4) (actual time=0.289..0.289 rows=1000 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 44kB"
"        ->  Seq Scan on dependent  (cost=0.00..21.00 rows=1000 width=4) (actual time=0.012..0.142 rows=1000 loops=1)"
"Planning Time: 0.261 ms"
"Execution Time: 2.042 ms"
```

Although there is a bitmap index on dependent.essn, the planner chooses to do a join between employee and dependent on ssn as it finds using seq scan and join is better than using bitmap index

Fourth scenario: with hash: on dependent.essn

Changes in flags: no flags changed

The query plan and actual measurements (The explain analyze output):

```
"Hash Semi Join  (cost=33.50..189.75 rows=1000 width=42) (actual time=0.314..2.046 rows=1000 loops=1)"
"  Hash Cond: (employee.ssn = dependent.essn)"
"  ->  Seq Scan on employee  (cost=0.00..132.00 rows=5000 width=46) (actual time=0.020..0.693 rows=5000 loops=1)"
"  ->  Hash  (cost=21.00..21.00 rows=1000 width=4) (actual time=0.281..0.281 rows=1000 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 44kB"
"        ->  Seq Scan on dependent  (cost=0.00..21.00 rows=1000 width=4) (actual time=0.010..0.134 rows=1000 loops=1)"
"Planning Time: 0.252 ms"
"Execution Time: 2.119 ms"
```

Although there is a bitmap index on dependent.essn, the planner chooses to do a join between employee and dependent on ssn as it finds using seq scan and join is better than using bitmap index

The best scenario in performance and justification:

The no index scenario is the one with the least total estimated cost

That's because joining the dependent and employee tables will scan through the whole tables so using seq scan (no index) is to join is better than using index scan

---

Query 6:

```
select dnumber, count(*)

from department, employee

where dnumber=dno

and

salary > 40000

and

dno in (

  select dno

  from employee

  group by dno

  having count (*) > 5)

group by dnumber;
```

First scenario: without any index (only seq scans)

Changes in flags: No flags are changed

The query plan and the actual measurements (The explain analyze output):

```
"HashAggregate  (cost=329.52..329.82 rows=30 width=12) (actual time=6.421..6.431 rows=30
loops=1)"
"  Group Key: department.dnumber"
"  ->  Hash Join  (cost=159.11..324.58 rows=987 width=4) (actual time=2.338..5.412 rows=2960
loops=1)"
"        Hash Cond: (employee.dno = department.dnumber)"
"        ->  Seq Scan on employee  (cost=0.00..144.50 rows=2960 width=4) (actual
time=0.030..1.668 rows=2960 loops=1)"
"              Filter: (salary > 40000)"
"              Rows Removed by Filter: 2040"
"        ->  Hash  (cost=158.99..158.99 rows=10 width=8) (actual time=2.289..2.289 rows=30
loops=1)"
"              Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"              ->  Hash Join  (cost=157.60..158.99 rows=10 width=8) (actual time=2.270..2.285
rows=30 loops=1)"
"                    Hash Cond: (department.dnumber = employee_1.dno)"
"                    ->  Seq Scan on department  (cost=0.00..1.30 rows=30 width=4) (actual
time=0.008..0.012 rows=30 loops=1)"
"                    ->  Hash  (cost=157.47..157.47 rows=10 width=4) (actual time=2.253..2.253
rows=30 loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                          ->  HashAggregate  (cost=157.00..157.38 rows=10 width=4) (actual
time=2.235..2.241 rows=30 loops=1)"
"                                Group Key: employee_1.dno"
"                                Filter: (count(*) > 5)"
"                                ->  Seq Scan on employee employee_1  (cost=0.00..132.00 rows=5000
width=4) (actual time=0.004..0.623 rows=5000 loops=1)"
"Planning Time: 0.351 ms"
"Execution Time: 6.528 ms"
```

Second scenario: with btree: on employee.dno, department.dnumer and employee.salary

Changes in flags: no changes

The query plan and actual measurements (The explain analyze output):

```
"GroupAggregate  (cost=0.70..413.92 rows=30 width=12) (actual time=0.420..7.571 rows=30 loops=1)"
"  Group Key: department.dnumber"
"  ->  Nested Loop  (cost=0.70..408.67 rows=989 width=4) (actual time=0.192..7.032 rows=2960 loops=1)"
"        ->  Merge Join  (cost=0.42..262.55 rows=10 width=8) (actual time=0.168..3.967 rows=30 loops=1)"
"              Merge Cond: (department.dnumber = employee_1.dno)"
"              ->  Index Only Scan using department_dnumber_btree on department  (cost=0.14..12.59 rows=30 width=4) (actual time=0.017..0.053 rows=30 loops=1)"
"                    Heap Fetches: 30"
"              ->  GroupAggregate  (cost=0.28..249.66 rows=10 width=4) (actual time=0.146..3.863 rows=30 loops=1)"
"                    Group Key: employee_1.dno"
"                    Filter: (count(*) > 5)"
"                    ->  Index Only Scan using employee_dno_btree on employee employee_1  (cost=0.28..224.28 rows=5000 width=4) (actual time=0.025..2.962 rows=5000 loops=1)"
"                          Heap Fetches: 5000"
"        ->  Index Scan using employee_dno_btree on employee  (cost=0.28..13.62 rows=99 width=4) (actual time=0.012..0.089 rows=99 loops=30)"
"              Index Cond: (dno = department.dnumber)"
"              Filter: (salary > 40000)"
"              Rows Removed by Filter: 68"
"Planning Time: 0.877 ms"
"Execution Time: 7.684 ms"
```

Third scenario: with bitmap: on employee.dno

Changes in flags: No flags changes

The query plan and actual measurements (The explain analyze output):

```
"GroupAggregate  (cost=161.22..318.36 rows=30 width=12) (actual time=4.076..8.517 rows=30
loops=1)"
"  Group Key: department.dnumber"
"  ->  Nested Loop  (cost=161.22..313.13 rows=987 width=4) (actual time=3.921..7.959
rows=2960 loops=1)"
"        ->  Merge Join  (cost=159.68..159.98 rows=10 width=8) (actual time=3.819..3.880 rows=30
loops=1)"
"              Merge Cond: (department.dnumber = employee_1.dno)"
"              ->  Sort  (cost=2.04..2.11 rows=30 width=4) (actual time=0.051..0.063 rows=30
loops=1)"
"                    Sort Key: department.dnumber"
"                    Sort Method: quicksort  Memory: 26kB"
"                    ->  Seq Scan on department  (cost=0.00..1.30 rows=30 width=4) (actual
time=0.025..0.031 rows=30 loops=1)"
"              ->  Sort  (cost=157.64..157.67 rows=10 width=4) (actual time=3.763..3.769 rows=30
loops=1)"
"                    Sort Key: employee_1.dno"
"                    Sort Method: quicksort  Memory: 26kB"
"                    ->  HashAggregate  (cost=157.00..157.38 rows=10 width=4) (actual
time=3.732..3.742 rows=30 loops=1)"
"                          Group Key: employee_1.dno"
"                          Filter: (count(*) > 5)"
"                          ->  Seq Scan on employee employee_1  (cost=0.00..132.00 rows=5000 width=4)
(actual time=0.013..0.992 rows=5000 loops=1)"
"        ->  Bitmap Heap Scan on employee  (cost=1.54..14.32 rows=99 width=4) (actual
time=0.051..0.115 rows=99 loops=30)"
```

```
"            Recheck Cond: (dno = department.dnumber)"
"            Filter: (salary > 40000)"
"            Rows Removed by Filter: 68"
"            Heap Blocks: exact=111"
"            ->  Bitmap Index Scan on employee_dno_gin  (cost=0.00..1.52 rows=167 width=0) (actual time=0.042..0.042 rows=167 loops=30)"
"                  Index Cond: (dno = department.dnumber)"
"Planning Time: 0.574 ms"
"Execution Time: 8.656 ms"
```

Fourth scenario: with hash: on employee.dno

Changes in flags: No changes

The query plan and actual measurements (The explain analyze output):

```
"HashAggregate  (cost=329.52..329.82 rows=30 width=12) (actual time=8.907..8.915 rows=30 loops=1)"
"  Group Key: department.dnumber"
"  ->  Hash Join  (cost=159.11..324.58 rows=987 width=4) (actual time=4.009..7.803 rows=2960 loops=1)"
"        Hash Cond: (employee.dno = department.dnumber)"
"        ->  Seq Scan on employee  (cost=0.00..144.50 rows=2960 width=4) (actual time=0.030..2.315 rows=2960 loops=1)"
"              Filter: (salary > 40000)"
"              Rows Removed by Filter: 2040"
"        ->  Hash  (cost=158.99..158.99 rows=10 width=8) (actual time=3.967..3.968 rows=30 loops=1)"
"              Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"              ->  Hash Join  (cost=157.60..158.99 rows=10 width=8) (actual time=3.924..3.951 rows=30 loops=1)"
"                    Hash Cond: (department.dnumber = employee_1.dno)"
"                 ->  Seq Scan on department  (cost=0.00..1.30 rows=30 width=4) (actual
time=0.013..0.019 rows=30 loops=1)"

