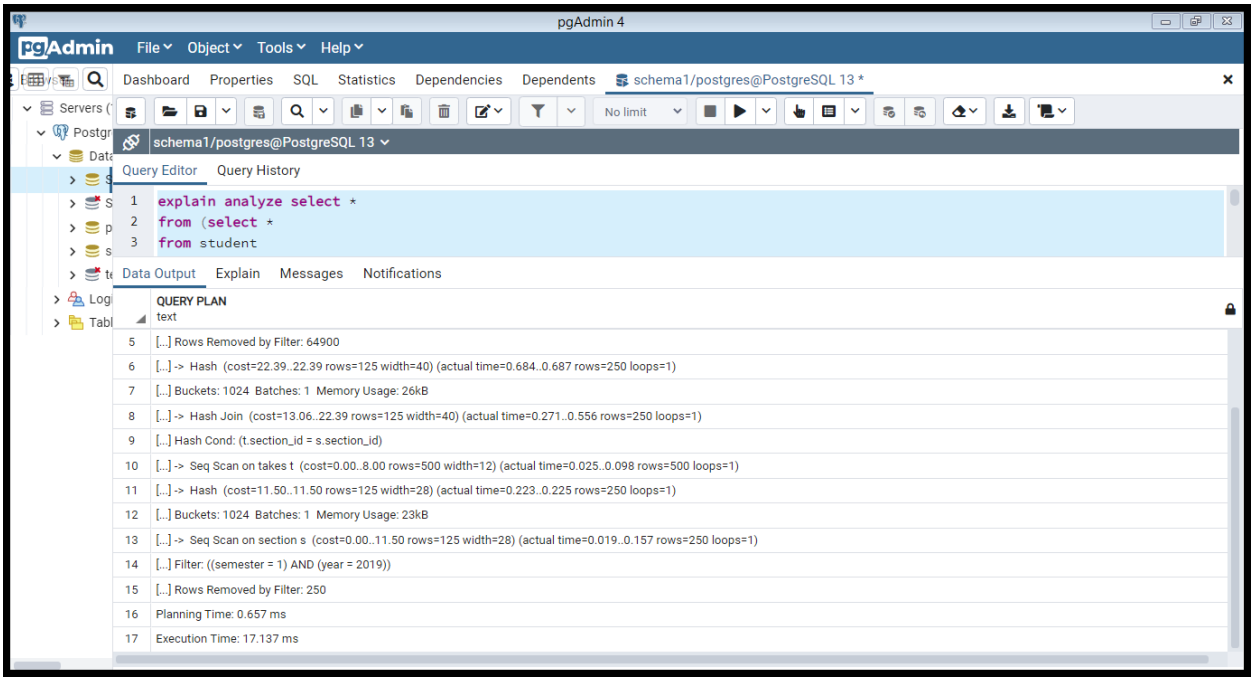
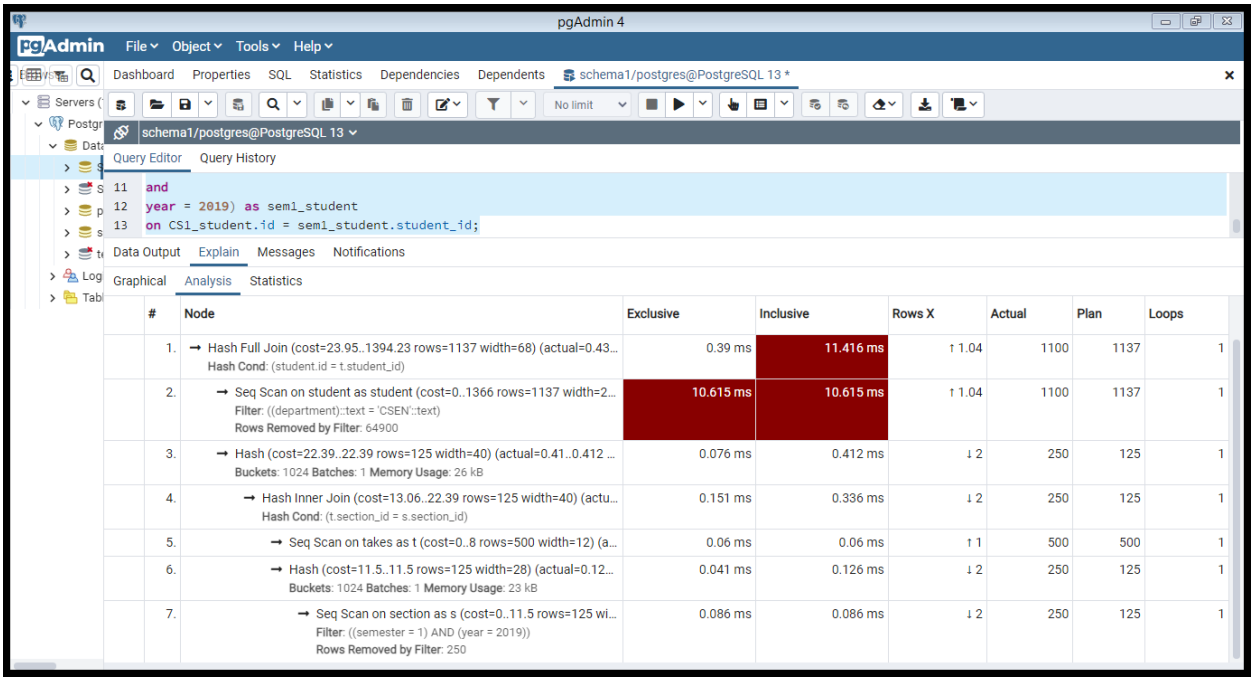
Query 1 with no indecies

Planning and execution time :



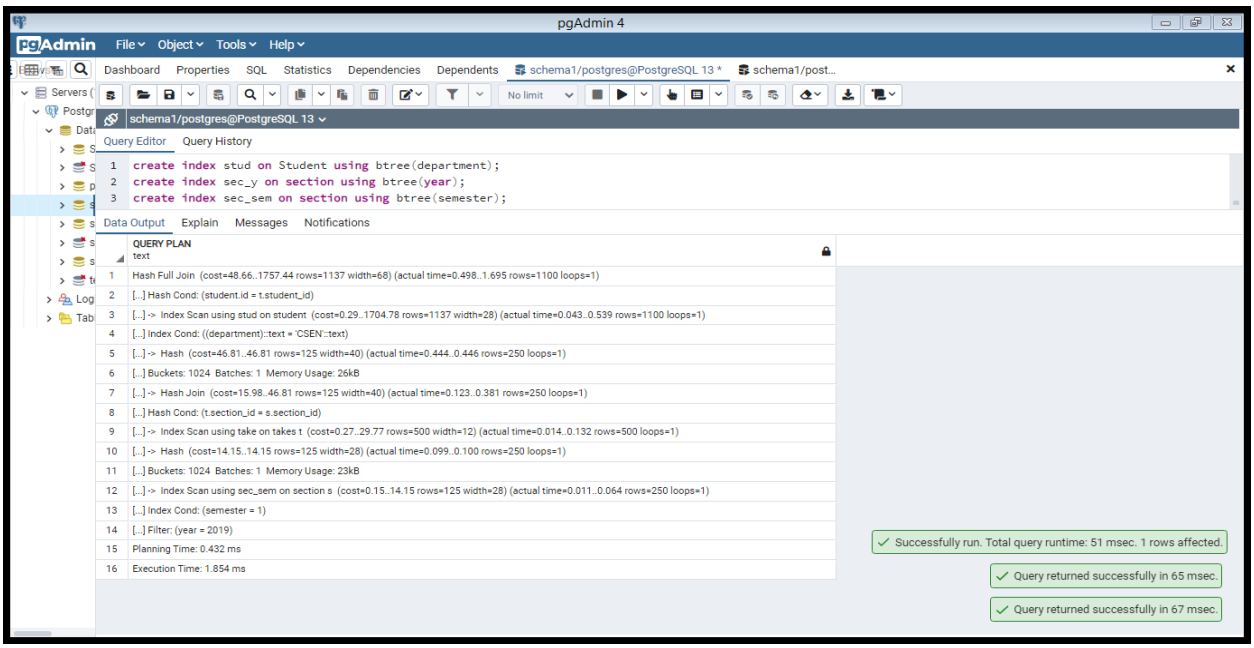
Execution Plan and costs :



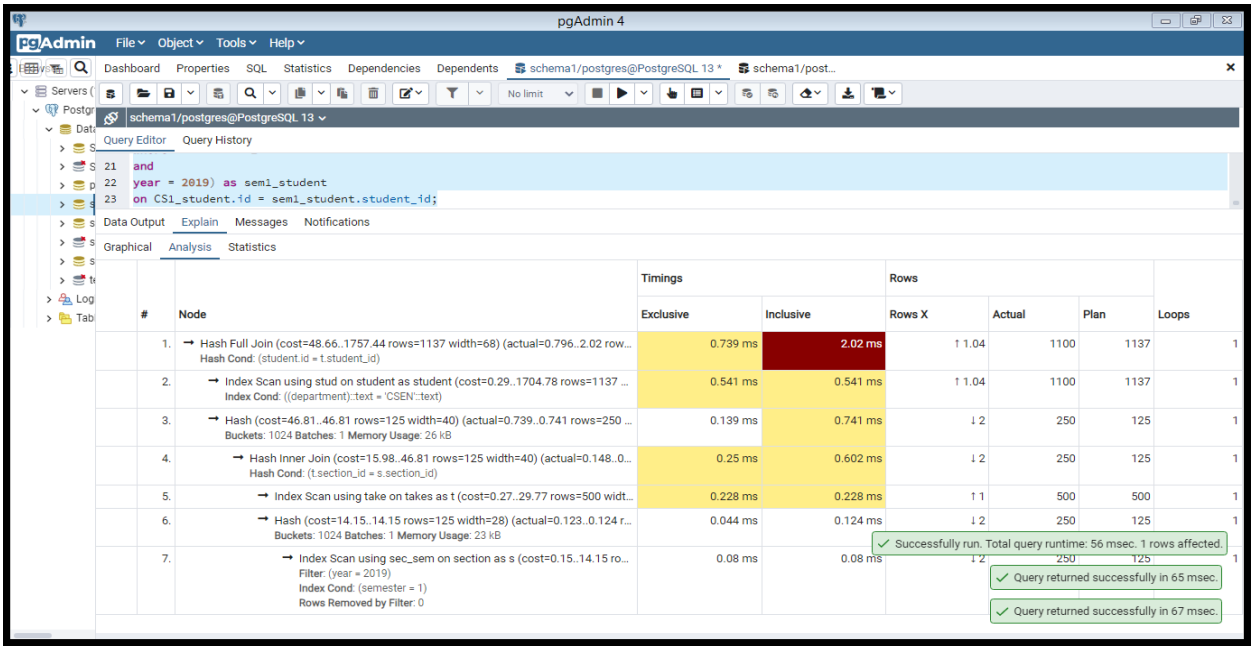
Query 1 using B+ Tree on : Student(department) , section(year) , section(semester), takes(section_id)

Disabled flags: seqscan , bitmapscan

Planning and execution time:



Execution plan and costs:

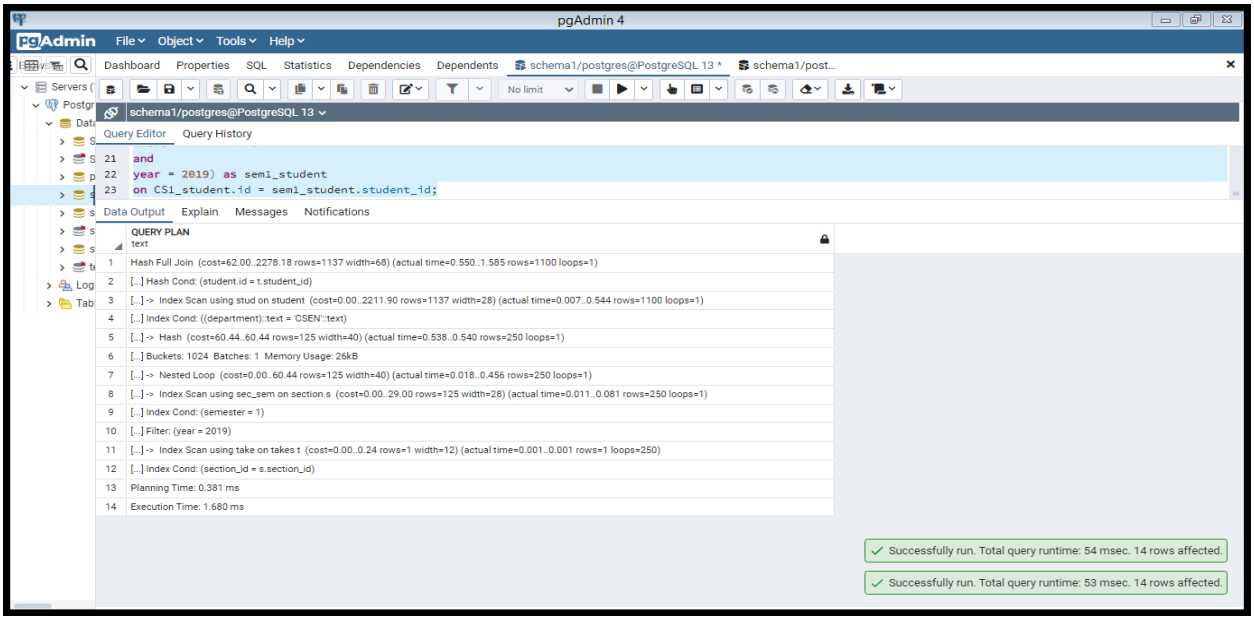


Here the B+ tree optimized the query as we put it on the columns that we search on like semester and year of section as B+ tree search in O(logn) time so it is better than seqscan

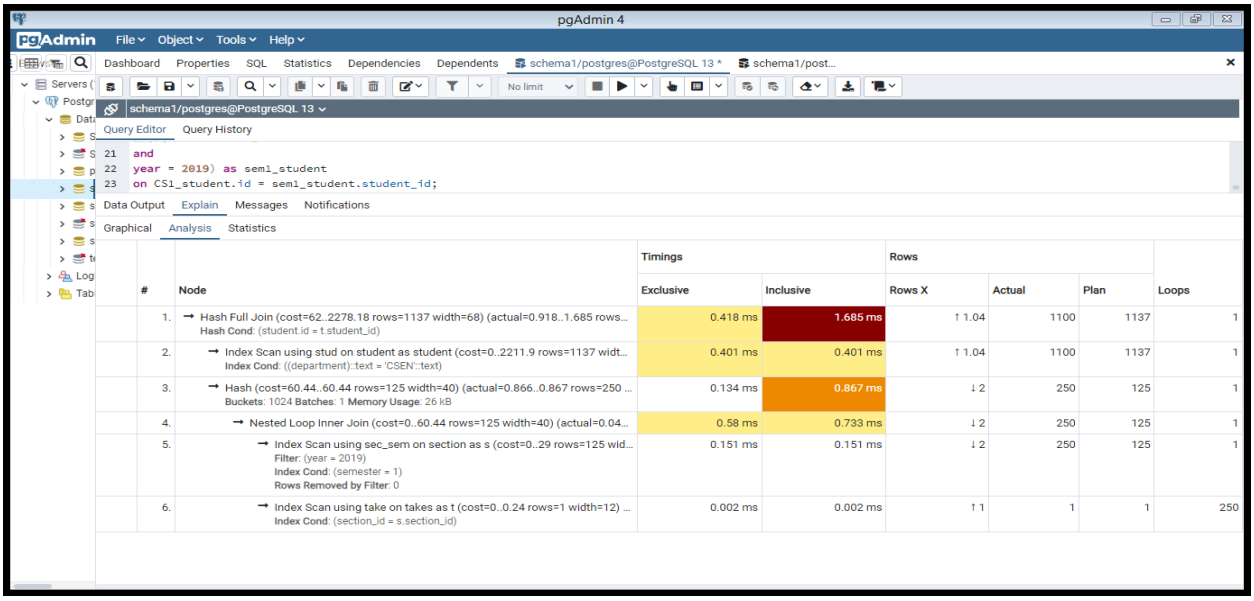
Query 1 using hash indecies on : Student(department) , section(year) , section(semester), takes(section_id)

Disabled flags: seqscan, bitmapscan

Planning and execution time:



Execution plan and costs:

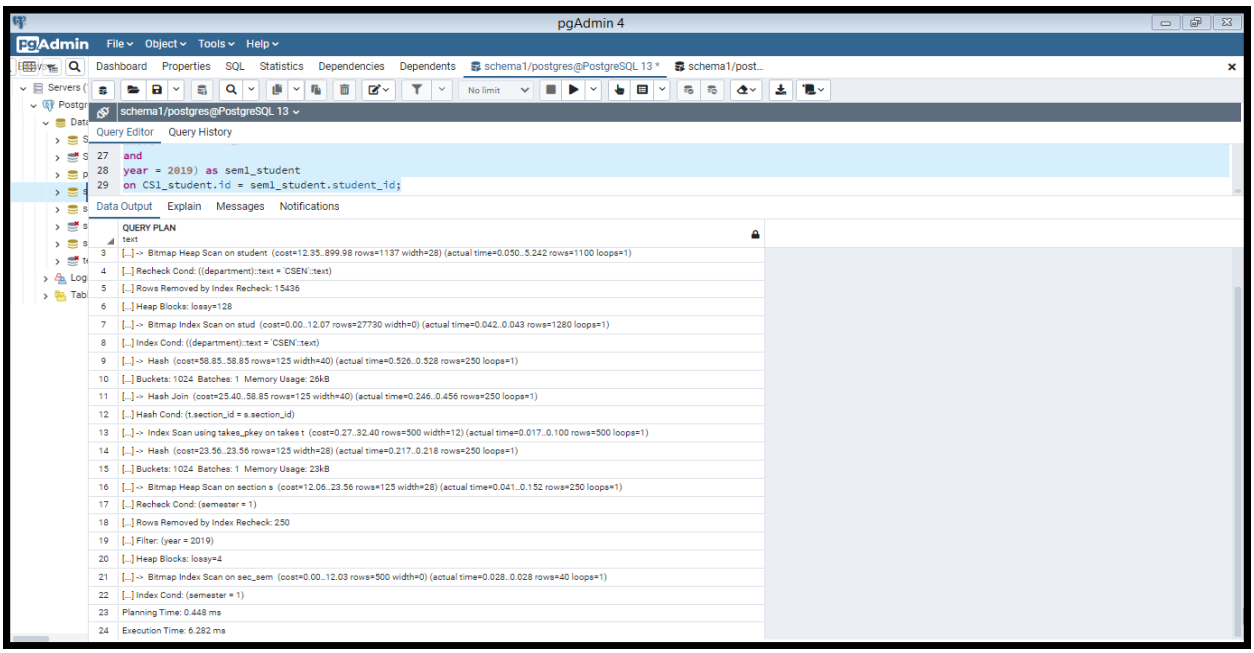


Here the hash index increased performance as it search in O(1) time so it helped in searching from semster=1 and year=2019

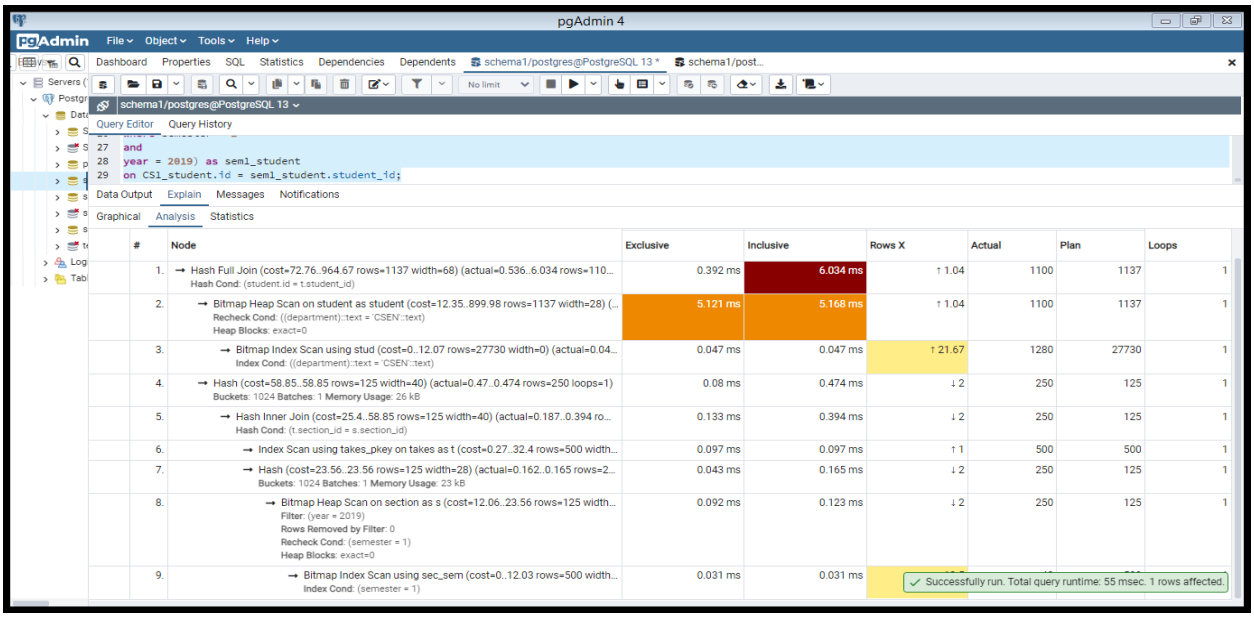
Query1 using BRIN indecies on : Student(Department) , section(year) , section(semester), takes(section_id)

Disabled flags : seqscan

Planning and execution time :

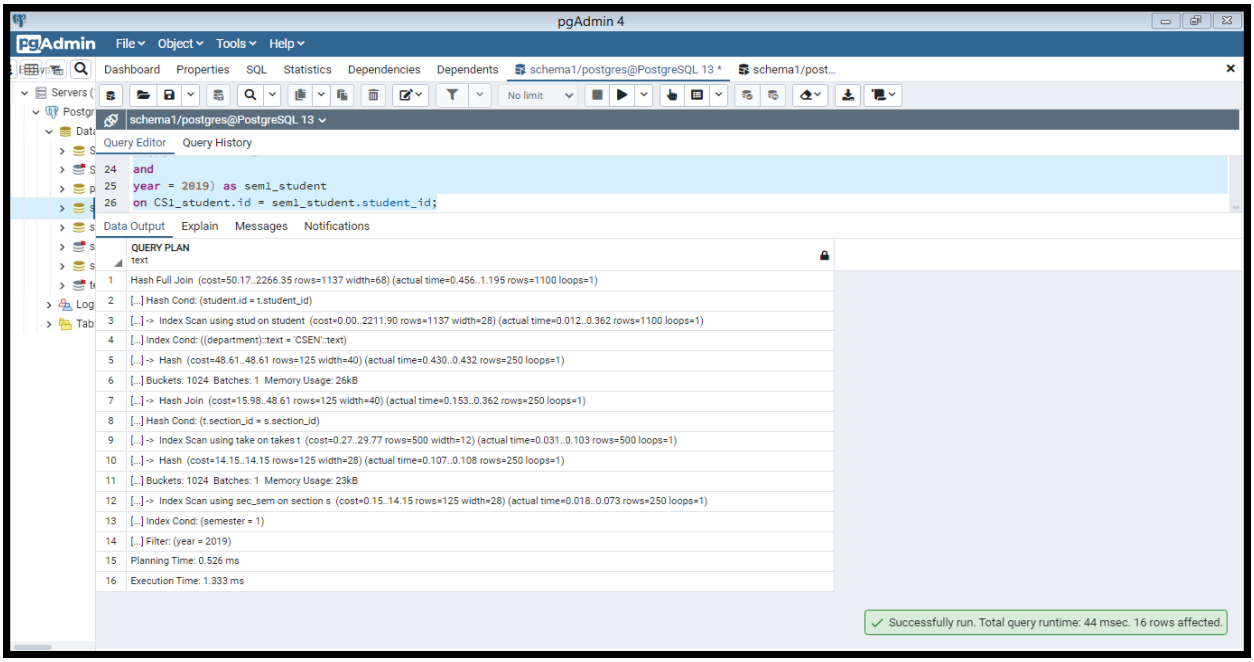


Execution plan and costs:

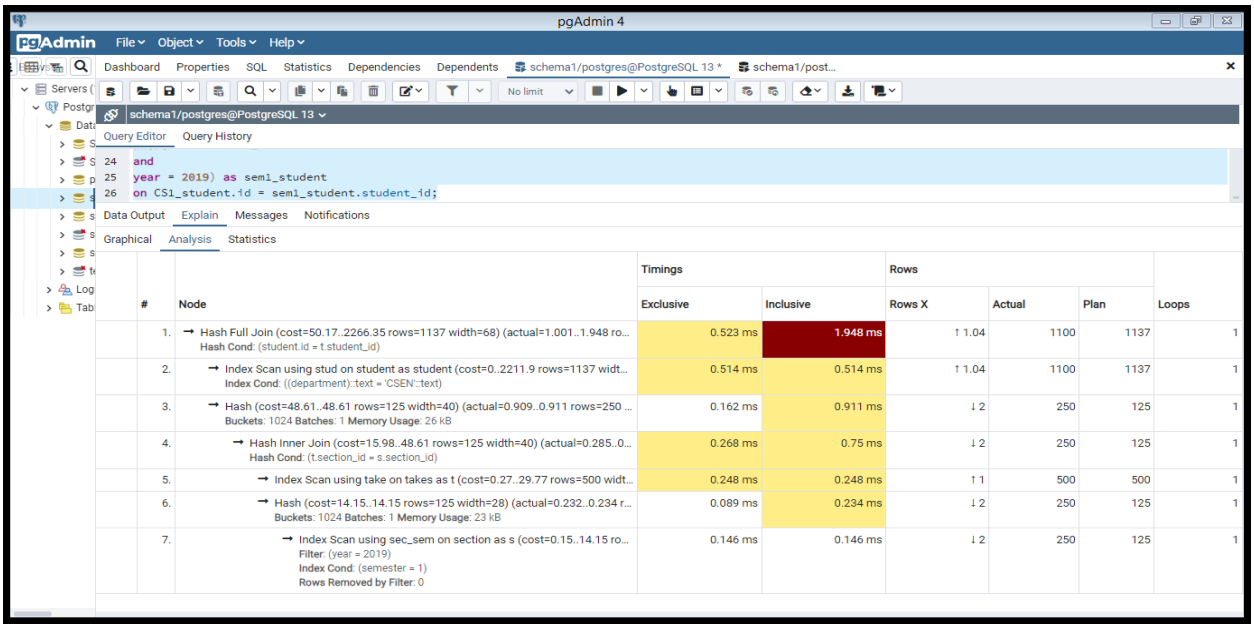


The BRIN index here increased performance a little bit as BRIN is used in small range queries in big ranges so it does not help that much in exact value queries

Query1 using mixed indecies : Hash indecies on : Student(Department),section(year) and B+ tree on : section(semester) , takes(section_id)
Disabled flags: seqscan and bitmapscan
Planning and execution time:



Execution plan and costs:



Both B+ tree and hash increased performance as the search in O(logn) and O(1) time

Query1 after optimization :

select * from

((select * from section where year=2019 and semester=1) as s inner join takes t on t.section_id = s.section_id) as sem1

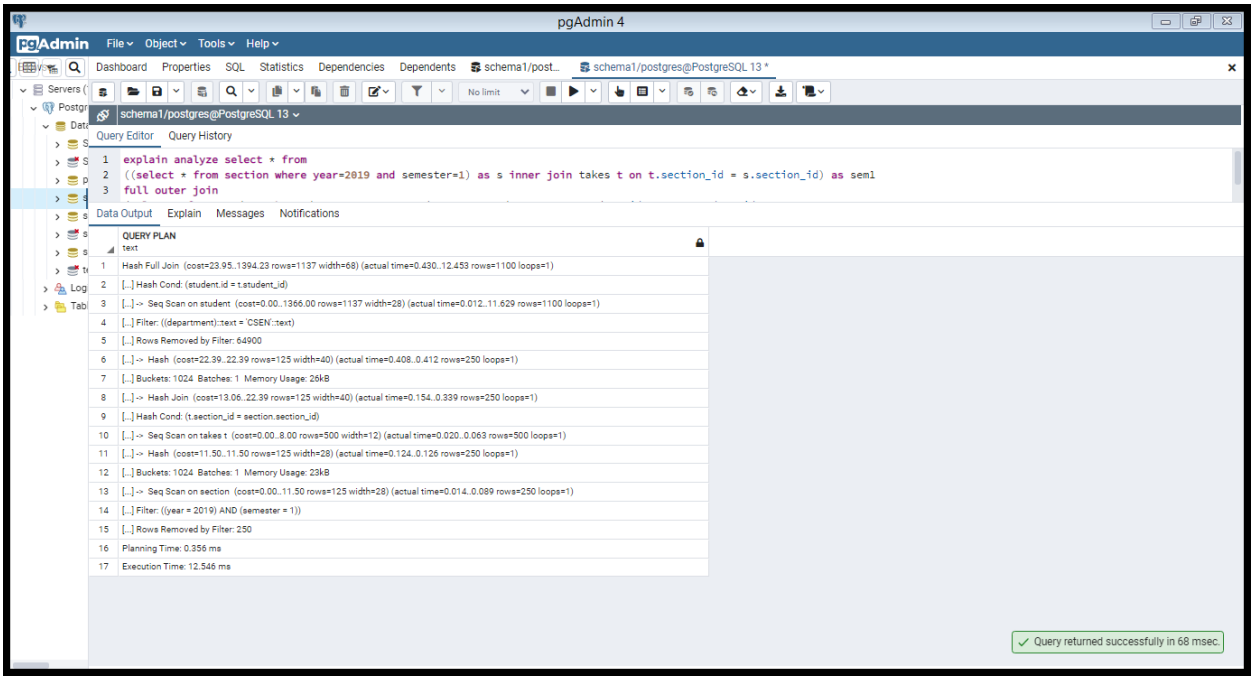
full outer join

(select * from student where department = 'CSEN') as CS1_student on CS1_student.id = sem1.student_id;

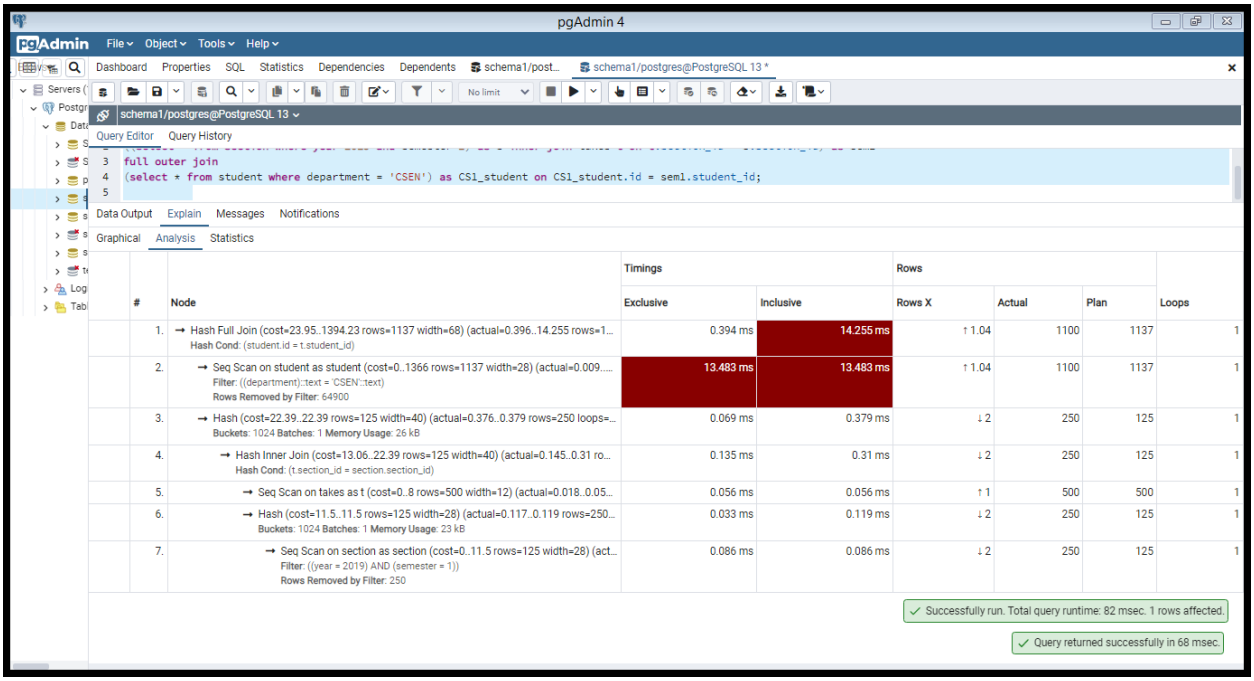
We pushed down selections

Query1 Optimized with no idex:

Planning and execution time:



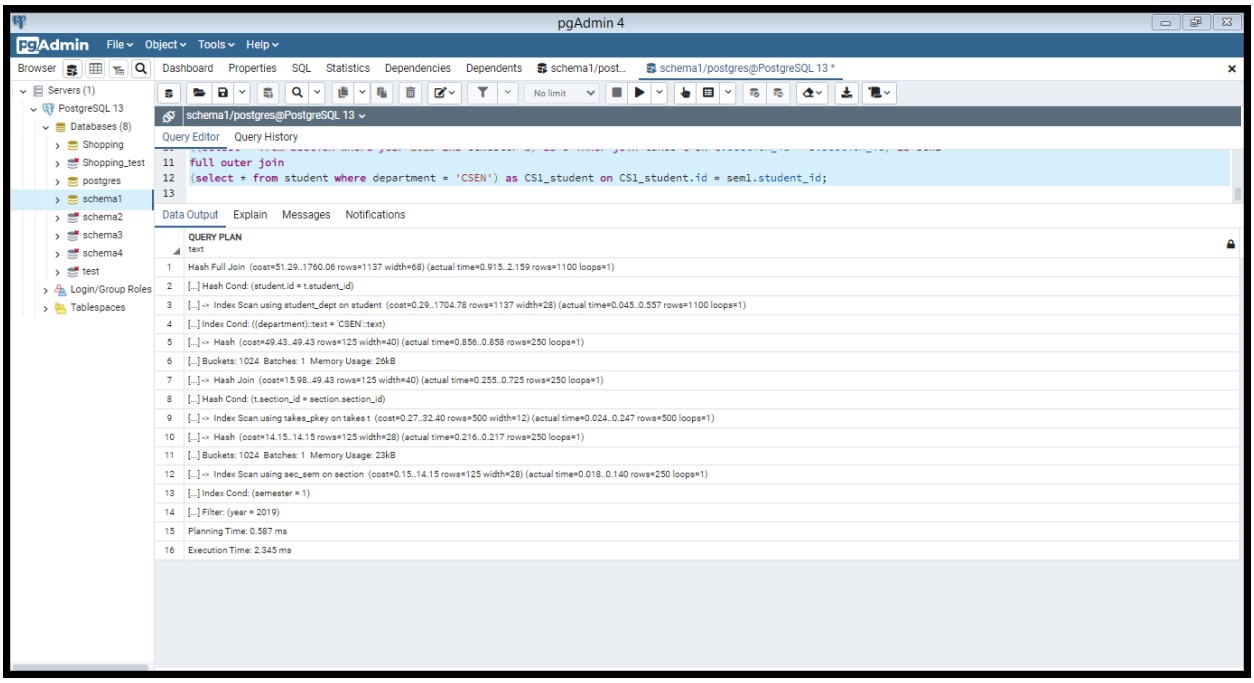
Execution plan and costs :



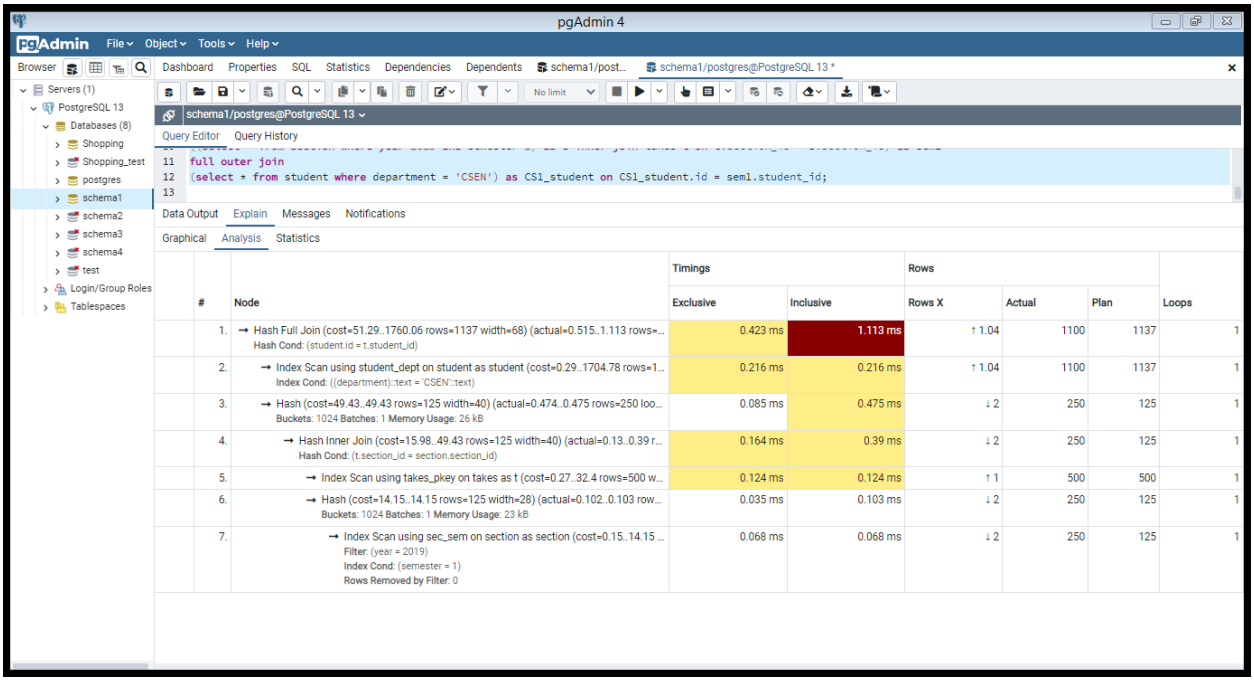
Query1 optimized using B+ tree on : Student(department) , section(year) , section(semester)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



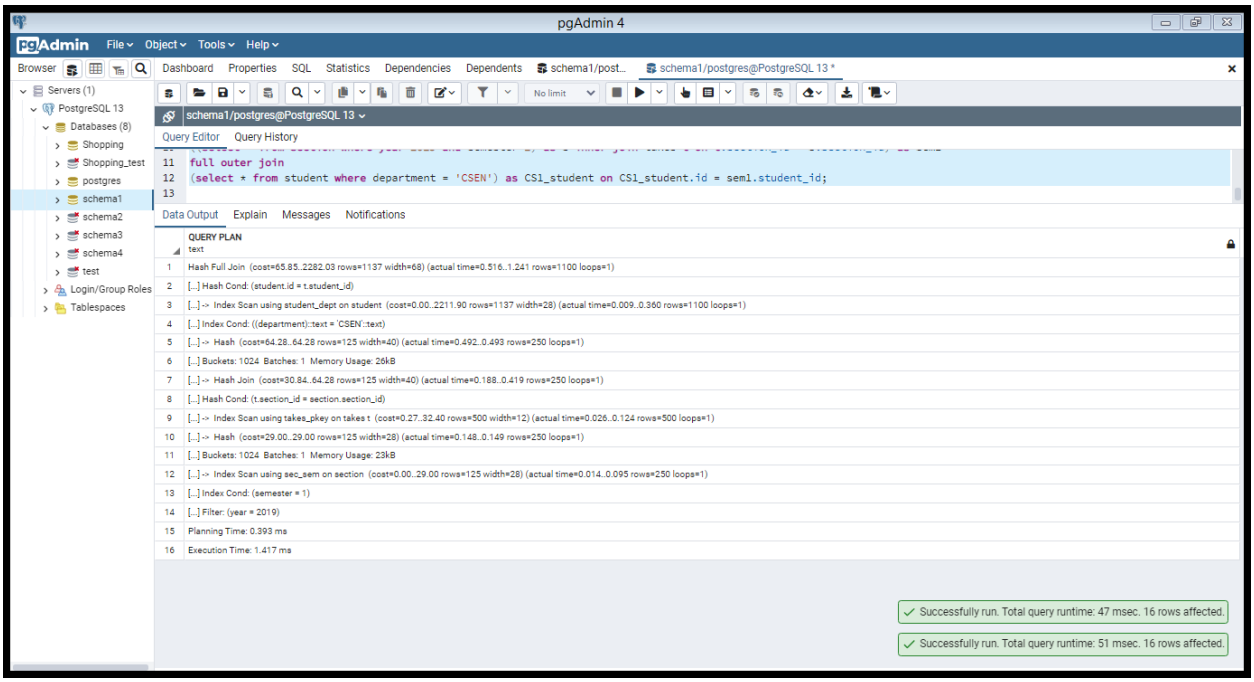
Execution plan and costs:



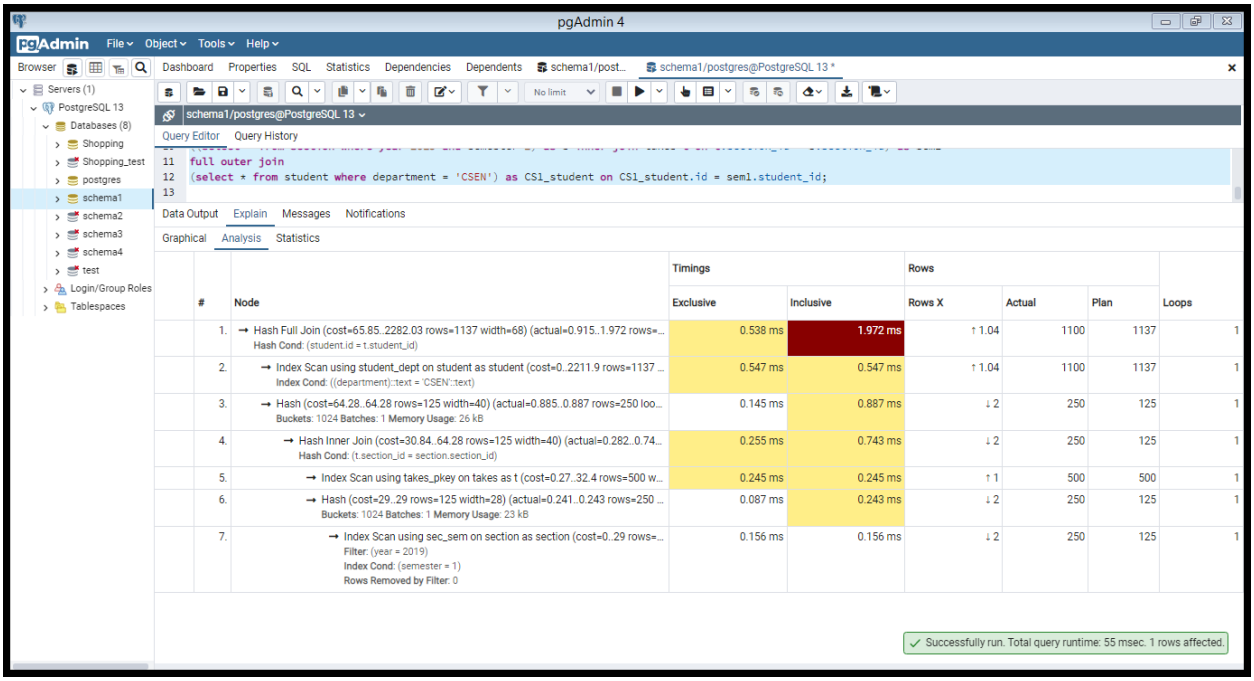
Query1 optimized using hash indecies on : Student(department) , section(year) , section(semester)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



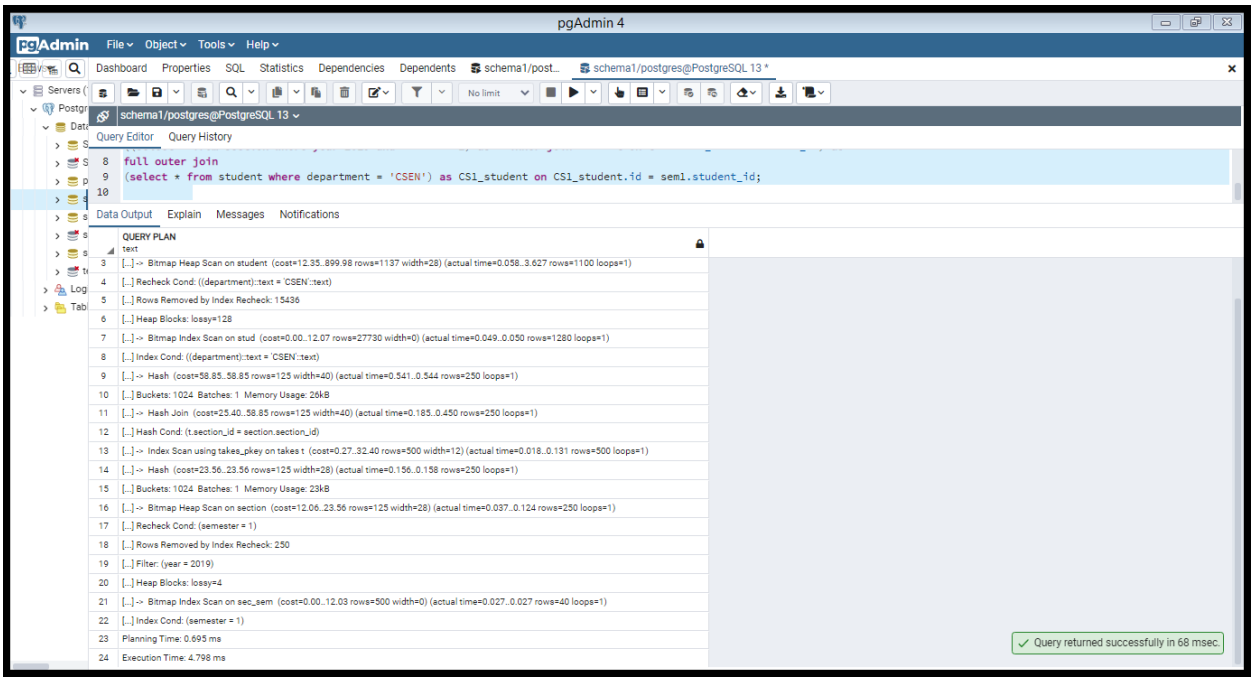
Execution plan and costs:



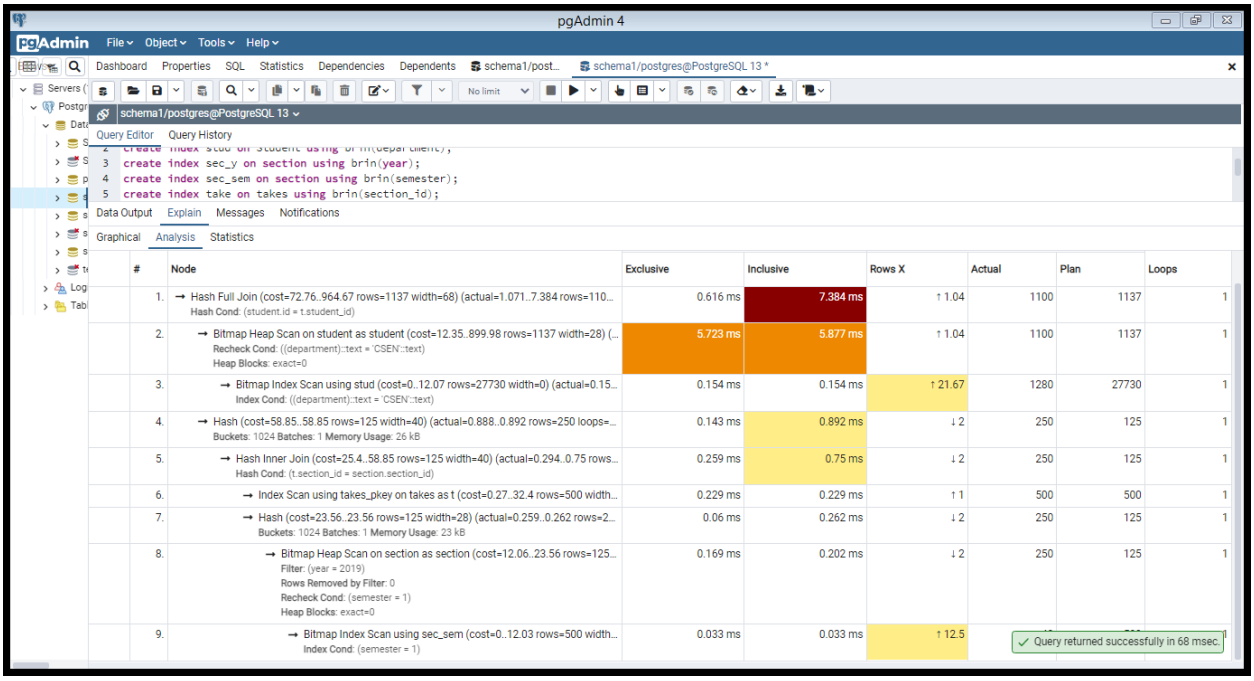
Query1 optimized using BRIN indecies on : Student(Department) , section(year) , section(semester),takes(section_id)

Disabled flags : seqscan

Planning and execution time:



Execution plan and costs:

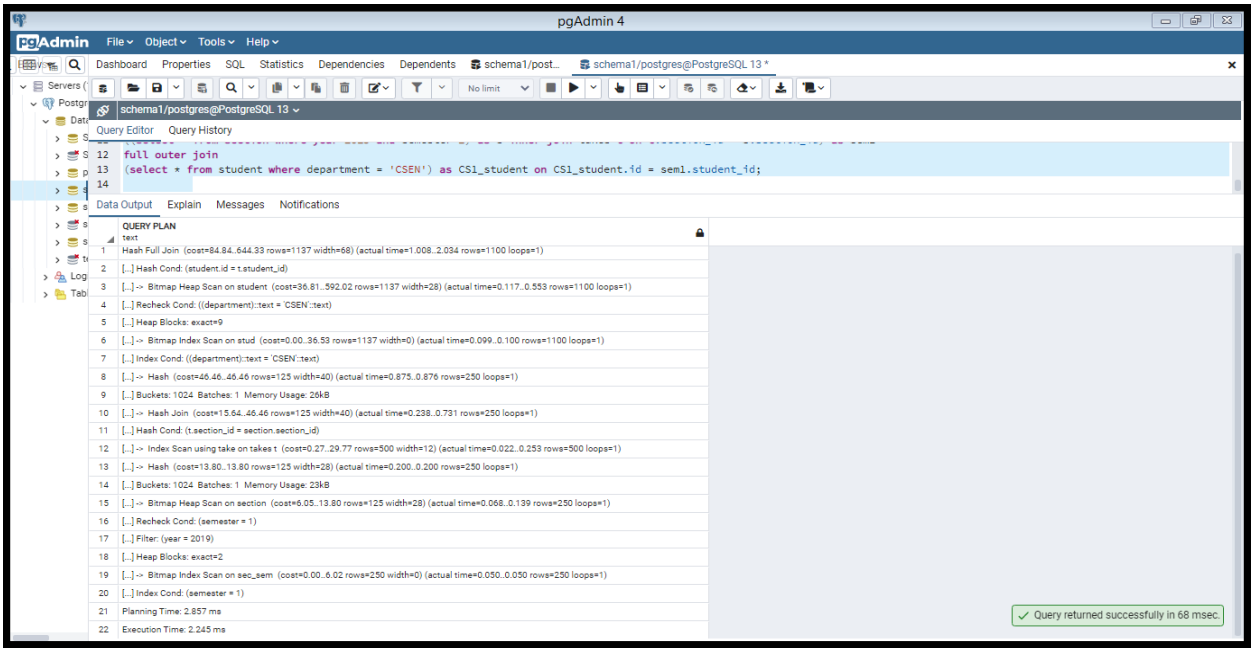


Query1 optimized using mixed indecies :

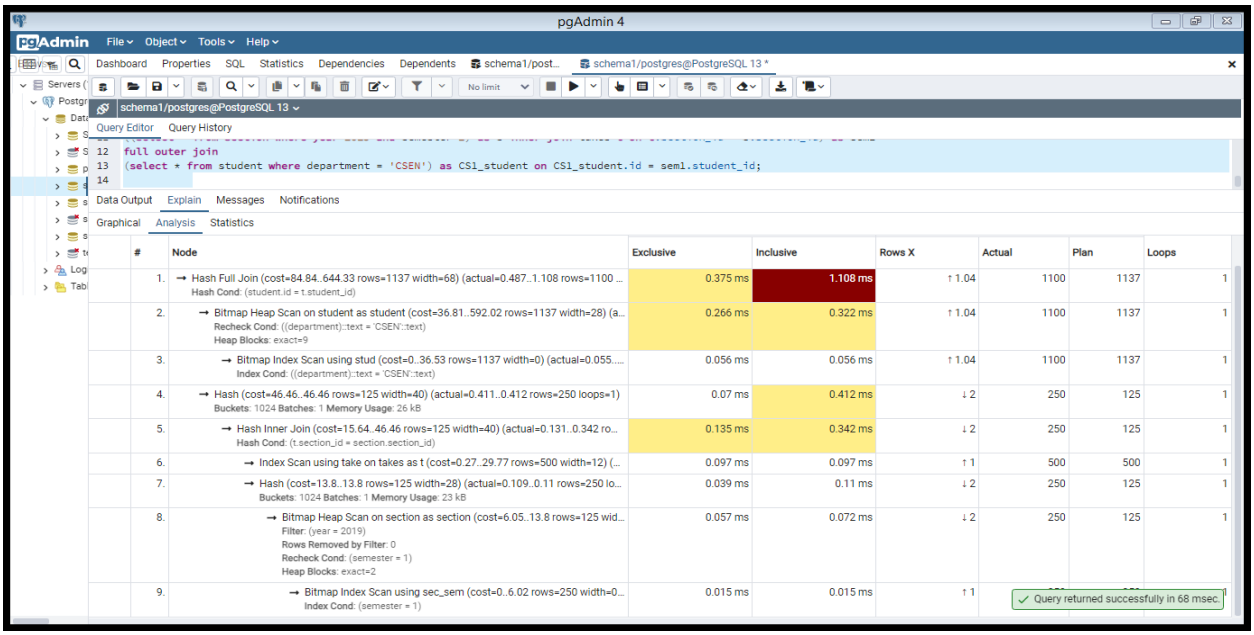
Hash indecies on : Student(Department),section(year) and B+ tree on : section(semester),takes(section_id)

Disabled flags : seqscan and bitmapscan

Planning and execution time:

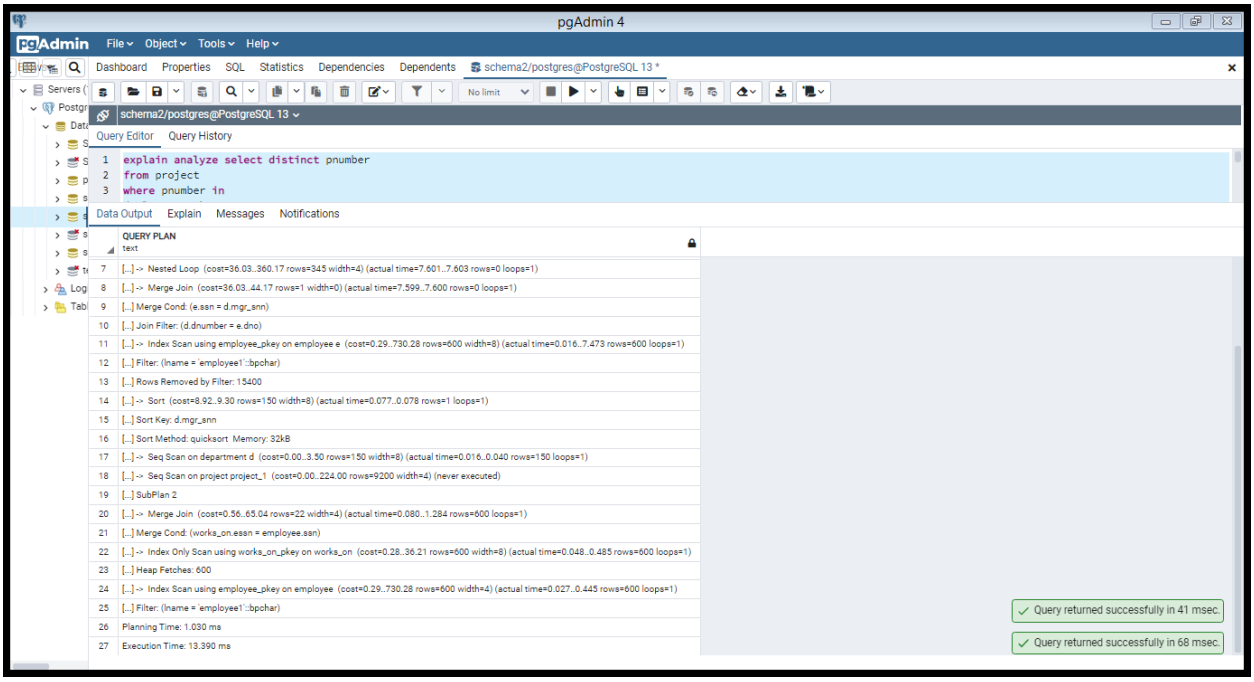


Execution plan and costs:

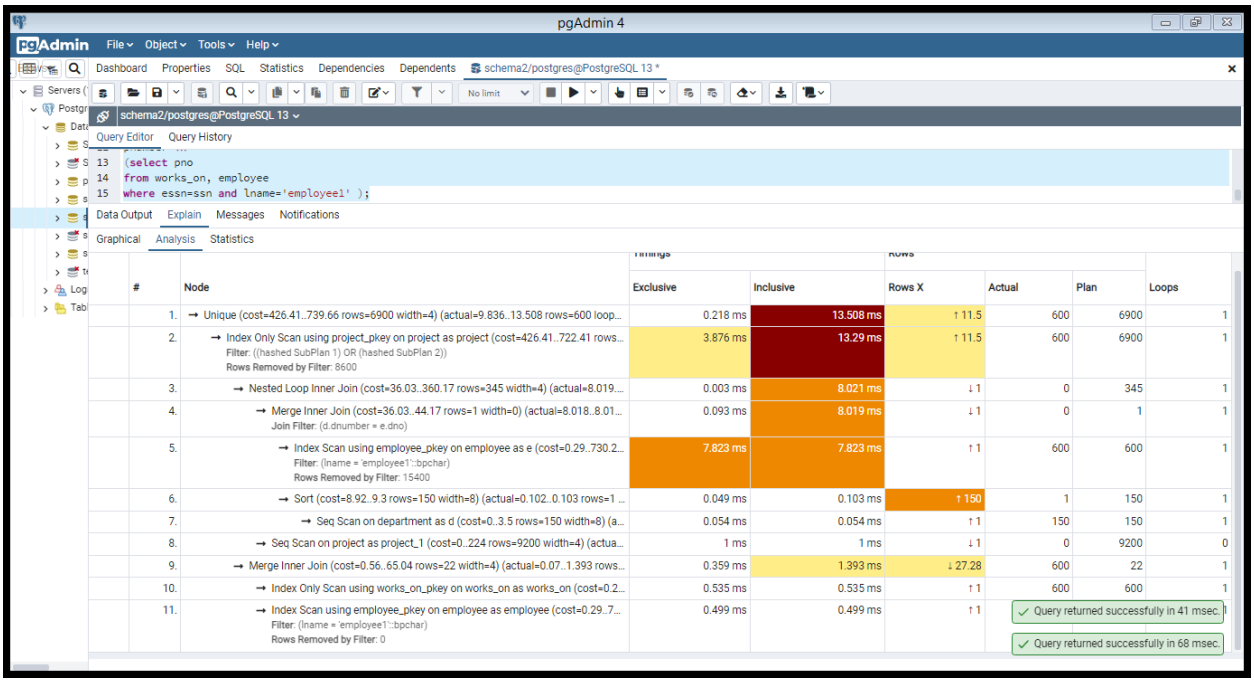


Query2 with no index:

Planning and execution time:



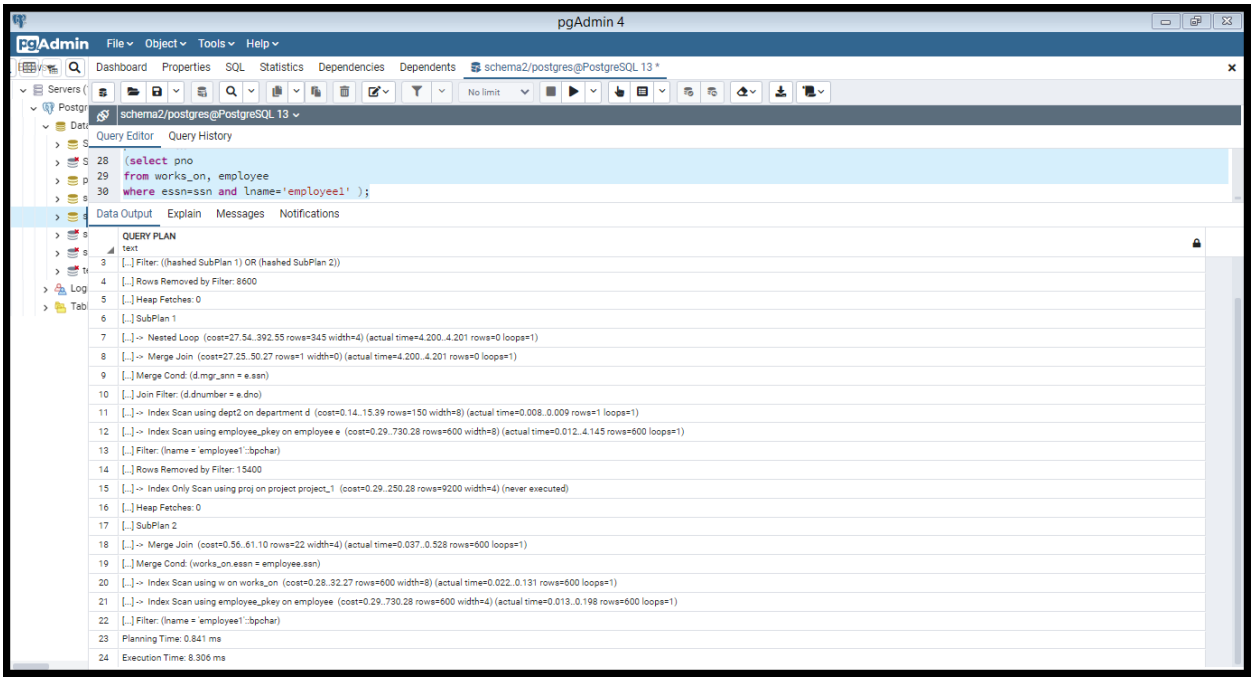
Execution plan and costs:



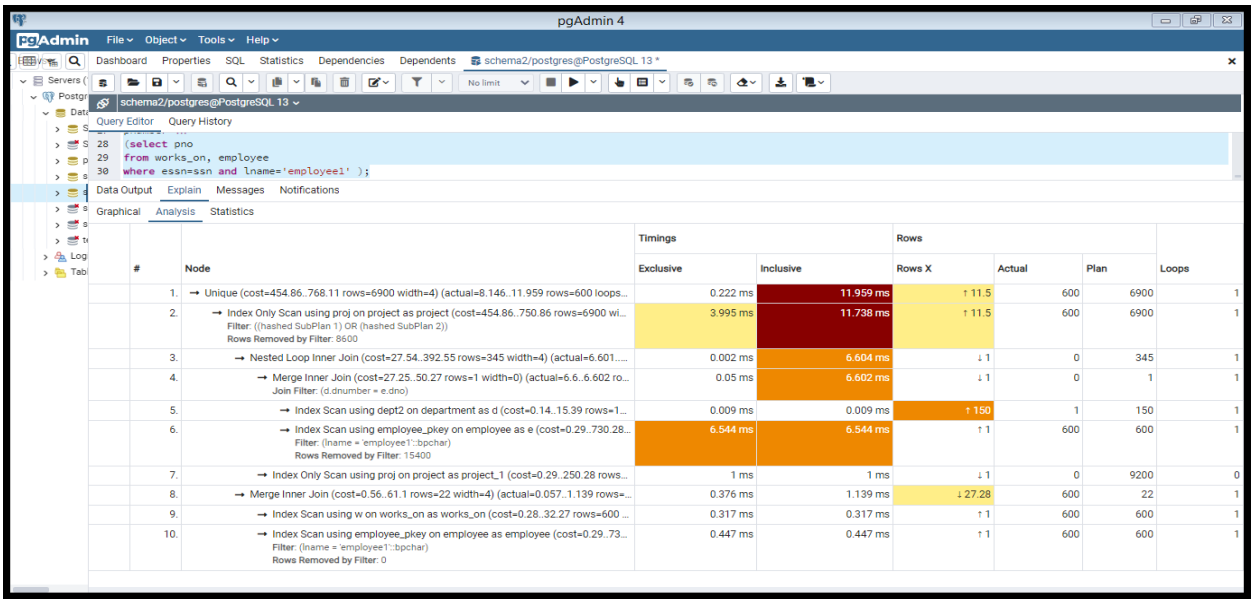
Query2 using B+ tree on : Department(Dnumber),Department(mgr_snn), Project(Pnumber) , Employee(Lname), Employee(dno), works_on(essn)

Disabled flags : seqscan , bitmapscan

Planning and execution time:



Execution plan and costs:

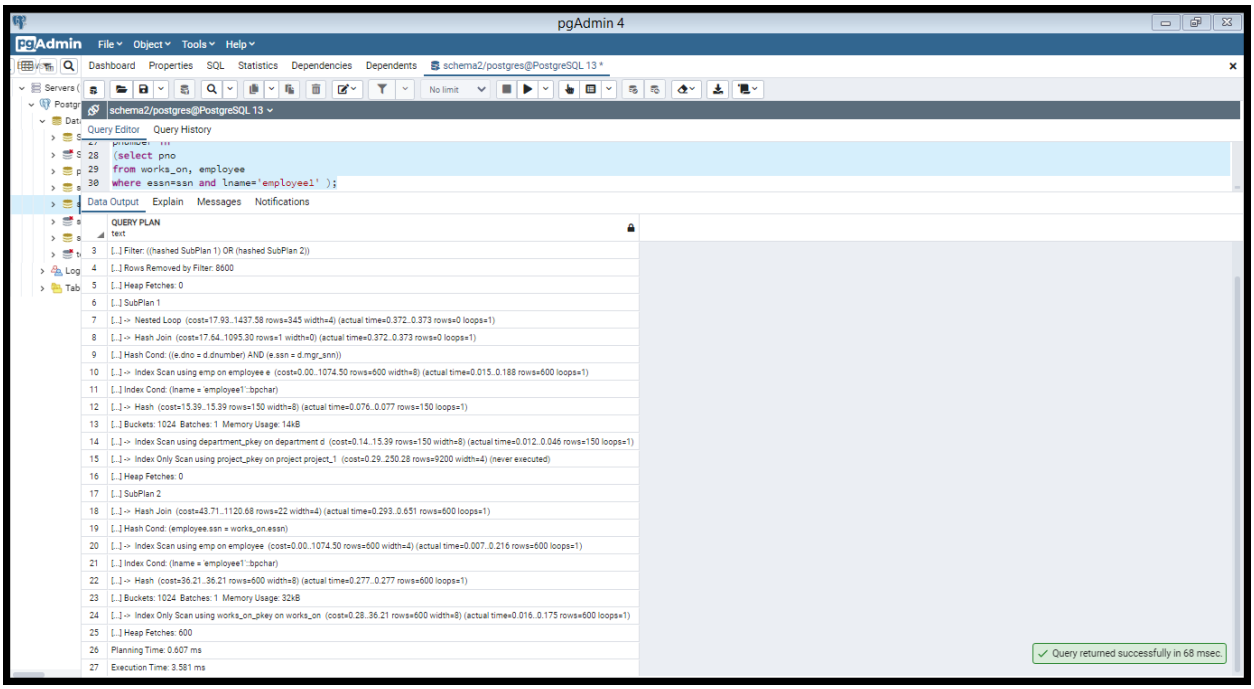


Here the B+ tree helped in the (in) statement because we check every Pnumber also it helped in the join between Employee and department as B+ tree search in O(logn)

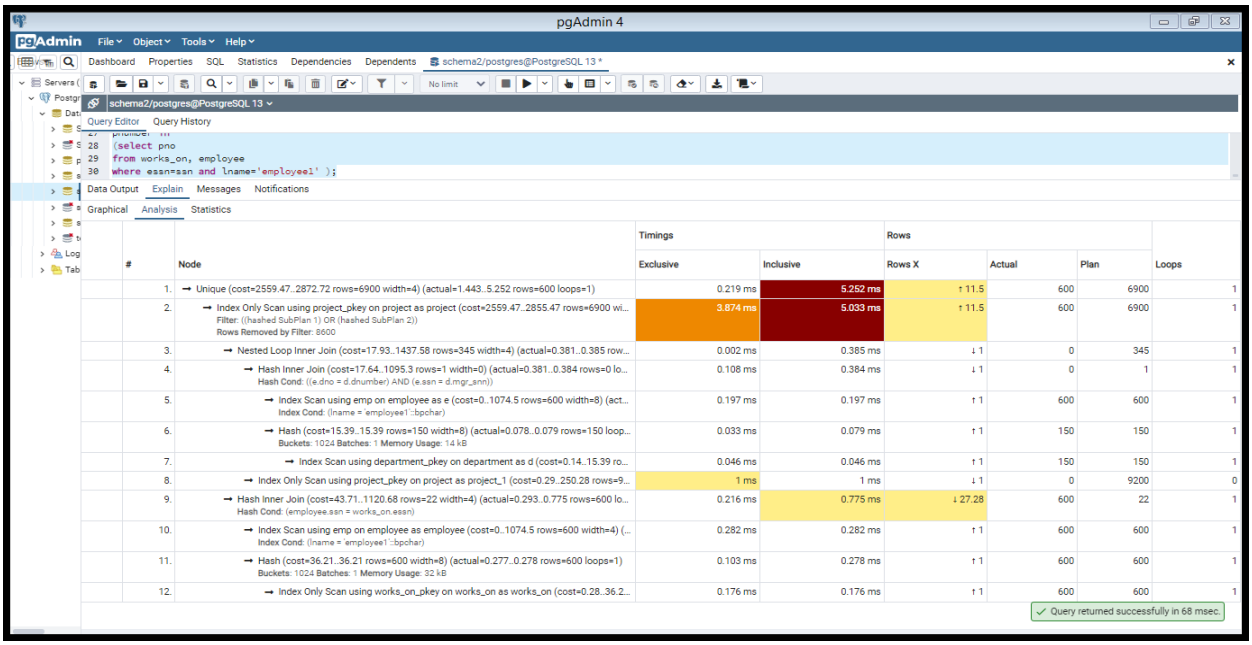
Query2 using hash indecies on : Employee(Lname), Employee(dno)

Disabled flags: seqscan,bitmapscan

Planning and execution time:



Execution plan and costs:

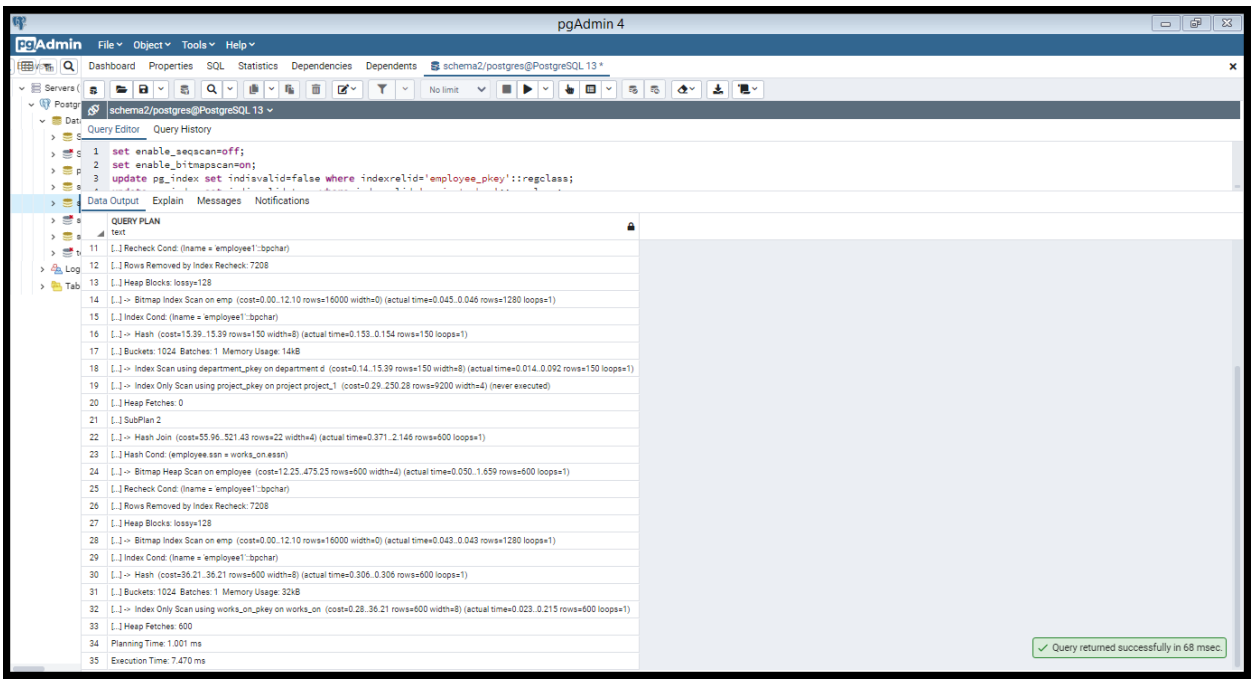


Here the hash helped in finding the Employee with lname=”employee1” as it search in O(1) time

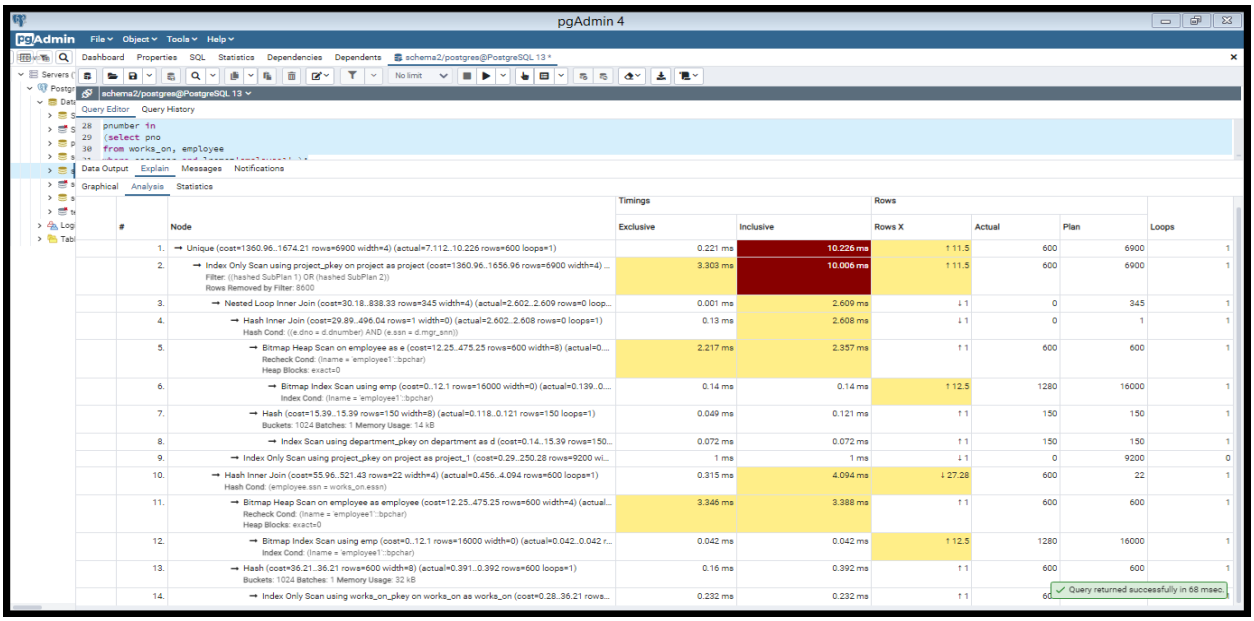
Query2 using BRIN indecies on : Department(Dnumber) , Employee(Lname),Employee(dno)

Disabled flags: seqscan

Planning and execution time:



Execution plan and costs:

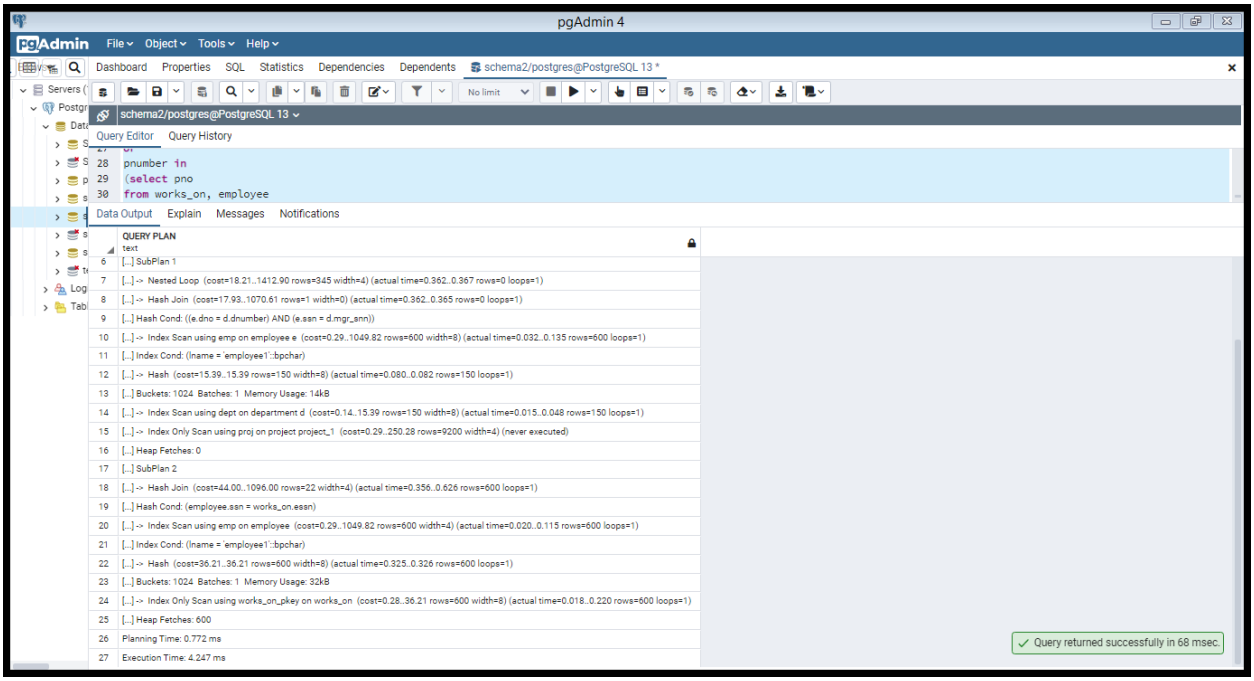


Here the Brin helped in finding the Employee name as it sorted the names and then found the needed name : employee1 as brin is used in small range queries in sorted data

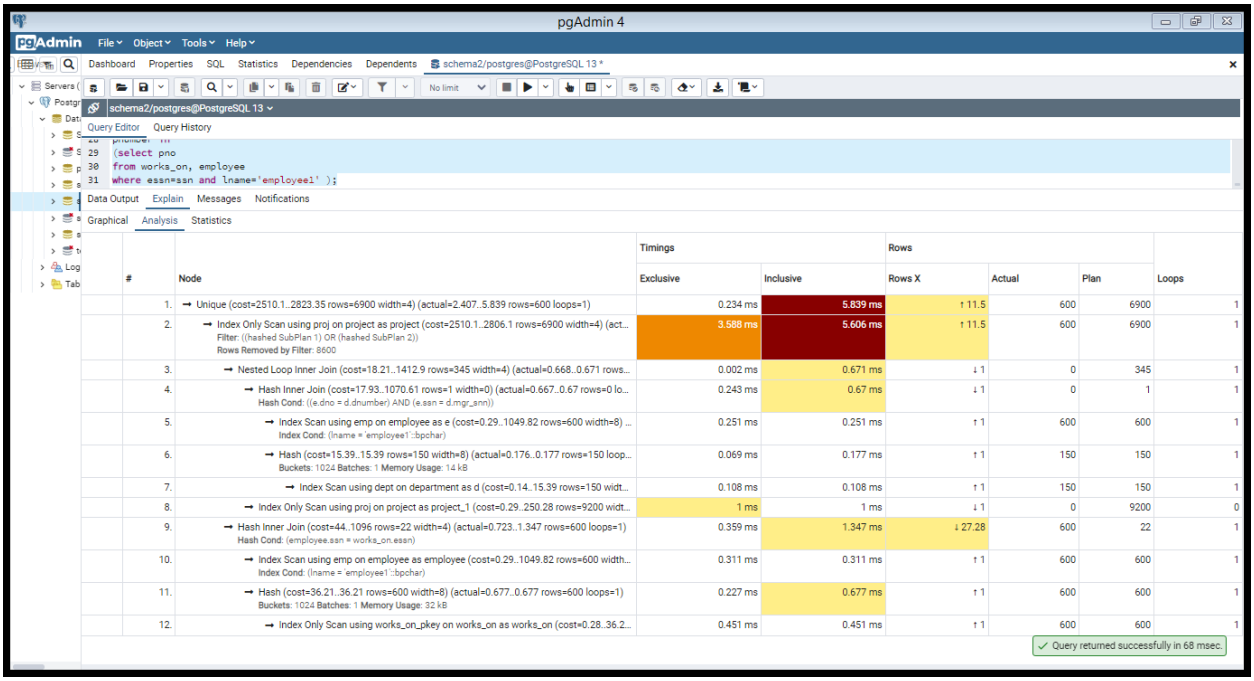
Query2 using mixed indecies :

Disabled flags: seqscan and bitmapscan

Hash index on : Employee(dno)
B+ tree on : Employee(Lname), Project(Pnumber),Department(dnumber)
Planning and execution time:



Execution plan and costs:



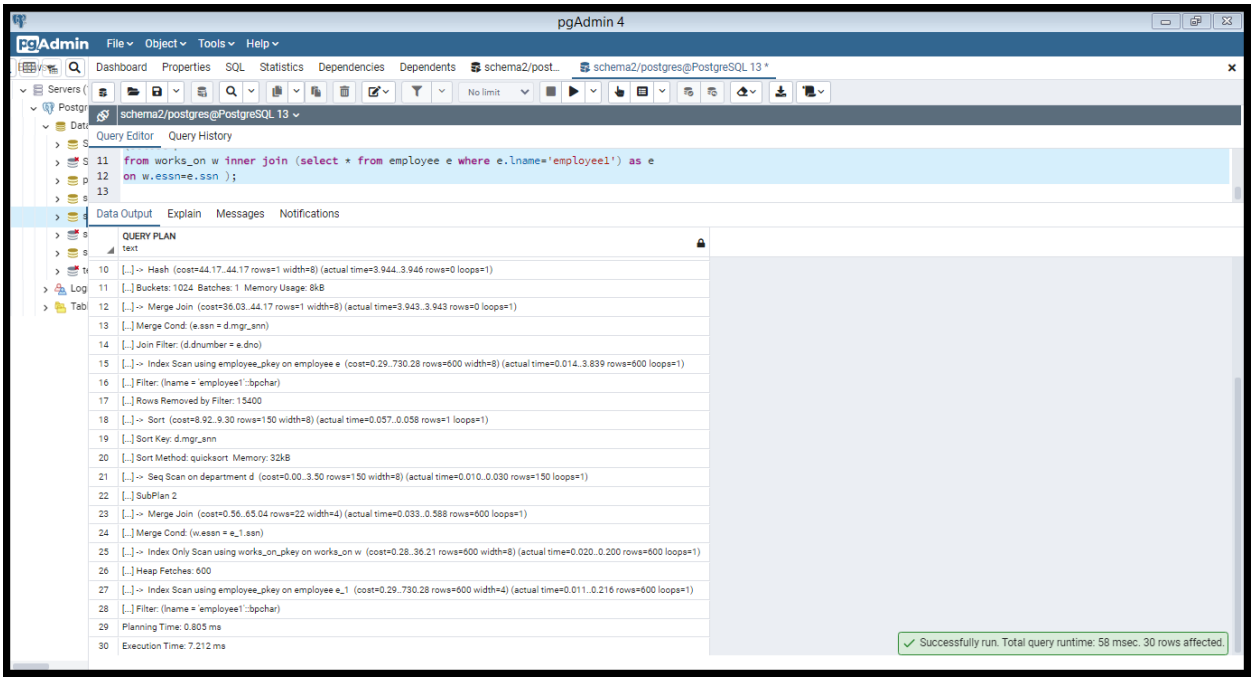
Query2 after optimization:

```
select distinct pnumber

from project

where pnumber in

(select pnumber

from ((select Pnumber,Dnumber as p1 from project) as p inner join  department d on p.p1=d.Dnumber) as d inner join  (select * from employee e where e.lname='employee1') as e

on e.dno=d.dnumber and  d.mgr_snn=e.ssn

)

or

pnumber in

(select pno

from works_on w inner join (select * from employee e where e.lname='employee1') as e

on w.essn=e.ssn );
```

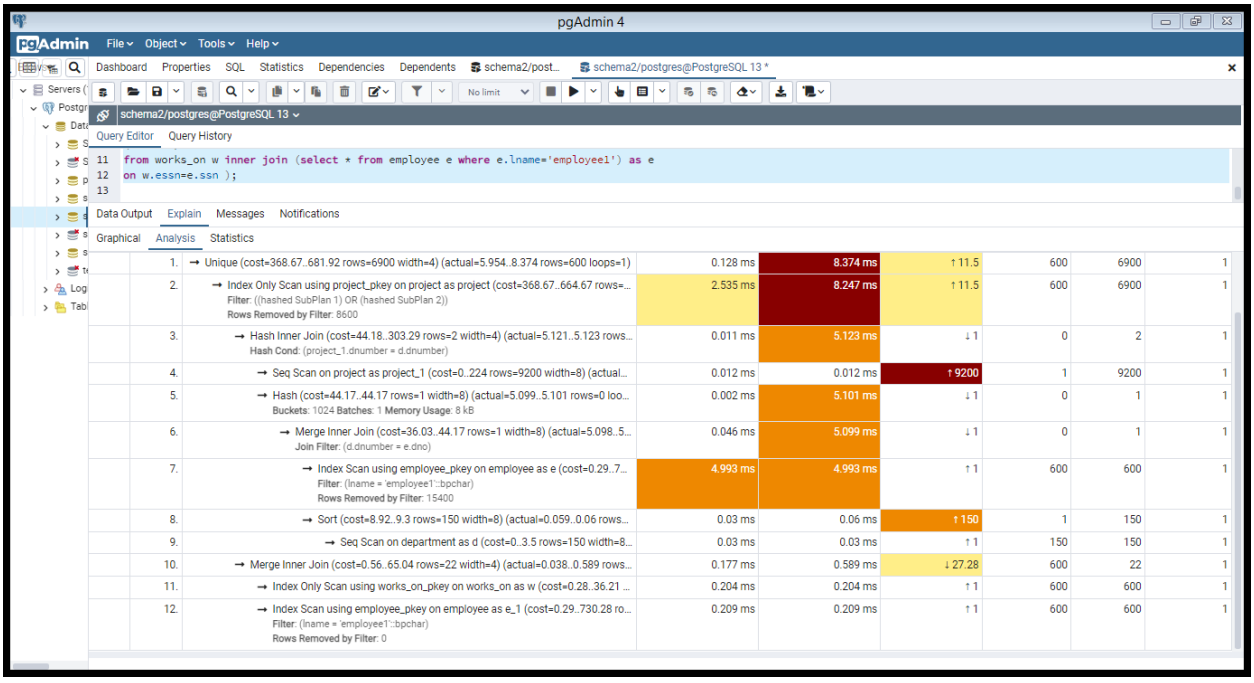
We changed the cross product with inner joins

Query2 optimized with no index:

Planning and execution time :



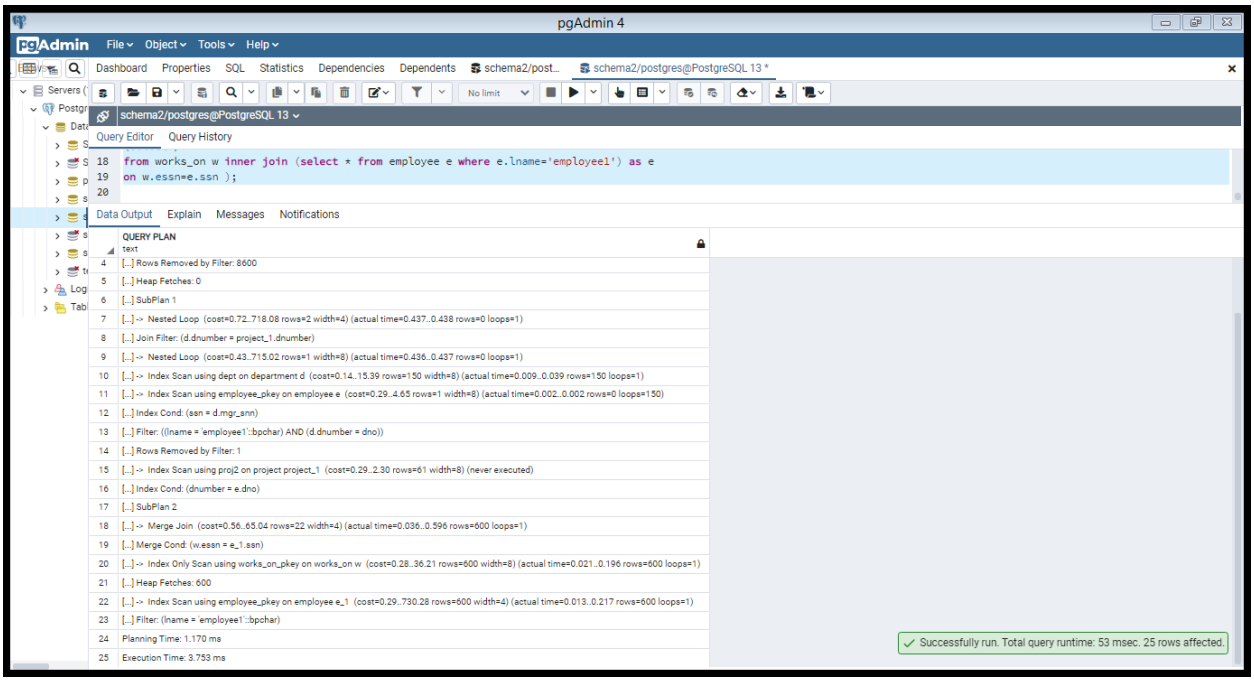
Execution plan and costs:



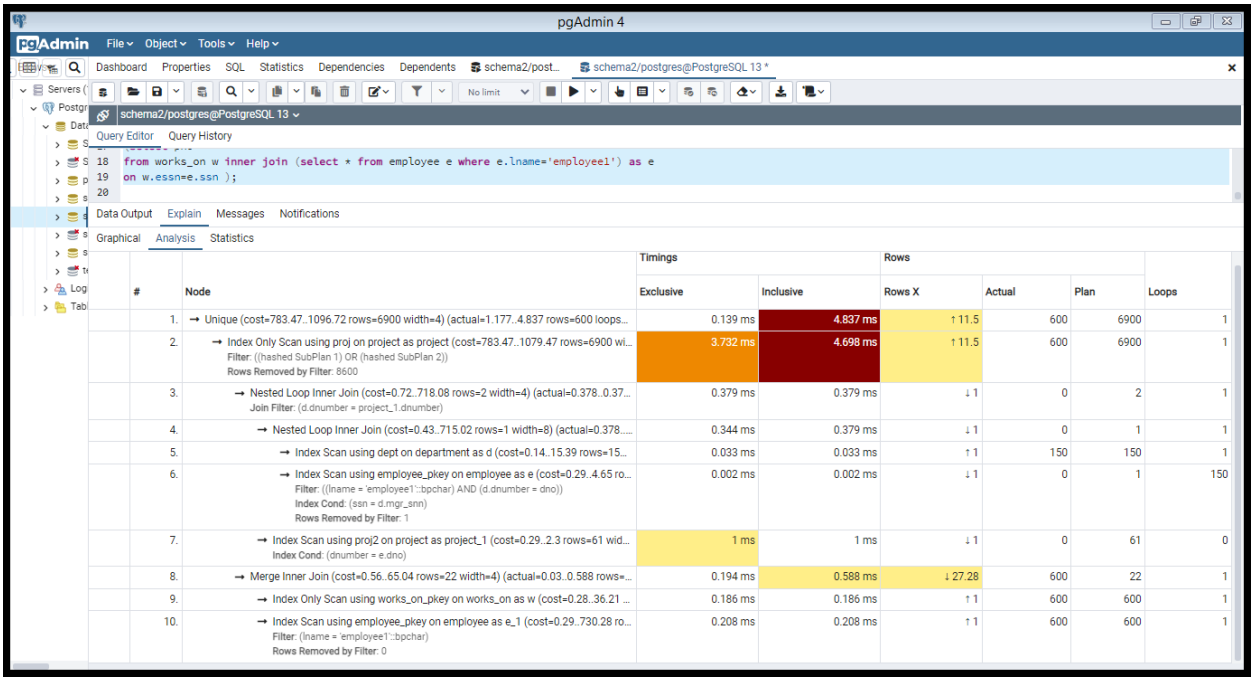
Query2 optimized using B+ tree on : Department(Dnumber) , Project(Dnumber) , Project(Pnumber) , Employee(Lname)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



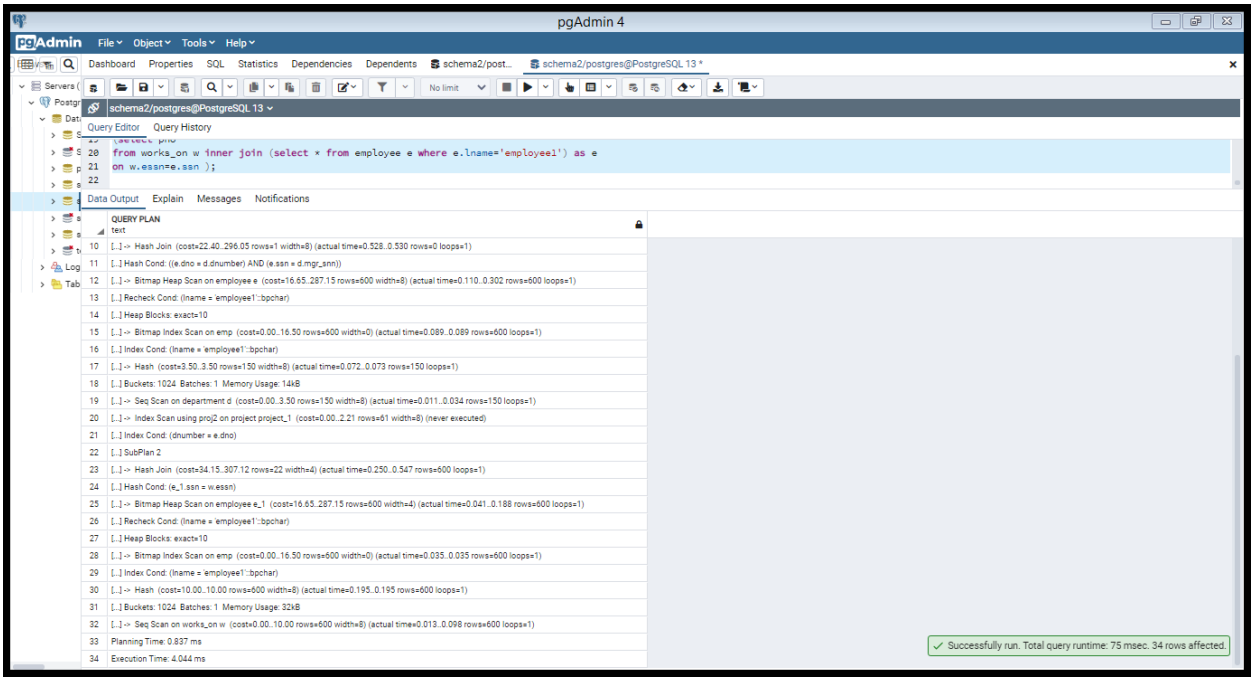
Execution plan and costs:



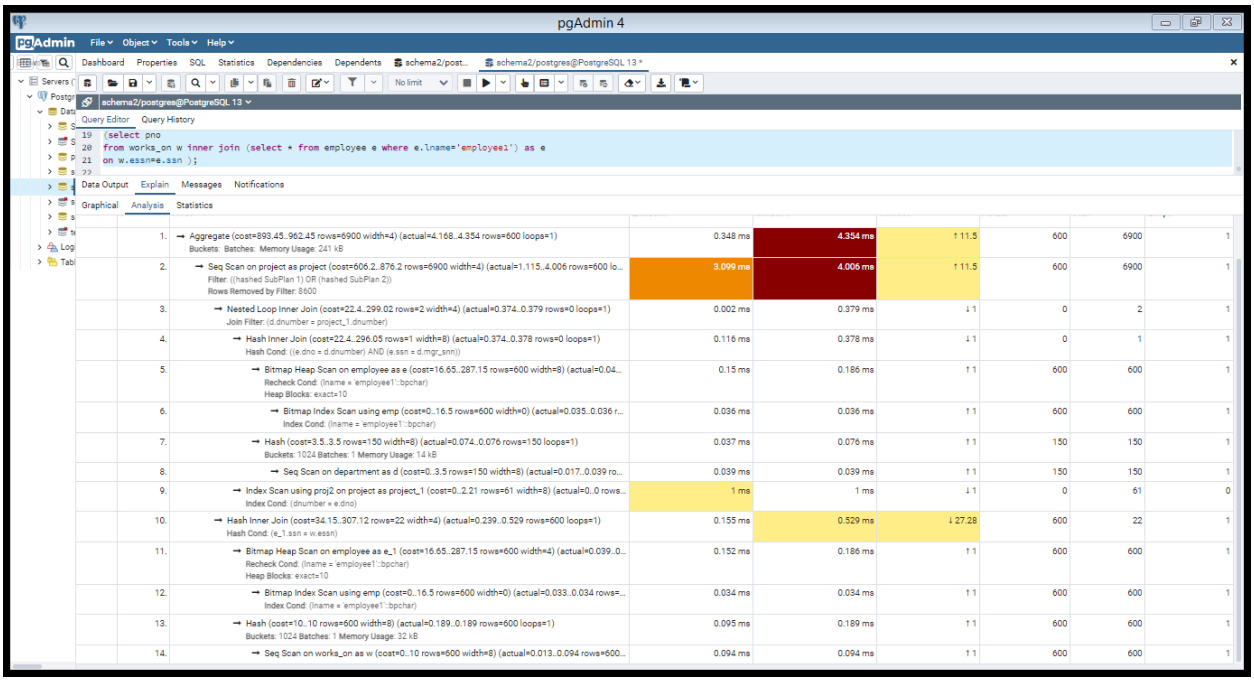
Query2 optimized using hash indecies on : Department(Dnumber) , Project(Pnumber) , Project(Dnumber), Employee(Lname)

Disabled flags: seqscan and bitmapscan

planning and execution time:

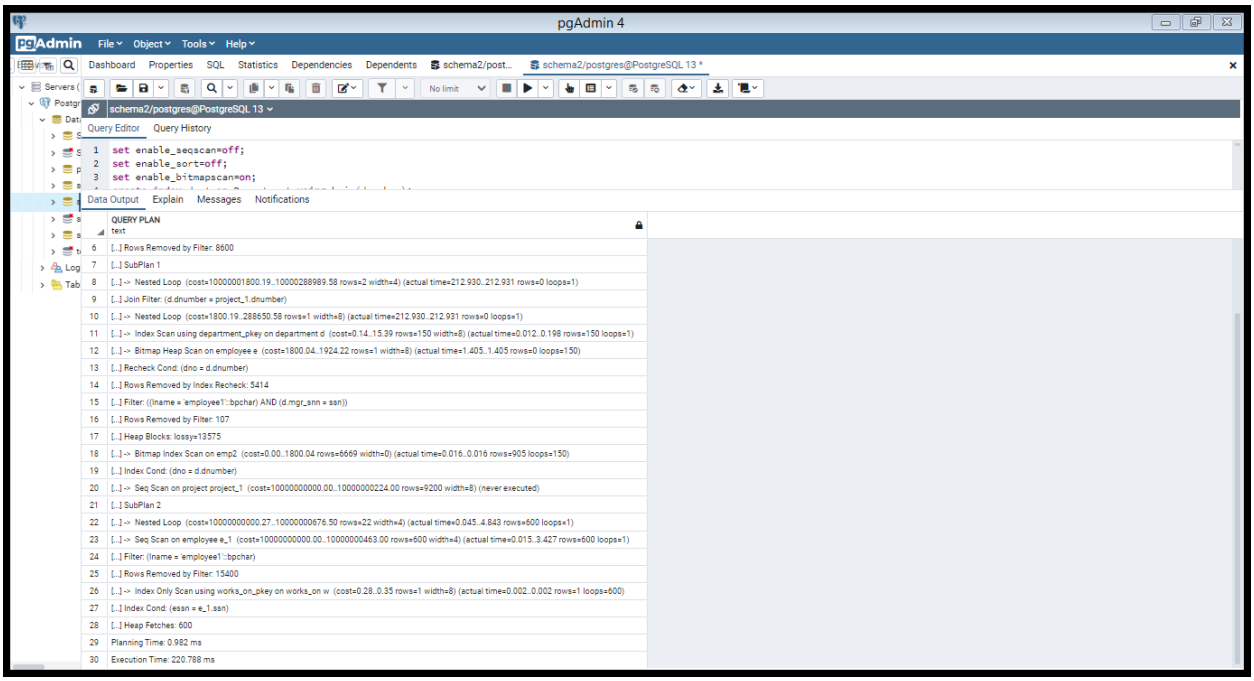


execution plan and costs:

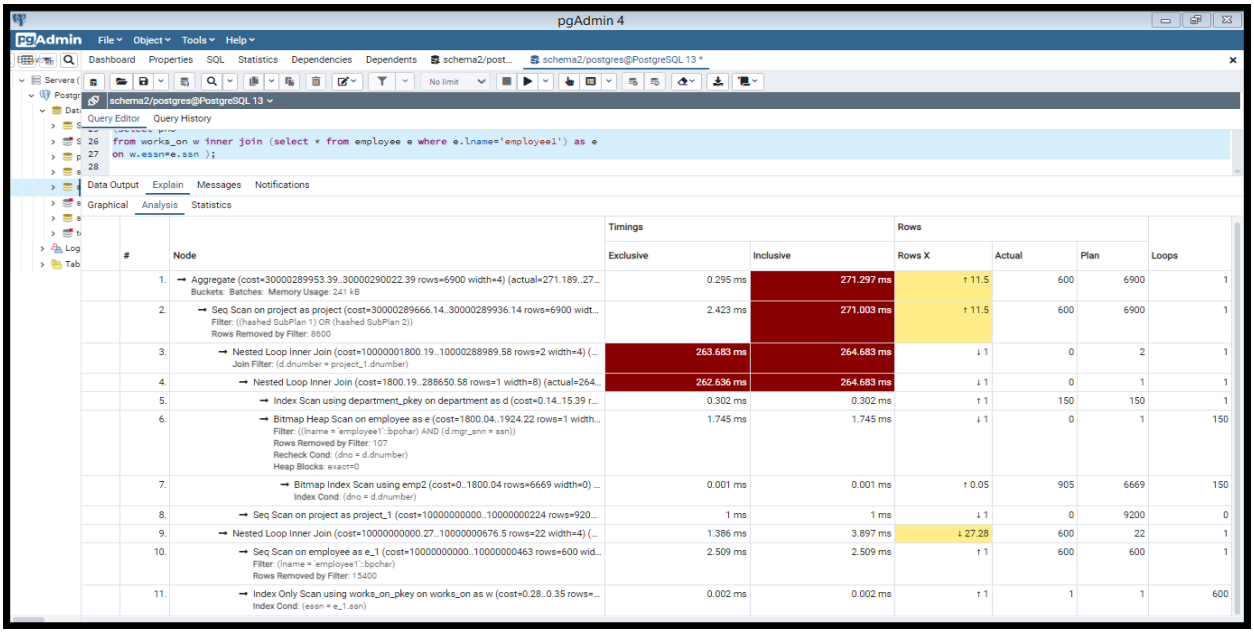


Query2 optimize using BRIN indecies on : Department(Dnumber) , Project(Pnumber) , Project(Dnumber), Employee(Lname)

Planning and execution time:



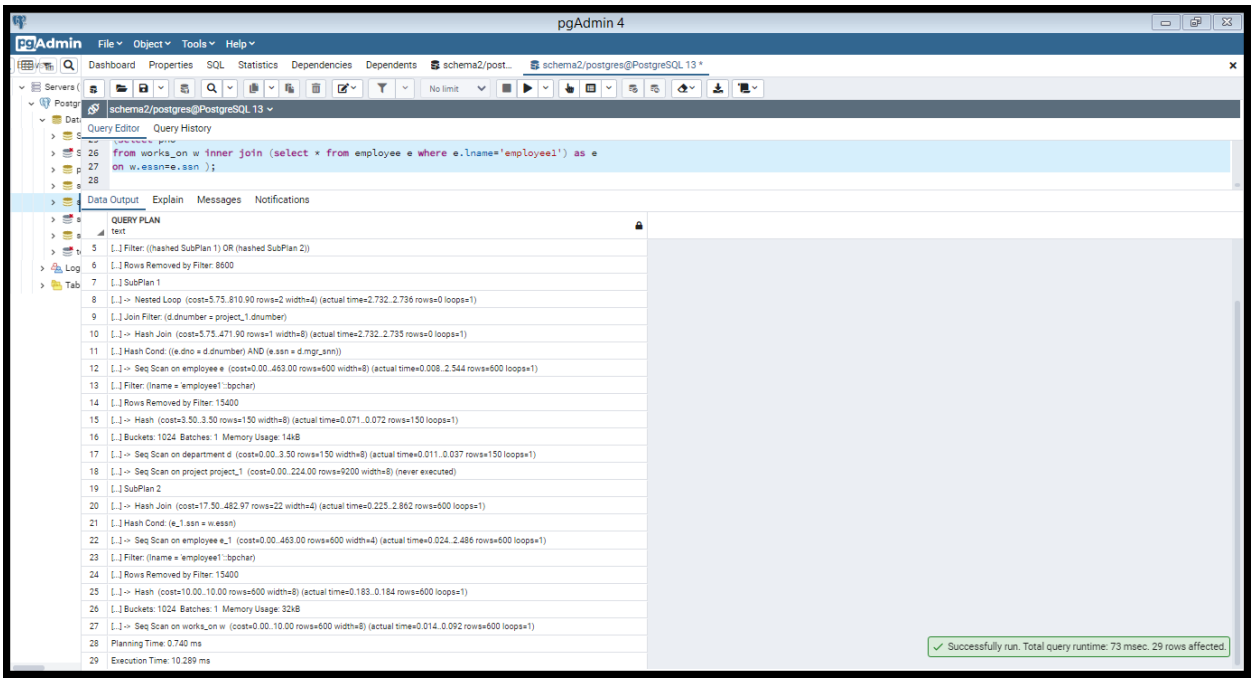
Execution plan and costs:



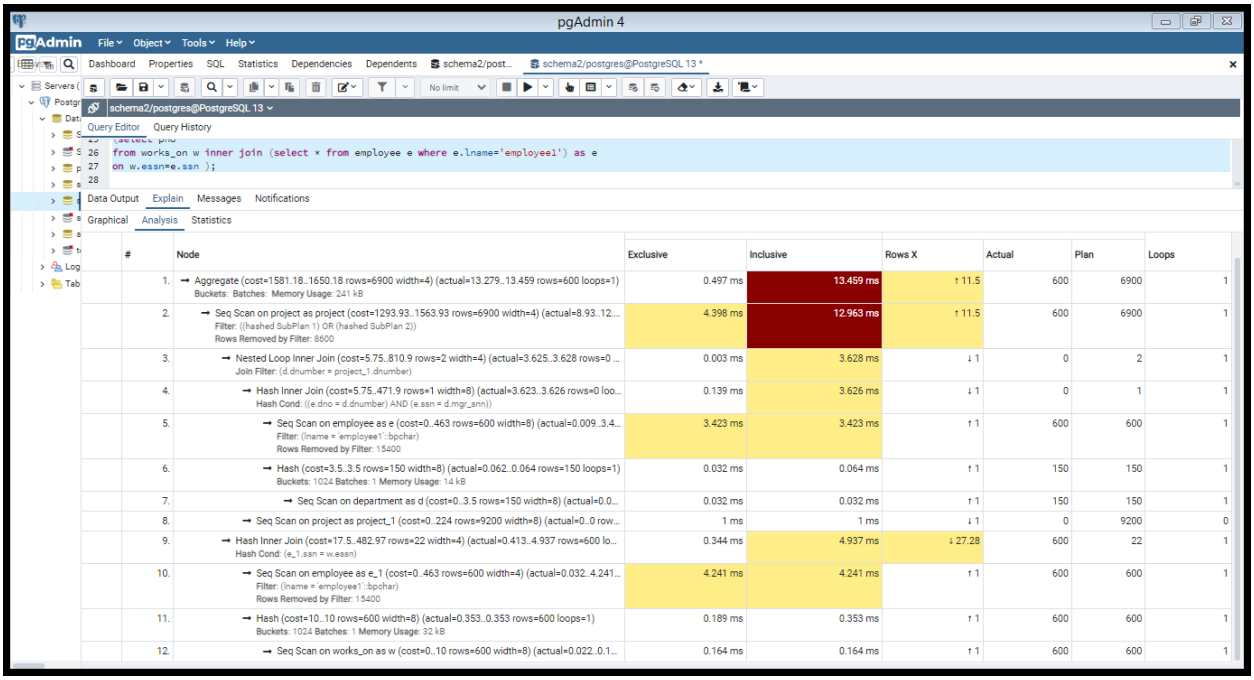
Using brin with seqscan off was very bad in time and costs as here the BRIN did not help in finding the Employee with lname=employee1 as it does not help in joins as BRIN is used in small range queries

Using brin with seqscan on

Planning and execution time:

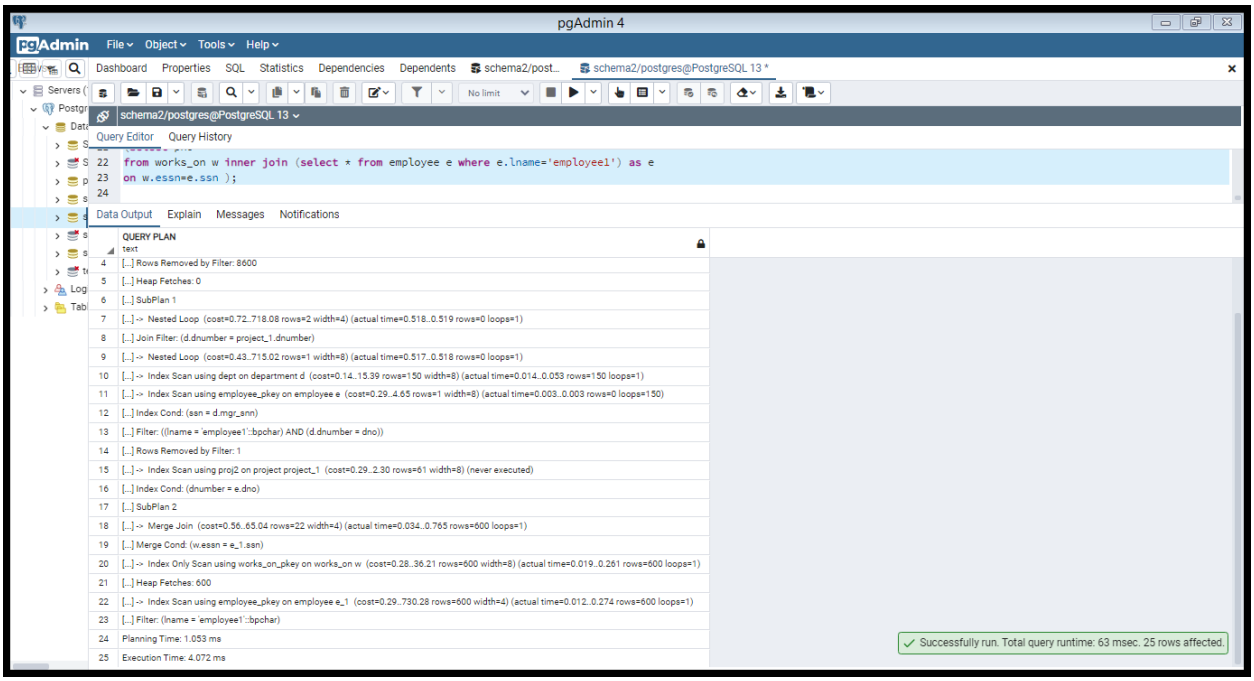


Execution plan and costs:

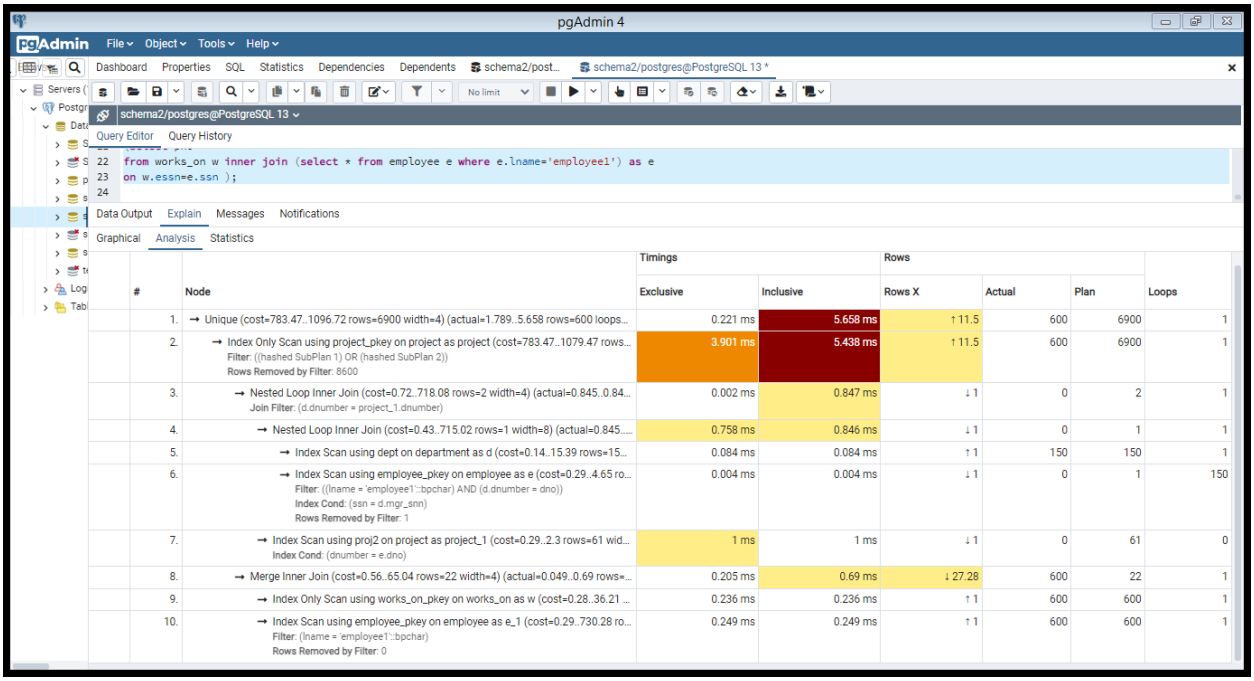


Query2 optimized using mixed indecies : Hash index on : Project(Pnumber), Employee(Lname)
B+ tree on : Department(Dnumber), Project(Dnumber)

Planning and execution times:

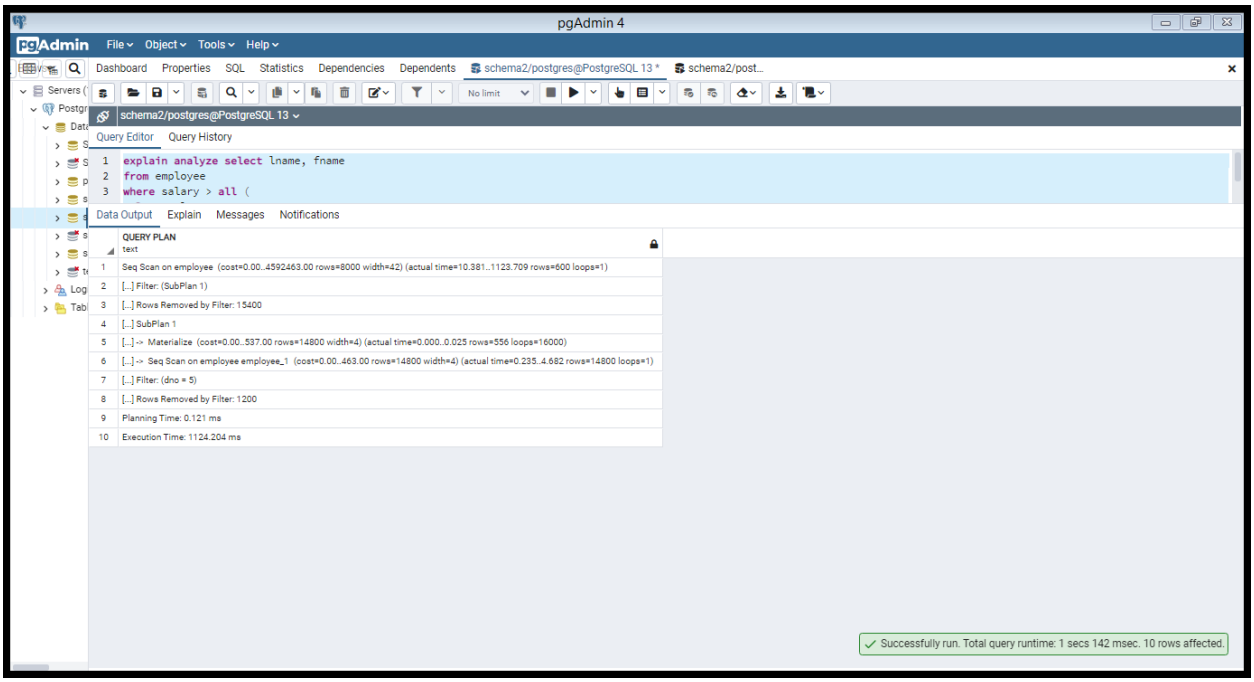


Execution plan and costs:

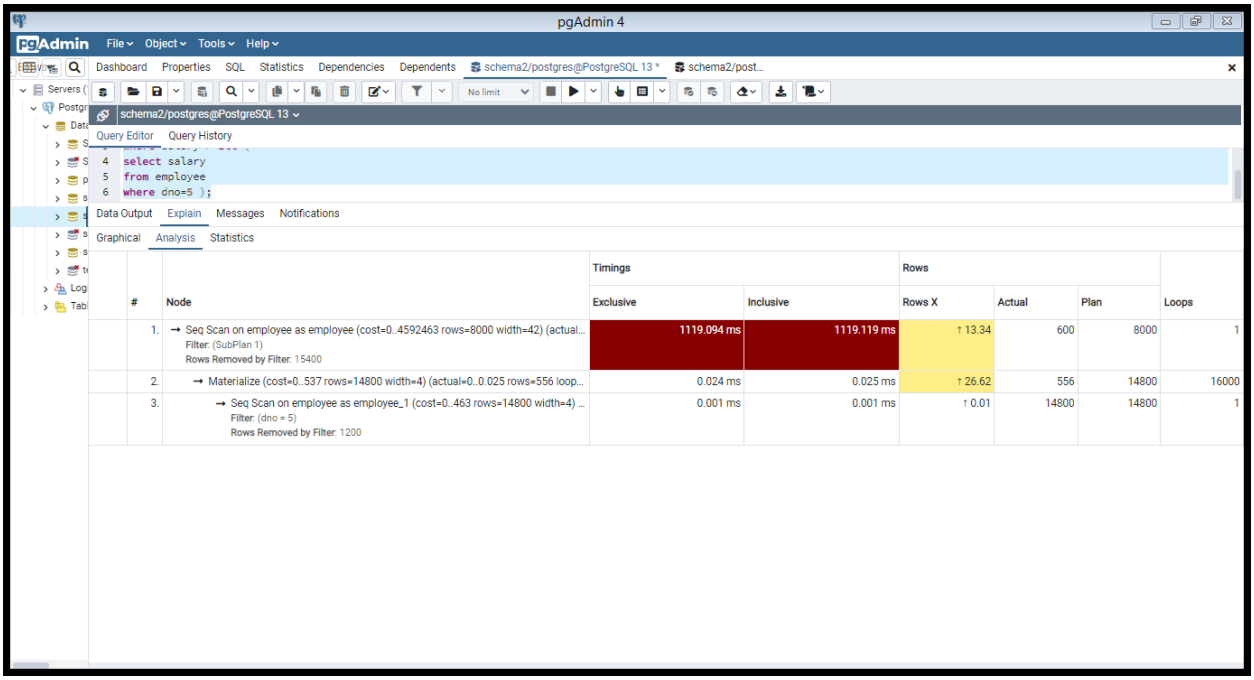


Query3 with no index

Planning and execution times:



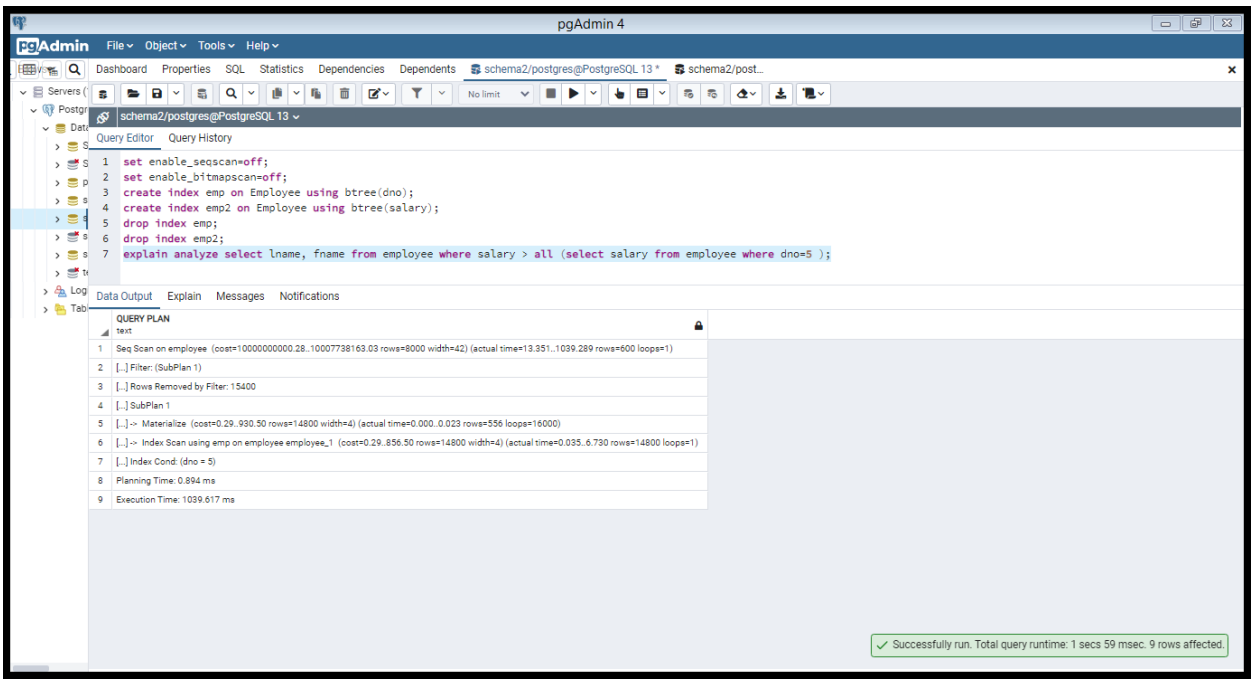
Execution plan and costs:



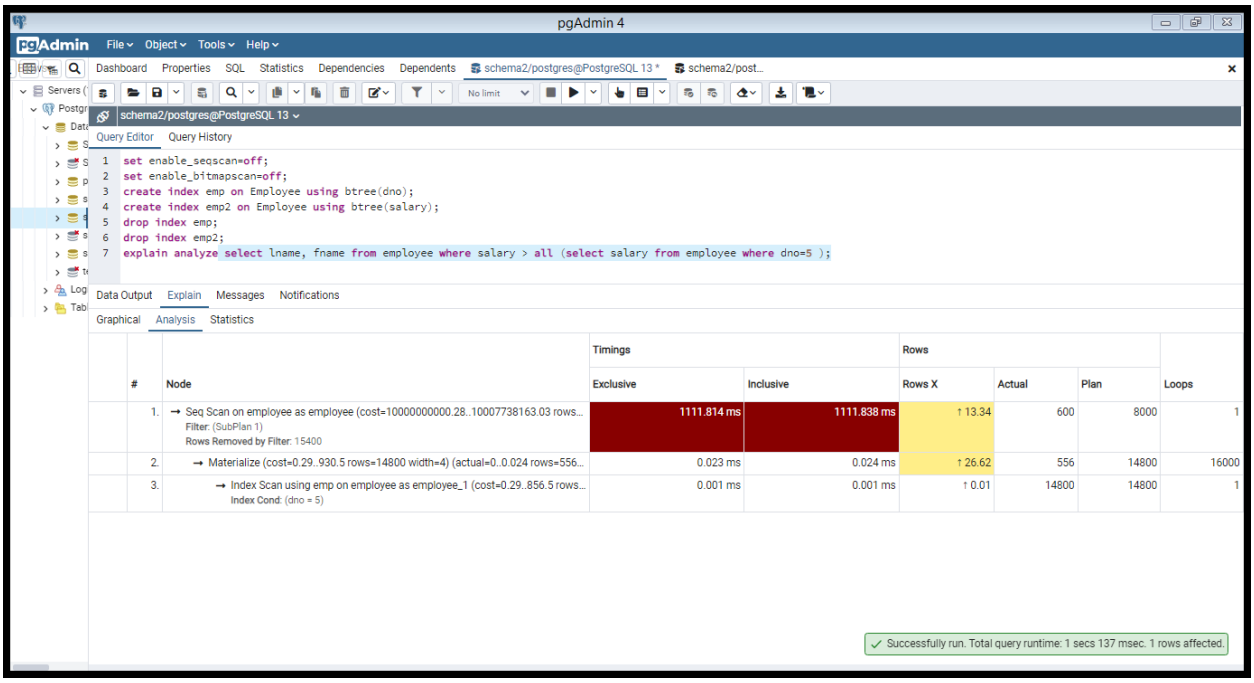
Query3 using B+ tree on : Employee(salary) , Employee(dno)

With seqscan disabled:

Planning and execution time:

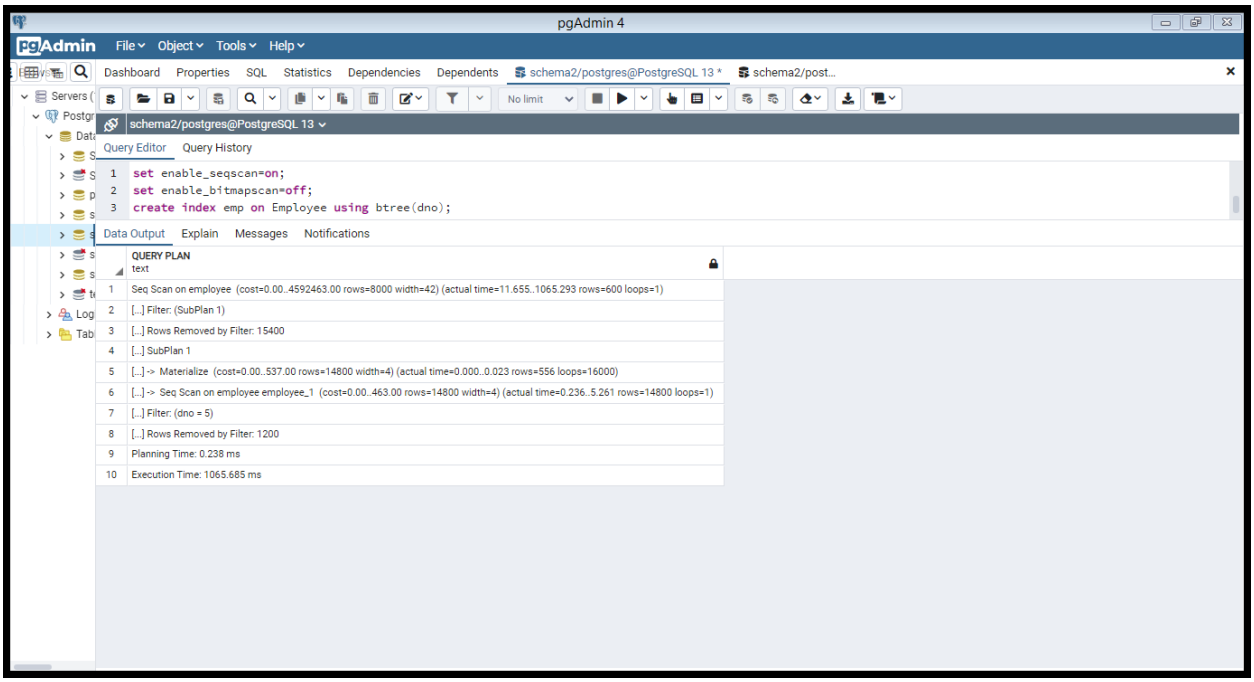


Execution plan and costs:

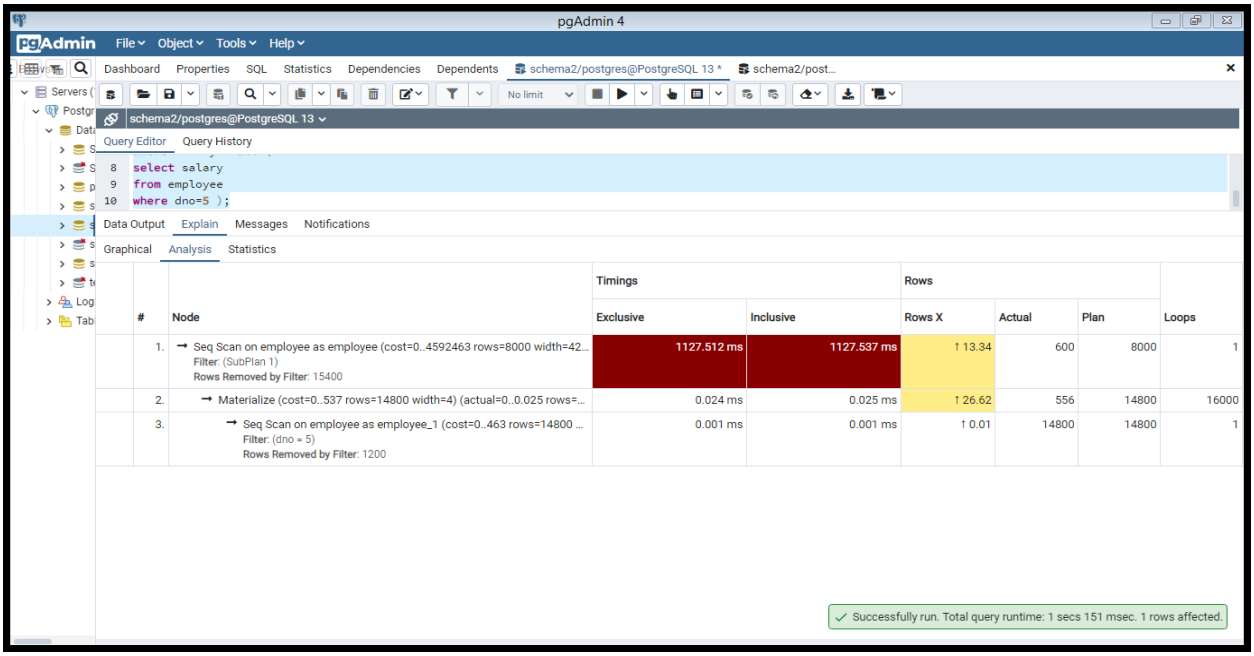


With seqscan enabled:

Planning and execution time:



Execution plan and costs:

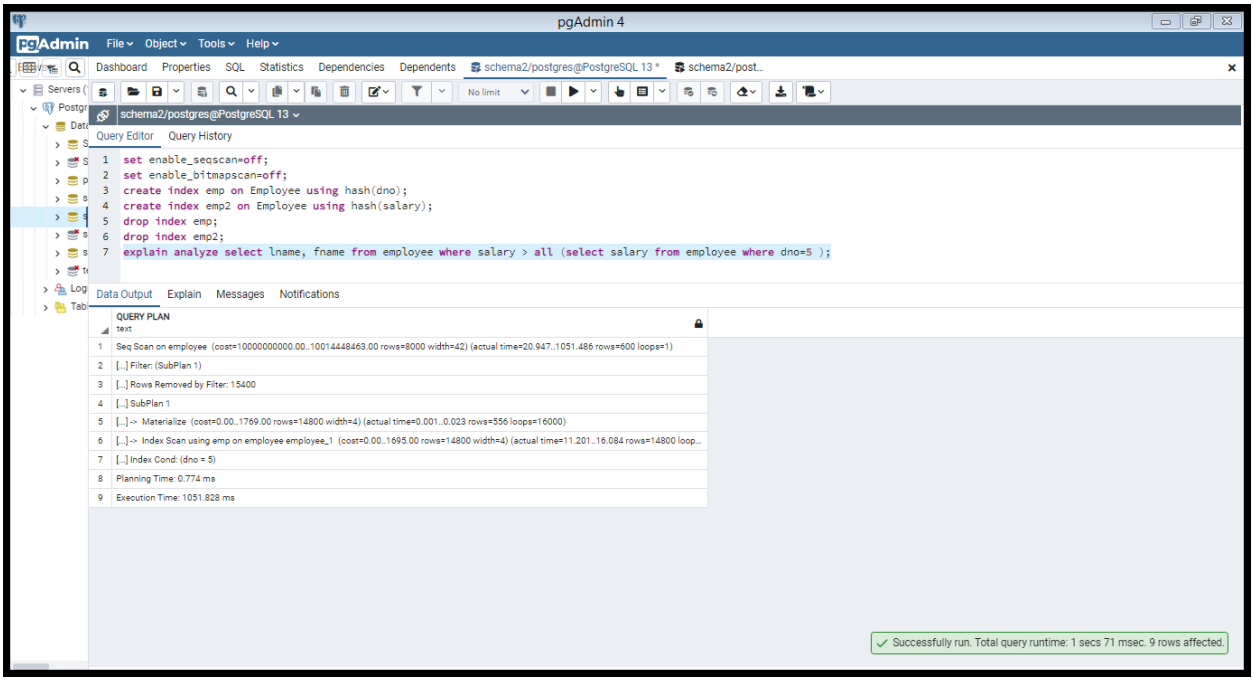


Here the cost without seqscan was so high and the execution time was enhanced but with seqscan the cost was normal because in this query due to (all) it will check every salary in the table so the index did not help

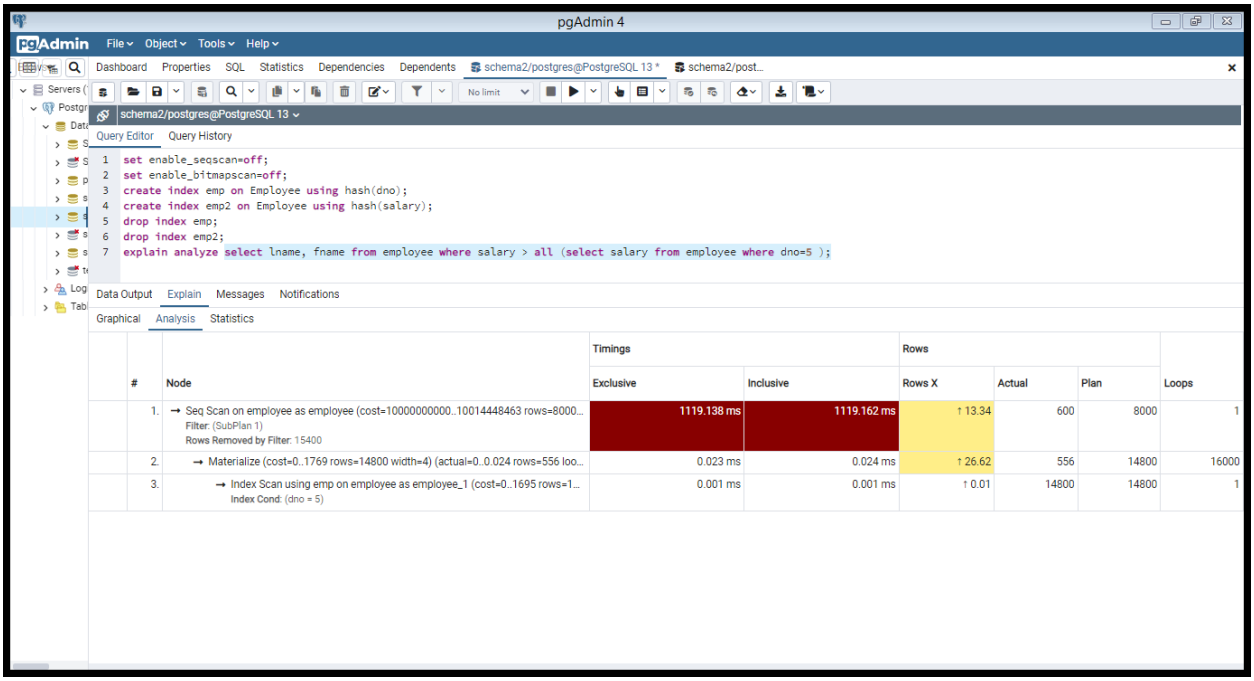
Query3 using hash indecies on : Employee(salary) , Employee(dno)

With seqscan disabled

Planning and execution time:

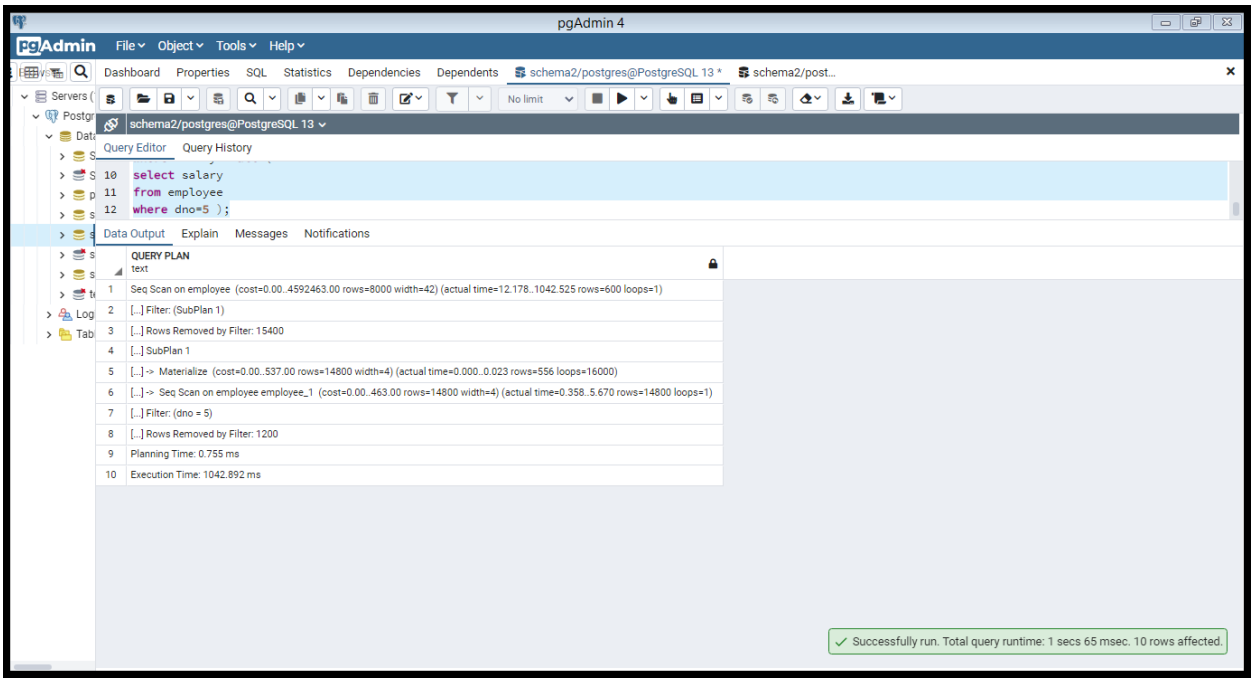


Execution plan and costs:

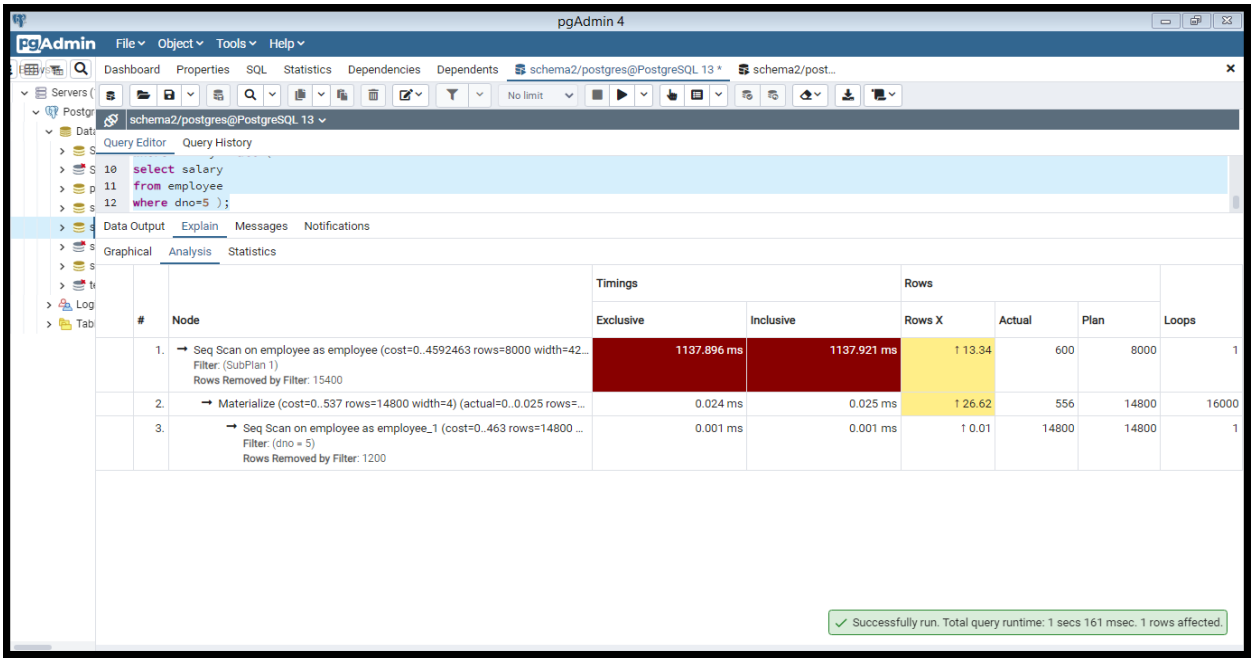


With seqscan enabled :

Planning and execution time:



Execution plan and costs:

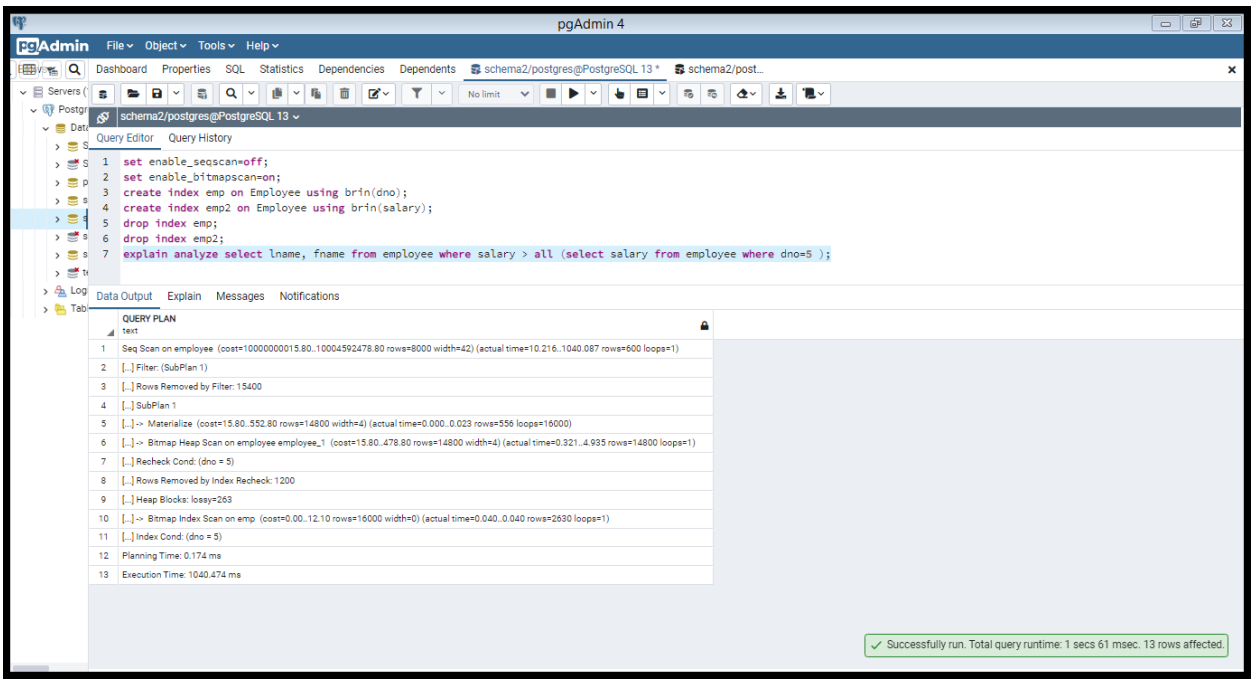


Here the cost without seqscan was so high and the execution time was enhanced but with seqscan the cost was normal because in this query due to (all) it will check every salary in the table so the index did not help

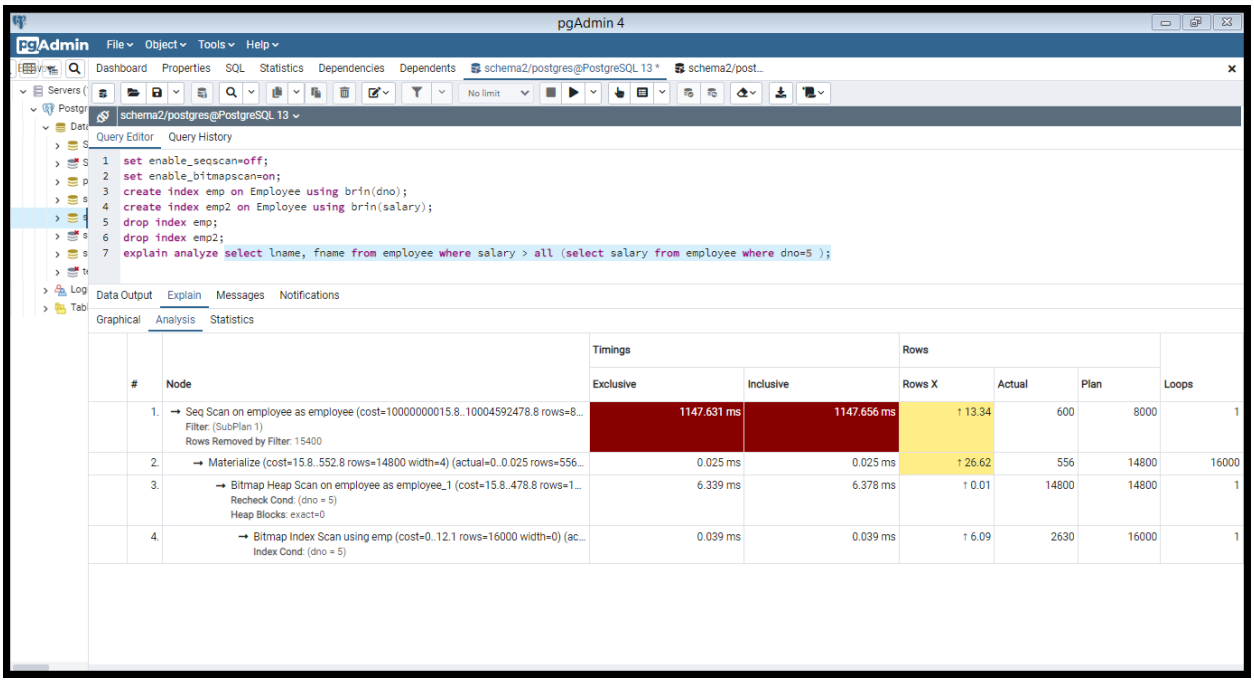
Query 3 using BRIN indecies on : Employee(dno) and Employee(salary)

With seqscan disabled

Planning and execution time:

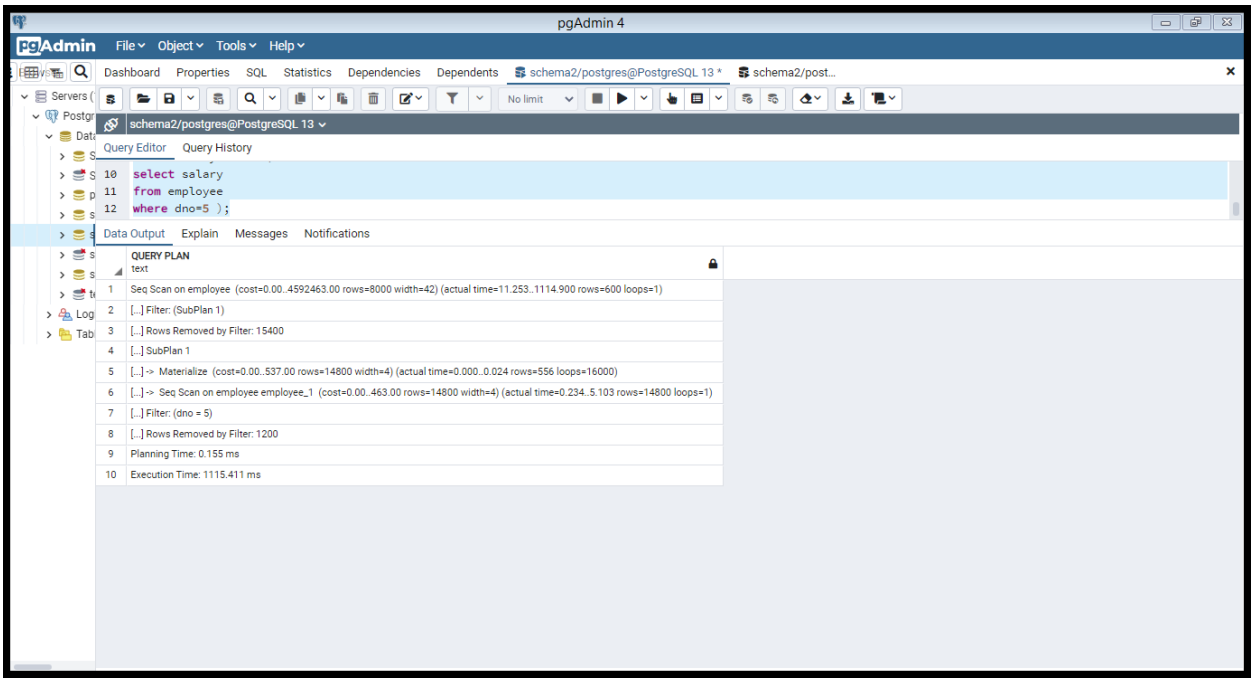


Execution plan and costs:

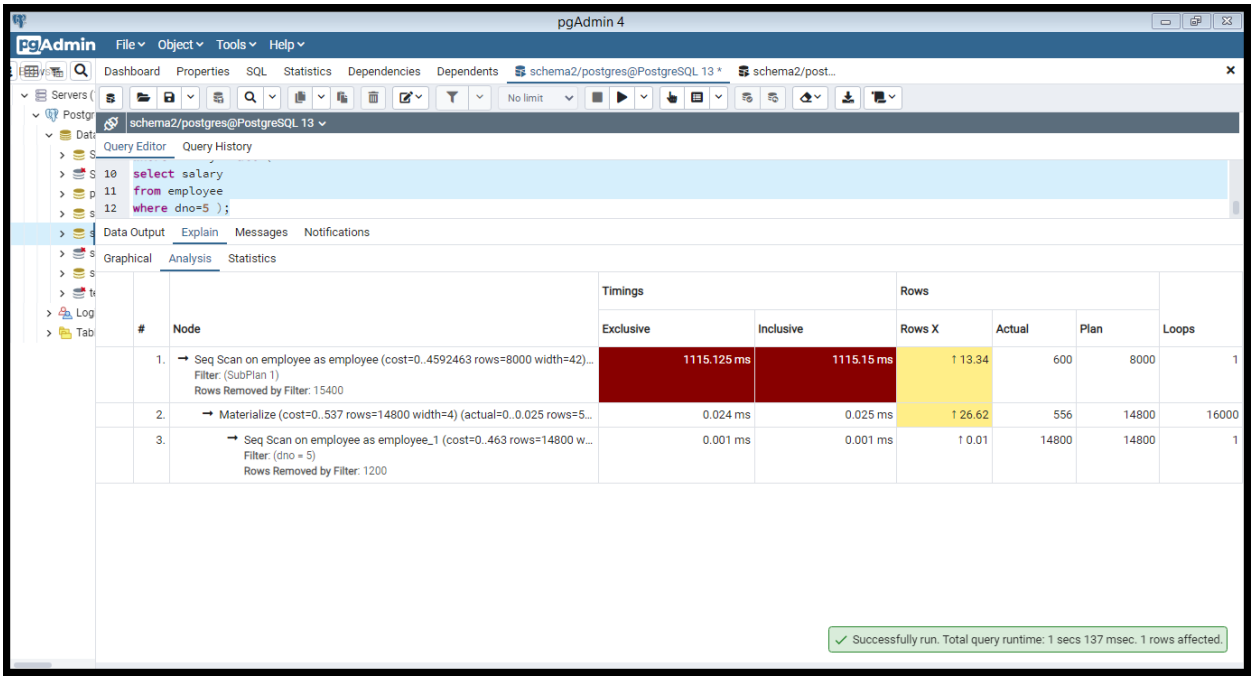


With seqscan enabled

Planning and execution time:



Execution plan and costs:

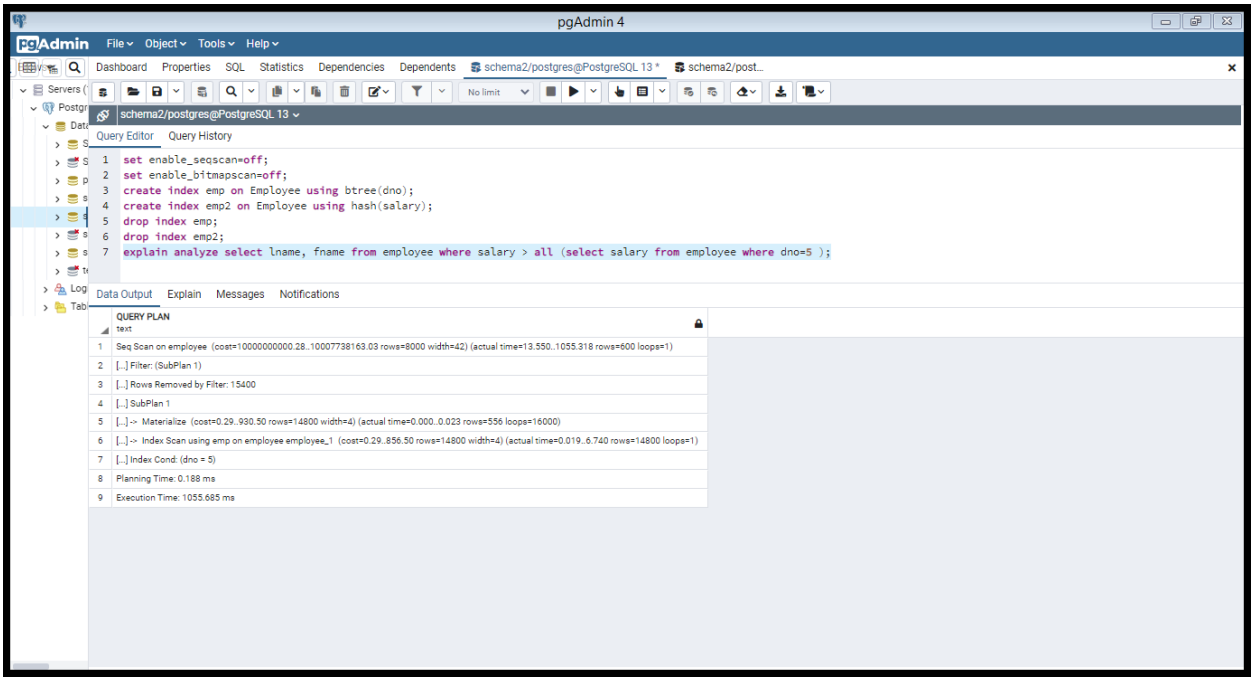


Query3 using mixed indecies :

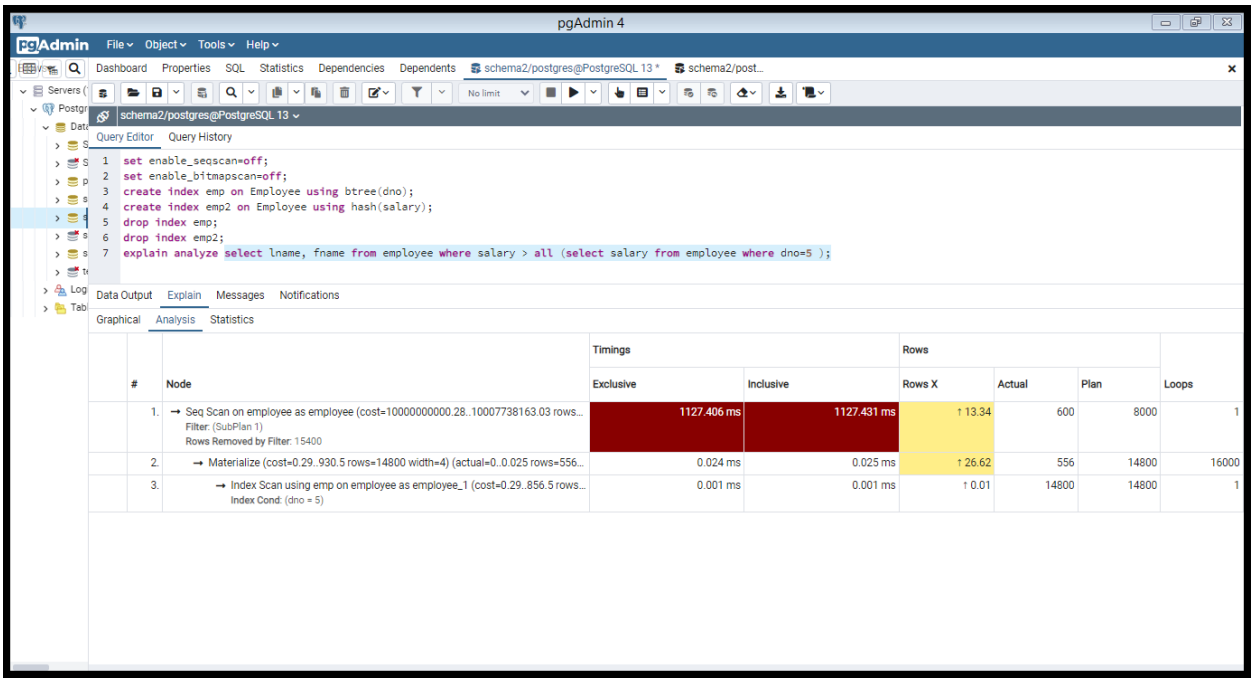
B+ tree on Employee(dno) and hash on Employee(salary)

With seqscan disabled

Planning and execution time:

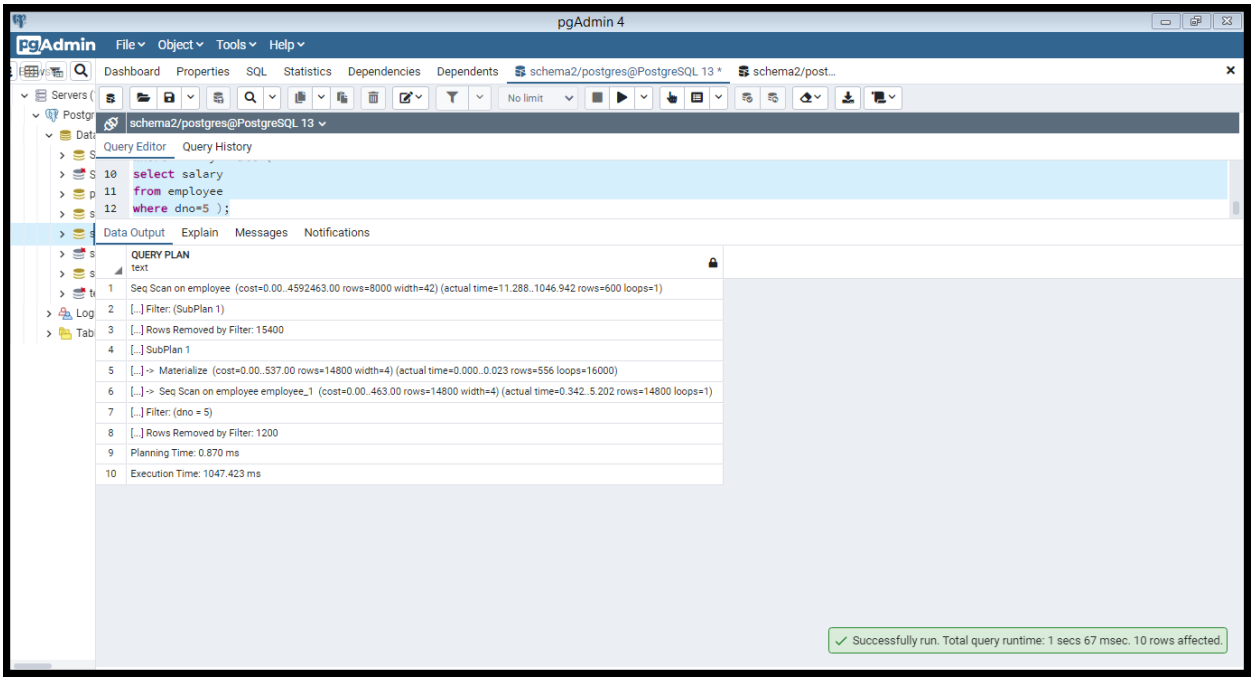


Execution plan and costs:

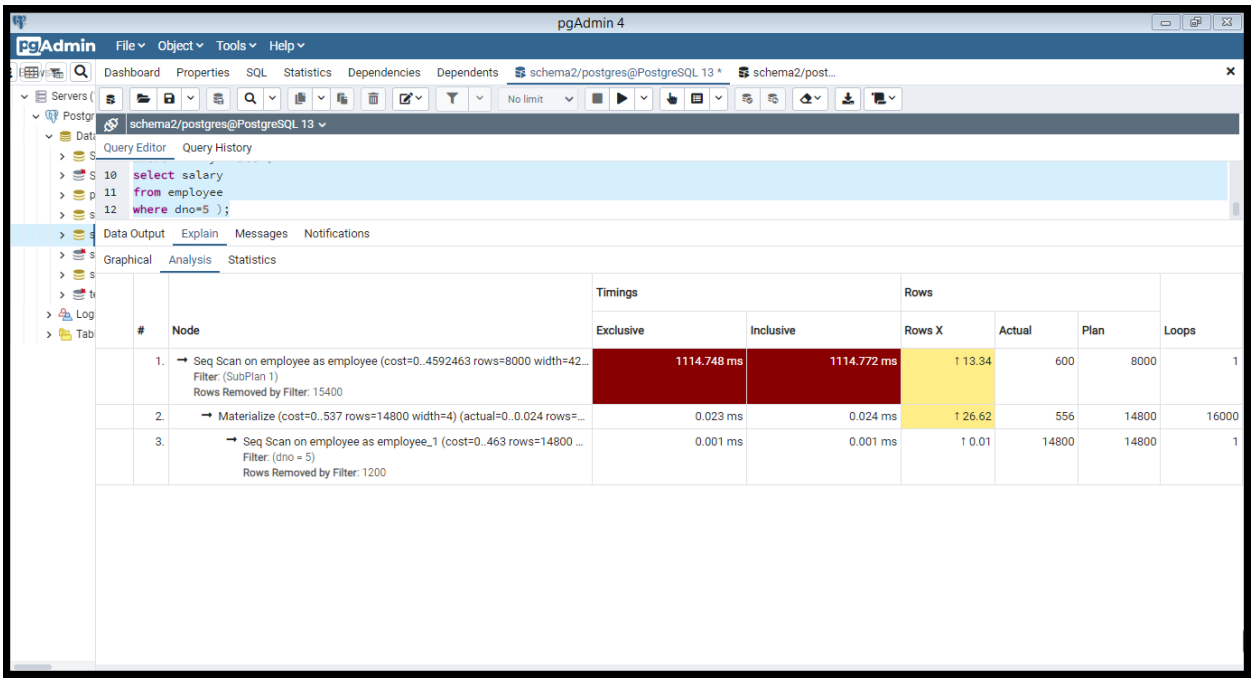


With seqscan enabled:

Planning and execution time:



Execution plan and costs:



Here in query 3 using indecies made the cost worse but it decreased the execution time

But if we used seqscan instead with no indecies it decrease the cost but increase the execution time

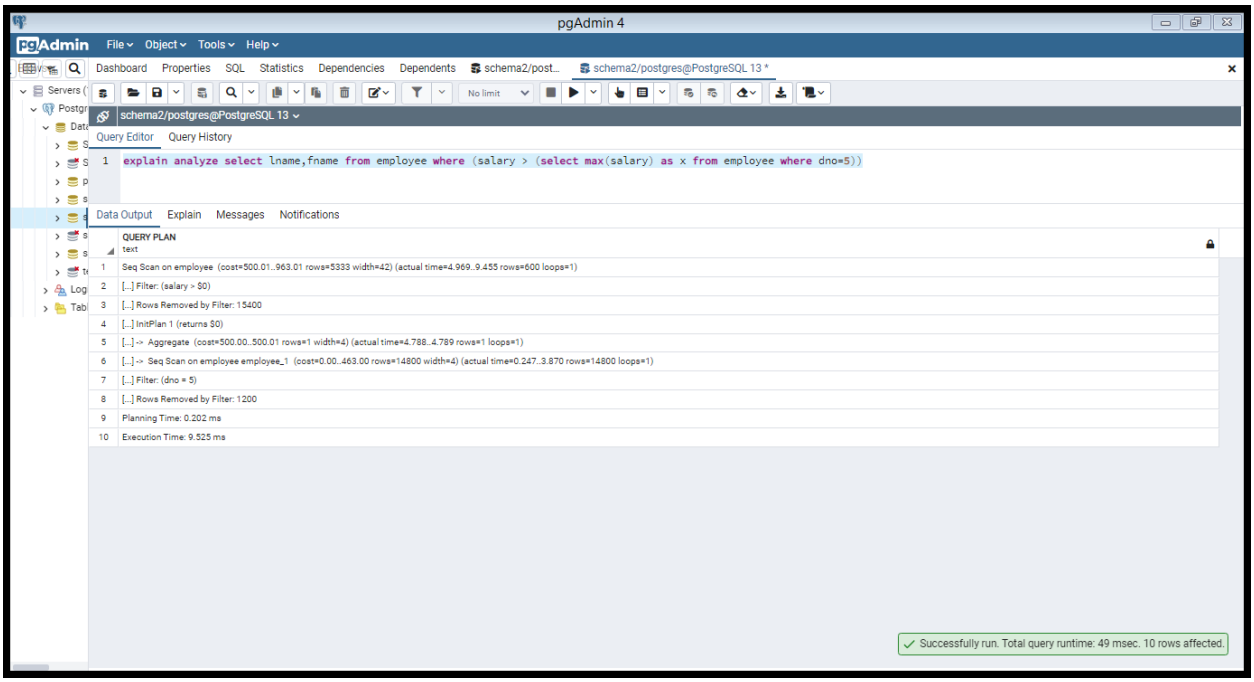
Query3 after optimization:

select lname,fname from employee where (salary > (select max(salary) as x from employee where dno=5))

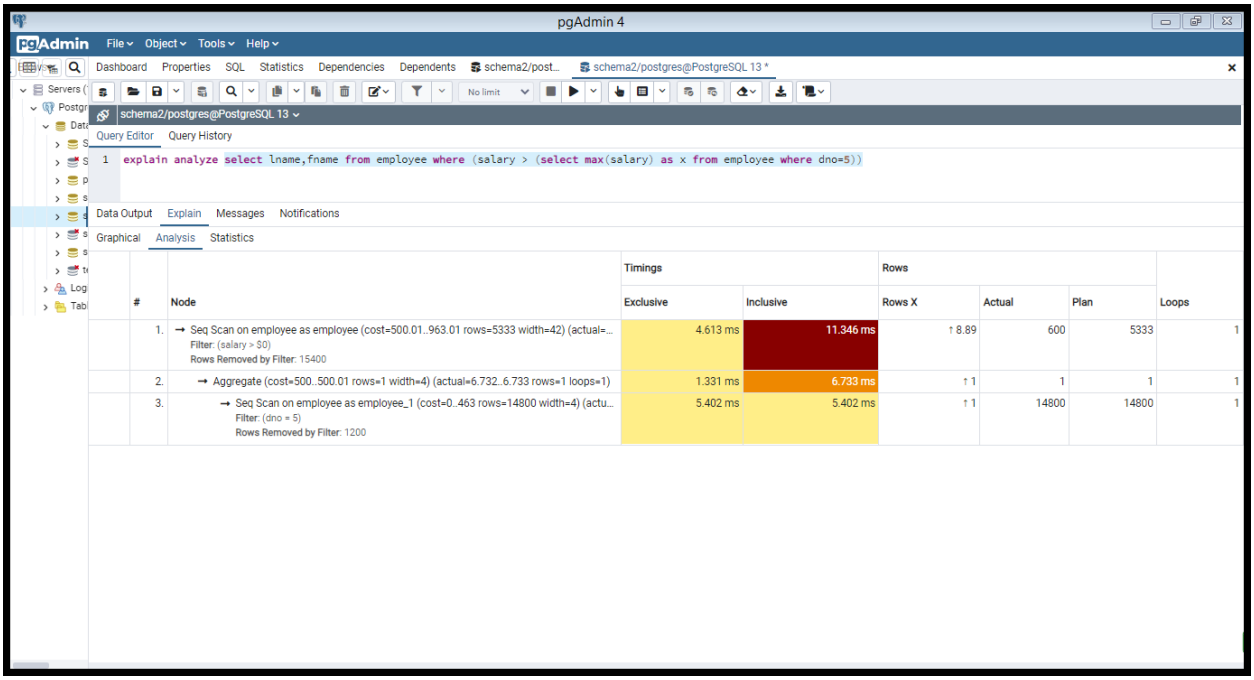
we replaced the check done by (all) with max as aggregate functions is better as it can use the indecies made on the column

Query 3 optimized with no index

Planning and execution time:



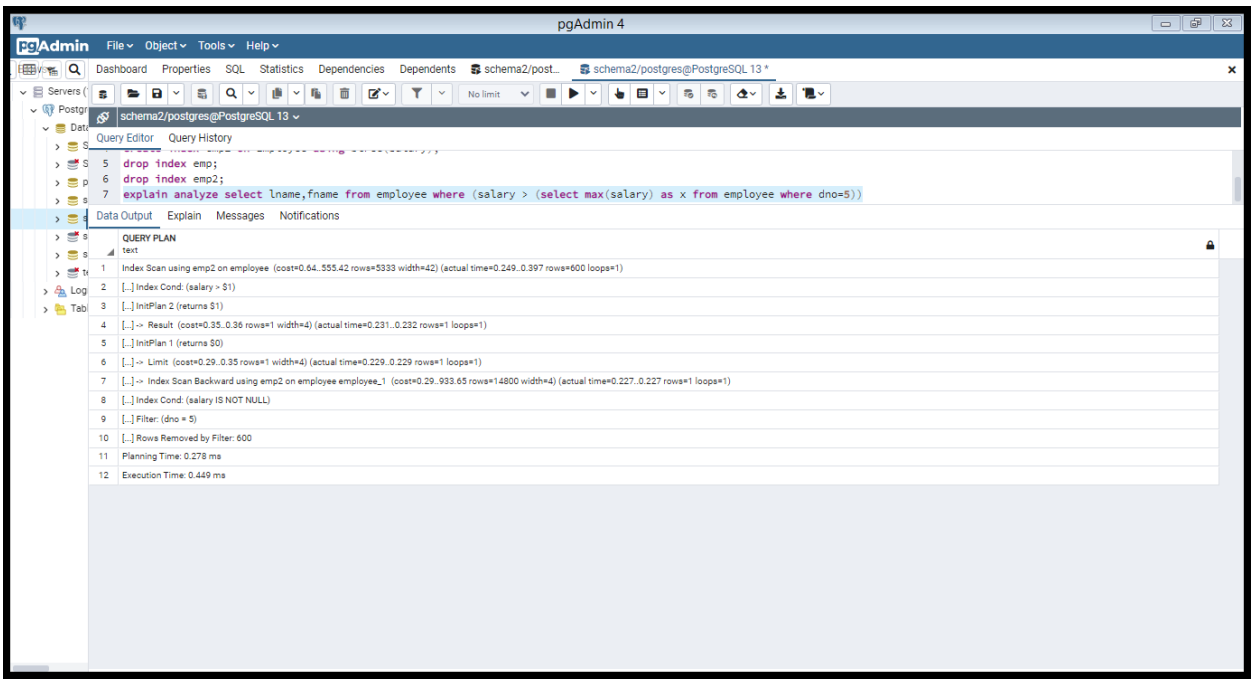
Execution plan and costs:



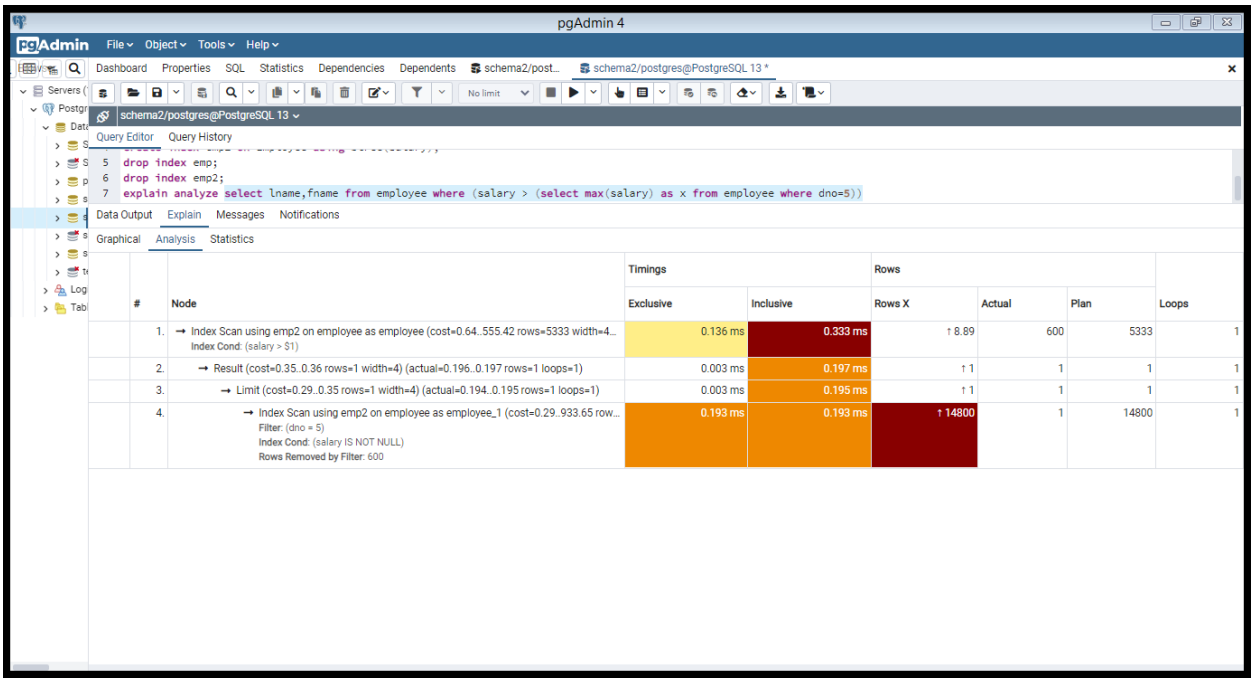
Query3 optimized using B+ tree on : Employee(dno) and Employee(salary)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



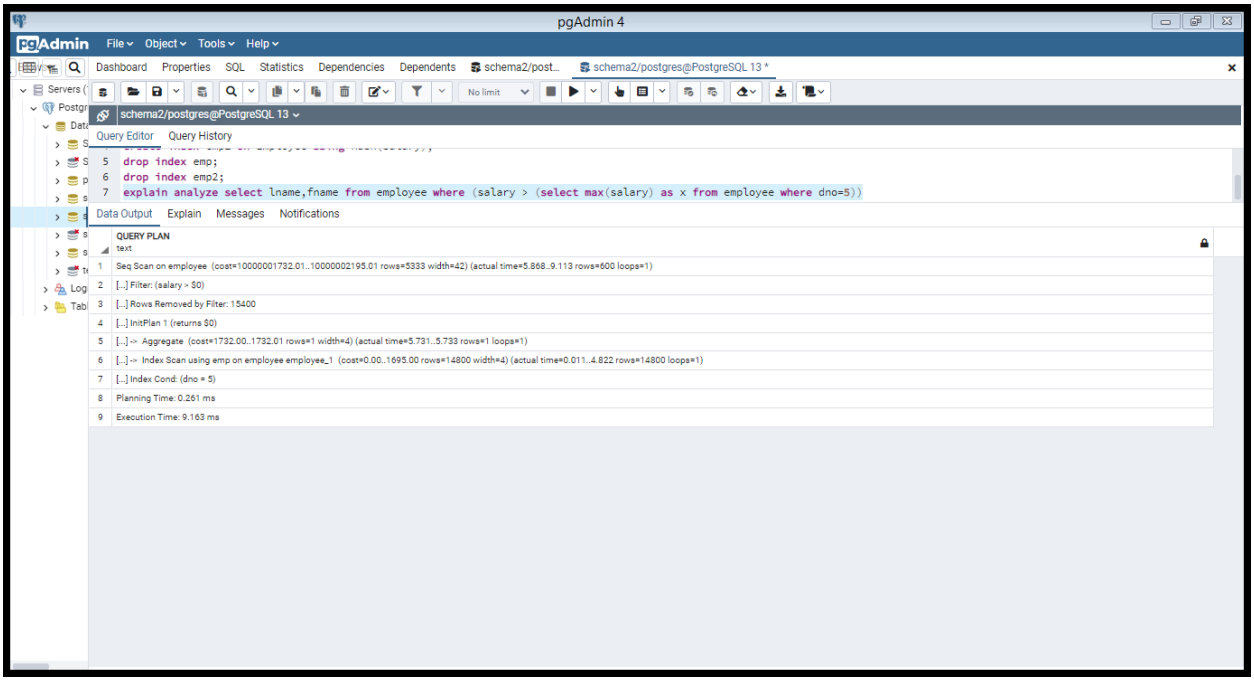
Execution plan and costs:



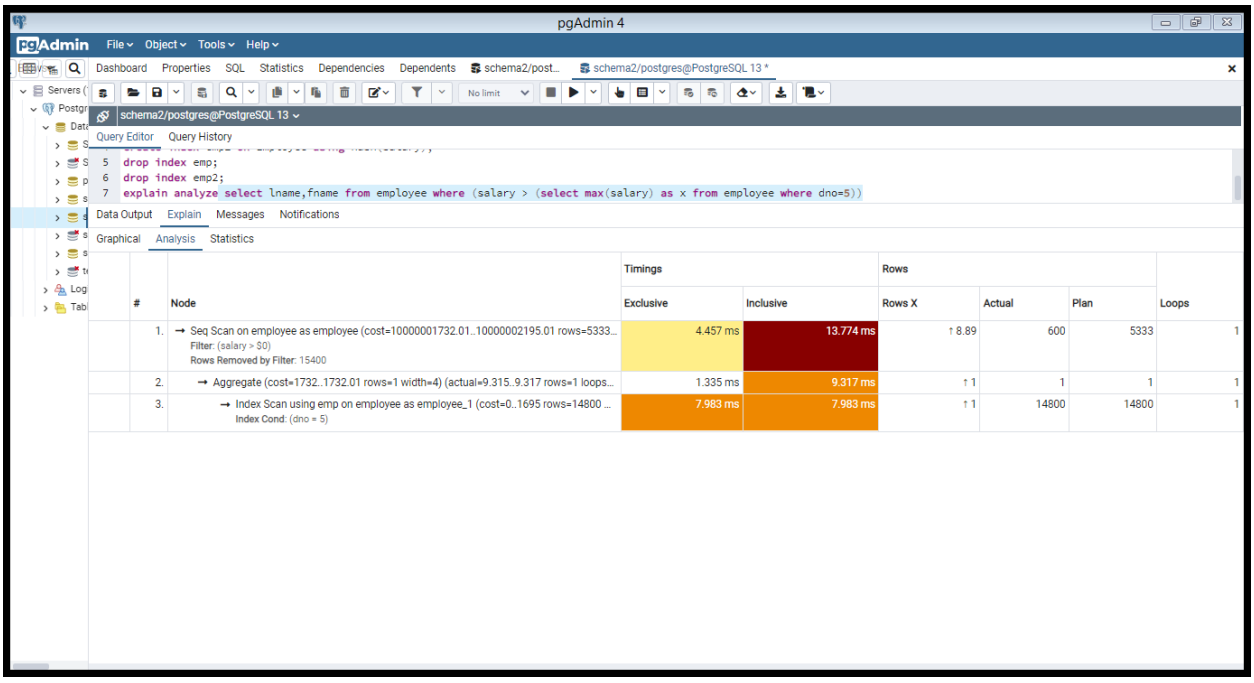
Using the B+ tree index improved performance as it helps in aggregate function like max as it search in O(logn) so it found the max number rapidly and used it in the query