"                 ->  Hash  (cost=157.47..157.47 rows=10 width=4) (actual time=3.898..3.899
rows=30 loops=1)"

"                      Buckets: 1024  Batches: 1  Memory Usage: 10kB"

"                      ->  HashAggregate  (cost=157.00..157.38 rows=10 width=4) (actual
time=3.864..3.876 rows=30 loops=1)"

"                            Group Key: employee_1.dno"

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

"                            ->  Seq Scan on employee employee_1  (cost=0.00..132.00 rows=5000
width=4) (actual time=0.008..1.064 rows=5000 loops=1)"

"Planning Time: 2.592 ms"

"Execution Time: 9.035 ms"
```

Although there is a hash index on employee.dno, the planner chooses to use seq scans only and not use the index as the whole tables should be scanned so using the index will only add an overhead

The best scenario in performance and justification:

The btree index scenario is the one with the least total estimated cost

That’s because the planner exploits the btree indicies on columns employee.dno & department.dnumber and do index scan(sorted scan) and do merge join between them(faster than ordinary join) and also it uses the index on employee.salary to retrieve only the employees with salary > 40000 faster

Schema 5:

Description of the data set:

1- Soccer_country table: 195 countries

2- player_mast table: 10000 players distributed among the 195 countries equally(around 51 for each country)

3- Match_mast table: 5000 matches

4- playing_pos table: 11 playing position

5- coach_mast table: 195 coaches one from each country

6- soccer_city table: 390 cities two for each country

7- soccer_venue table: 390 venues one for each city

8- referee_mast table: 585 referees

9- ass_referee_mast table: 1170 assistant referees

10- Goal_details table: 1950 goals

11- peanlities table: 10000 penalities

Changes to SQL and justification: Dropping all the primary key constraints from all the tables after populating the data set (to guarantee all data integrity) to remove any automatically created index

Query 13:

```
select country_name as team

from soccer_country

where country_id in(select team_id from match_details where
play_stage='a'and win_lose='W');
```

First scenario: without any index(only seq scans):

Changes in flags: no change

The query plan and actual measurements (The output of explain analyze):

```
"Hash Join  (cost=127.64..134.27 rows=195 width=10) (actual time=69.934..70.121 rows=195 loops=1)"
"  Hash Cond: (soccer_country.country_id = match_details.team_id)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.025..0.056 rows=195 loops=1)"
"  ->  Hash  (cost=125.20..125.20 rows=195 width=4) (actual time=69.893..69.893 rows=195 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"        ->  HashAggregate  (cost=123.25..125.20 rows=195 width=4) (actual time=69.742..69.786 rows=195 loops=1)"
"              Group Key: match_details.team_id"
"              ->  Seq Scan on match_details  (cost=0.00..117.00 rows=2500 width=4) (actual time=0.019..2.213 rows=2500 loops=1)"
"                    Filter: ((play_stage = 'a'::bpchar) AND (win_lose = 'W'::bpchar))"
"                    Rows Removed by Filter: 2500"
"Planning Time: 0.555 ms"
"Execution Time: 70.203 ms"
```

Second scenario: with btree: on match_details.team_id

Changes in flags: No changes

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop Semi Join  (cost=0.28..112.79 rows=195 width=10) (actual time=0.048..2.837 rows=195 loops=1)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.024..0.097 rows=195 loops=1)"
"  ->  Index Scan using match_details_team_id_btree on match_details  (cost=0.28..2.06 rows=13 width=4) (actual time=0.012..0.012 rows=1 loops=195)"
"        Index Cond: (team_id = soccer_country.country_id)"
"        Filter: ((play_stage = 'a'::bpchar) AND (win_lose = 'W'::bpchar))"
"Planning Time: 5.329 ms"
"Execution Time: 2.907 ms"
```

Third scenario: with bitmap(GIN): on match_dtails.team_id and for removing the seq scan and uses index scan instead creating a btree index on country_id

Changes in flags: SET enable_seqscan = OFF;

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop Semi Join  (cost=0.44..203.59 rows=195 width=10) (actual time=0.086..6.382 rows=195 loops=1)"
"  ->  Index Scan using soccer_country_country_id_btree on soccer_country  (cost=0.14..16.07 rows=195 width=14) (actual time=0.015..0.190 rows=195 loops=1)"
"  ->  Bitmap Heap Scan on match_details  (cost=0.30..4.75 rows=13 width=4) (actual time=0.022..0.022 rows=1 loops=195)"
"        Recheck Cond: (team_id = soccer_country.country_id)"
"        Filter: ((play_stage = 'a'::bpchar) AND (win_lose = 'W'::bpchar))"
"        Heap Blocks: exact=195"
"        ->  Bitmap Index Scan on match_details_team_id_gin  (cost=0.00..0.29 rows=26 width=0) (actual time=0.014..0.014 rows=26 loops=195)"
"              Index Cond: (team_id = soccer_country.country_id)"
"Planning Time: 3.315 ms"
"Execution Time: 6.515 ms"
```

Third scenario: with hash: on match_details.team_id

Changes in flags: No changes

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop Semi Join  (cost=0.00..69.28 rows=195 width=10) (actual time=0.363..4.173 rows=195 loops=1)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.033..0.075 rows=195 loops=1)"
"  ->  Index Scan using match_details_team_id_hash on match_details  (cost=0.00..2.19 rows=13 width=4) (actual time=0.020..0.020 rows=1 loops=195)"
"        Index Cond: (team_id = soccer_country.country_id)"
"        Filter: ((play_stage = 'a'::bpchar) AND (win_lose = 'W'::bpchar))"
"        Rows Removed by Filter: 12"
"Planning Time: 5.039 ms"
"Execution Time: 4.244 ms"
```

The best scenario in performance and justification:

The hash index scenario is the one with the least total estimated cost
That’s because the planner do a nested loop join with seq scan on soccer_country and then uses the hash index on match_details.team_id to check the rows that matches the join condition (team_id = country_id) in O(1)

Query 14:

```
select match_no
from match_details
where team_id=(
select country_id
from soccer_country
where country_name='Germany1')
or
```

```
team_id=(
select country_id
from soccer_country
where country_name='Germany2')
group by match_no
having count(distinct team_id)=1;
```

First scenario: without any index (seq scans only):

Changes in flags: No were flags changed

The query plan and actual measurements (The output of explain analyze):

```
"GroupAggregate  (cost=127.32..128.34 rows=1 width=4) (actual time=3.510..3.688 rows=26 loops=1)"
"  Group Key: match_details.match_no"
"  Filter: (count(DISTINCT match_details.team_id) = 1)"
"  Rows Removed by Filter: 13"
"  InitPlan 1 (returns $0)"
"    ->  Seq Scan on soccer_country  (cost=0.00..4.44 rows=1 width=4) (actual time=0.021..0.057 rows=1 loops=1)"
"          Filter: ((country_name)::text = 'Germany1'::text)"
"          Rows Removed by Filter: 194"
"  InitPlan 2 (returns $1)"
"    ->  Seq Scan on soccer_country soccer_country_1  (cost=0.00..4.44 rows=1 width=4) (actual time=0.014..0.050 rows=1 loops=1)"
"          Filter: ((country_name)::text = 'Germany2'::text)"
"          Rows Removed by Filter: 194"
"  ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=3.460..3.471 rows=52 loops=1)"
"        Sort Key: match_details.match_no"
"        Sort Method: quicksort  Memory: 27kB"
"       ->  Seq Scan on match_details  (cost=0.00..117.00 rows=51 width=8) (actual time=0.158..3.373 rows=52 loops=1)"
"             Filter: ((team_id = $0) OR (team_id = $1))"
"             Rows Removed by Filter: 4948"
"Planning Time: 1.302 ms"
"Execution Time: 4.038 ms"
```

Second scenario: with Btree: on match_mast.match_no , soccer_country_country_name

Changes in flags: No changes in flags

The query plan and actual measurements (The explain analyze output):

```
"GroupAggregate  (cost=16.61..226.50 rows=1 width=4) (actual time=0.343..5.200 rows=26 loops=1)"
"  Group Key: match_details.match_no"
"  Filter: (count(DISTINCT match_details.team_id) = 1)"
"  Rows Removed by Filter: 13"
"  InitPlan 1 (returns $0)"
"    ->  Index Scan using soccer_country_country_name_btree on soccer_country  (cost=0.14..8.16 rows=1 width=4) (actual time=0.033..0.034 rows=1 loops=1)"
"          Index Cond: ((country_name)::text = 'Germany1'::text)"
"  InitPlan 2 (returns $1)"
"    ->  Index Scan using soccer_country_country_name_btree on soccer_country soccer_country_1  (cost=0.14..8.16 rows=1 width=4) (actual time=0.012..0.013 rows=1 loops=1)"
"          Index Cond: ((country_name)::text = 'Germany2'::text)"
"  ->  Index Scan using match_details_match_no_btree on match_details  (cost=0.28..209.28 rows=51 width=8) (actual time=0.129..4.867 rows=52 loops=1)"
"        Filter: ((team_id = $0) OR (team_id = $1))"
"        Rows Removed by Filter: 4948"
```