Query3 optimized using hash indecies on : Employee(dno) and Employee(salary)

Planning and execution time:



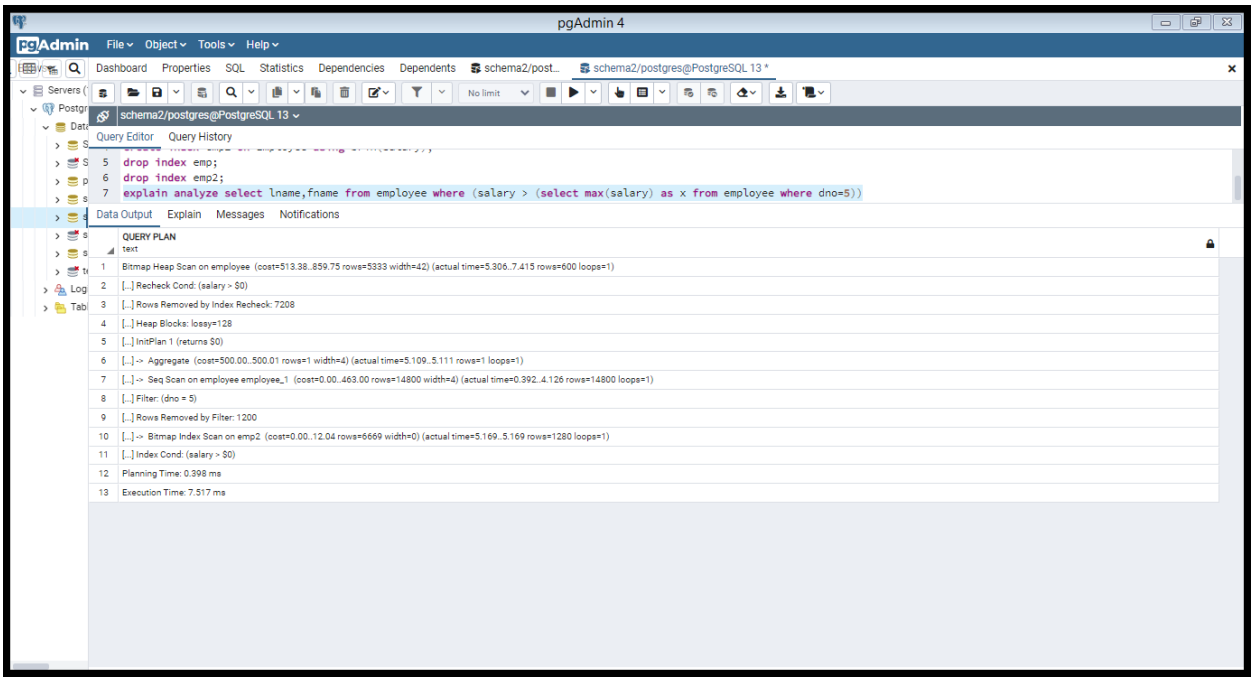
Execution plan and costs:



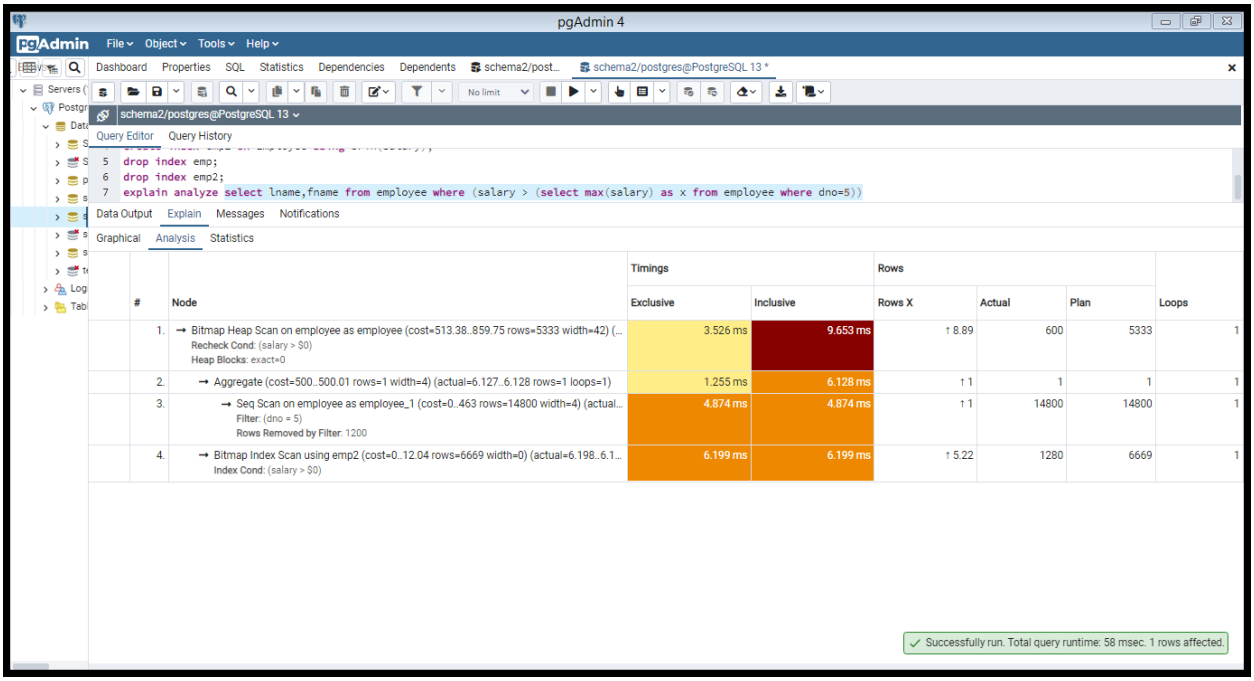
Here using hash did not help in the max function as the hash is used in exact value queries it does not help in searching for the max number and does not help in range queries so it increased the cost

Query3 optimized using BRIN indecies on: Employee(dno) and Employee(salary)

Planning and execution time:



Execution plan and costs:



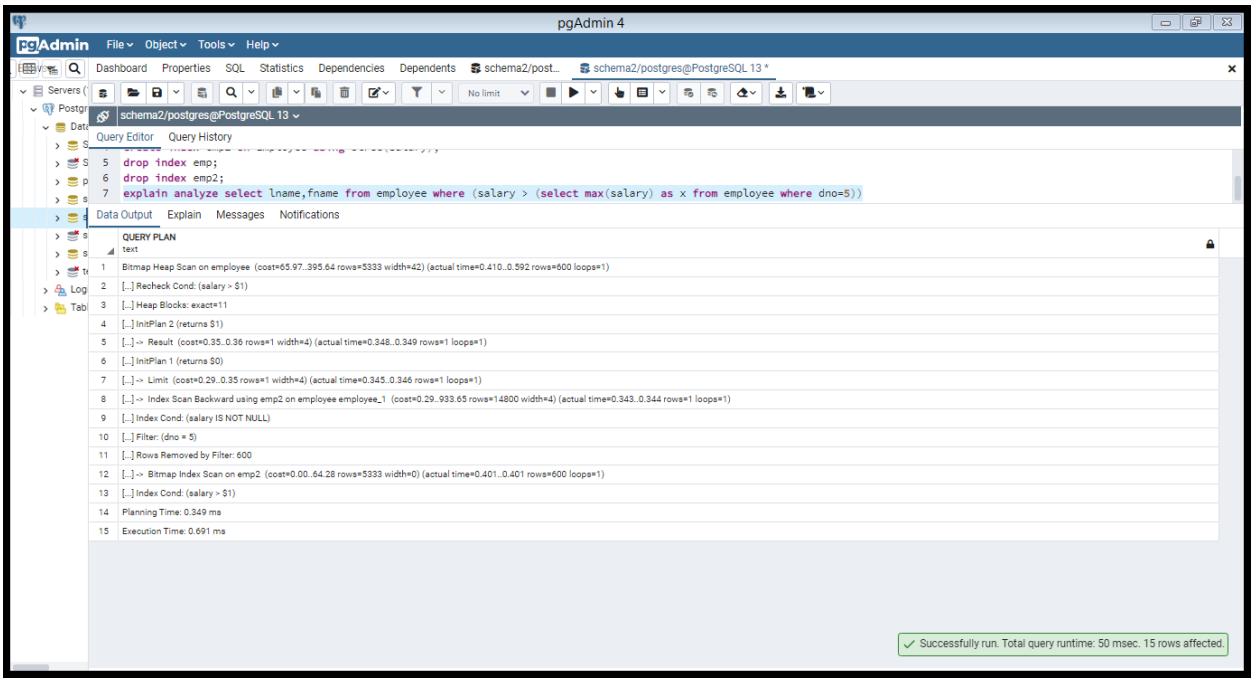
Here the BRIN index increased performance slightly as BRIN works in query that has small range in a big range so it was not that much effective

Query3 optimized using mixed indecies

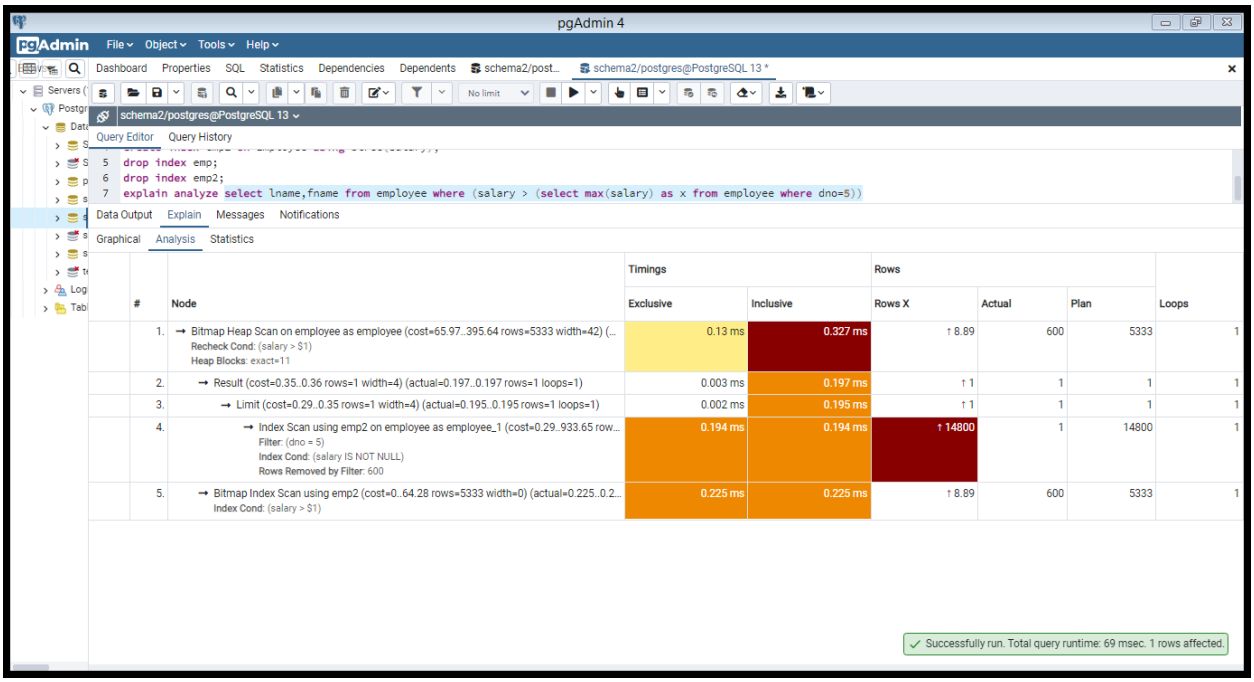
Disabled flags: seqscan

B+ tree on Employee(salary) and BRIN on Employee(dno)

Planning and execution time:

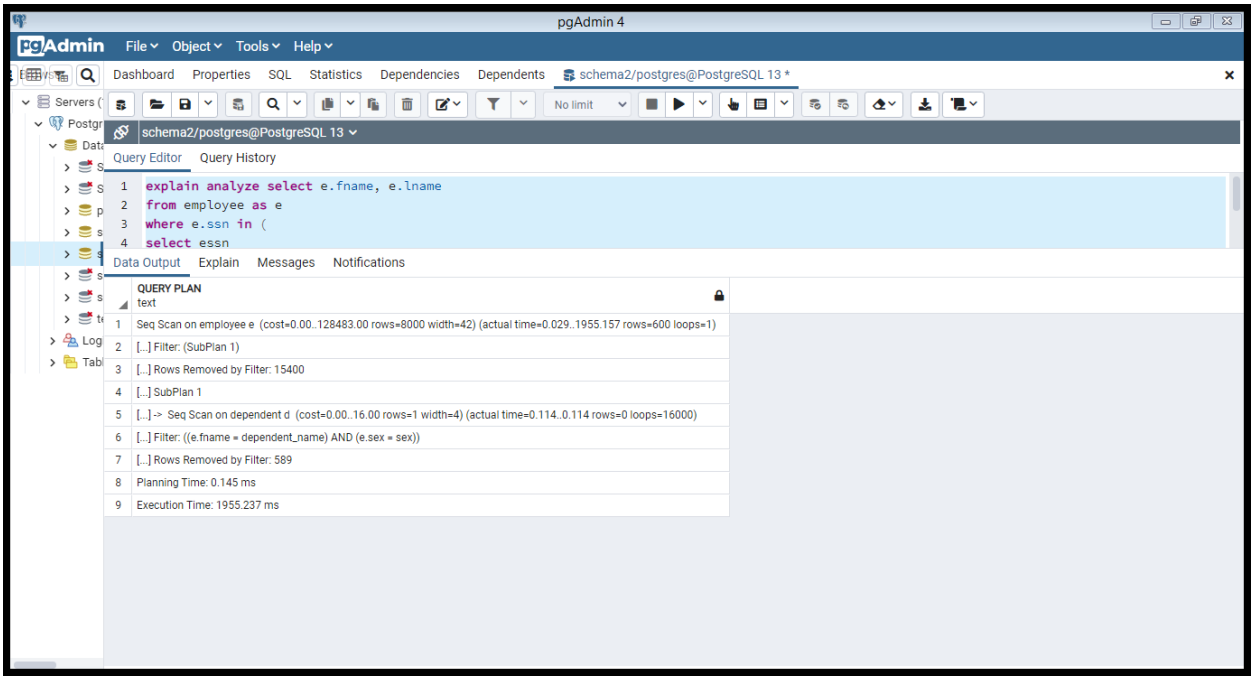


Execution plan and costs:

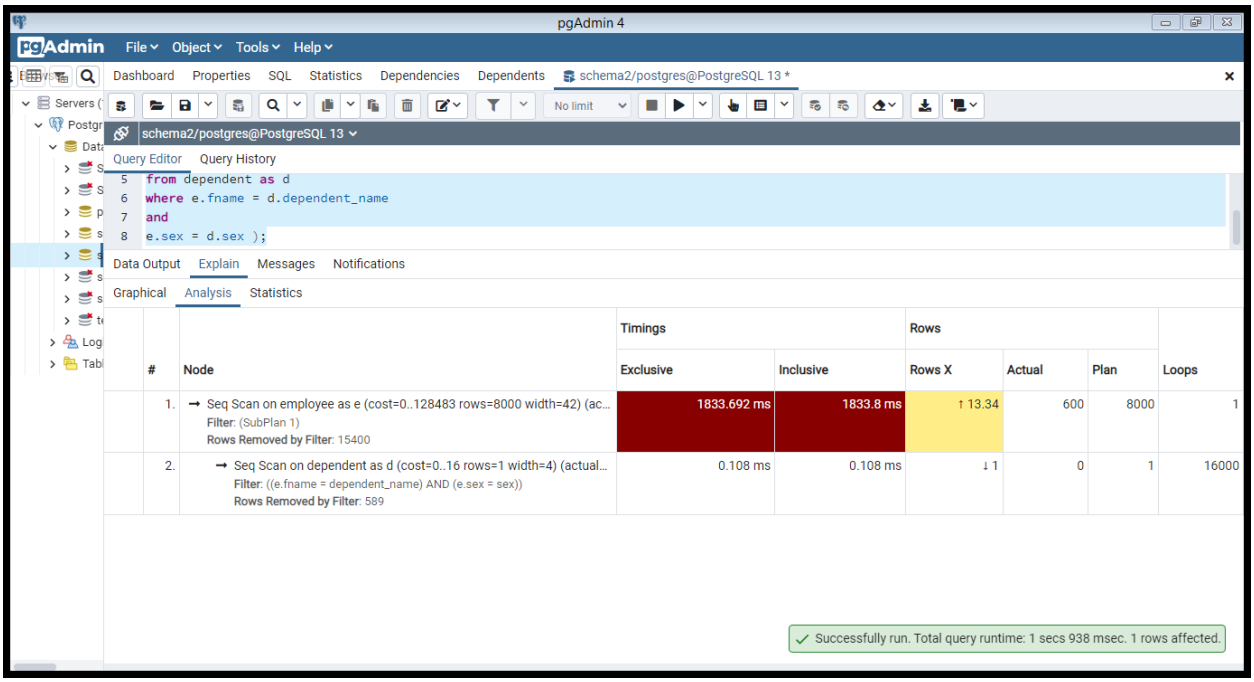


Query4 with no index

Planning and execution time:



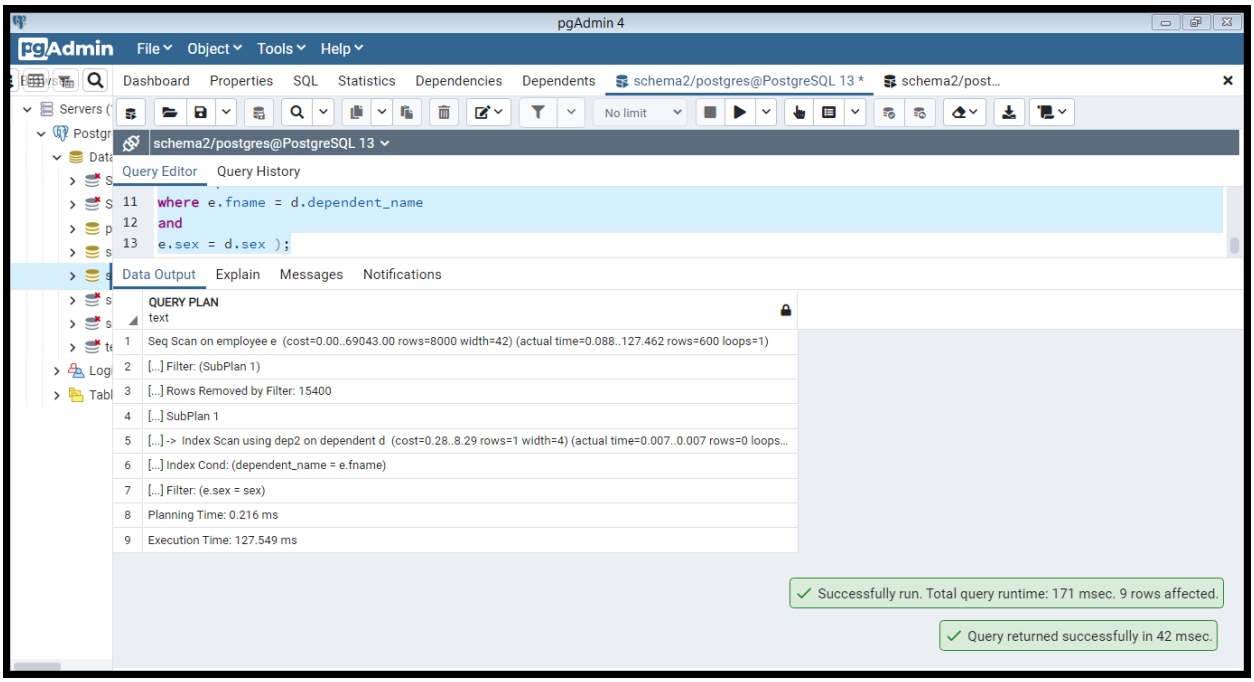
Execution plan and costs:



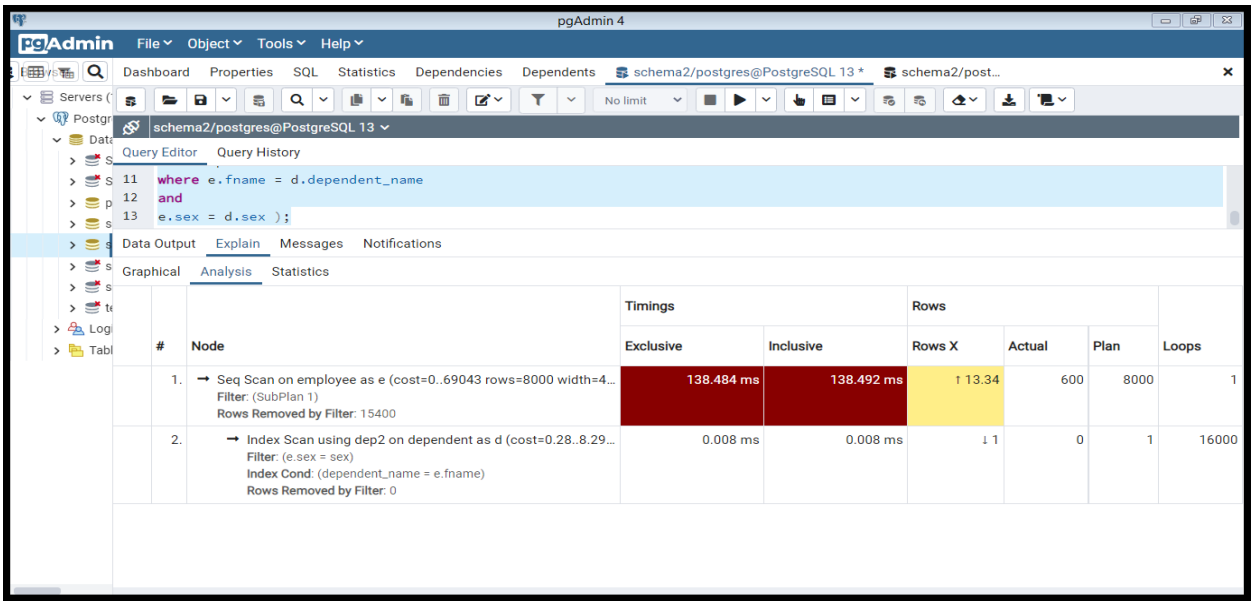
Query 4 using B+ tree indecies on : Employee(sex), Employee(fname) , Dependent(dependent_name),Dependent(sex)

Disabled flags: seqscan and bitmap scan

Planning and execution time:



Execution plan and costs:

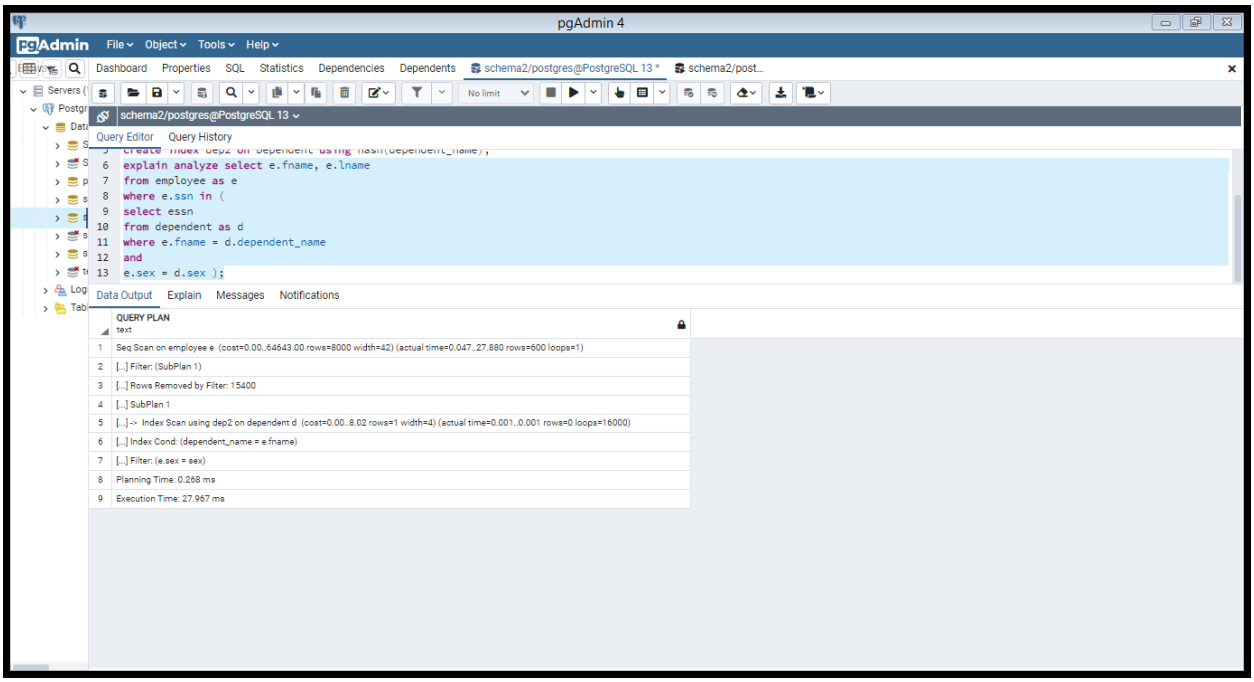


Here the B+ tree helped in finding the dependent as it search in O(logn) time so it increase performance

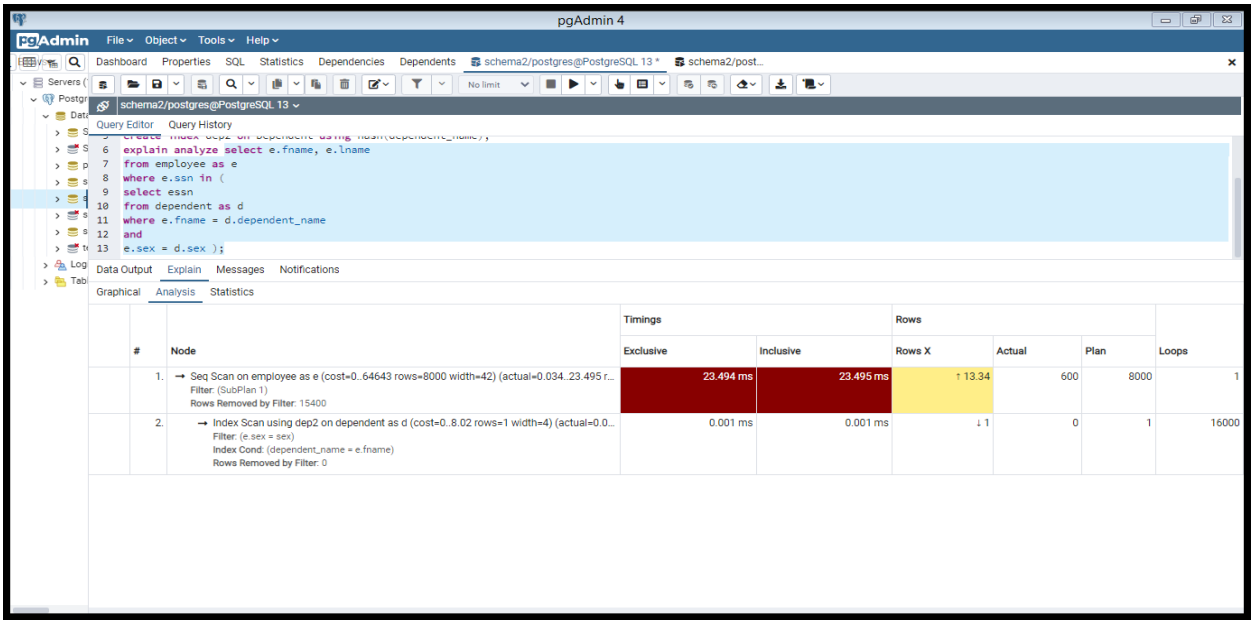
Query4 using hash indecies on : Employee(sex), Employee(fname) , Dependent(dependent_name),Dependent(sex)

Disabled flags : seqscan and bitmap scan

Planning and execution time:



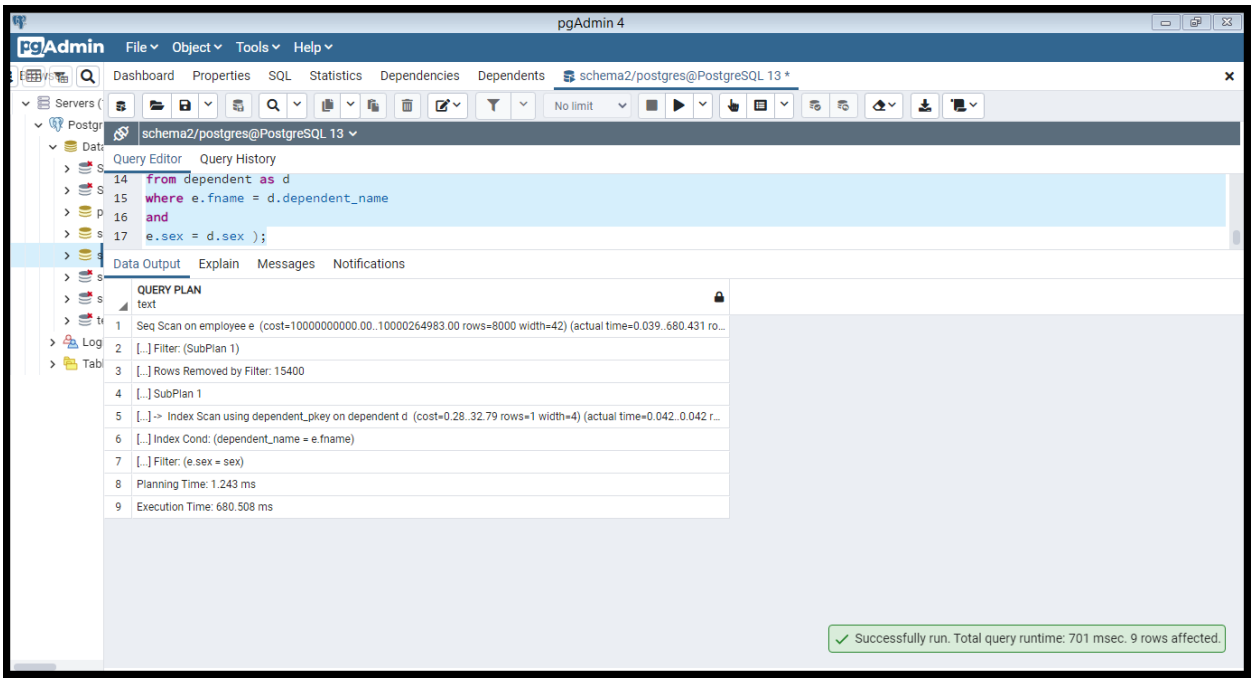
Execution plan and costs:



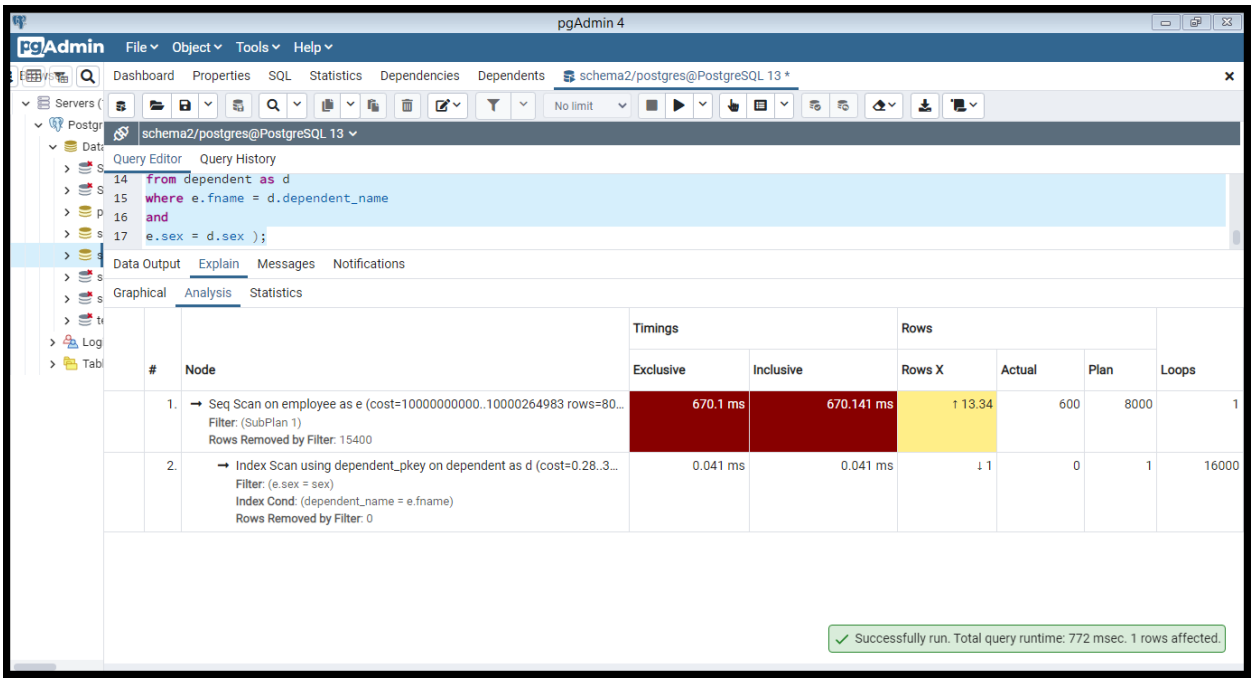
Here the hash improved the performance as it helps in searching for the dependent who has the same name and sex as it search in O(1) time

Query4 using BRIN indecies on : Employee(sex), Employee(name) , Dependent(dependent_name),Dependent(sex)

Planning and execution time:



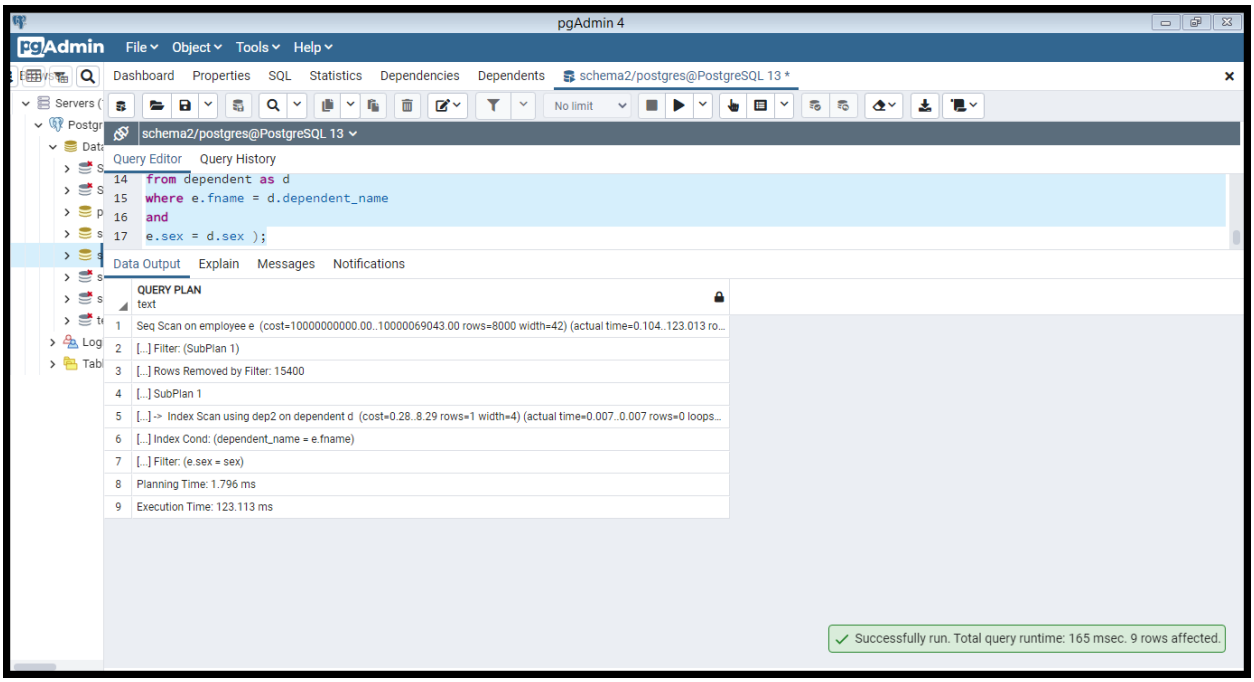
Execution plan and costs:



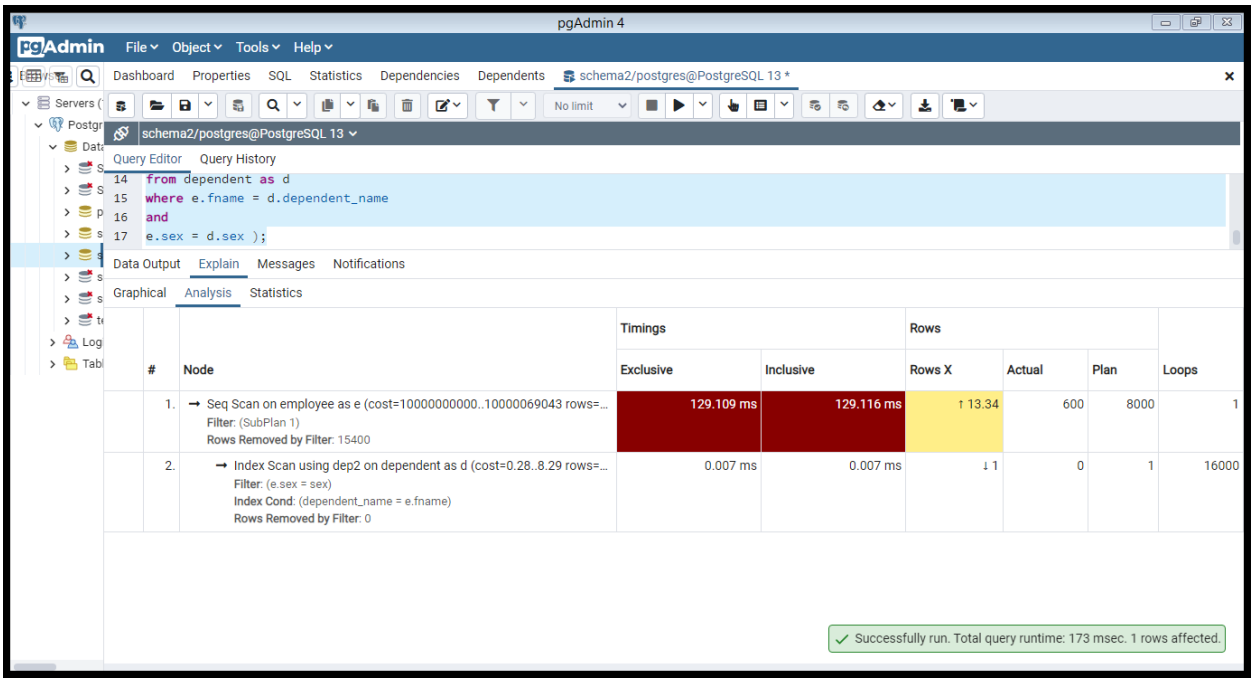
Here brin increased the cost and increased execution time as it is used in small range queries only

Query4 using mixed indecies : hash index on Employee(sex),Dependent(sex)
B+ tree on Emloyee(fname) , Dependent(dependent_name)

Planning and execution time:



Execution plan and costs:



The indecies in this query increased performance but it also increased the cost too much and when I use seqscans only the execution time increase and cost decrease

Query4 after optimization :

Select distinct fname,lname

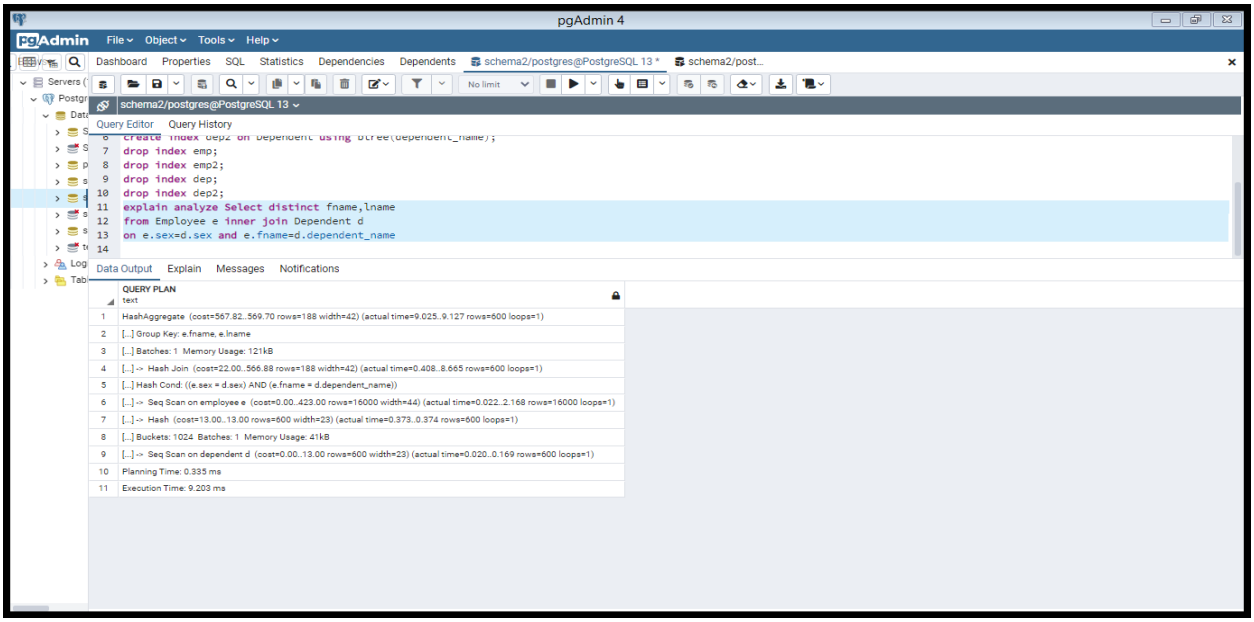
from Employee e inner join Dependent d

on e.sex=d.sex and e.fname=d.dependent_name

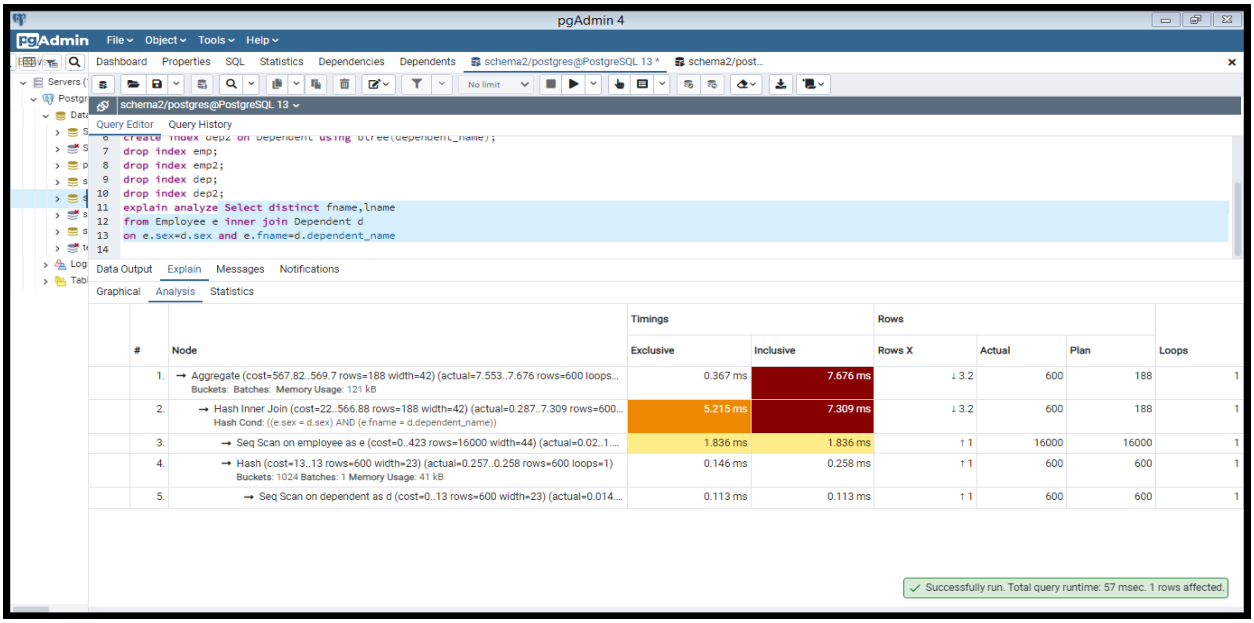
we replaced the sub query with inner join as it is faster