"Planning Time: 3.246 ms"

"Execution Time: 5.328 ms"

Third scenario: with bitmap: on match_details.team_id , soccer_country.country_name

Changes in flags: No change

The query plan and actual measurement (The explain analyze output):

```
"GroupAggregate  (cost=86.87..87.89 rows=1 width=4) (actual time=0.261..0.428 rows=26 loops=1)"

"  Group Key: match_details.match_no"

"  Filter: (count(DISTINCT match_details.team_id) = 1)"

"  Rows Removed by Filter: 13"

"  InitPlan 1 (returns $0)"

"    ->  Bitmap Heap Scan on soccer_country  (cost=8.01..12.02 rows=1 width=4) (actual time=0.044..0.044 rows=1 loops=1)"

"          Recheck Cond: ((country_name)::text = 'Germany1'::text)"

"          Heap Blocks: exact=1"

"          ->  Bitmap Index Scan on soccer_country_country_name_gin  (cost=0.00..8.01 rows=1 width=0) (actual time=0.035..0.035 rows=1 loops=1)"

"                Index Cond: ((country_name)::text = 'Germany1'::text)"

"  InitPlan 2 (returns $1)"

"    ->  Bitmap Heap Scan on soccer_country soccer_country_1  (cost=8.01..12.02 rows=1 width=4) (actual time=0.012..0.012 rows=1 loops=1)"

"          Recheck Cond: ((country_name)::text = 'Germany2'::text)"

"          Heap Blocks: exact=1"

"          ->  Bitmap Index Scan on soccer_country_country_name_gin  (cost=0.00..8.01 rows=1 width=0) (actual time=0.009..0.009 rows=1 loops=1)"

"                Index Cond: ((country_name)::text = 'Germany2'::text)"

"  ->  Sort  (cost=62.83..62.95 rows=51 width=8) (actual time=0.232..0.241 rows=52 loops=1)"
"        Sort Key: match_details.match_no"
"        Sort Method: quicksort  Memory: 27kB"
"        ->  Bitmap Heap Scan on match_details  (cost=16.41..61.38 rows=51 width=8) (actual time=0.118..0.189 rows=52 loops=1)"
"              Recheck Cond: ((team_id = $0) OR (team_id = $1))"
"              Heap Blocks: exact=26"
"              ->  BitmapOr  (cost=16.41..16.41 rows=51 width=0) (actual time=0.106..0.106 rows=0 loops=1)"
"                    ->  Bitmap Index Scan on match_details_team_id_gin  (cost=0.00..8.19 rows=26 width=0) (actual time=0.079..0.079 rows=26 loops=1)"
"                          Index Cond: (team_id = $0)"
"                    ->  Bitmap Index Scan on match_details_team_id_gin  (cost=0.00..8.19 rows=26 width=0) (actual time=0.025..0.025 rows=26 loops=1)"
"                          Index Cond: (team_id = $1)"
"Planning Time: 2.344 ms"
"Execution Time: 0.631 ms"
```

Fourth scenario: with hash: on match_details.team_id, soccer_country.country_name

Changes in flags: no changes

The query plan and actual measurements (The explain analyze output):

```
"GroupAggregate  (cost=70.87..71.89 rows=1 width=4) (actual time=0.291..0.462 rows=26 loops=1)"
"  Group Key: match_details.match_no"
"  Filter: (count(DISTINCT match_details.team_id) = 1)"
"  Rows Removed by Filter: 13"
"  InitPlan 1 (returns $0)"
"    ->  Index Scan using soccer_country_country_name_hash on soccer_country  (cost=0.00..8.02 rows=1 width=4) (actual time=0.038..0.040 rows=1 loops=1)"
"          Index Cond: ((country_name)::text = 'Germany1'::text)"
"  InitPlan 2 (returns $1)"
"    ->  Index Scan using soccer_country_country_name_hash on soccer_country soccer_country_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.004..0.005 rows=1 loops=1)"
"          Index Cond: ((country_name)::text = 'Germany2'::text)"
"  ->  Sort  (cost=54.83..54.96 rows=51 width=8) (actual time=0.262..0.269 rows=52 loops=1)"
"        Sort Key: match_details.match_no"
"        Sort Method: quicksort  Memory: 27kB"
"        ->  Bitmap Heap Scan on match_details  (cost=8.42..53.38 rows=51 width=8) (actual time=0.111..0.215 rows=52 loops=1)"
"              Recheck Cond: ((team_id = $0) OR (team_id = $1))"
"              Heap Blocks: exact=26"
"              ->  BitmapOr  (cost=8.42..8.42 rows=51 width=0) (actual time=0.092..0.092 rows=0 loops=1)"
"                    ->  Bitmap Index Scan on match_details_team_id_hash  (cost=0.00..4.20 rows=26 width=0) (actual time=0.076..0.076 rows=26 loops=1)"
"                          Index Cond: (team_id = $0)"
"                    ->  Bitmap Index Scan on match_details_team_id_hash  (cost=0.00..4.20 rows=26 width=0) (actual time=0.015..0.015 rows=26 loops=1)"
"                          Index Cond: (team_id = $1)"
"Planning Time: 0.543 ms"
"Execution Time: 0.605 ms"
```

The best scenario in performance and justification:

The hash index scenario is the one with the least total estimated cost then after it the bitmap index (the difference in est. Cost is not too much, the planner uses the same approach in both)

That’s because the planner uses the hash index on country_name to retrive ‘Germany1’ and ‘Germany2’ ids fast, then it uses the hash index on team_id to get the match_no with team_id = Germany1’s id OR Germany2’s id

Query 15:

```
select match_no, play_stage, play_date, results, goal_score

from match_mast

where match_no in(select match_no from match_details where
team_id=(select country_id from soccer_country where
country_name='Germany1')

Or

team_id=(  select country_id   from soccer_country   where
country_name='Germany2')group by match_no having count(distinct
team_id)=1);
```

First scenario: without index (only seq scans):

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

"Nested Loop (cost=127.60..136.65 rows=1 width=22) (actual time=3.693..4.241 rows=26 loops=1)"

" -> GroupAggregate (cost=127.32..128.34 rows=1 width=4) (actual time=3.656..3.899 rows=26 loops=1)"

"   Group Key: match_details.match_no"

"   Filter: (count(DISTINCT match_details.team_id) = 1)"

"   Rows Removed by Filter: 13"

"   InitPlan 1 (returns $0)"

"   -> Seq Scan on soccer_country (cost=0.00..4.44 rows=1 width=4) (actual time=0.021..0.067 rows=1 loops=1)"

"   Filter: ((country_name)::text = 'Germany1'::text)"

"   Rows Removed by Filter: 194"

"   InitPlan 2 (returns $1)"

```
"         ->  Seq Scan on soccer_country soccer_country_1  (cost=0.00..4.44 rows=1 width=4) (actual time=0.017..0.059 rows=1 loops=1)"
"               Filter: ((country_name)::text = 'Germany2'::text)"
"               Rows Removed by Filter: 194"
"       ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=3.609..3.619 rows=52 loops=1)"
"             Sort Key: match_details.match_no"
"             Sort Method: quicksort  Memory: 27kB"
"             ->  Seq Scan on match_details  (cost=0.00..117.00 rows=51 width=8) (actual time=0.172..3.541 rows=52 loops=1)"
"                   Filter: ((team_id = $0) OR (team_id = $1))"
"                   Rows Removed by Filter: 4948"
"  ->  Index Scan using match_mast_pkey on match_mast  (cost=0.28..8.30 rows=1 width=22) (actual time=0.011..0.011 rows=1 loops=26)"
"        Index Cond: (match_no = match_details.match_no)"
"Planning Time: 0.505 ms"
"Execution Time: 4.376 ms"
```

Second scenario: with btree: on match_mast.match_no, match_details.match_no , soccer_country.country_name