Query4 optimized with no index

Planning and execution time:



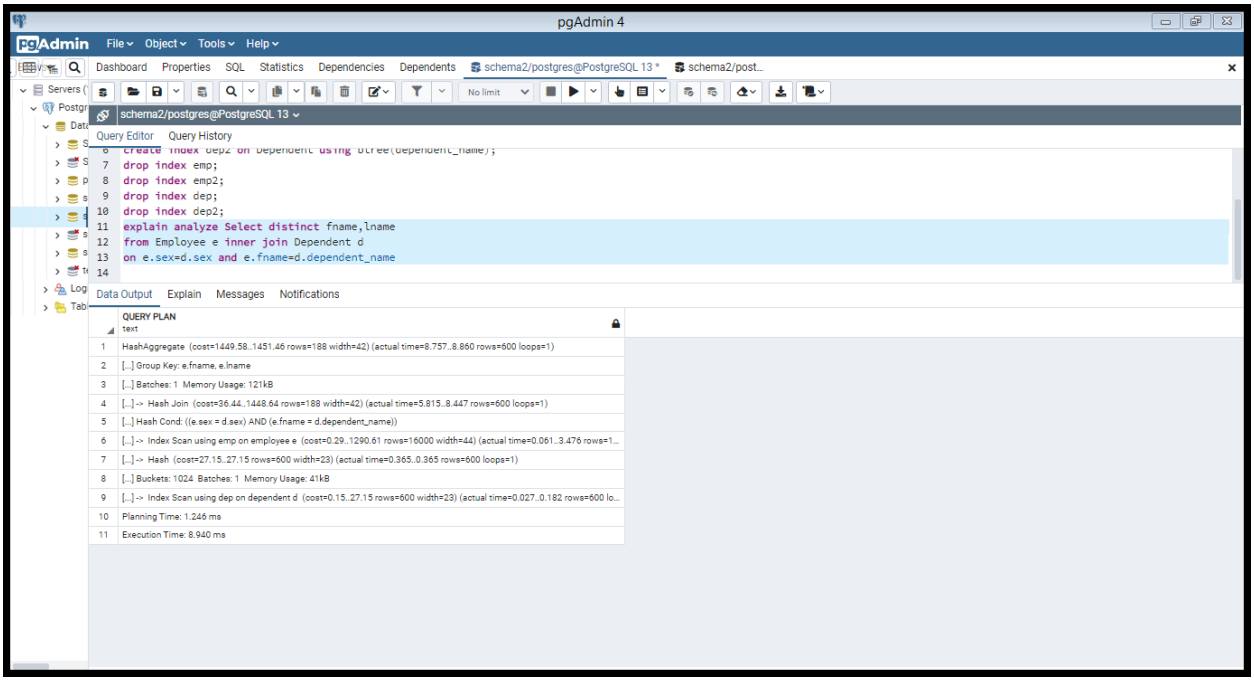
Execution plan and costs:



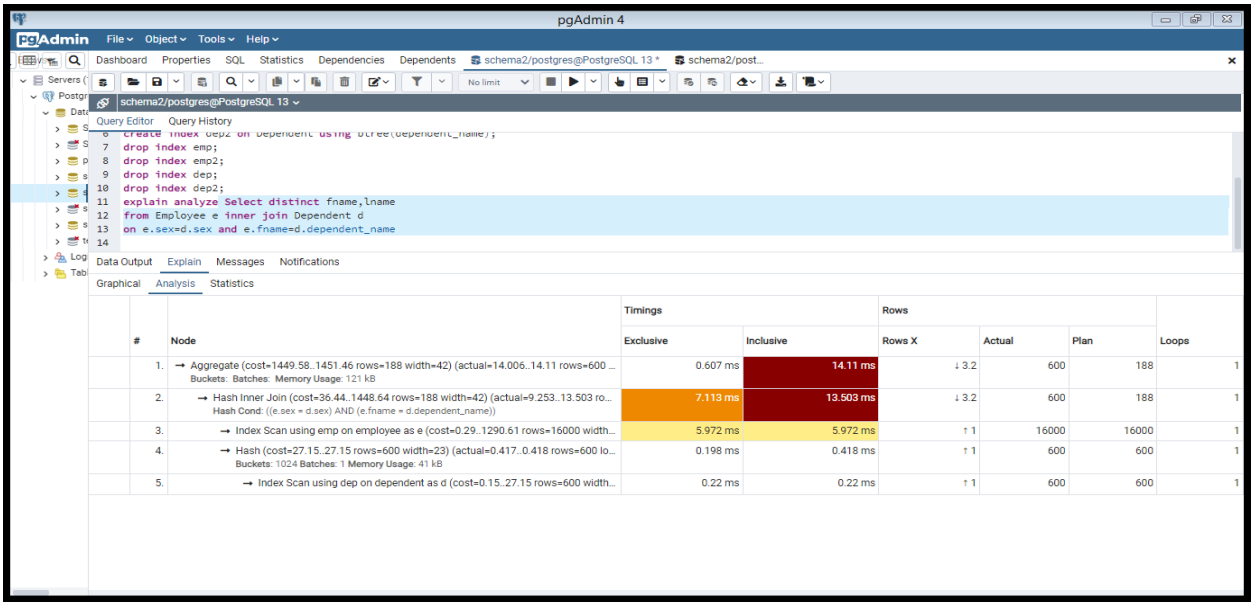
Query4 optimized using B+ tree on: Dependent(dependent_name),Dependent(sex), Employee(fname), Employee(sex)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



Execution plan and costs:

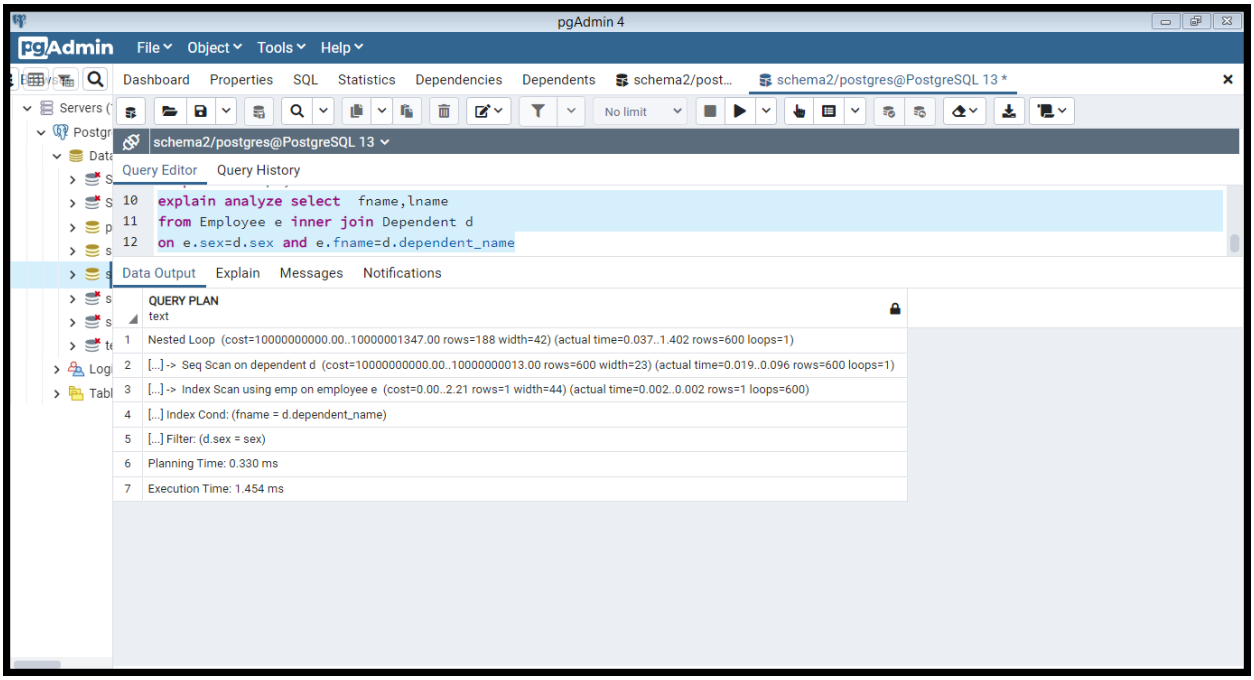


here the B+ tree did not improve the execution time that much because in the join it needs to check every row from 1 table with all rows of the other tables and it also checks on 2 conditions sex and name so it did not help that much

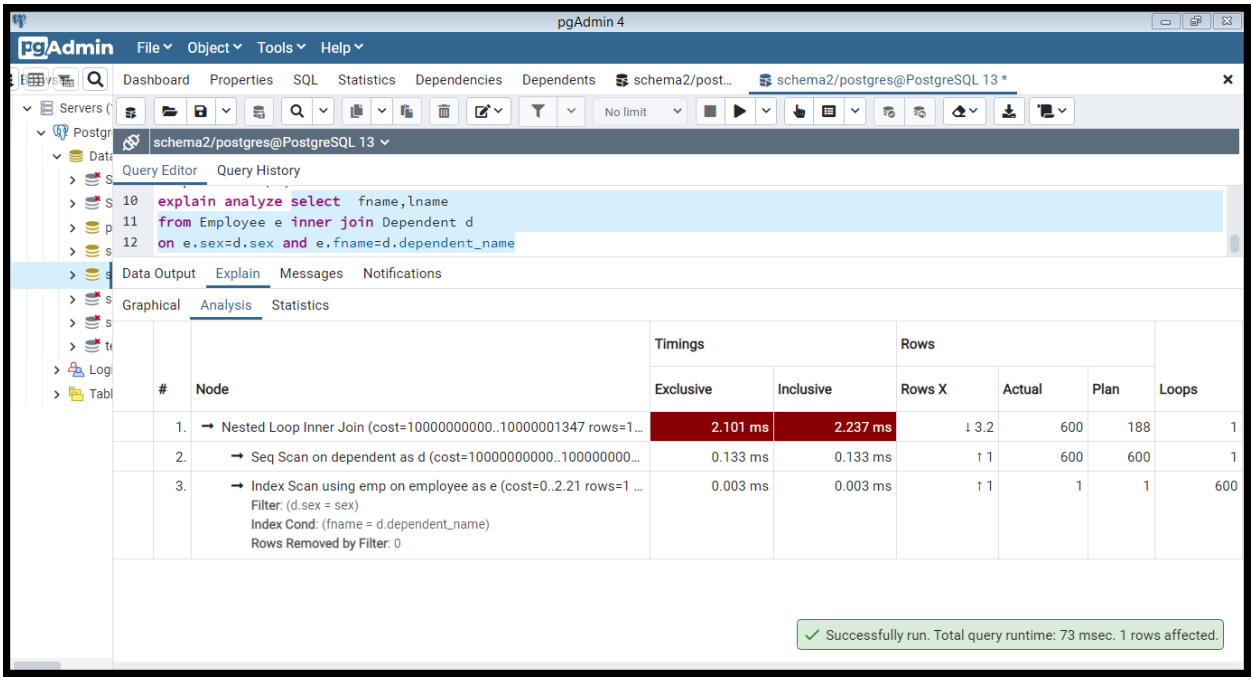
Query4 optimized using hash indecies on : Employee(sex), Employee(fname)

Disabled flags: seqscan and bitmap scan

Planning and execution time:



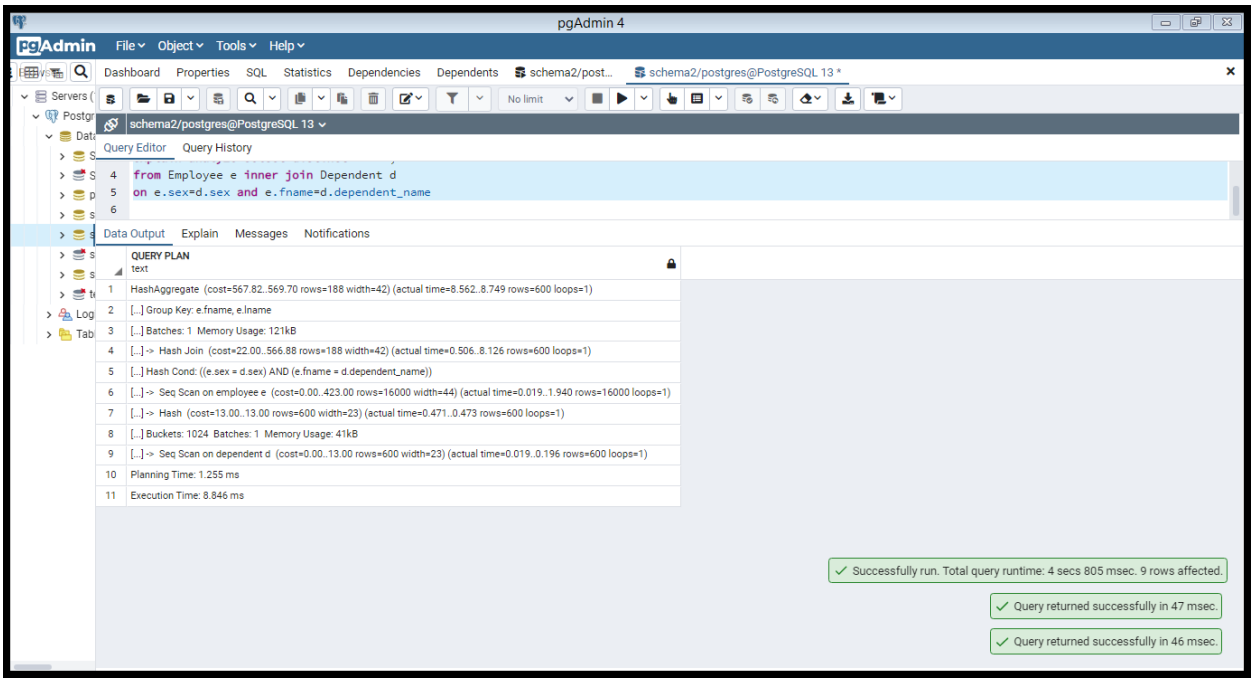
Execution plan and costs:



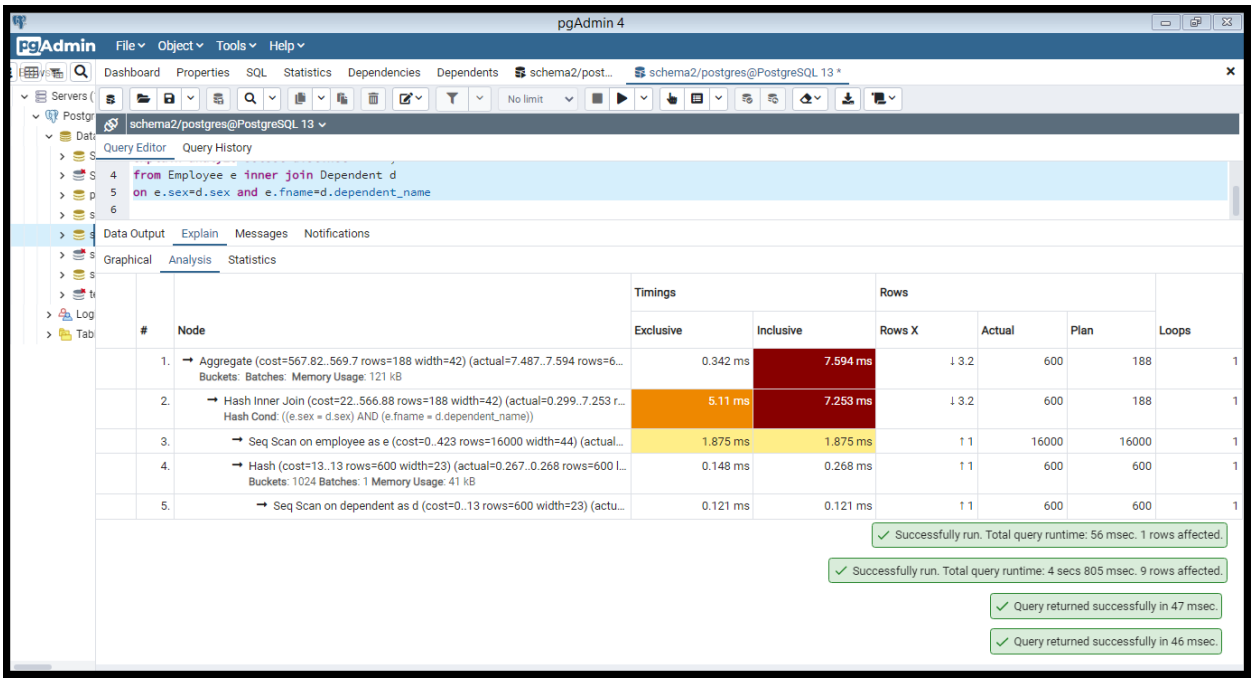
Here the cost increased too much as I disabled seqscan but the performance increased

With seqscan enabled:

Planning and execution time:



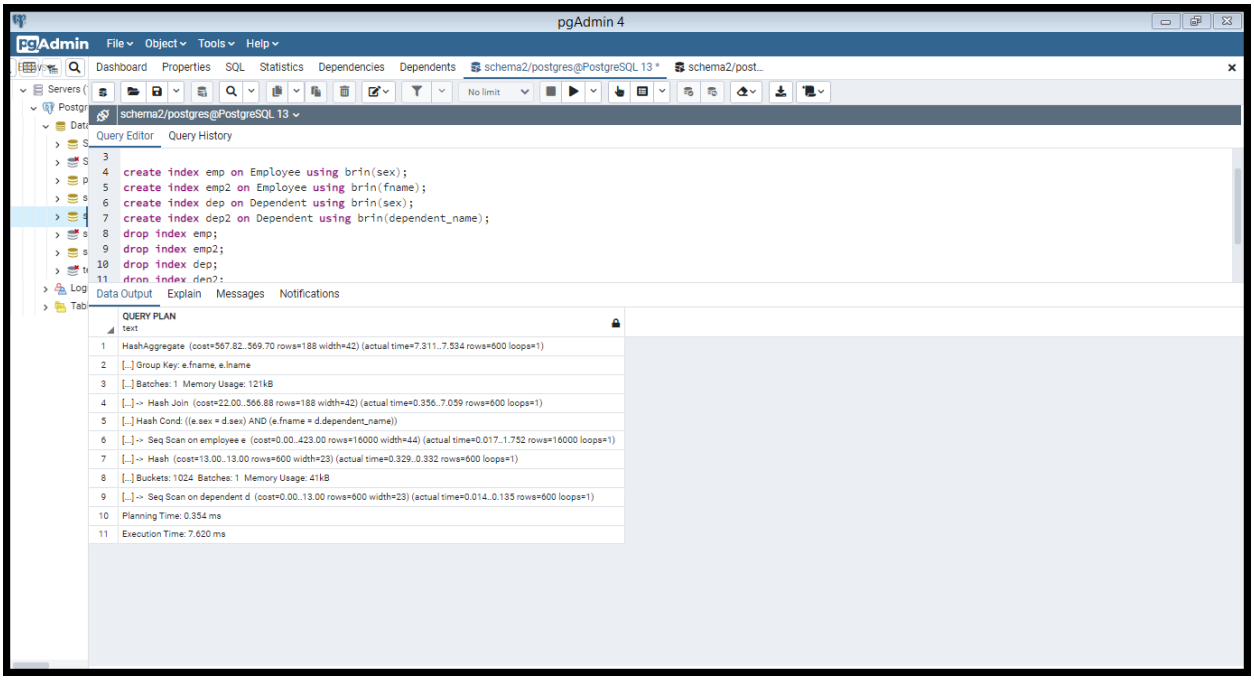
Execution plan and costs:



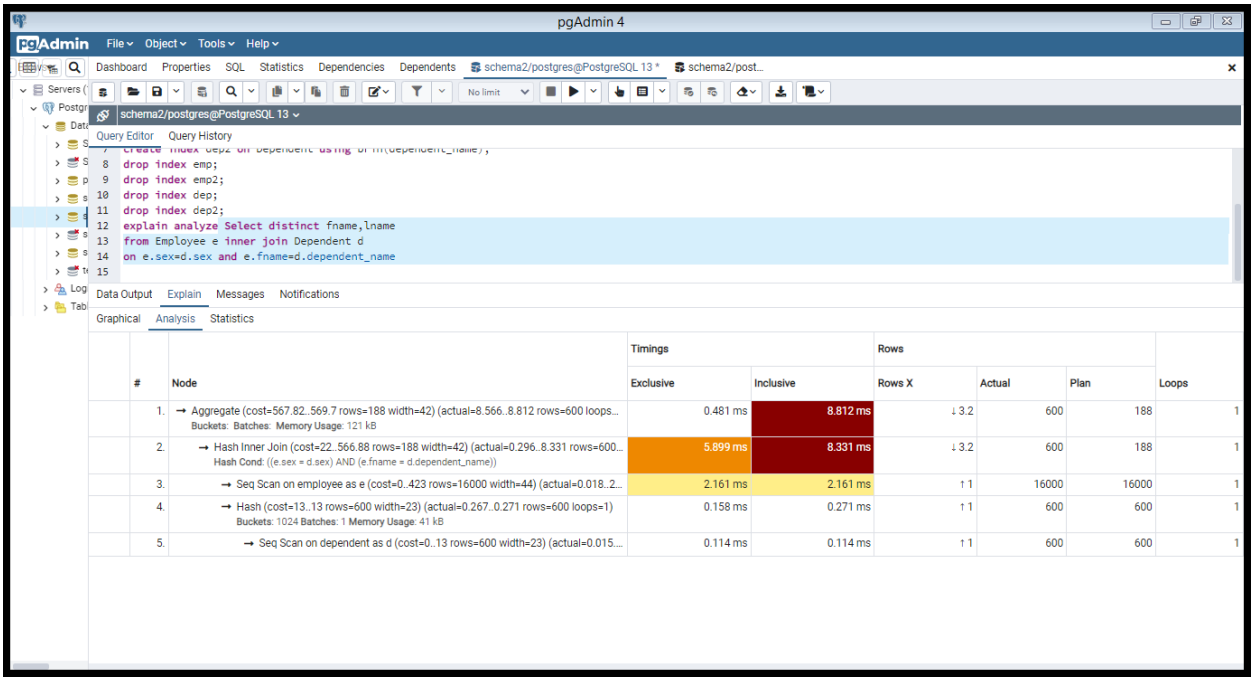
Query4 optimized using BRIN indecies on : Employee(sex), Employee(fname) , Dependent(dependent_name),Dependent(sex)

With seqscan enabled:

Planning and execution time:

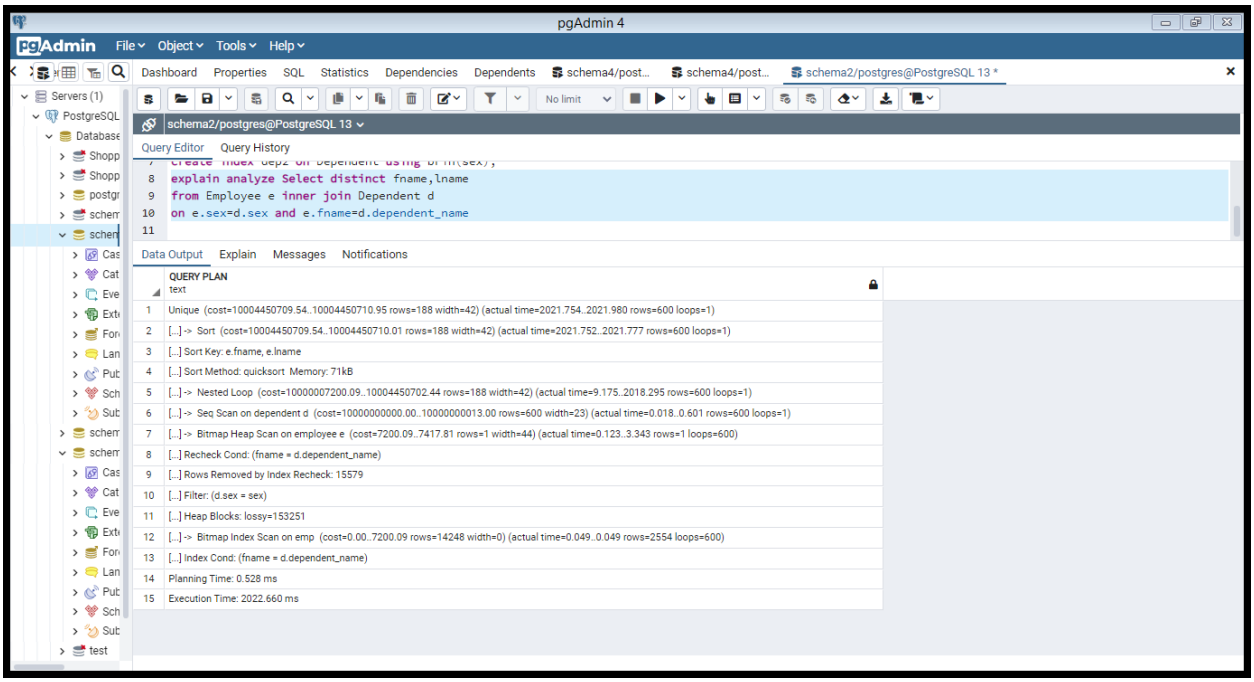


execution plan and costs:



With seq scan disabled :

Planning and execution time:



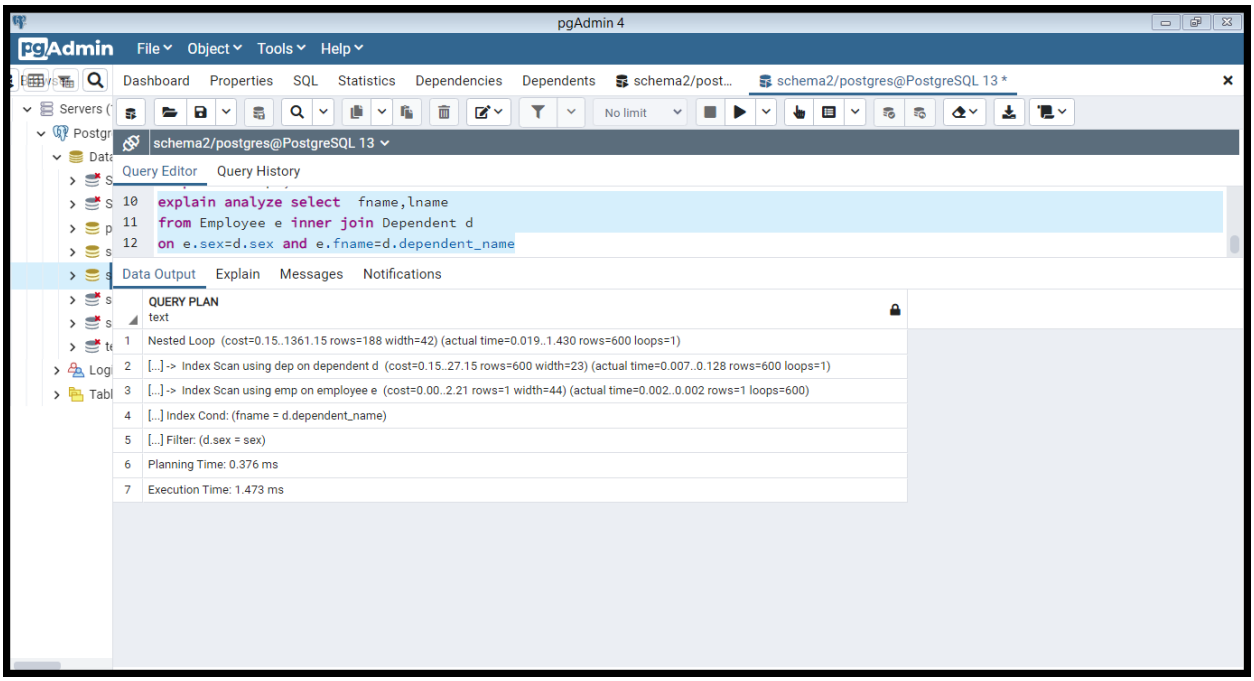
Both the cost and execution time increased too much as brin is used in small range queries only so it made It worse in this query

Query4 optimized using mixed indecies :

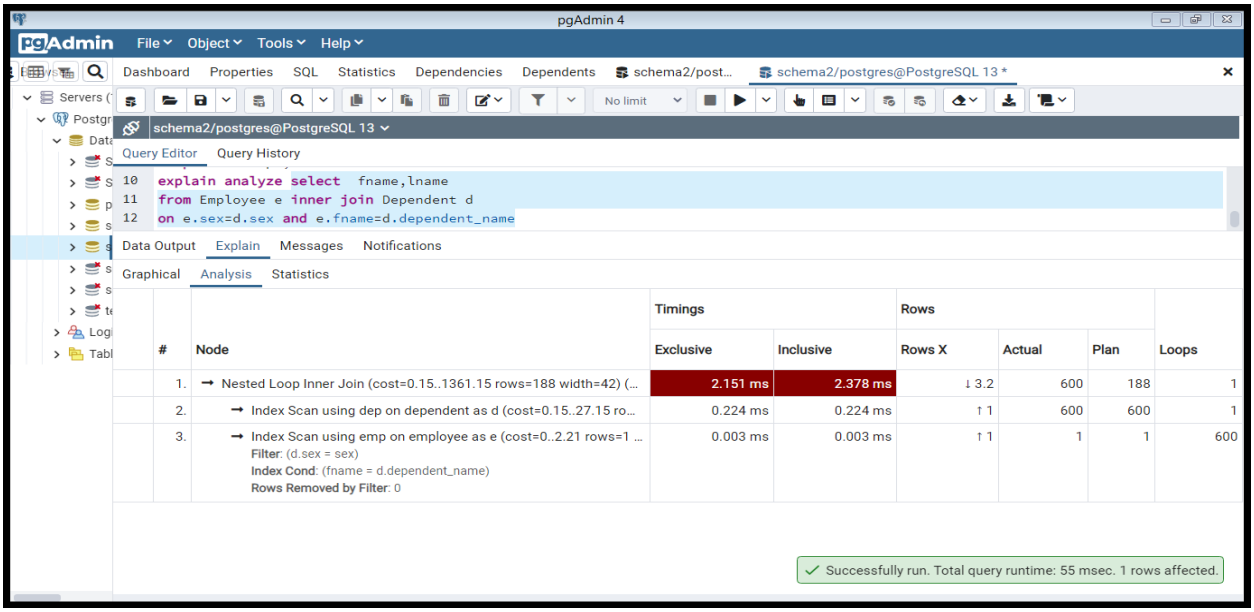
Hash index on Employee(sex), Employee(name) and B+ tree on Dependent(sex), Dependent(dependent_name)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



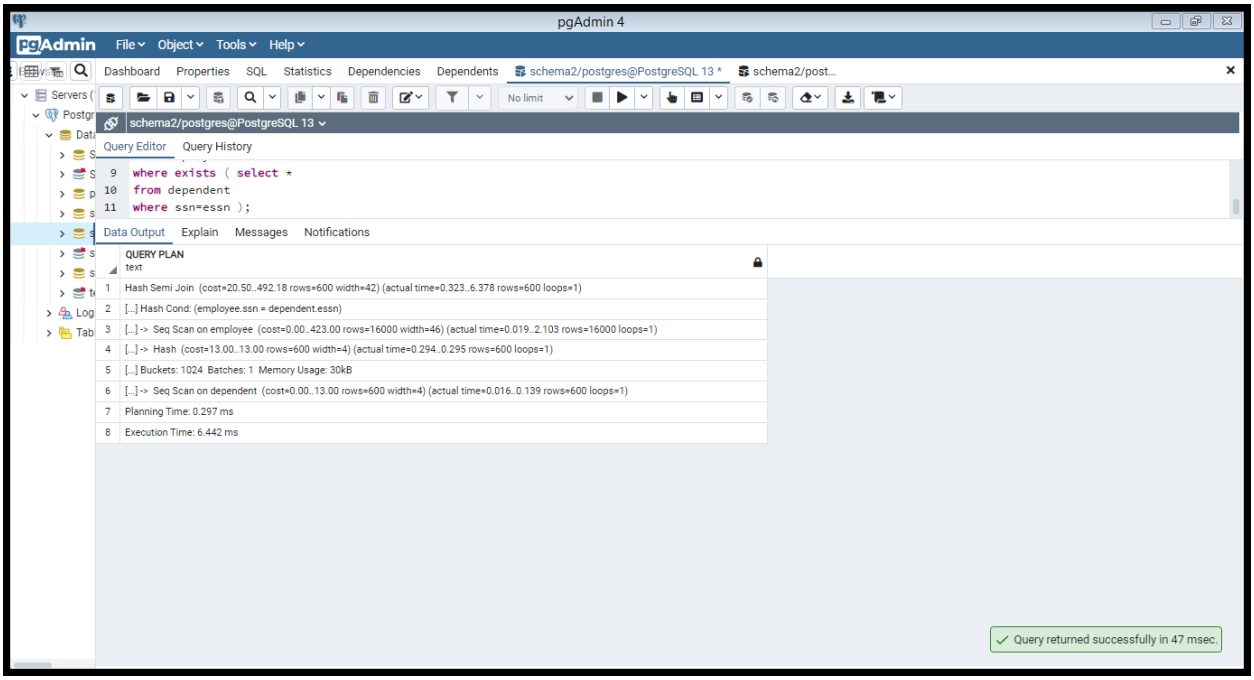
Execution plan and costs:



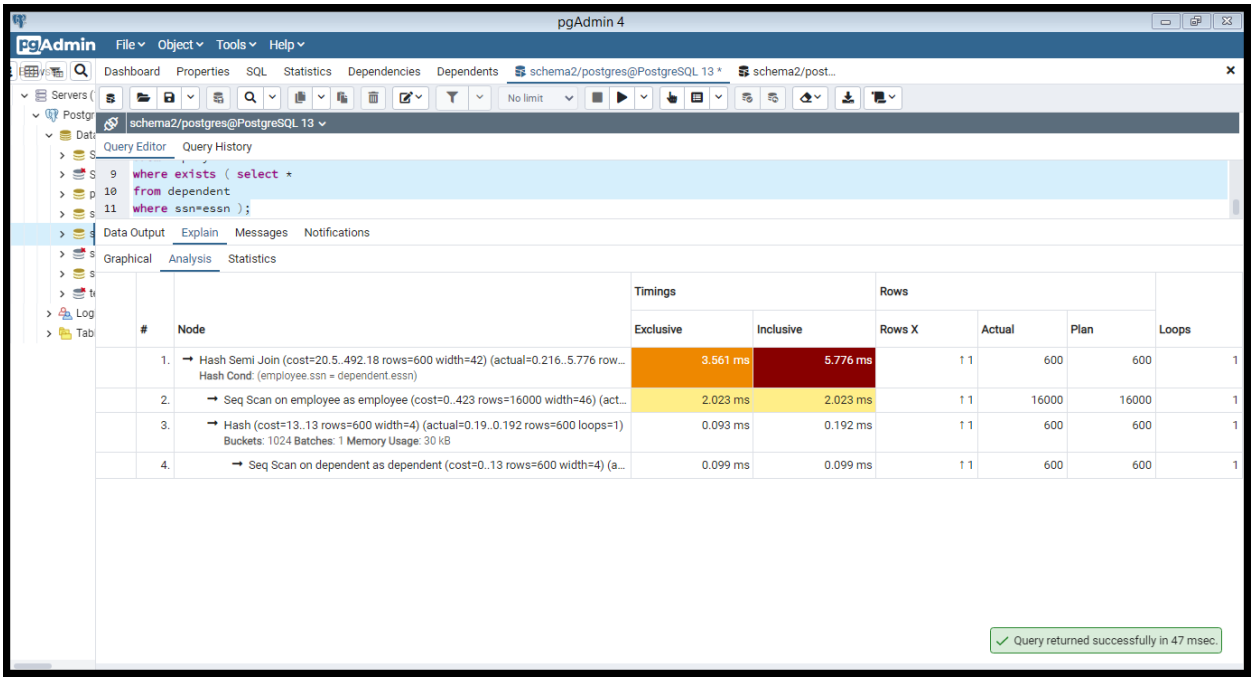
Here the hash increased performance as it helped in searching for the dependent as it search in O(1) time

Query5 with no index

Planning and execution time:



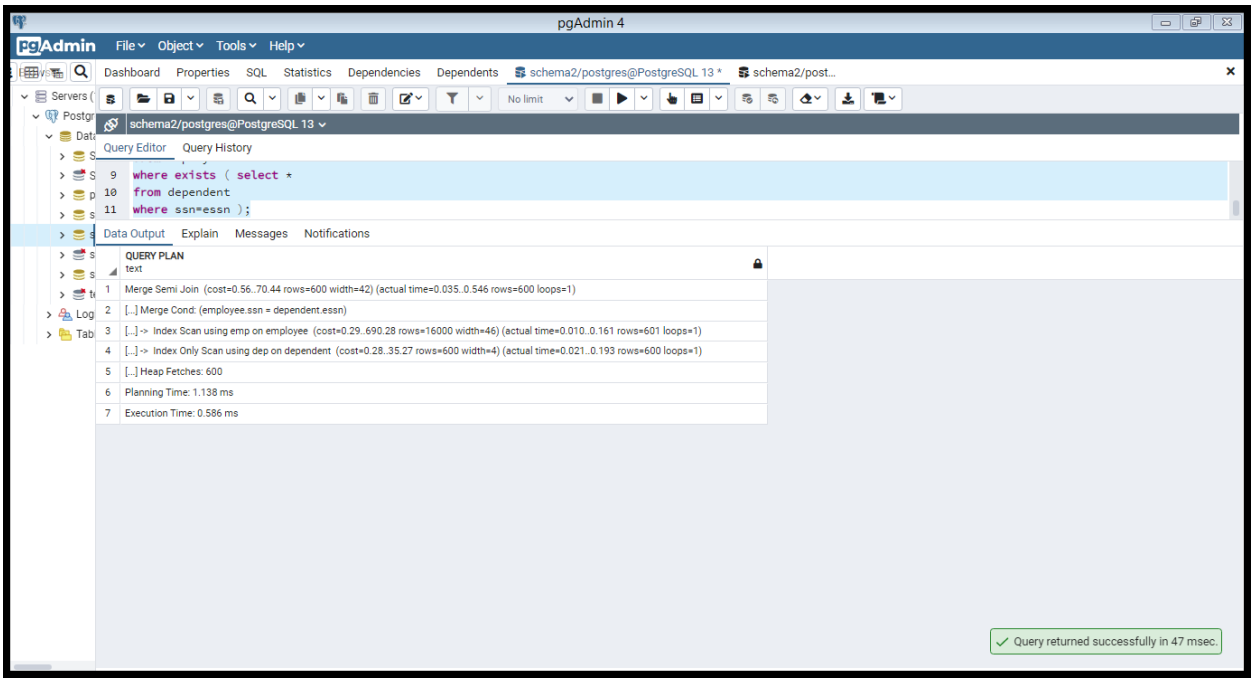
Execution plan and costs:



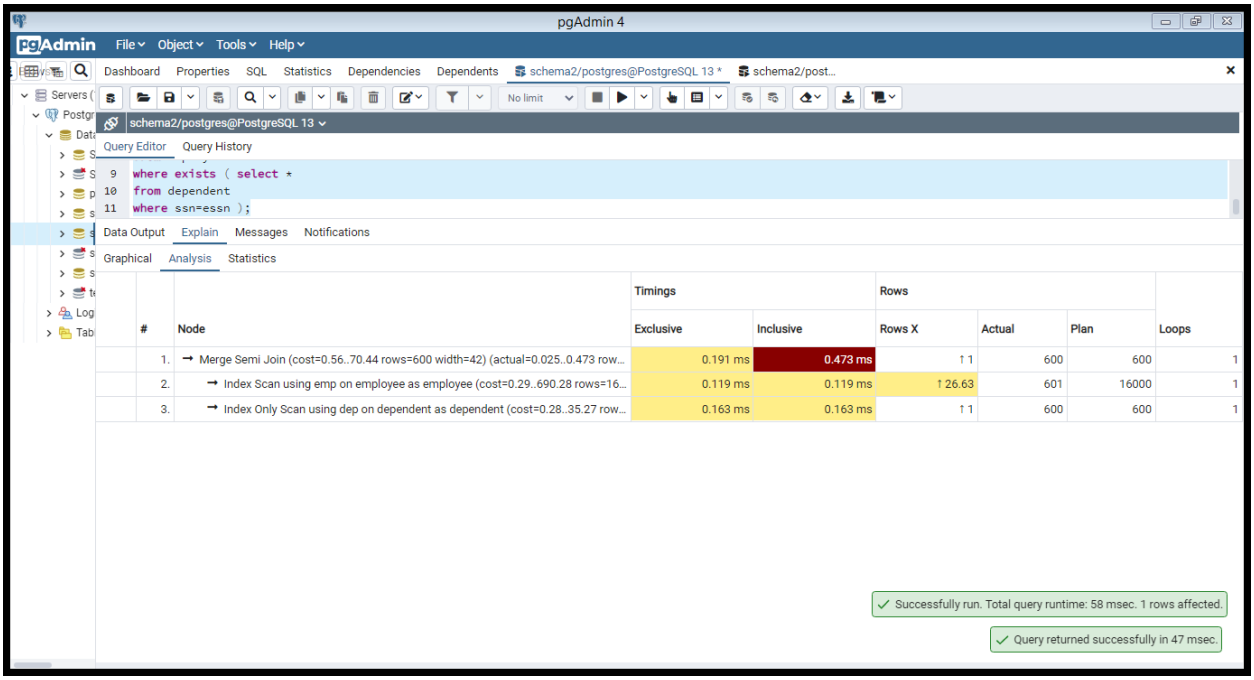
Query5 with B+ tree indecies on : Dependent(essn),Employee(ssn)

Disabled flags: seqscan and bitmap scan

Planning and execution time:



Execution plan and costs:

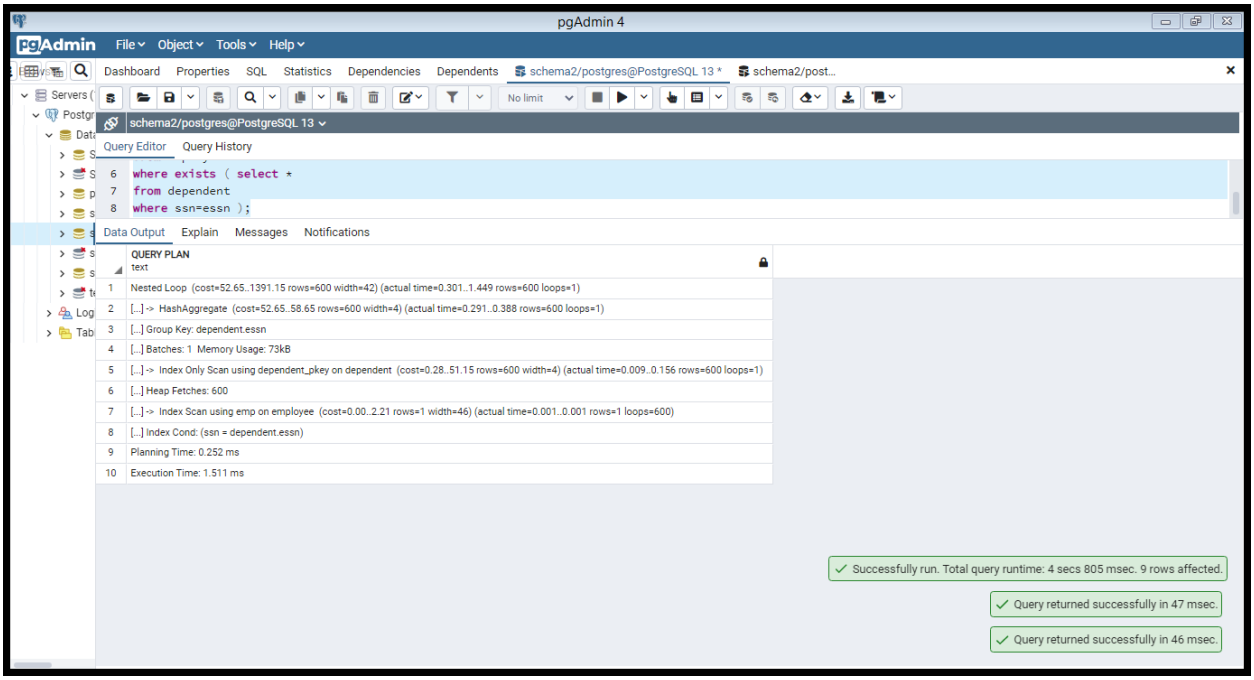


here the B+ tree helped in this query as it search in O(logn) time

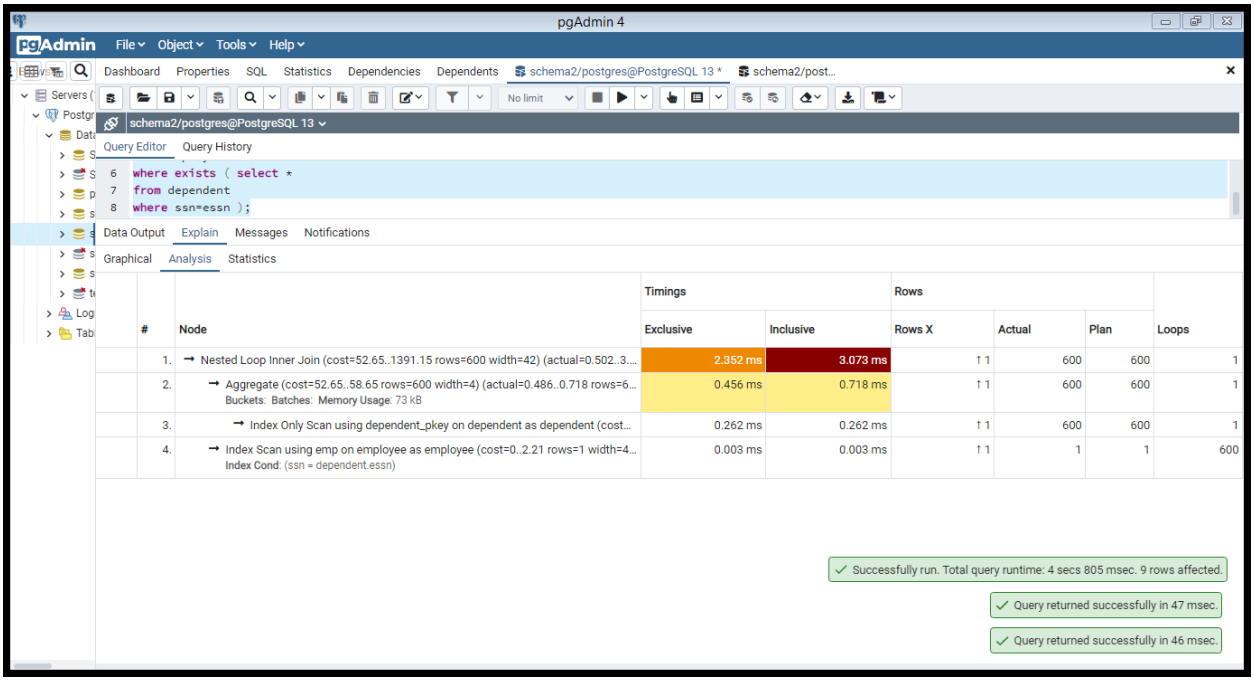
Query5 using hash indecies on : Dependent(essn) , Employee(ssn)