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop  (cost=16.60..234.53 rows=1 width=22) (actual time=0.350..4.321 rows=26 loops=1)"
"  ->  GroupAggregate  (cost=16.32..226.21 rows=1 width=4) (actual time=0.266..3.826 rows=26 loops=1)"
"        Group Key: match_details.match_no"
"        Filter: (count(DISTINCT match_details.team_id) = 1)"
"        Rows Removed by Filter: 13"
"       InitPlan 1 (returns $0)"
"         ->  Index Scan using soccer_country_country_name_hash on soccer_country
(cost=0.00..8.02 rows=1 width=4) (actual time=0.022..0.024 rows=1 loops=1)"
"               Index Cond: ((country_name)::text = 'Germany1'::text)"
"       InitPlan 2 (returns $1)"
"         ->  Index Scan using soccer_country_country_name_hash on soccer_country
soccer_country_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.008..0.008 rows=1
loops=1)"
"               Index Cond: ((country_name)::text = 'Germany2'::text)"
"       ->  Index Scan using match_details_match_no_btree on match_details  (cost=0.28..209.28
rows=51 width=8) (actual time=0.075..3.521 rows=52 loops=1)"
"             Filter: ((team_id = $0) OR (team_id = $1))"
"             Rows Removed by Filter: 4948"
"  ->  Index Scan using match_mast_match_no_btree on match_mast  (cost=0.28..8.30 rows=1
width=22) (actual time=0.016..0.017 rows=1 loops=26)"
"       Index Cond: (match_no = match_details.match_no)"
"Planning Time: 2.754 ms"
"Execution Time: 4.470 ms"
```

Third scenario: with bitmap: on match_details.team_id, country_name , and to set the seq scan off and force only index scan a btree index on match_mast.match_no

Flags changes: SET enable_seqscan = OFF;

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop  (cost=79.14..88.20 rows=1 width=22) (actual time=0.341..0.754 rows=26
loops=1)"
"  ->  GroupAggregate  (cost=78.86..79.88 rows=1 width=4) (actual time=0.266..0.435 rows=26
loops=1)"
"       Group Key: match_details.match_no"
"       Filter: (count(DISTINCT match_details.team_id) = 1)"
"       Rows Removed by Filter: 13"
"       InitPlan 1 (returns $0)"

"         ->  Index Scan using soccer_country_country_name_hash on soccer_country
(cost=0.00..8.02 rows=1 width=4) (actual time=0.026..0.027 rows=1 loops=1)"

"               Index Cond: ((country_name)::text = 'Germany1'::text)"

"       InitPlan 2 (returns $1)"

"         ->  Index Scan using soccer_country_country_name_hash on soccer_country
soccer_country_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.010..0.011 rows=1
loops=1)"

"               Index Cond: ((country_name)::text = 'Germany2'::text)"

"       ->  Sort  (cost=62.83..62.95 rows=51 width=8) (actual time=0.234..0.240 rows=52
loops=1)"

"             Sort Key: match_details.match_no"

"             Sort Method: quicksort  Memory: 27kB"

"             ->  Bitmap Heap Scan on match_details  (cost=16.41..61.38 rows=51 width=8) (actual
time=0.114..0.190 rows=52 loops=1)"

"                   Recheck Cond: ((team_id = $0) OR (team_id = $1))"

"                   Heap Blocks: exact=26"

"                   ->  BitmapOr  (cost=16.41..16.41 rows=51 width=0) (actual time=0.101..0.101
rows=0 loops=1)"

"                         ->  Bitmap Index Scan on match_details_team_id_gin  (cost=0.00..8.19
rows=26 width=0) (actual time=0.076..0.076 rows=26 loops=1)"

"                               Index Cond: (team_id = $0)"

"                         ->  Bitmap Index Scan on match_details_team_id_gin  (cost=0.00..8.19
rows=26 width=0) (actual time=0.025..0.025 rows=26 loops=1)"

"                               Index Cond: (team_id = $1)"

"  ->  Index Scan using match_mast_match_no_btree on match_mast  (cost=0.28..8.30 rows=1
width=22) (actual time=0.011..0.011 rows=1 loops=26)"

"       Index Cond: (match_no = match_details.match_no)"

"Planning Time: 5.234 ms"

"Execution Time: 0.980 ms"
```

Fourth scenario: with hash: match_details_team_id, soccer_country_country_name, and to set the seq scan off and force only index scan a btree index on match_mast.match_no

Flags changes: SET enable_seqscan = OFF;

The query plan and actual measurements (The explain analyze output):

```
"Nested Loop  (cost=71.15..80.21 rows=1 width=22) (actual time=0.227..0.574 rows=26 loops=1)"
"  ->  GroupAggregate  (cost=70.87..71.89 rows=1 width=4) (actual time=0.184..0.321 rows=26 loops=1)"
"        Group Key: match_details.match_no"
"        Filter: (count(DISTINCT match_details.team_id) = 1)"
"        Rows Removed by Filter: 13"
"        InitPlan 1 (returns $0)"
"          ->  Index Scan using soccer_country_country_name_hash on soccer_country  (cost=0.00..8.02 rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)"
"                Index Cond: ((country_name)::text = 'Germany1'::text)"
"        InitPlan 2 (returns $1)"
"          ->  Index Scan using soccer_country_country_name_hash on soccer_country soccer_country_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.005..0.005 rows=1 loops=1)"
"                Index Cond: ((country_name)::text = 'Germany2'::text)"
"        ->  Sort  (cost=54.83..54.96 rows=51 width=8) (actual time=0.160..0.167 rows=52 loops=1)"
"              Sort Key: match_details.match_no"
"              Sort Method: quicksort  Memory: 27kB"
"              ->  Bitmap Heap Scan on match_details  (cost=8.42..53.38 rows=51 width=8) (actual time=0.072..0.130 rows=52 loops=1)"
"                    Recheck Cond: ((team_id = $0) OR (team_id = $1))"
"                    Heap Blocks: exact=26"
"                    ->  BitmapOr  (cost=8.42..8.42 rows=51 width=0) (actual time=0.061..0.061 rows=0 loops=1)"
"                          ->  Bitmap Index Scan on match_details_team_id_hash  (cost=0.00..4.20 rows=26 width=0) (actual time=0.050..0.050 rows=26 loops=1)"
"                                Index Cond: (team_id = $0)"
"                          ->  Bitmap Index Scan on match_details_team_id_hash  (cost=0.00..4.20 rows=26 width=0) (actual time=0.011..0.011 rows=26 loops=1)"
"                                Index Cond: (team_id = $1)"
"  ->  Index Scan using match_mast_match_no_btree on match_mast  (cost=0.28..8.30 rows=1 width=22) (actual time=0.008..0.008 rows=1 loops=26)"
"        Index Cond: (match_no = match_details.match_no)"
"Planning Time: 4.026 ms"
"Execution Time: 0.690 ms"
```

The best scenario in performance and justification:

The hash index scenario is the one with the least total estimated cost

That's because the planner uses the hash index on country_name to retrive 'Germany1' and 'Germany2' ids fast, then it uses the hash index on team_id to get the match_no with team_id = Germany1's id OR Germany2's id

---

Query 16:

```
select country_name
from soccer_country
where country_id in
( select team_id
from goal_details
where match_no=( select match_no from match_mast where audonce=( select max(audonce) from match_mast) orderby audonce desc));
```

First scenario: without index (only seq scans):

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

```
"Hash Join  (cost=414.08..420.72 rows=195 width=10) (actual time=13.135..13.321 rows=195
loops=1)"

"  Hash Cond: (soccer_country.country_id = goal_details.team_id)"

"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual
time=0.027..0.058 rows=195 loops=1)"

"  ->  Hash  (cost=411.65..411.65 rows=195 width=4) (actual time=13.090..13.090 rows=195
loops=1)"

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

"        ->  HashAggregate  (cost=409.70..411.65 rows=195 width=4) (actual time=12.936..12.980
rows=195 loops=1)"

"              Group Key: goal_details.team_id"

"              ->  Hash Semi Join  (cost=341.51..404.82 rows=1950 width=4) (actual
time=9.526..11.649 rows=1950 loops=1)"

"                    Hash Cond: (goal_details.match_no = "ANY_subquery".match_no)"

"                    ->  Seq Scan on goal_details  (cost=0.00..36.50 rows=1950 width=8) (actual
time=0.017..0.439 rows=1950 loops=1)"

"                    ->  Hash  (cost=279.01..279.01 rows=5000 width=4) (actual time=9.479..9.480
rows=5000 loops=1)"

"                          Buckets: 8192  Batches: 1  Memory Usage: 245kB"

"                          ->  Subquery Scan on "ANY_subquery"  (cost=114.51..279.01 rows=5000
width=4) (actual time=3.007..6.927 rows=5000 loops=1)"

"                                ->  Seq Scan on match_mast  (cost=114.51..229.01 rows=5000 width=9)
(actual time=3.005..6.414 rows=5000 loops=1)"

"                                      Filter: (audonce = $0)"

"                                      InitPlan 1 (returns $0)"
"                              ->  Aggregate  (cost=114.50..114.51 rows=1 width=32) (actual time=2.978..2.978 rows=1 loops=1)"
"                                  ->  Seq Scan on match_mast match_mast_1  (cost=0.00..102.00 rows=5000 width=5) (actual time=0.008..0.840 rows=5000 loops=1)"
"Planning Time: 1.922 ms"
"Execution Time: 13.546 ms"
```