Disabled flags: seqscan and bitmapscan

Planning and execution time:



Execution plan and costs:

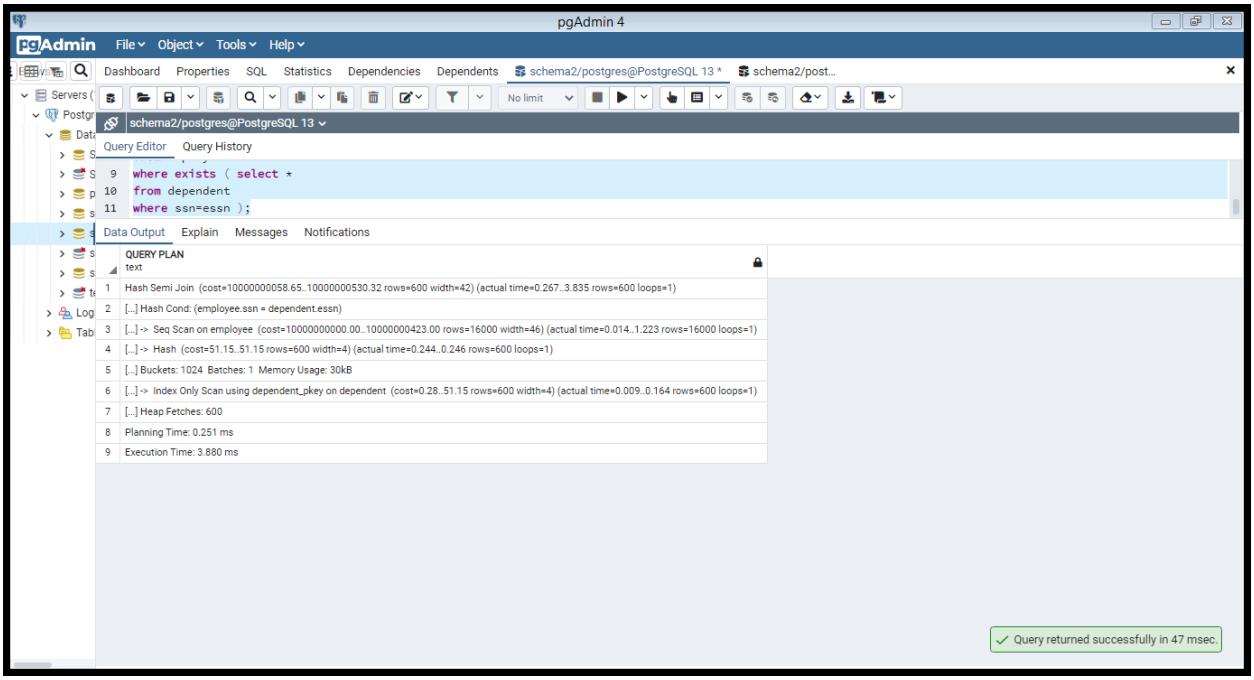


Here the hash improved performance as hash is used in exact value queries as it search in O(1) it helped in the join using the ssn

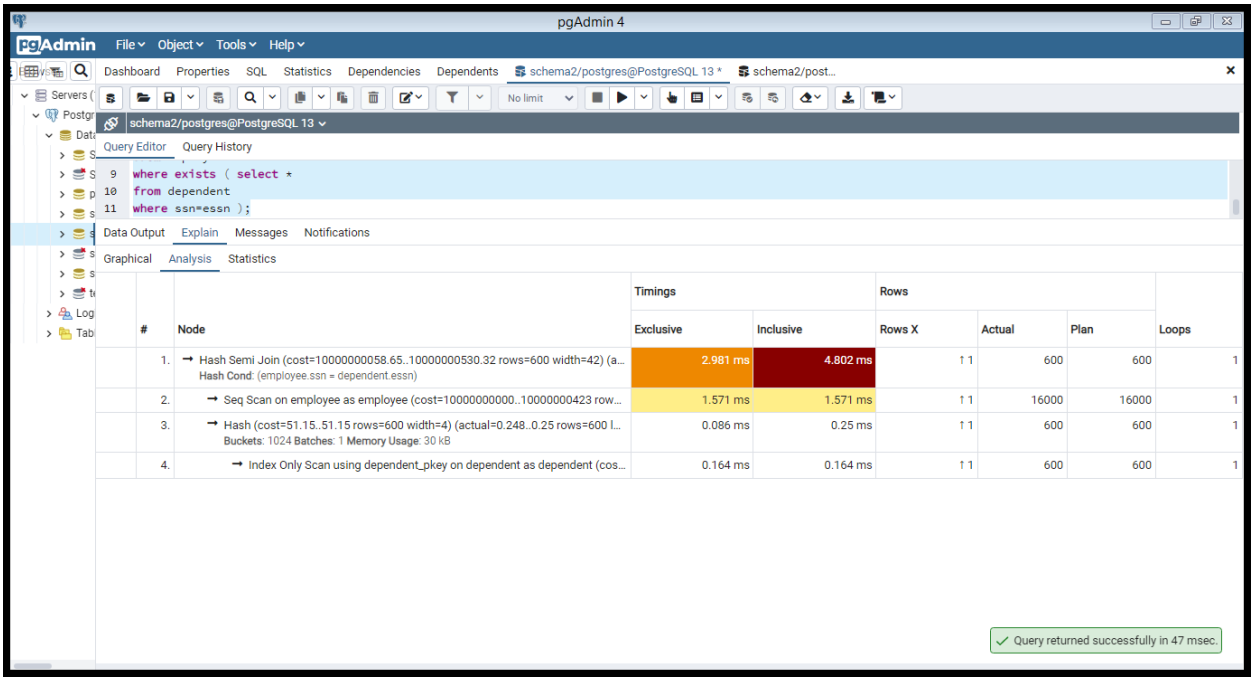
Query5 using BRIN indecies on : Dependent(essn) , Employee(ssn)

With seqscan disabled

Planning and execution time:



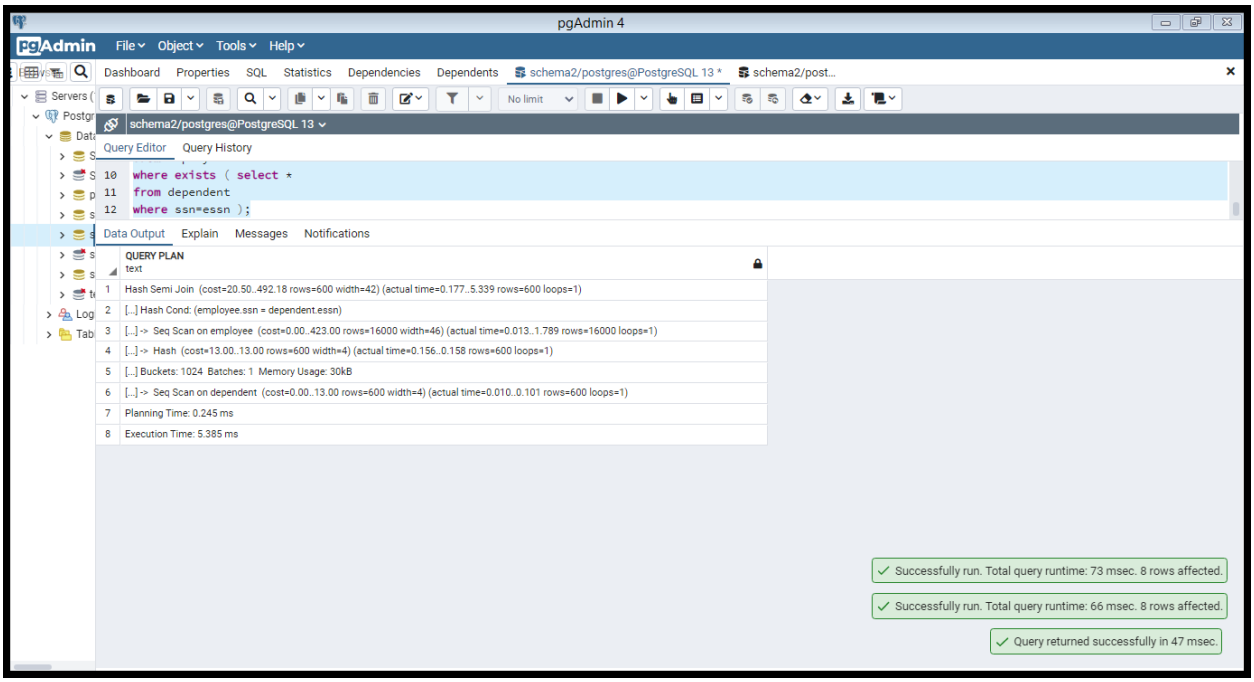
Execution plan and costs:



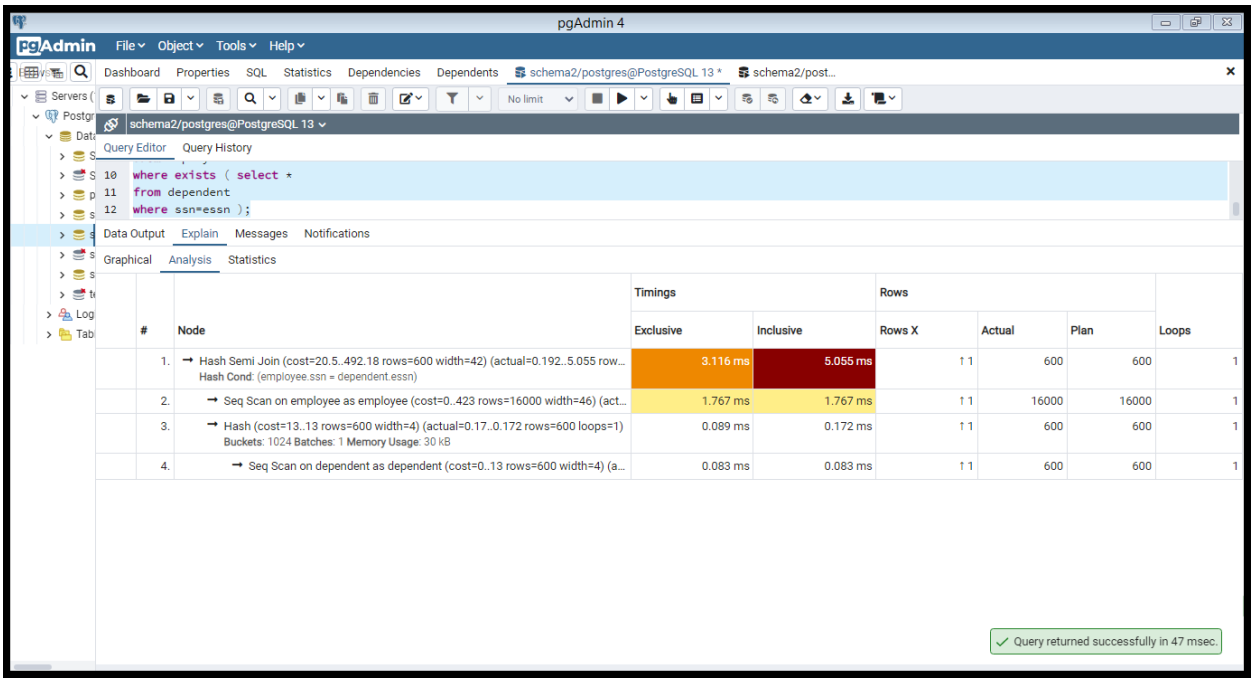
Here the brin increased the cost to much as it is used in small range queries only

With seqscan enabled:

Planning and execution time:

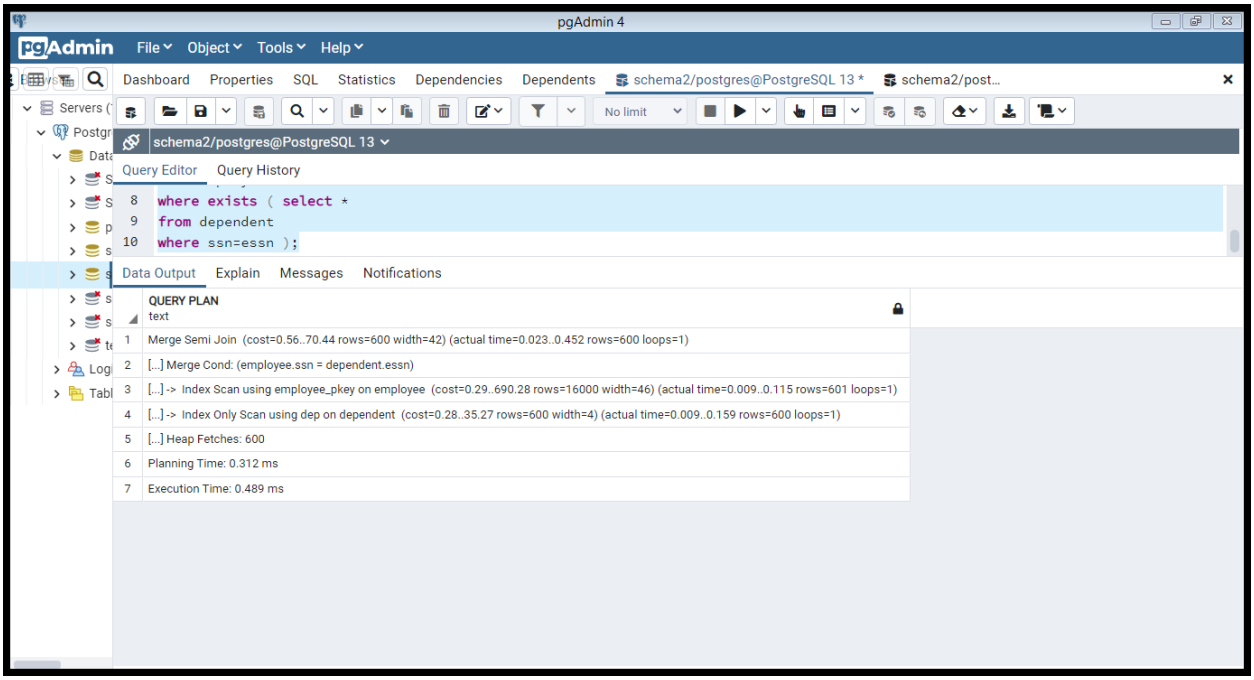


Execution plan and costs:

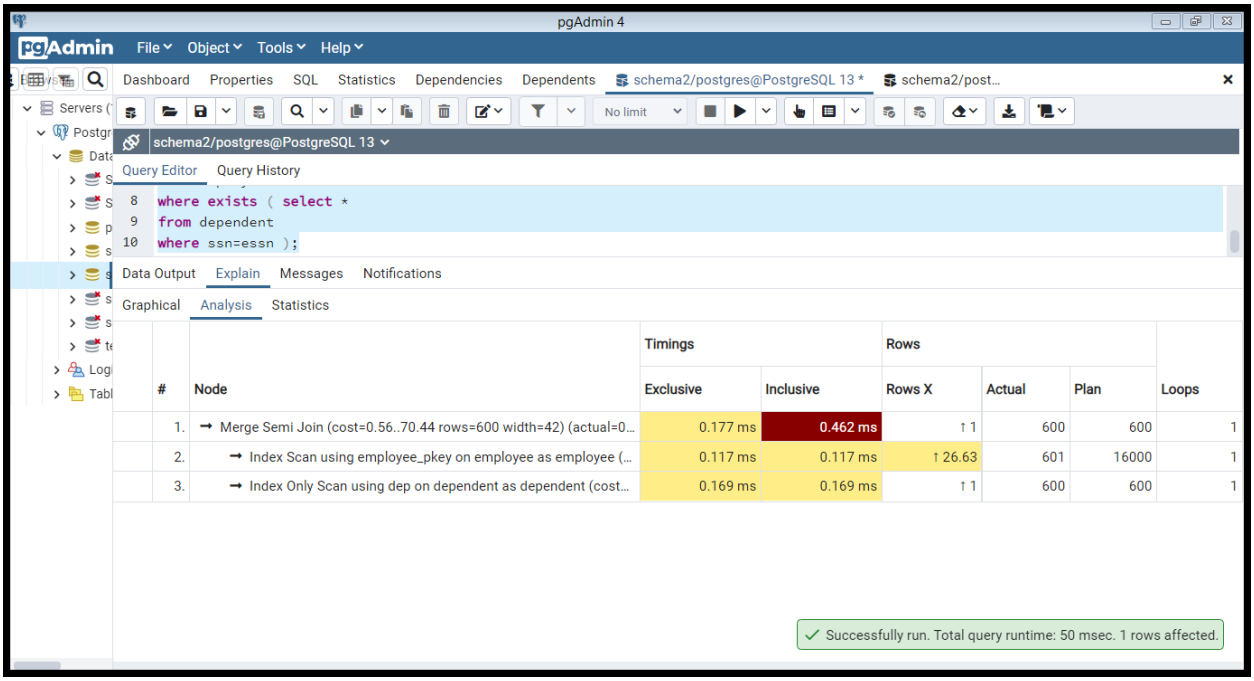


Query5 using mixed indecies : hash on Employee(ssn) and B+ tree on Dependent(essn)

Planning and execution time :



Execution plan and costs:

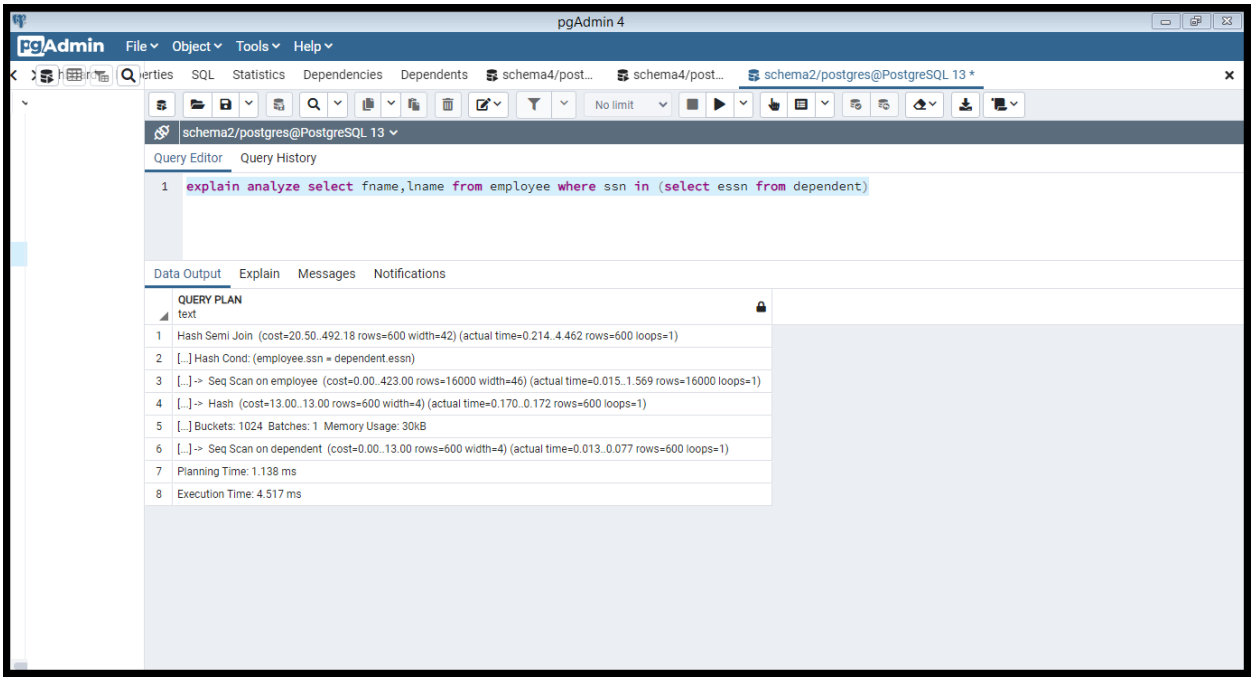


Query5 after optimization :

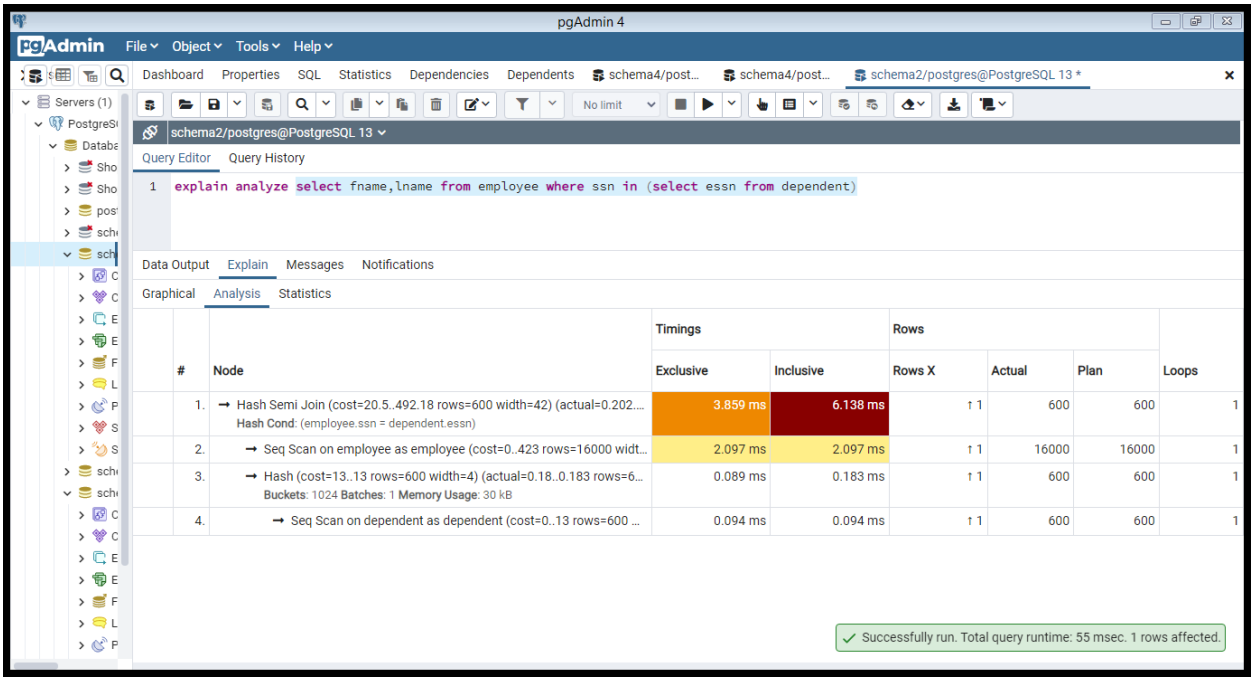
It can not be optimized too much as exists statement is optimized but we can use IN statement instead that might be better or equal to it

select fname,lname from employee where ssn in (select essn from dependent)

Query5 optimized with no index
Planning and execution time:

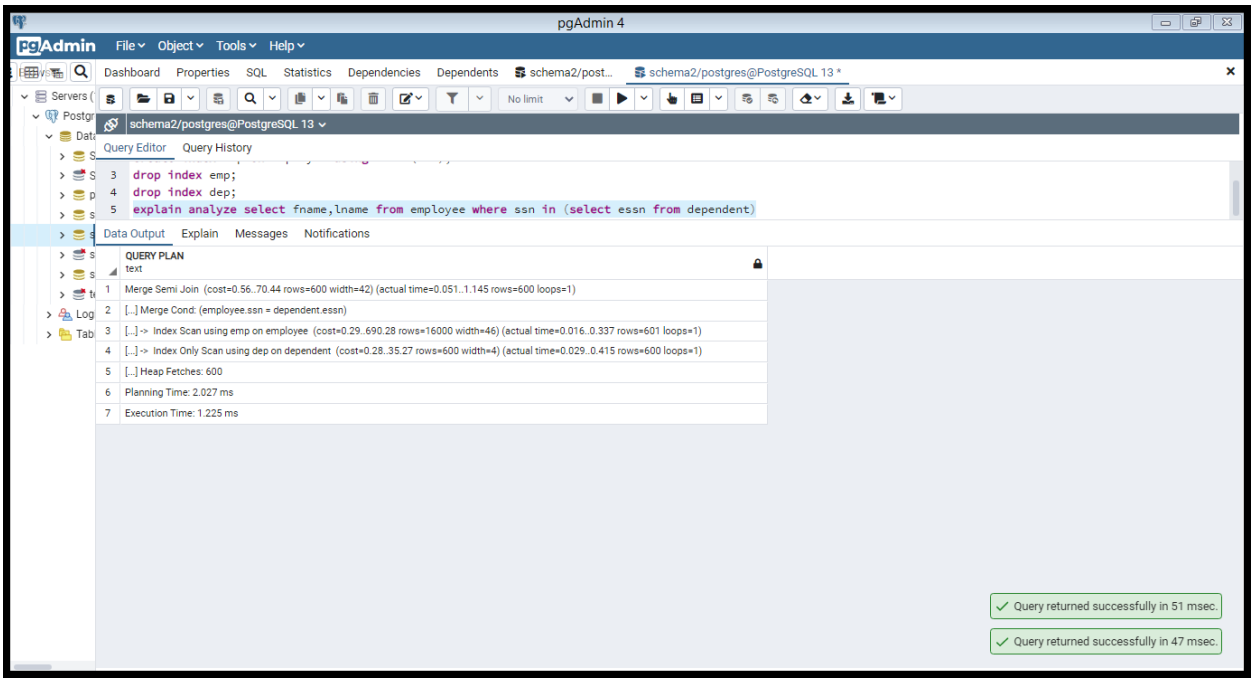


Execution plan and costs:

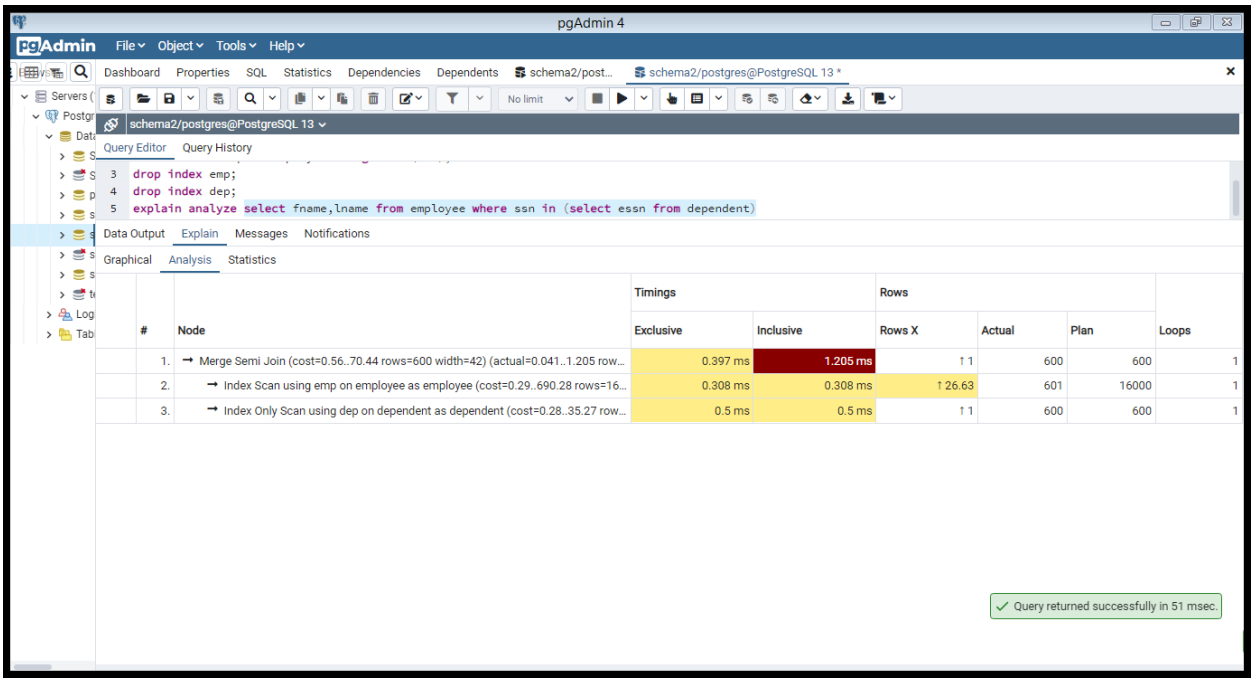


Query5 optimized using B+ tree indecies on : Dependent(essn) ,Employee(ssn)

Planning and execution time:

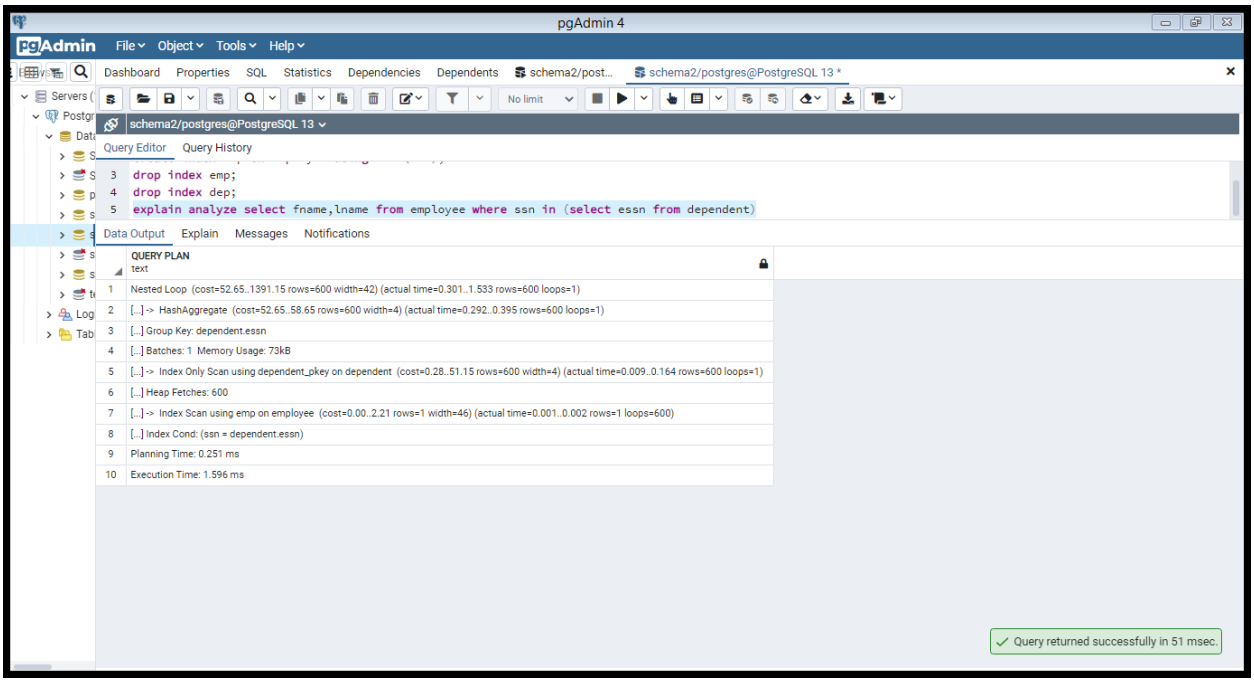


Execution plan and costs:

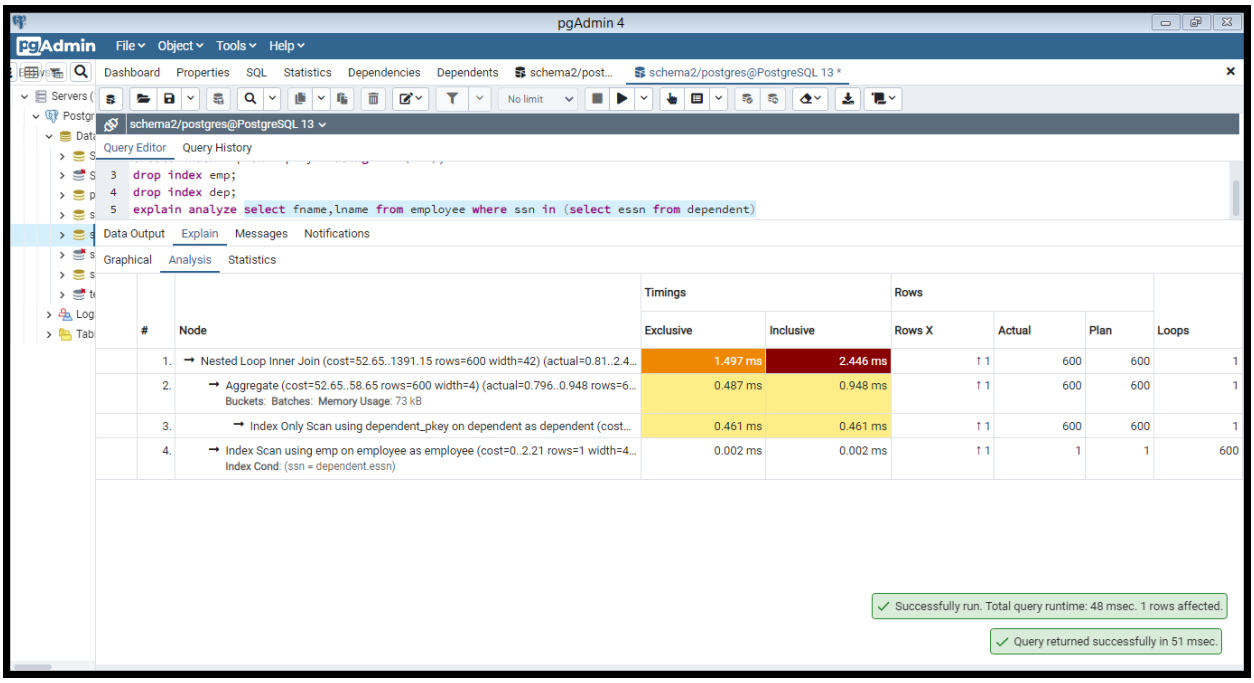


Query5 optimized using hash indecies on : Dependent(essn) , Employee(ssn)

Planning and execution time:



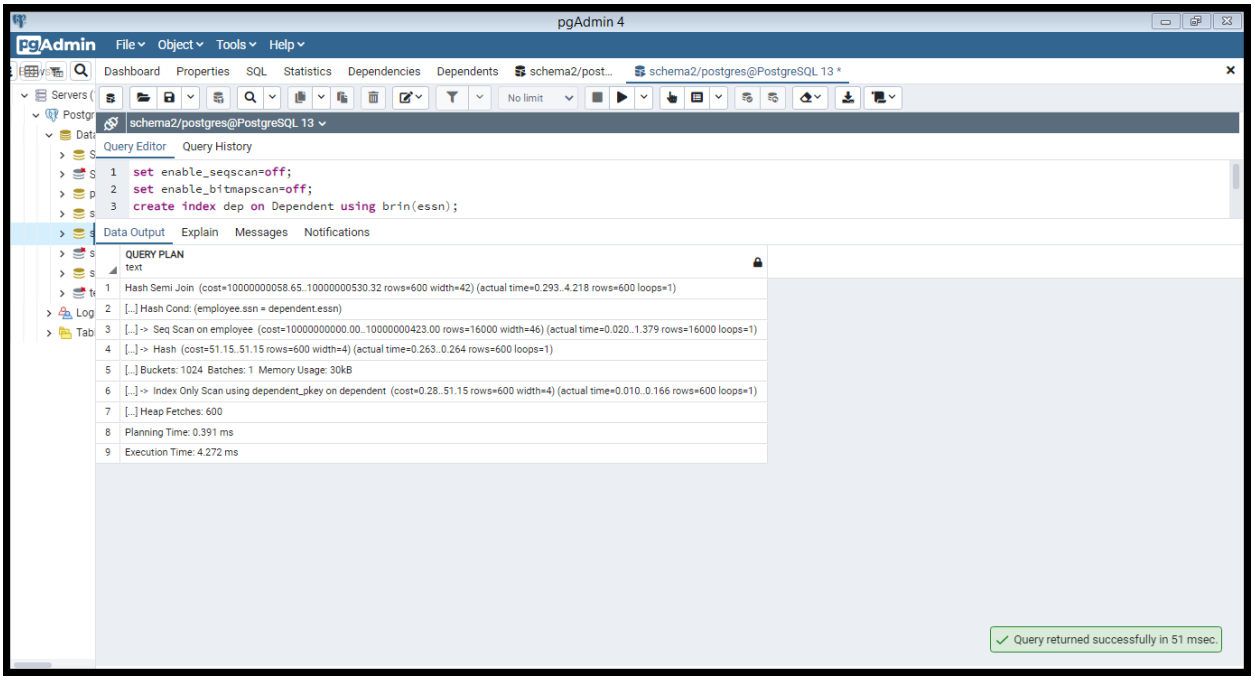
Execution plan and costs:



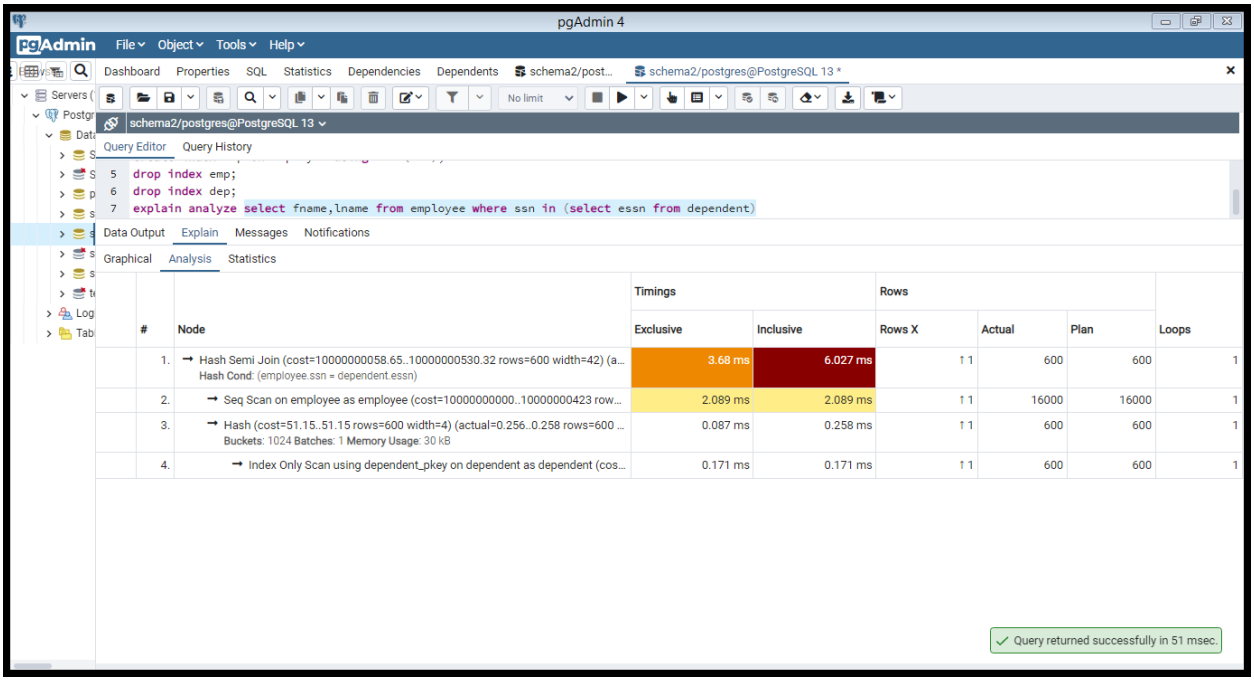
Query5 optimized using BRIN indecies on: Dependent(essn) , Employee(ssn)

With seqscan disabled

Planning and execution time :



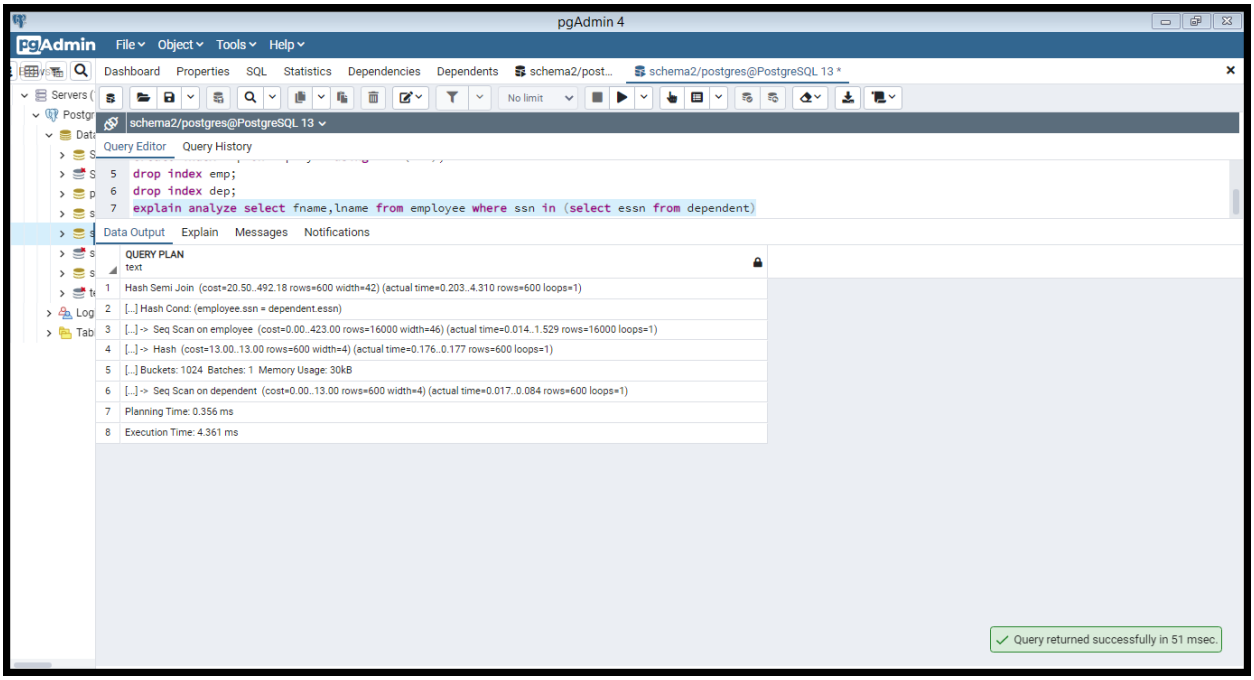
Execution plan and costs:



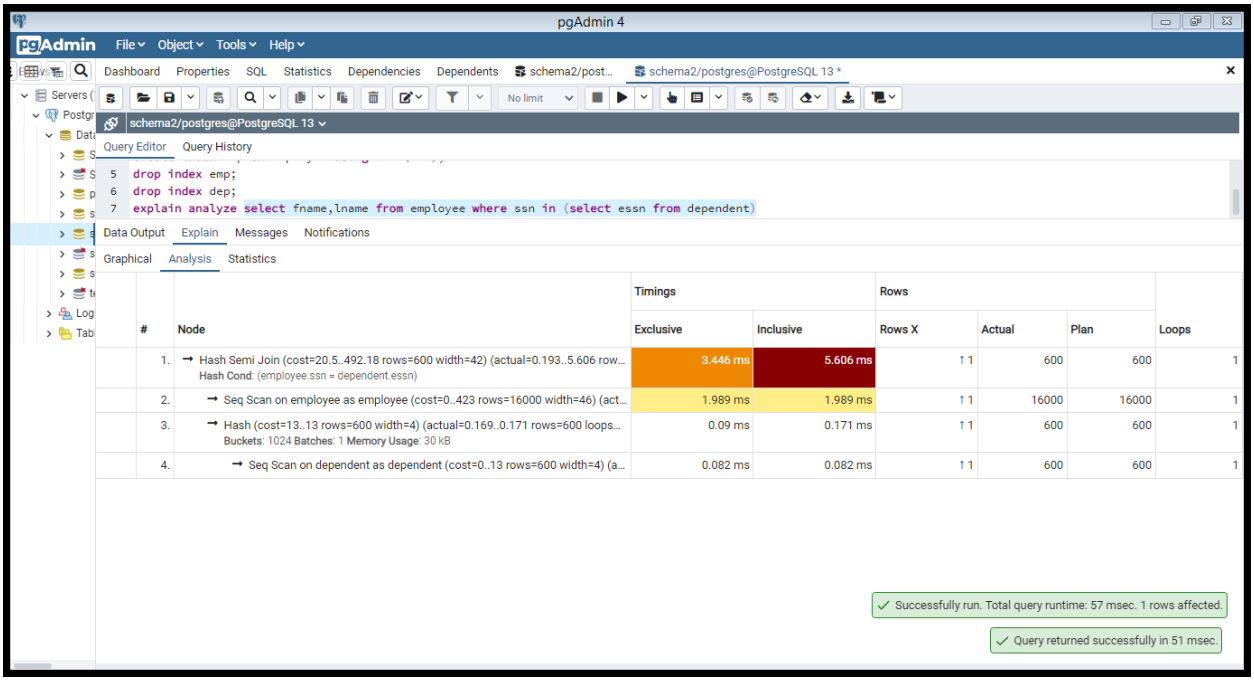
Here the brin increased the cost to much as it is used in small range queries only