Second scenario: with btree: on match_mast.audonce

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

```
"Hash Join  (cost=299.91..306.54 rows=195 width=10) (actual time=19.071..19.175 rows=195 loops=1)"
"  Hash Cond: (soccer_country.country_id = goal_details.team_id)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.023..0.042 rows=195 loops=1)"
"  ->  Hash  (cost=297.47..297.47 rows=195 width=4) (actual time=19.035..19.035 rows=195 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"        ->  HashAggregate  (cost=295.52..297.47 rows=195 width=4) (actual time=18.952..18.985 rows=195 loops=1)"
"              Group Key: goal_details.team_id"
"              ->  Hash Semi Join  (cost=227.33..290.65 rows=1950 width=4) (actual time=3.807..5.036 rows=1950 loops=1)"
"                    Hash Cond: (goal_details.match_no = "ANY_subquery".match_no)"
"                    ->  Seq Scan on goal_details  (cost=0.00..36.50 rows=1950 width=8) (actual time=0.014..0.277 rows=1950 loops=1)"
"                    ->  Hash  (cost=164.83..164.83 rows=5000 width=4) (actual time=3.752..3.753 rows=5000 loops=1)"
"                          Buckets: 8192  Batches: 1  Memory Usage: 245kB"
"                    ->  Subquery Scan on "ANY_subquery"  (cost=0.33..164.83 rows=5000 width=4) (actual time=0.178..2.539 rows=5000 loops=1)"
"                          ->  Seq Scan on match_mast  (cost=0.33..114.83 rows=5000 width=9) (actual time=0.177..2.181 rows=5000 loops=1)"
"                                Filter: (audonce = $1)"
"                                InitPlan 2 (returns $1)"
"                                  ->  Result  (cost=0.32..0.33 rows=1 width=32) (actual time=0.157..0.158 rows=1 loops=1)"
"                                        InitPlan 1 (returns $0)"
"                                          ->  Limit  (cost=0.28..0.32 rows=1 width=5) (actual time=0.152..0.152 rows=1 loops=1)"
"                                                ->  Index Only Scan Backward using match_mast_audonce_btree on match_mast match_mast_1  (cost=0.28..206.78 rows=5000 width=5) (actual time=0.151..0.151 rows=1 loops=1)"
"                                                      Index Cond: (audonce IS NOT NULL)"
"                                                      Heap Fetches: 1"
"Planning Time: 5.699 ms"
"Execution Time: 19.422 ms"
```

Third scenario: with bitmap on match_mast.audonce:

```
"Hash Join  (cost=414.08..420.72 rows=195 width=10) (actual time=12.810..13.005 rows=195 loops=1)"
"  Hash Cond: (soccer_country.country_id = goal_details.team_id)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.025..0.053 rows=195 loops=1)"
"  ->  Hash  (cost=411.65..411.65 rows=195 width=4) (actual time=12.765..12.765 rows=195 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"        ->  HashAggregate  (cost=409.70..411.65 rows=195 width=4) (actual time=12.618..12.664 rows=195 loops=1)"
"            Group Key: goal_details.team_id"
"            ->  Hash Semi Join  (cost=341.51..404.82 rows=1950 width=4) (actual time=9.223..11.297 rows=1950 loops=1)"
"                  Hash Cond: (goal_details.match_no = "ANY_subquery".match_no)"
"                  ->  Seq Scan on goal_details  (cost=0.00..36.50 rows=1950 width=8) (actual time=0.013..0.403 rows=1950 loops=1)"
"                  ->  Hash  (cost=279.01..279.01 rows=5000 width=4) (actual time=9.180..9.180 rows=5000 loops=1)"
"                        Buckets: 8192  Batches: 1  Memory Usage: 245kB"
"                        ->  Subquery Scan on "ANY_subquery"  (cost=114.51..279.01 rows=5000 width=4) (actual time=3.012..6.796 rows=5000 loops=1)"
"                              ->  Seq Scan on match_mast  (cost=114.51..229.01 rows=5000 width=9) (actual time=3.011..6.260 rows=5000 loops=1)"
"                                    Filter: (audonce = $0)"
"                                    InitPlan 1 (returns $0)"
"                                      ->  Aggregate  (cost=114.50..114.51 rows=1 width=32) (actual time=2.984..2.985 rows=1 loops=1)"
"                                            ->  Seq Scan on match_mast match_mast_1  (cost=0.00..102.00 rows=5000 width=5) (actual time=0.008..0.811 rows=5000 loops=1)"
"Planning Time: 2.711 ms"
"Execution Time: 13.236 ms"
```

Although there is a bitmap index on audonce column the planner chooses to de seq scans as retieving data by it will be faster than the index

Fourth scenario: with hash: on match_mast.audonce

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

```
"Hash Join  (cost=414.08..420.72 rows=195 width=10) (actual time=10.005..10.106 rows=195 loops=1)"
```

```
"  Hash Cond: (soccer_country.country_id = goal_details.team_id)"
"  ->  Seq Scan on soccer_country  (cost=0.00..3.95 rows=195 width=14) (actual time=0.035..0.055 rows=195 loops=1)"
"  ->  Hash  (cost=411.65..411.65 rows=195 width=4) (actual time=9.948..9.948 rows=195 loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"        ->  HashAggregate  (cost=409.70..411.65 rows=195 width=4) (actual time=9.865..9.895 rows=195 loops=1)"
"              Group Key: goal_details.team_id"
"              ->  Hash Semi Join  (cost=341.51..404.82 rows=1950 width=4) (actual time=7.882..9.090 rows=1950 loops=1)"
"                    Hash Cond: (goal_details.match_no = "ANY_subquery".match_no)"
"                    ->  Seq Scan on goal_details  (cost=0.00..36.50 rows=1950 width=8) (actual time=0.016..0.267 rows=1950 loops=1)"
"                    ->  Hash  (cost=279.01..279.01 rows=5000 width=4) (actual time=7.802..7.802 rows=5000 loops=1)"
"                          Buckets: 8192  Batches: 1  Memory Usage: 245kB"
"                          ->  Subquery Scan on "ANY_subquery"  (cost=114.51..279.01 rows=5000 width=4) (actual time=3.313..6.140 rows=5000 loops=1)"
"                                ->  Seq Scan on match_mast  (cost=114.51..229.01 rows=5000 width=9) (actual time=3.310..5.686 rows=5000 loops=1)"
"                                      Filter: (audonce = $0)"
"                                      InitPlan 1 (returns $0)"
"                                        ->  Aggregate  (cost=114.50..114.51 rows=1 width=32) (actual time=3.268..3.268 rows=1 loops=1)"
"                                              ->  Seq Scan on match_mast match_mast_1  (cost=0.00..102.00 rows=5000 width=5) (actual time=0.013..0.879 rows=5000 loops=1)"
"Planning Time: 3.288 ms"
"Execution Time: 10.327 ms"
```

Although there is a bitmap index on audonce column the planner chooses to de seq scans as retieving data by it will be faster than the index

The best scenario in performance and justification:

The btree index scenario is the one with the least total estimated cost

That’s because creating index on match_mast.audonce makes the planner able to get the max(audonce) fast.

Query 17:

```
SELECT player_name

FROM player_mast

WHERE player_id=( SELECT player_id FROM goal_details where match_no=(SELECT match_no FROM match_details WHERE team_id=( SELECT country_id FROM soccer_country WHERE country_name='Germany1') or team_id=(SELECT country_id FROM soccer_country WHERE country_name='Germany2') GROUP BY match_no HAVING COUNT(DISTINCT team_id)=2)

AND

team_id=(SELECT team_id FROM soccer_country a, soccer_team b WHERE a.country_id=b.team_id AND country_name='Germany2')

AND goal_time=( SELECT max(goal_time) FROM goal_details WHERE match_no=(SELECT match_no FROM match_details WHERE team_id=( SELECT country_id FROM soccer_country WHERE country_name='Germany1') or team_id=(SELECT country_id FROM soccer_country WHERE country_name='Germany2') GROUP BY match_no HAVING COUNT(DISTINCT team_id)=2)

AND team_id=( SELECT team_id FROM soccer_country a, soccer_team b WHERE a.country_id=b.team_id AND country_name='Germany2')));
```

First scenario: without index (only seq scans):

Flags changes: No change

The query plan and actual measurements (The explain analyze output):

```
"Seq Scan on player_mast  (cost=315.12..544.12 rows=1 width=11) (actual time=12.708..15.335 rows=1 loops=1)"
```

```
"  Filter: (player_id = $9)"

"  Rows Removed by Filter: 9999"

"  InitPlan 8 (returns $9)"

"    ->  Nested Loop  (cost=298.37..315.12 rows=1 width=4) (actual time=12.457..12.682 rows=1
loops=1)"

"          InitPlan 1 (returns $0)"

"            ->  Hash Join  (cost=4.45..9.14 rows=1 width=4) (actual time=0.124..0.203 rows=1
loops=1)"

"                  Hash Cond: (b.team_id = a.country_id)"

"                  ->  Seq Scan on soccer_team b  (cost=0.00..3.95 rows=195 width=4) (actual
time=0.018..0.036 rows=195 loops=1)"

"                  ->  Hash  (cost=4.44..4.44 rows=1 width=4) (actual time=0.078..0.079 rows=1
loops=1)"

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

"                        ->  Seq Scan on soccer_country a  (cost=0.00..4.44 rows=1 width=4) (actual
time=0.023..0.070 rows=1 loops=1)"

"                              Filter: ((country_name)::text = 'Germany2'::text)"

"                              Rows Removed by Filter: 194"

"          InitPlan 5 (returns $5)"

"            ->  Aggregate  (cost=157.54..157.55 rows=1 width=32) (actual time=8.828..8.829 rows=1
loops=1)"

"                  InitPlan 2 (returns $1)"

"                    ->  Hash Join  (cost=4.45..9.14 rows=1 width=4) (actual time=4.507..4.558 rows=1
loops=1)"

"                          Hash Cond: (b_1.team_id = a_1.country_id)"

"                          ->  Seq Scan on soccer_team b_1  (cost=0.00..3.95 rows=195 width=4) (actual
time=0.017..0.034 rows=195 loops=1)"

"                          ->  Hash  (cost=4.44..4.44 rows=1 width=4) (actual time=0.081..0.081 rows=1
loops=1)"

"                                Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                              ->  Seq Scan on soccer_country a_1  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.043..0.071 rows=1 loops=1)"
"                                    Filter: ((country_name)::text = 'Germany2'::text)"
"                                    Rows Removed by Filter: 194"
"                 ->  Nested Loop  (cost=131.67..148.40 rows=1 width=5) (actual time=8.548..8.819
rows=1 loops=1)"
"                       ->  GroupAggregate  (cost=127.32..128.34 rows=1 width=4) (actual
time=3.891..4.032 rows=13 loops=1)"
"                             Group Key: match_details.match_no"
"                             Filter: (count(DISTINCT match_details.team_id) = 2)"
"                             Rows Removed by Filter: 26"
"                             InitPlan 3 (returns $2)"
"                               ->  Seq Scan on soccer_country  (cost=0.00..4.44 rows=1 width=4) (actual
time=0.016..0.056 rows=1 loops=1)"
"                                     Filter: ((country_name)::text = 'Germany1'::text)"
"                                     Rows Removed by Filter: 194"
"                             InitPlan 4 (returns $3)"
"                               ->  Seq Scan on soccer_country soccer_country_1  (cost=0.00..4.44 rows=1
width=4) (actual time=0.044..0.086 rows=1 loops=1)"
"                                     Filter: ((country_name)::text = 'Germany2'::text)"
"                                     Rows Removed by Filter: 194"
"                             ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=3.841..3.846
rows=52 loops=1)"
"                                   Sort Key: match_details.match_no"
"                                   Sort Method: quicksort  Memory: 27kB"
"                                   ->  Seq Scan on match_details  (cost=0.00..117.00 rows=51 width=8)
(actual time=0.170..3.785 rows=52 loops=1)"
"                                         Filter: ((team_id = $2) OR (team_id = $3))"
"                                         Rows Removed by Filter: 4948"
"                      ->  Bitmap Heap Scan on goal_details  (cost=4.35..20.04 rows=1 width=9) (actual
time=0.360..0.361 rows=0 loops=13)"
"                            Recheck Cond: (match_no = match_details.match_no)"
"                            Filter: (team_id = $1)"
"                            Rows Removed by Filter: 1"
"                            Heap Blocks: exact=10"
"                            ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35
rows=10 width=0) (actual time=0.006..0.006 rows=1 loops=13)"
"                                  Index Cond: (match_no = match_details.match_no)"
"          ->  GroupAggregate  (cost=127.32..128.34 rows=1 width=4) (actual time=3.317..3.441
rows=13 loops=1)"
"                Group Key: match_details_1.match_no"
"                Filter: (count(DISTINCT match_details_1.team_id) = 2)"
"                Rows Removed by Filter: 26"
"                InitPlan 6 (returns $6)"
"                  ->  Seq Scan on soccer_country soccer_country_2  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.020..0.065 rows=1 loops=1)"
"                        Filter: ((country_name)::text = 'Germany1'::text)"
"                        Rows Removed by Filter: 194"
"                InitPlan 7 (returns $7)"
"                  ->  Seq Scan on soccer_country soccer_country_3  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.013..0.056 rows=1 loops=1)"
"                        Filter: ((country_name)::text = 'Germany2'::text)"
"                        Rows Removed by Filter: 194"
"                ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=3.272..3.279 rows=52
loops=1)"
"                      Sort Key: match_details_1.match_no"
"                      Sort Method: quicksort  Memory: 27kB"
"                    ->  Seq Scan on match_details match_details_1  (cost=0.00..117.00 rows=51 width=8) (actual time=0.148..3.200 rows=52 loops=1)"
"                          Filter: ((team_id = $6) OR (team_id = $7))"
"                          Rows Removed by Filter: 4948"
"          ->  Bitmap Heap Scan on goal_details goal_details_1  (cost=4.35..20.06 rows=1 width=8) (actual time=0.703..0.704 rows=0 loops=13)"
"                Recheck Cond: (match_no = match_details_1.match_no)"
"                Filter: ((team_id = $0) AND (goal_time = $5))"
"                Rows Removed by Filter: 1"
"                Heap Blocks: exact=10"
"                ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35 rows=10 width=0) (actual time=0.006..0.006 rows=1 loops=13)"
"                      Index Cond: (match_no = match_details_1.match_no)"
"Planning Time: 76.071 ms"
"Execution Time: 15.652 ms"
```

Second scenario: with btree:on player_mast_player_id & goal_details_match_no

Flags changes: no changes

The query plan and actual measurements (The explain analyze output):

```
"Index Scan using player_mast_player_id_btree on player_mast  (cost=315.41..323.42 rows=1 width=11) (actual time=17.059..17.061 rows=1 loops=1)"
"  Index Cond: (player_id = $9)"
"  InitPlan 8 (returns $9)"
"    ->  Nested Loop  (cost=298.37..315.12 rows=1 width=4) (actual time=16.119..16.977 rows=1 loops=1)"
"          InitPlan 1 (returns $0)"
"            ->  Hash Join  (cost=4.45..9.14 rows=1 width=4) (actual time=8.734..8.785 rows=1 loops=1)"
"                  Hash Cond: (b.team_id = a.country_id)"
"                  ->  Seq Scan on soccer_team b  (cost=0.00..3.95 rows=195 width=4) (actual
time=0.020..0.038 rows=195 loops=1)"
"                  ->  Hash  (cost=4.44..4.44 rows=1 width=4) (actual time=0.078..0.078 rows=1
loops=1)"
"                        Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                        ->  Seq Scan on soccer_country a  (cost=0.00..4.44 rows=1 width=4) (actual
time=0.042..0.068 rows=1 loops=1)"
"                              Filter: ((country_name)::text = 'Germany2'::text)"
"                              Rows Removed by Filter: 194"
"    InitPlan 5 (returns $5)"
"      ->  Aggregate  (cost=157.54..157.55 rows=1 width=32) (actual time=3.214..3.214 rows=1
loops=1)"
"            InitPlan 2 (returns $1)"
"              ->  Hash Join  (cost=4.45..9.14 rows=1 width=4) (actual time=0.101..0.173 rows=1
loops=1)"
"                    Hash Cond: (b_1.team_id = a_1.country_id)"
"                    ->  Seq Scan on soccer_team b_1  (cost=0.00..3.95 rows=195 width=4) (actual
time=0.015..0.038 rows=195 loops=1)"
"                    ->  Hash  (cost=4.44..4.44 rows=1 width=4) (actual time=0.065..0.065 rows=1
loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                          ->  Seq Scan on soccer_country a_1  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.017..0.055 rows=1 loops=1)"
"                                Filter: ((country_name)::text = 'Germany2'::text)"
"                                Rows Removed by Filter: 194"
"            ->  Nested Loop  (cost=131.67..148.40 rows=1 width=5) (actual time=2.730..3.206
rows=1 loops=1)"
"                  ->  GroupAggregate  (cost=127.32..128.34 rows=1 width=4) (actual
time=2.479..2.718 rows=13 loops=1)"
"                        Group Key: match_details.match_no"
"                              Filter: (count(DISTINCT match_details.team_id) = 2)"
"                              Rows Removed by Filter: 26"
"                              InitPlan 3 (returns $2)"
"                                ->  Seq Scan on soccer_country  (cost=0.00..4.44 rows=1 width=4) (actual
time=0.008..0.032 rows=1 loops=1)"
"                                      Filter: ((country_name)::text = 'Germany1'::text)"
"                                      Rows Removed by Filter: 194"
"                              InitPlan 4 (returns $3)"
"                                ->  Seq Scan on soccer_country soccer_country_1  (cost=0.00..4.44 rows=1
width=4) (actual time=0.008..0.032 rows=1 loops=1)"
"                                      Filter: ((country_name)::text = 'Germany2'::text)"
"                                      Rows Removed by Filter: 194"
"                              ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=2.437..2.446
rows=52 loops=1)"
"                                    Sort Key: match_details.match_no"
"                                    Sort Method: quicksort  Memory: 27kB"
"                                    ->  Seq Scan on match_details  (cost=0.00..117.00 rows=51 width=8)
(actual time=0.080..2.403 rows=52 loops=1)"
"                                          Filter: ((team_id = $2) OR (team_id = $3))"
"                                          Rows Removed by Filter: 4948"
"                        ->  Bitmap Heap Scan on goal_details  (cost=4.35..20.04 rows=1 width=9) (actual
time=0.025..0.027 rows=0 loops=13)"
"                              Recheck Cond: (match_no = match_details.match_no)"
"                              Filter: (team_id = $1)"
"                              Rows Removed by Filter: 1"
"                              Heap Blocks: exact=10"
"                              ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35
rows=10 width=0) (actual time=0.009..0.009 rows=1 loops=13)"
"                                    Index Cond: (match_no = match_details.match_no)"
"        ->  GroupAggregate  (cost=127.32..128.34 rows=1 width=4) (actual time=4.016..4.614
rows=13 loops=1)"
"              Group Key: match_details_1.match_no"
"              Filter: (count(DISTINCT match_details_1.team_id) = 2)"
"              Rows Removed by Filter: 26"
"              InitPlan 6 (returns $6)"
"                ->  Seq Scan on soccer_country soccer_country_2  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.023..0.063 rows=1 loops=1)"
"                      Filter: ((country_name)::text = 'Germany1'::text)"
"                      Rows Removed by Filter: 194"
"              InitPlan 7 (returns $7)"
"                ->  Seq Scan on soccer_country soccer_country_3  (cost=0.00..4.44 rows=1 width=4)
(actual time=0.014..0.057 rows=1 loops=1)"
"                      Filter: ((country_name)::text = 'Germany2'::text)"
"                      Rows Removed by Filter: 194"
"              ->  Sort  (cost=118.45..118.57 rows=51 width=8) (actual time=3.764..3.781 rows=52
loops=1)"
"                    Sort Key: match_details_1.match_no"
"                    Sort Method: quicksort  Memory: 27kB"
"                    ->  Seq Scan on match_details match_details_1  (cost=0.00..117.00 rows=51
width=8) (actual time=0.172..3.681 rows=52 loops=1)"
"                          Filter: ((team_id = $6) OR (team_id = $7))"
"                          Rows Removed by Filter: 4948"
"        ->  Bitmap Heap Scan on goal_details goal_details_1  (cost=4.35..20.06 rows=1 width=8)
(actual time=0.938..0.939 rows=0 loops=13)"
"              Recheck Cond: (match_no = match_details_1.match_no)"
"              Filter: ((team_id = $0) AND (goal_time = $5))"
"              Rows Removed by Filter: 1"
"              Heap Blocks: exact=10"
"              ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35 rows=10
width=0) (actual time=0.011..0.011 rows=1 loops=13)"

"                    Index Cond: (match_no = match_details_1.match_no)"

"Planning Time: 4.735 ms"

"Execution Time: 17.534 ms"
```

Third scenario: with bitmap: on soccer_country_country_name

Flags changes: No change

The query plan and actual measurements (The explain analyze output):

```
"Index Scan using player_mast_player_id_btree on player_mast  (cost=549.40..557.42 rows=1
width=11) (actual time=18.489..18.489 rows=1 loops=1)"

"  Index Cond: (player_id = $11)"

"  InitPlan 8 (returns $11)"

"    ->  Nested Loop  (cost=323.49..549.12 rows=1 width=4) (actual time=13.328..18.124 rows=1
loops=1)"

"          InitPlan 1 (returns $1)"

"            ->  Nested Loop  (cost=8.15..20.26 rows=1 width=4) (actual time=0.054..0.055 rows=1
loops=1)"

"                  ->  Bitmap Heap Scan on soccer_country a  (cost=8.01..12.02 rows=1 width=4)
(actual time=0.032..0.033 rows=1 loops=1)"

"                        Recheck Cond: ((country_name)::text = 'Germany2'::text)"

"                        Heap Blocks: exact=1"

"                        ->  Bitmap Index Scan on soccer_country_country_name_gin  (cost=0.00..8.01
rows=1 width=0) (actual time=0.017..0.017 rows=1 loops=1)"

"                              Index Cond: ((country_name)::text = 'Germany2'::text)"

"                  ->  Index Only Scan using soccer_team_team_id_btree on soccer_team b
(cost=0.14..8.16 rows=1 width=4) (actual time=0.017..0.017 rows=1 loops=1)"

"                        Index Cond: (team_id = a.country_id)"

"                        Heap Fetches: 1"
"         InitPlan 5 (returns $7)"
"           ->  Aggregate  (cost=274.54..274.55 rows=1 width=32) (actual time=5.781..5.781 rows=1
loops=1)"
"                 InitPlan 2 (returns $3)"
"                   ->  Nested Loop  (cost=8.15..20.26 rows=1 width=4) (actual time=0.439..0.441
rows=1 loops=1)"
"                         ->  Bitmap Heap Scan on soccer_country a_1  (cost=8.01..12.02 rows=1 width=4)
(actual time=0.017..0.018 rows=1 loops=1)"
"                               Recheck Cond: ((country_name)::text = 'Germany2'::text)"
"                               Heap Blocks: exact=1"
"                               ->  Bitmap Index Scan on soccer_country_country_name_gin
(cost=0.00..8.01 rows=1 width=0) (actual time=0.013..0.014 rows=1 loops=1)"
"                                     Index Cond: ((country_name)::text = 'Germany2'::text)"
"                         ->  Index Only Scan using soccer_team_team_id_btree on soccer_team b_1
(cost=0.14..8.16 rows=1 width=4) (actual time=0.416..0.416 rows=1 loops=1)"
"                               Index Cond: (team_id = a_1.country_id)"
"                               Heap Fetches: 1"
"                 ->  Nested Loop  (cost=28.68..254.27 rows=1 width=5) (actual time=0.741..5.773
rows=1 loops=1)"
"                       ->  GroupAggregate  (cost=24.32..234.22 rows=1 width=4) (actual
time=0.242..4.993 rows=13 loops=1)"
"                             Group Key: match_details.match_no"
"                             Filter: (count(DISTINCT match_details.team_id) = 2)"
"                             Rows Removed by Filter: 26"
"                             InitPlan 3 (returns $4)"
"                               ->  Bitmap Heap Scan on soccer_country  (cost=8.01..12.02 rows=1 width=4)
(actual time=0.016..0.017 rows=1 loops=1)"
"                                     Recheck Cond: ((country_name)::text = 'Germany1'::text)"
"                                     Heap Blocks: exact=1"
"                                  ->  Bitmap Index Scan on soccer_country_country_name_gin
(cost=0.00..8.01 rows=1 width=0) (actual time=0.013..0.013 rows=1 loops=1)"
"                                        Index Cond: ((country_name)::text = 'Germany1'::text)"
"                          InitPlan 4 (returns $5)"
"                            ->  Bitmap Heap Scan on soccer_country soccer_country_1  (cost=8.01..12.02
rows=1 width=4) (actual time=0.019..0.019 rows=1 loops=1)"
"                                  Recheck Cond: ((country_name)::text = 'Germany2'::text)"
"                                  Heap Blocks: exact=1"
"                                  ->  Bitmap Index Scan on soccer_country_country_name_gin
(cost=0.00..8.01 rows=1 width=0) (actual time=0.016..0.017 rows=1 loops=1)"
"                                        Index Cond: ((country_name)::text = 'Germany2'::text)"
"                          ->  Index Scan using match_details_match_no_btree on match_details
(cost=0.28..209.28 rows=51 width=8) (actual time=0.056..4.642 rows=52 loops=1)"
"                                Filter: ((team_id = $4) OR (team_id = $5))"
"                                Rows Removed by Filter: 4948"
"                    ->  Bitmap Heap Scan on goal_details  (cost=4.35..20.04 rows=1 width=9) (actual
time=0.045..0.047 rows=0 loops=13)"
"                          Recheck Cond: (match_no = match_details.match_no)"
"                          Filter: (team_id = $3)"
"                          Rows Removed by Filter: 1"
"                          Heap Blocks: exact=10"
"                          ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35
rows=10 width=0) (actual time=0.008..0.008 rows=1 loops=13)"
"                                Index Cond: (match_no = match_details.match_no)"
"        ->  GroupAggregate  (cost=24.32..234.22 rows=1 width=4) (actual time=7.416..11.915
rows=13 loops=1)"
"              Group Key: match_details_1.match_no"
"              Filter: (count(DISTINCT match_details_1.team_id) = 2)"
"              Rows Removed by Filter: 26"
"              InitPlan 6 (returns $8)"
"                ->  Bitmap Heap Scan on soccer_country soccer_country_2  (cost=8.01..12.02 rows=1 width=4) (actual time=0.055..0.056 rows=1 loops=1)"
"                      Recheck Cond: ((country_name)::text = 'Germany1'::text)"
"                      Heap Blocks: exact=1"
"                      ->  Bitmap Index Scan on soccer_country_country_name_gin  (cost=0.00..8.01 rows=1 width=0) (actual time=0.046..0.046 rows=1 loops=1)"
"                            Index Cond: ((country_name)::text = 'Germany1'::text)"
"              InitPlan 7 (returns $9)"
"                ->  Bitmap Heap Scan on soccer_country soccer_country_3  (cost=8.01..12.02 rows=1 width=4) (actual time=0.012..0.013 rows=1 loops=1)"
"                      Recheck Cond: ((country_name)::text = 'Germany2'::text)"
"                      Heap Blocks: exact=1"
"                      ->  Bitmap Index Scan on soccer_country_country_name_gin  (cost=0.00..8.01 rows=1 width=0) (actual time=0.010..0.010 rows=1 loops=1)"
"                            Index Cond: ((country_name)::text = 'Germany2'::text)"
"              ->  Index Scan using match_details_match_no_btree on match_details match_details_1  (cost=0.28..209.28 rows=51 width=8) (actual time=0.175..4.477 rows=52 loops=1)"
"                    Filter: ((team_id = $8) OR (team_id = $9))"
"                    Rows Removed by Filter: 4948"
"        ->  Bitmap Heap Scan on goal_details goal_details_1  (cost=4.35..20.06 rows=1 width=8) (actual time=0.461..0.463 rows=0 loops=13)"
"              Recheck Cond: (match_no = match_details_1.match_no)"
"              Filter: ((team_id = $1) AND (goal_time = $7))"
"              Rows Removed by Filter: 1"
"              Heap Blocks: exact=10"
"              ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35 rows=10 width=0) (actual time=0.008..0.008 rows=1 loops=13)"
"                    Index Cond: (match_no = match_details_1.match_no)"
```

```
"Planning Time: 2.167 ms"

"Execution Time: 158.821 ms"
```

Fourth scenario: with hash: on soccer_country_country_name:

Flags changes: No changes

The query plan and actual measurements (The explain analyze output):

```
"Index Scan using player_mast_player_id_btree on player_mast  (cost=525.38..533.40 rows=1 width=11) (actual time=10.929..10.930 rows=1 loops=1)"

"  Index Cond: (player_id = $11)"

"  InitPlan 8 (returns $11)"

"    ->  Nested Loop  (cost=299.48..525.10 rows=1 width=4) (actual time=6.143..10.893 rows=1 loops=1)"

"          InitPlan 1 (returns $1)"

"            ->  Nested Loop  (cost=0.14..16.26 rows=1 width=4) (actual time=0.019..0.020 rows=1 loops=1)"

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

"                        Index Cond: ((country_name)::text = 'Germany2'::text)"

"                  ->  Index Only Scan using soccer_team_team_id_btree on soccer_team b  (cost=0.14..8.16 rows=1 width=4) (actual time=0.012..0.012 rows=1 loops=1)"

"                        Index Cond: (team_id = a.country_id)"

"                        Heap Fetches: 1"

"          InitPlan 5 (returns $7)"

"            ->  Aggregate  (cost=262.53..262.54 rows=1 width=32) (actual time=5.807..5.807 rows=1 loops=1)"

"                  InitPlan 2 (returns $3)"

"                    ->  Nested Loop  (cost=0.14..16.26 rows=1 width=4) (actual time=0.021..0.022 rows=1 loops=1)"

"                          ->  Index Scan using soccer_country_country_name_hash on soccer_country a_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.004..0.005 rows=1 loops=1)"
"                              Index Cond: ((country_name)::text = 'Germany2'::text)"
"                         ->  Index Only Scan using soccer_team_team_id_btree on soccer_team b_1
(cost=0.14..8.16 rows=1 width=4) (actual time=0.015..0.015 rows=1 loops=1)"
"                              Index Cond: (team_id = a_1.country_id)"
"                              Heap Fetches: 1"
"                  ->  Nested Loop  (cost=20.67..246.27 rows=1 width=5) (actual time=0.271..5.800
rows=1 loops=1)"
"                        ->  GroupAggregate  (cost=16.32..226.21 rows=1 width=4) (actual
time=0.213..5.267 rows=13 loops=1)"
"                              Group Key: match_details.match_no"
"                              Filter: (count(DISTINCT match_details.team_id) = 2)"
"                              Rows Removed by Filter: 26"
"                              InitPlan 3 (returns $4)"
"                                ->  Index Scan using soccer_country_country_name_hash on soccer_country
(cost=0.00..8.02 rows=1 width=4) (actual time=0.018..0.019 rows=1 loops=1)"
"                                      Index Cond: ((country_name)::text = 'Germany1'::text)"
"                              InitPlan 4 (returns $5)"
"                                ->  Index Scan using soccer_country_country_name_hash on soccer_country
soccer_country_1  (cost=0.00..8.02 rows=1 width=4) (actual time=0.004..0.004 rows=1
loops=1)"
"                                      Index Cond: ((country_name)::text = 'Germany2'::text)"
"                              ->  Index Scan using match_details_match_no_btree on match_details
(cost=0.28..209.28 rows=51 width=8) (actual time=0.039..4.870 rows=52 loops=1)"
"                                    Filter: ((team_id = $4) OR (team_id = $5))"
"                                    Rows Removed by Filter: 4948"
"                        ->  Bitmap Heap Scan on goal_details  (cost=4.35..20.04 rows=1 width=9) (actual
time=0.018..0.020 rows=0 loops=13)"
"                              Recheck Cond: (match_no = match_details.match_no)"
"                              Filter: (team_id = $3)"
"                              Rows Removed by Filter: 1"
"                         Heap Blocks: exact=10"
"                         ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35
rows=10 width=0) (actual time=0.012..0.012 rows=1 loops=13)"
"                              Index Cond: (match_no = match_details.match_no)"
"        ->  GroupAggregate  (cost=16.32..226.21 rows=1 width=4) (actual time=0.253..4.653
rows=13 loops=1)"
"              Group Key: match_details_1.match_no"
"              Filter: (count(DISTINCT match_details_1.team_id) = 2)"
"              Rows Removed by Filter: 26"
"              InitPlan 6 (returns $8)"
"                ->  Index Scan using soccer_country_country_name_hash on soccer_country
soccer_country_2  (cost=0.00..8.02 rows=1 width=4) (actual time=0.015..0.016 rows=1
loops=1)"
"                      Index Cond: ((country_name)::text = 'Germany1'::text)"
"              InitPlan 7 (returns $9)"
"                ->  Index Scan using soccer_country_country_name_hash on soccer_country
soccer_country_3  (cost=0.00..8.02 rows=1 width=4) (actual time=0.010..0.011 rows=1
loops=1)"
"                      Index Cond: ((country_name)::text = 'Germany2'::text)"
"              ->  Index Scan using match_details_match_no_btree on match_details
match_details_1  (cost=0.28..209.28 rows=51 width=8) (actual time=0.062..4.316 rows=52
loops=1)"
"                    Filter: ((team_id = $8) OR (team_id = $9))"
"                    Rows Removed by Filter: 4948"
"        ->  Bitmap Heap Scan on goal_details goal_details_1  (cost=4.35..20.06 rows=1 width=8)
(actual time=0.462..0.465 rows=0 loops=13)"
"              Recheck Cond: (match_no = match_details_1.match_no)"
"              Filter: ((team_id = $1) AND (goal_time = $7))"
"              Rows Removed by Filter: 1"
"              Heap Blocks: exact=10"
"               ->  Bitmap Index Scan on goal_details_match_no_btree  (cost=0.00..4.35 rows=10 width=0) (actual time=0.009..0.009 rows=1 loops=13)"
"                     Index Cond: (match_no = match_details_1.match_no)"
"Planning Time: 1.913 ms"
"Execution Time: 11.231 ms"
```

The best scenario in performance and justification:

The btree index scenario is the one with the least total estimated cost

That’s because The planner uses the btree index on player_mast_player_id to scan the table faster and also uses the btree index on goal_details.match_no to retieve rows satisfying the condition (goal_details.match_no = match_details.match_no) faster than seq scan on the table then filter.