With seqscan enabled

Planning and execution time:

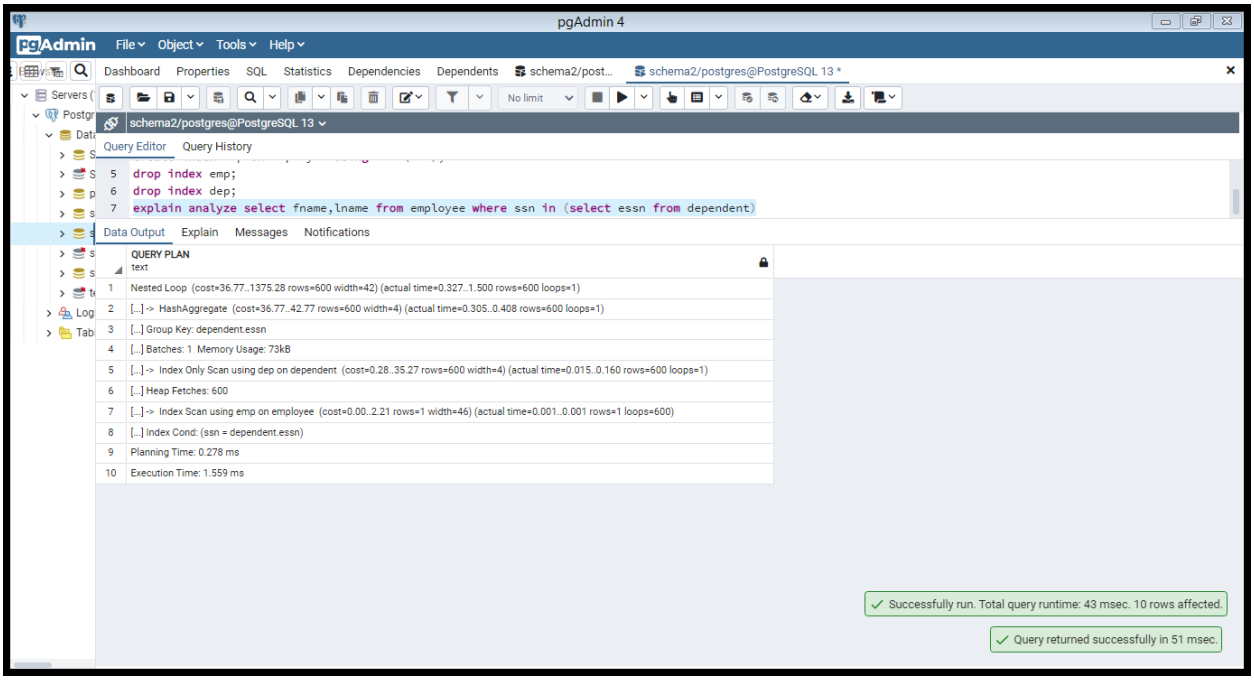


Execution plan and costs:

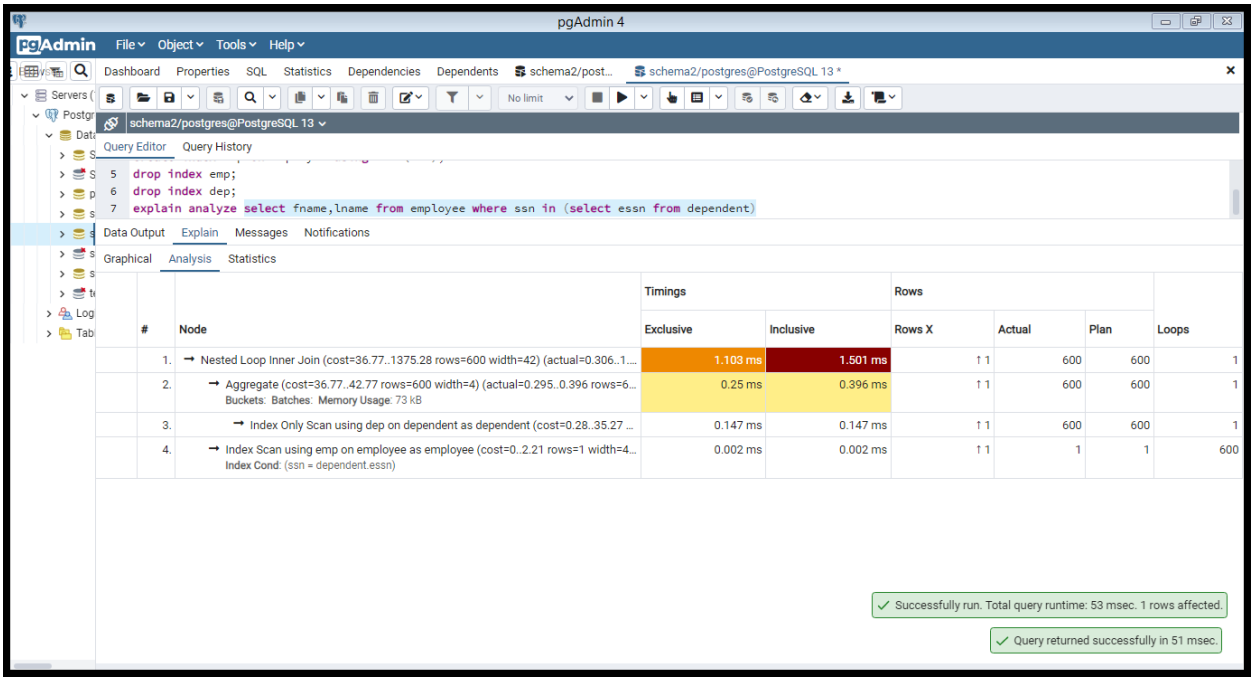


Query5 optimized using mixed indecies : hash on Employee(ssn) and B+ tree on Dependent(essn)

Planning and execution time:

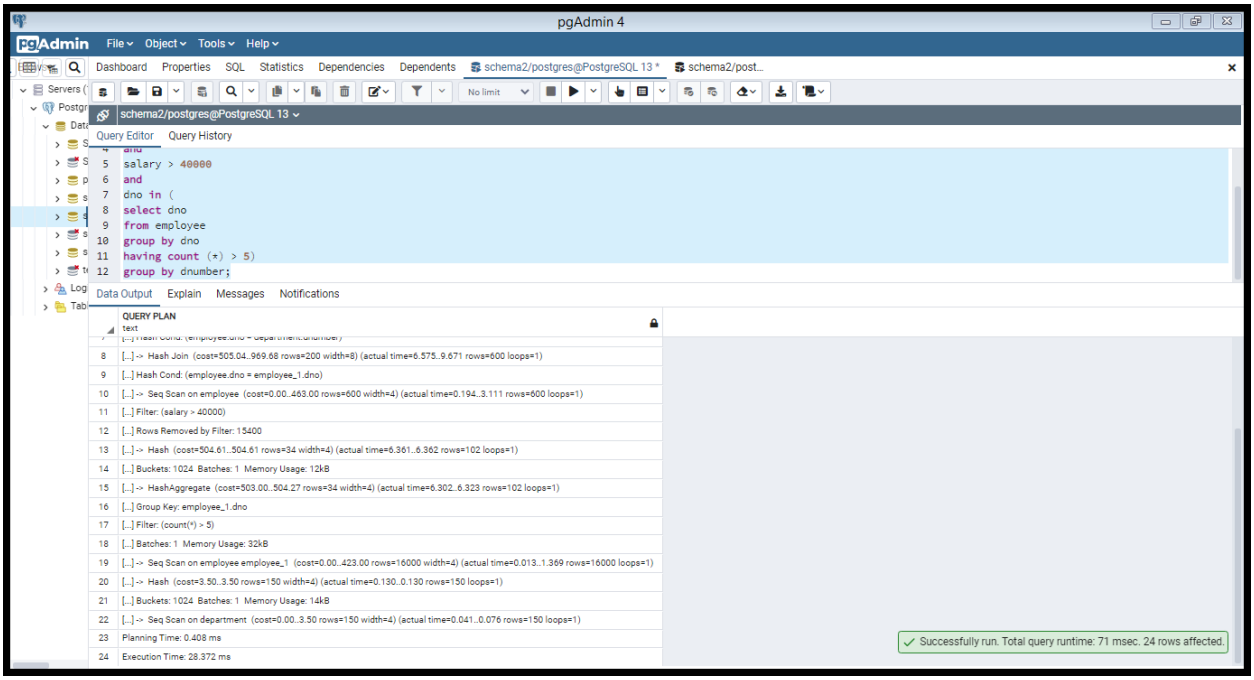


Execution plan and costs:

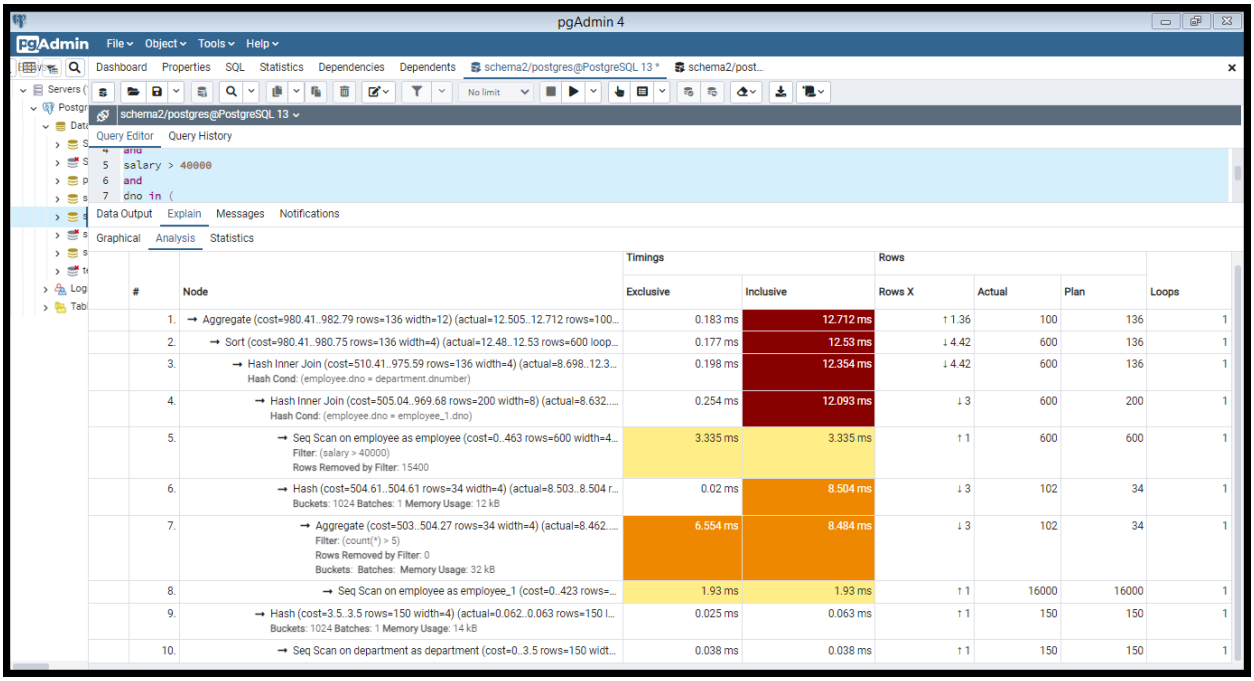


Query6 with no index

Planning and execution time:



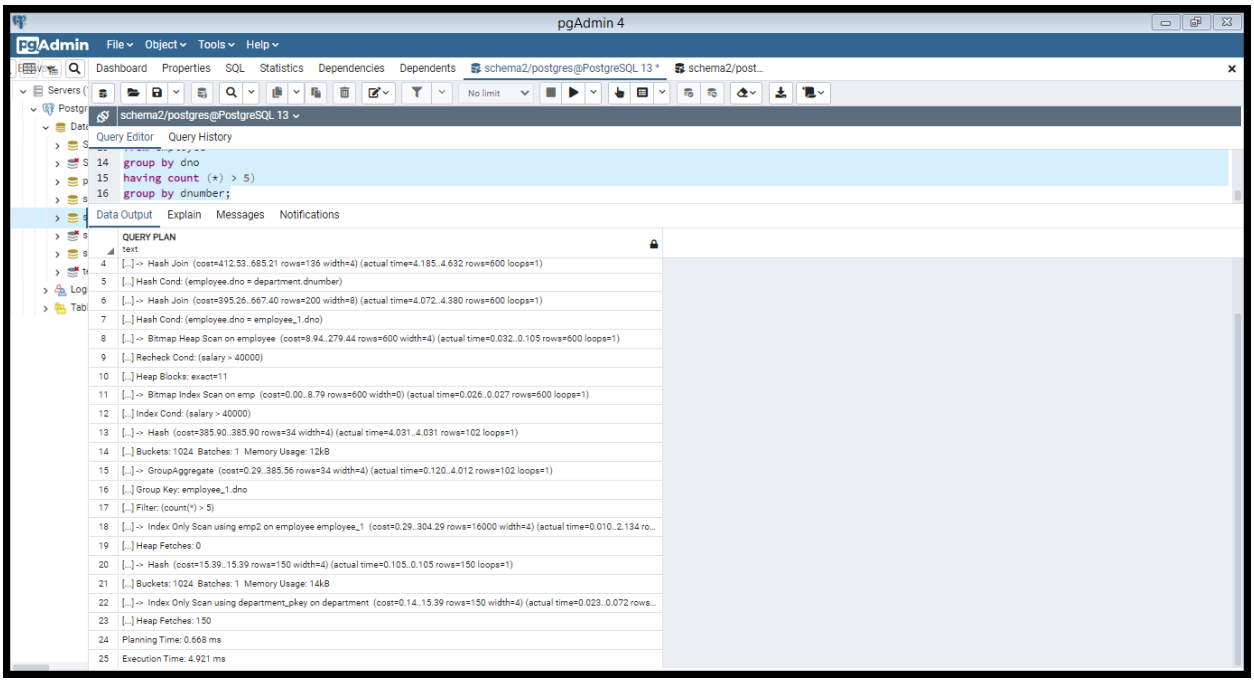
Execution plan and costs:



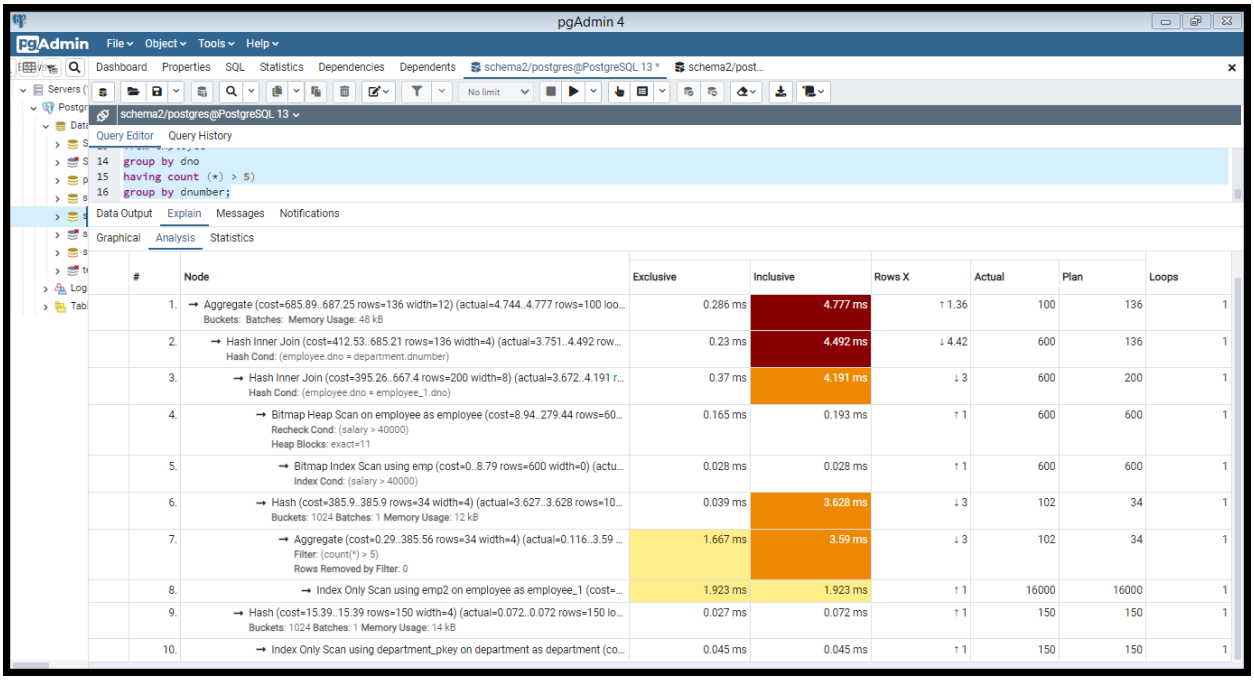
Query6 using B+ tree on : Employee(salary), Employee(dno)

Disabled flags: seqscan

Planning and execution time:



Execution plan and costs:

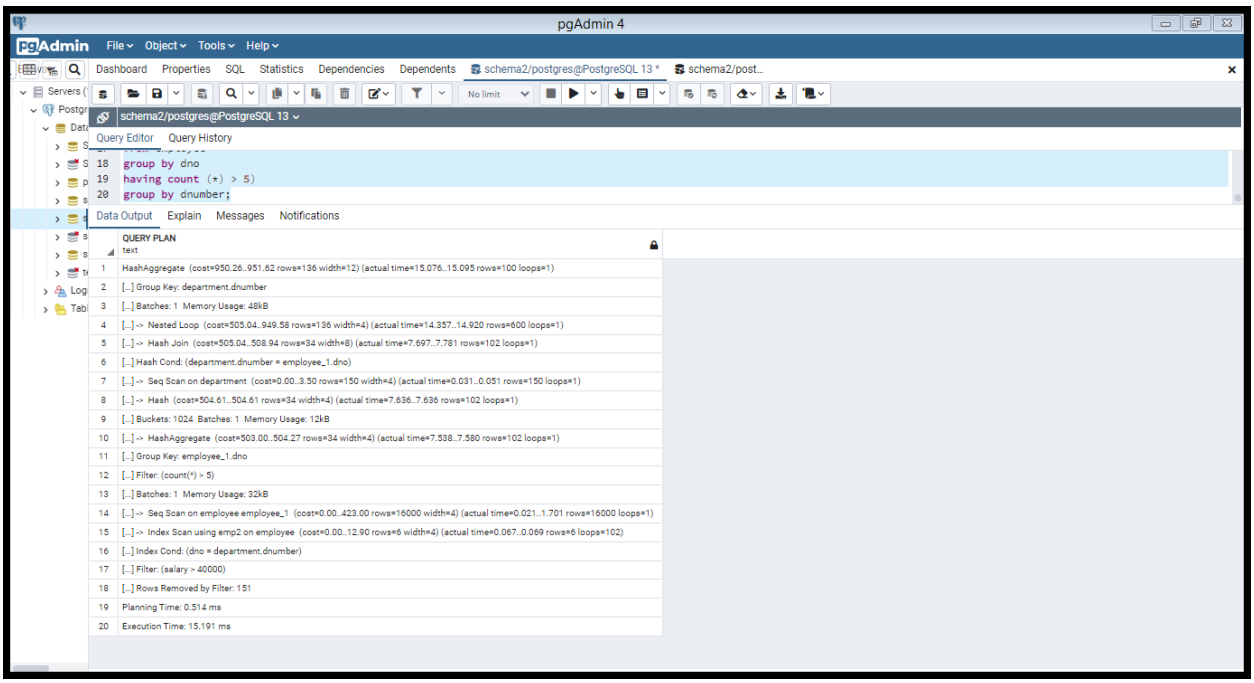


Here the B+ tree increased performance as it is used in range queries and it search in O(logn) time

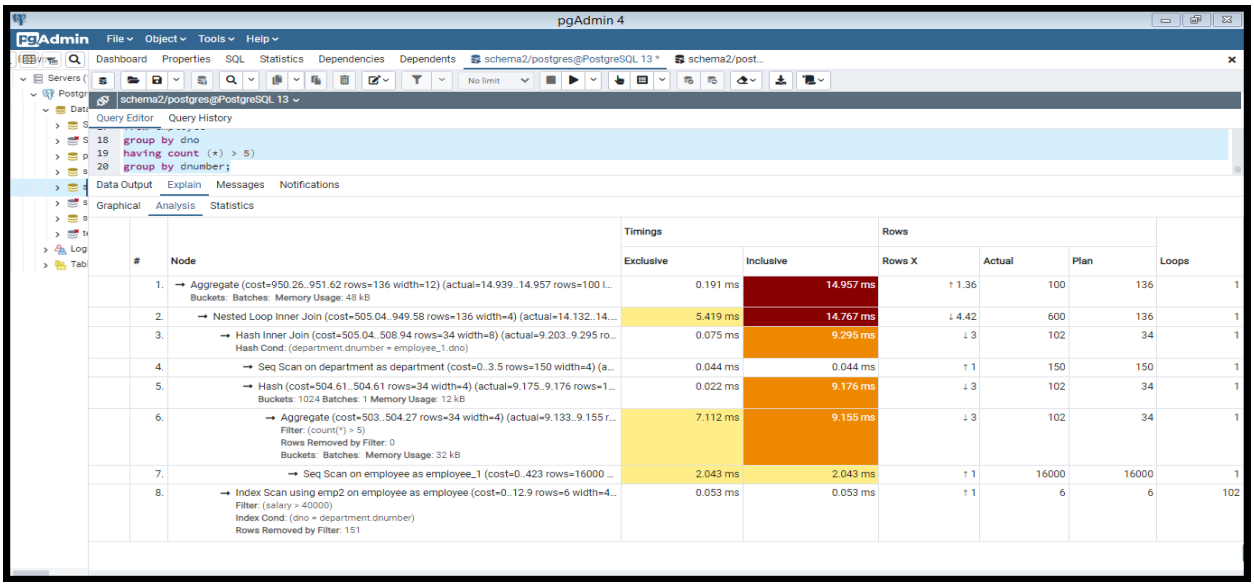
Query 6 using hash indecies on : Employee(salary), Employee(dno) ,Department(Dnumber)

Disabled flags: bitmapscan and index scan

Planning and execution time:



Execution plan and costs:

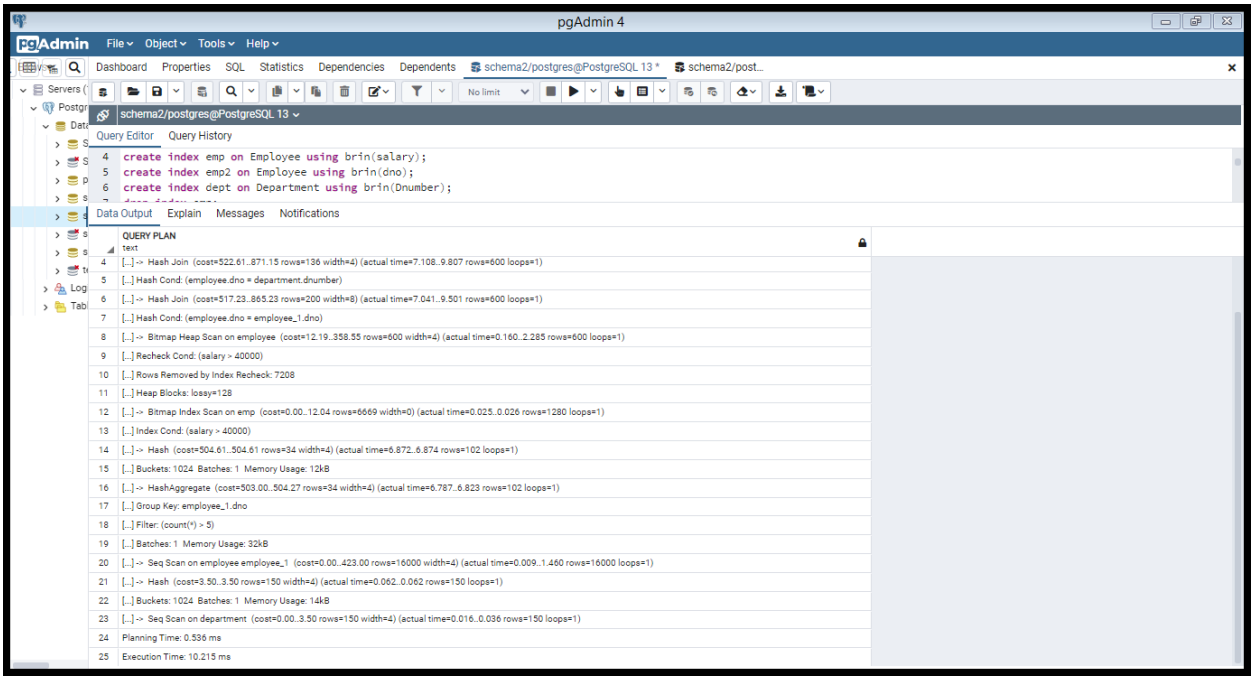


Here the hash did not help that much as it is used in exact value queries it does not help in range queries as it search in O(1) time

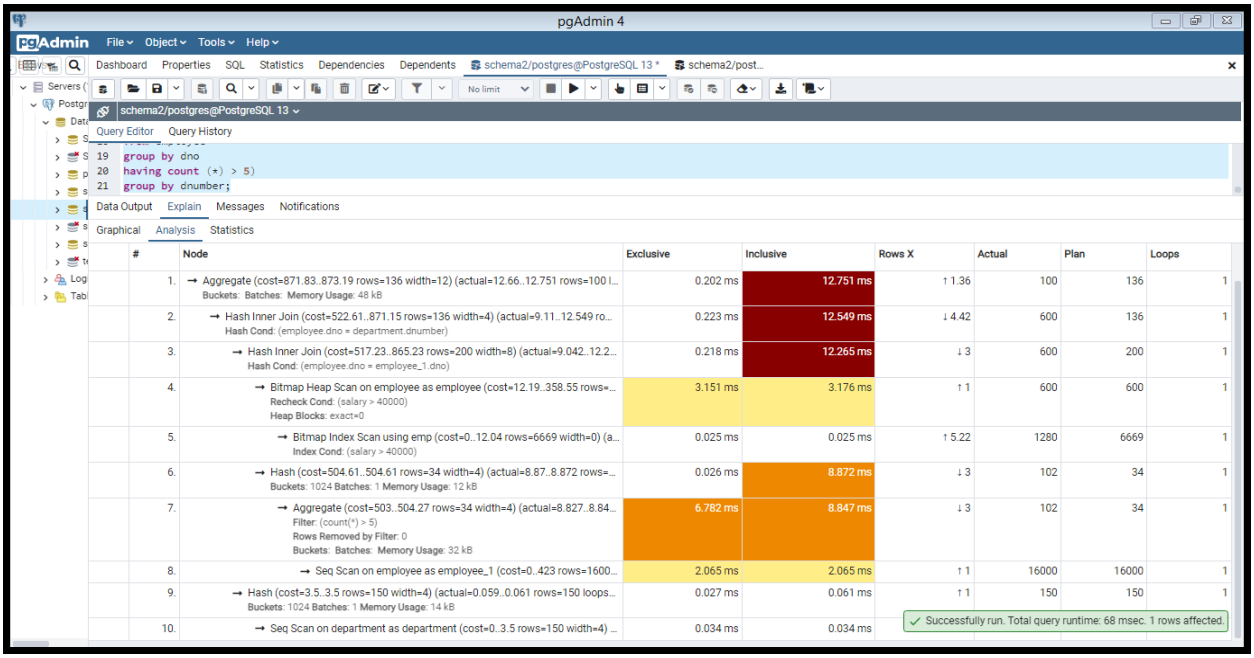
Query6 using BRIN indecies on : Employee(salary), Employee(dno) ,Department(Dnumber)

Disabled flags: seqscan

Planning and execution time:



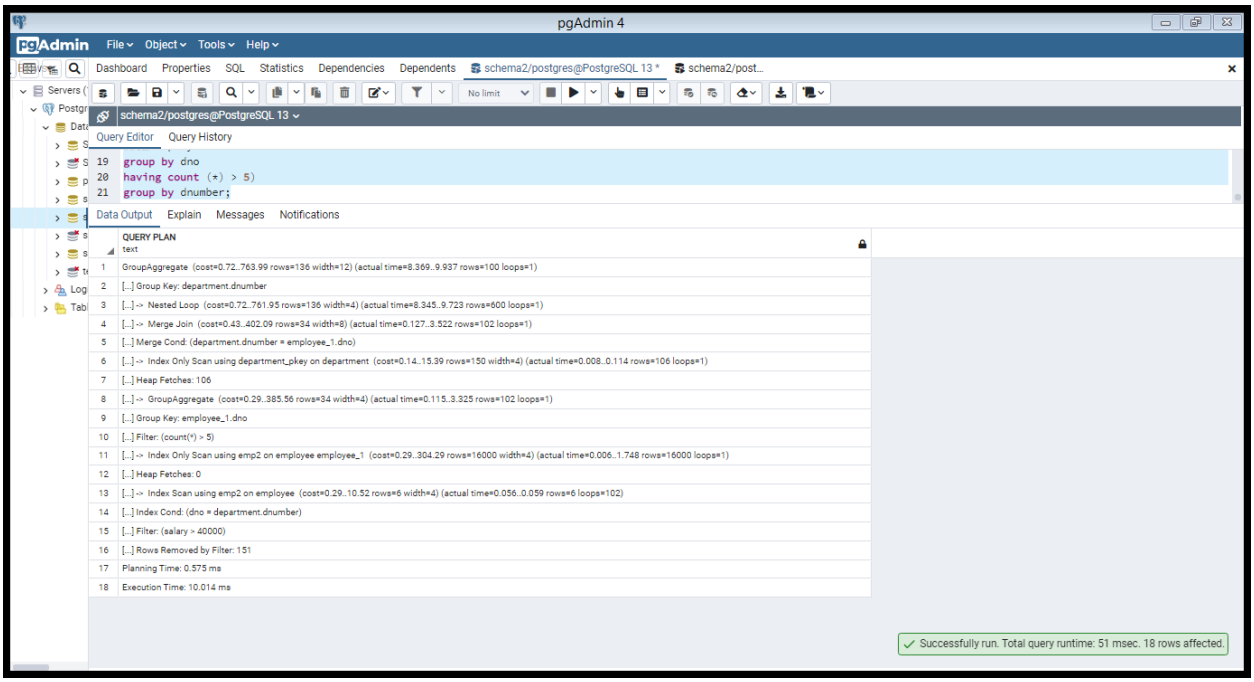
Execution plan and costs:



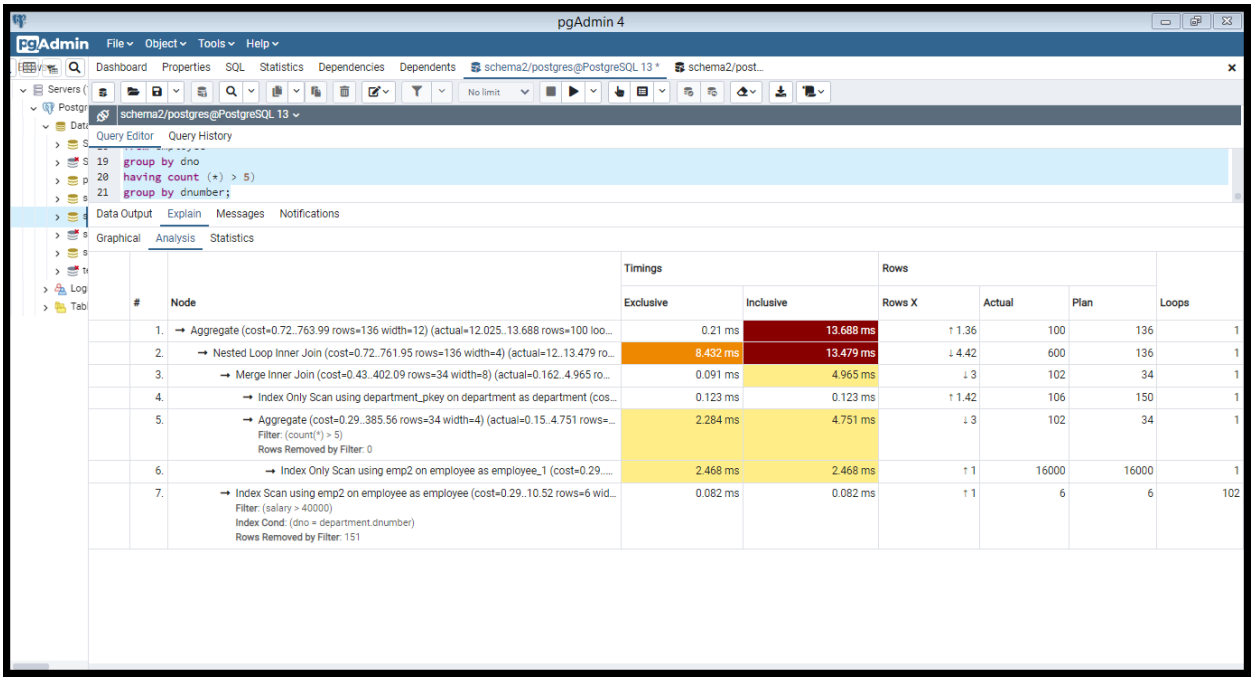
Here the BRIN increased performance as it is used in small range queries

Query6 using mixed indecies : B+ tree Employee(salary) , Employee(dno) and hash on Department(Dnumber)

Planning and execution time:



Execution plan and costs:



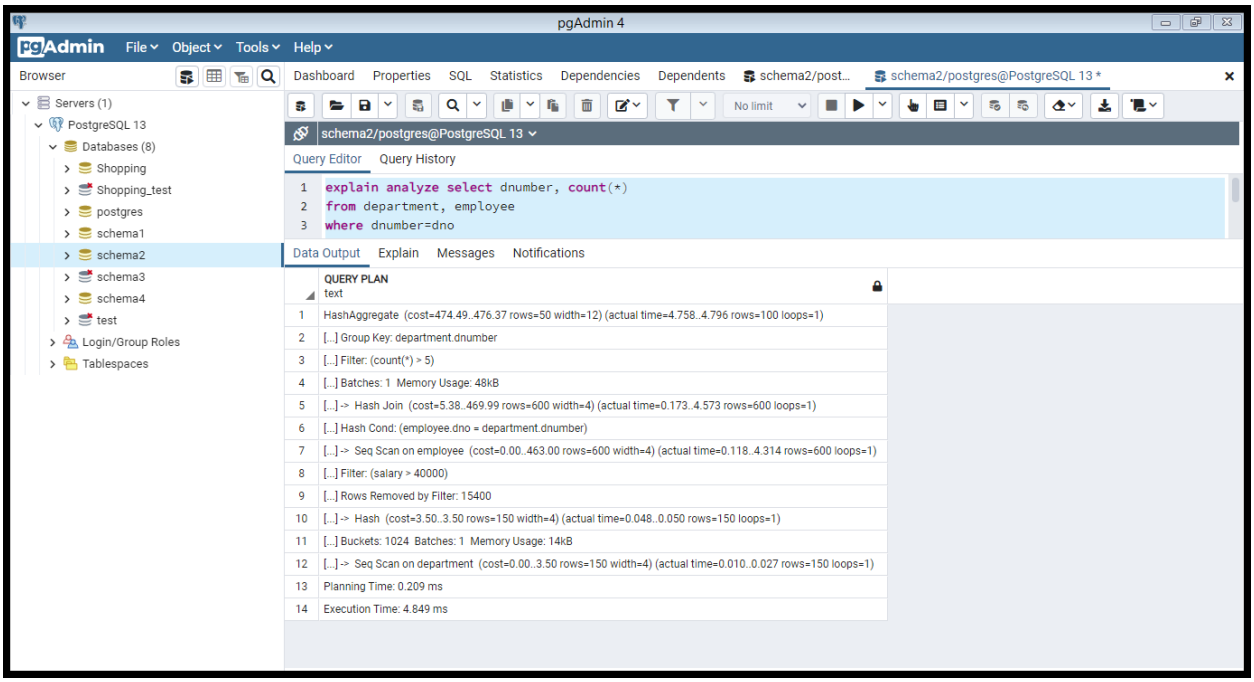
Query6 after optimization :

```
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
group by dnumber
having count(*)>5
```

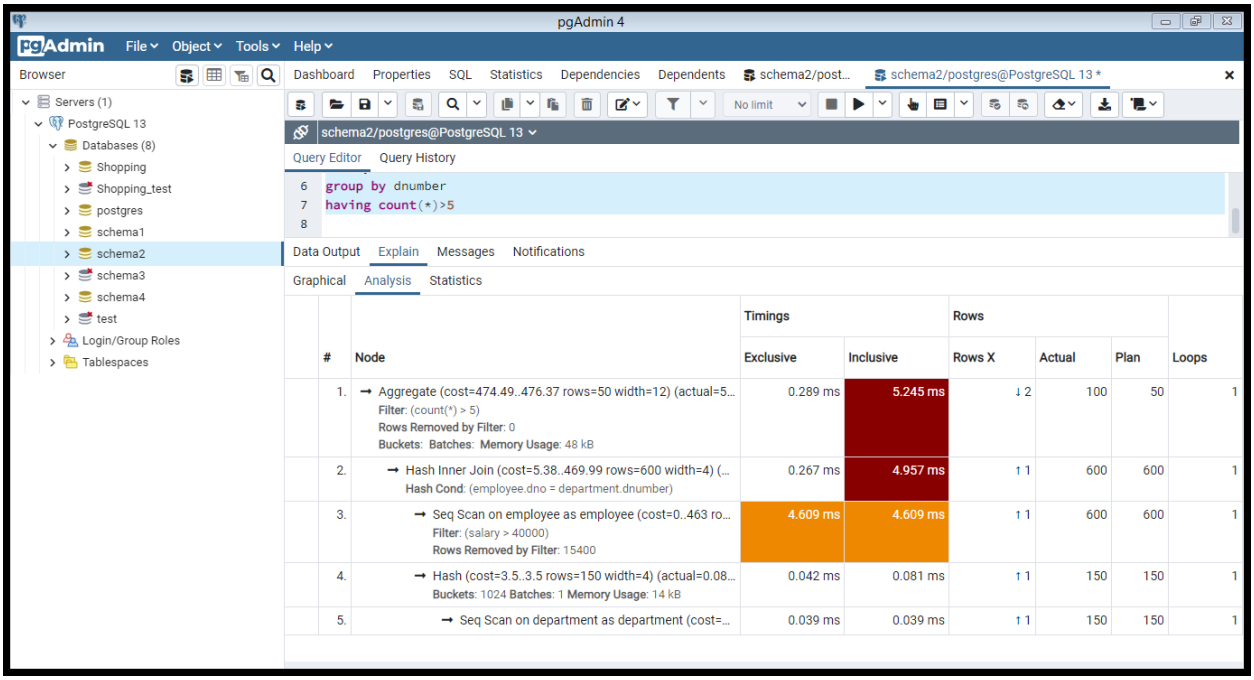
we removed the in statement as it did not affect the qeury at all it only make it worse

Query6 optimized with no index:

Planning and execution time:

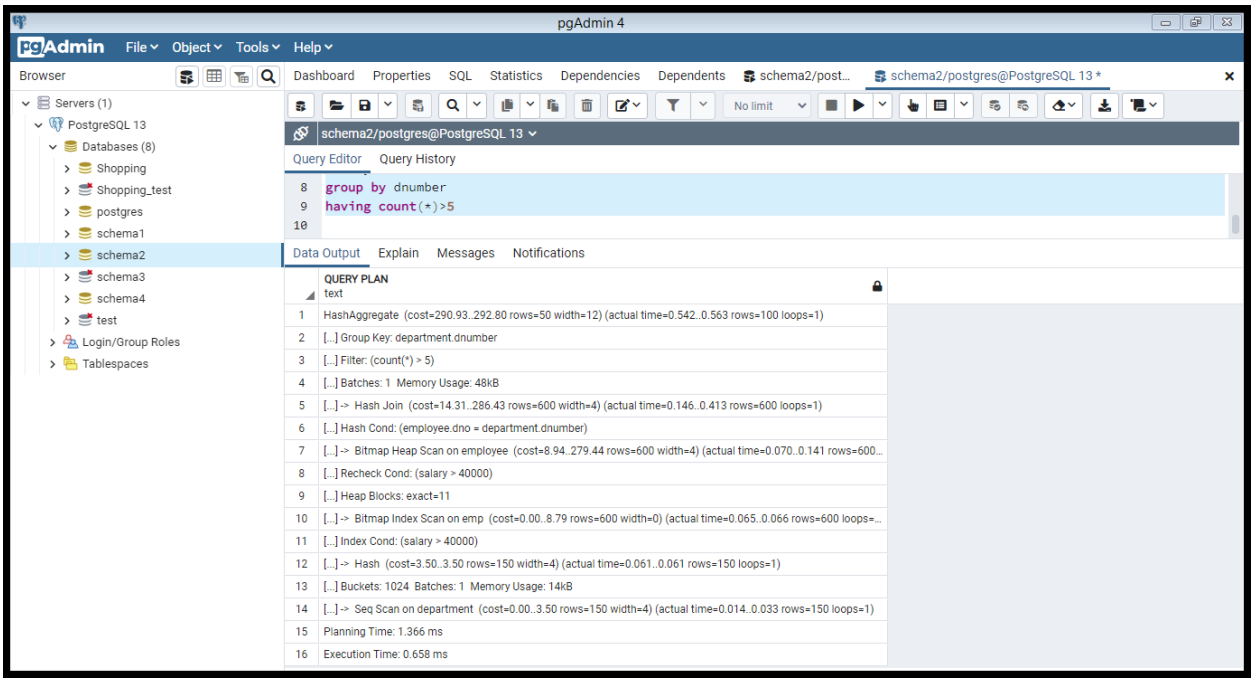


Execution plan and costs:

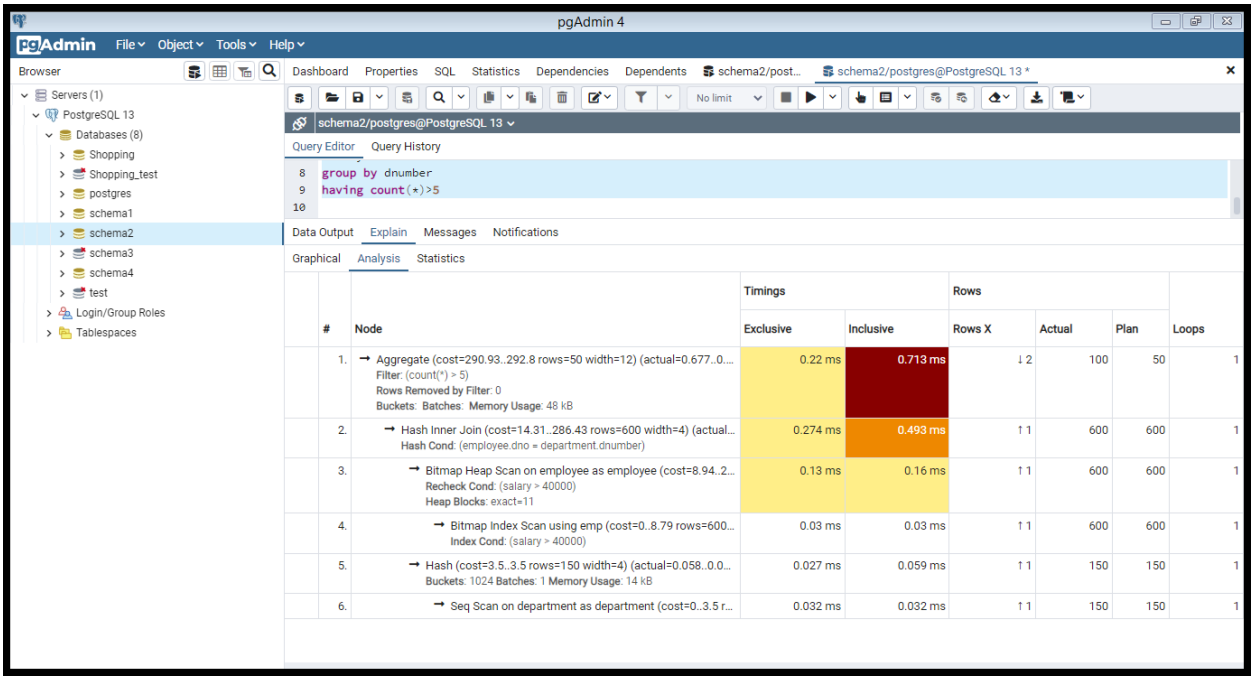


Query6 optimized using B+ tree on : Employee(salary), Department(Dnumber)

Planning and execution time:



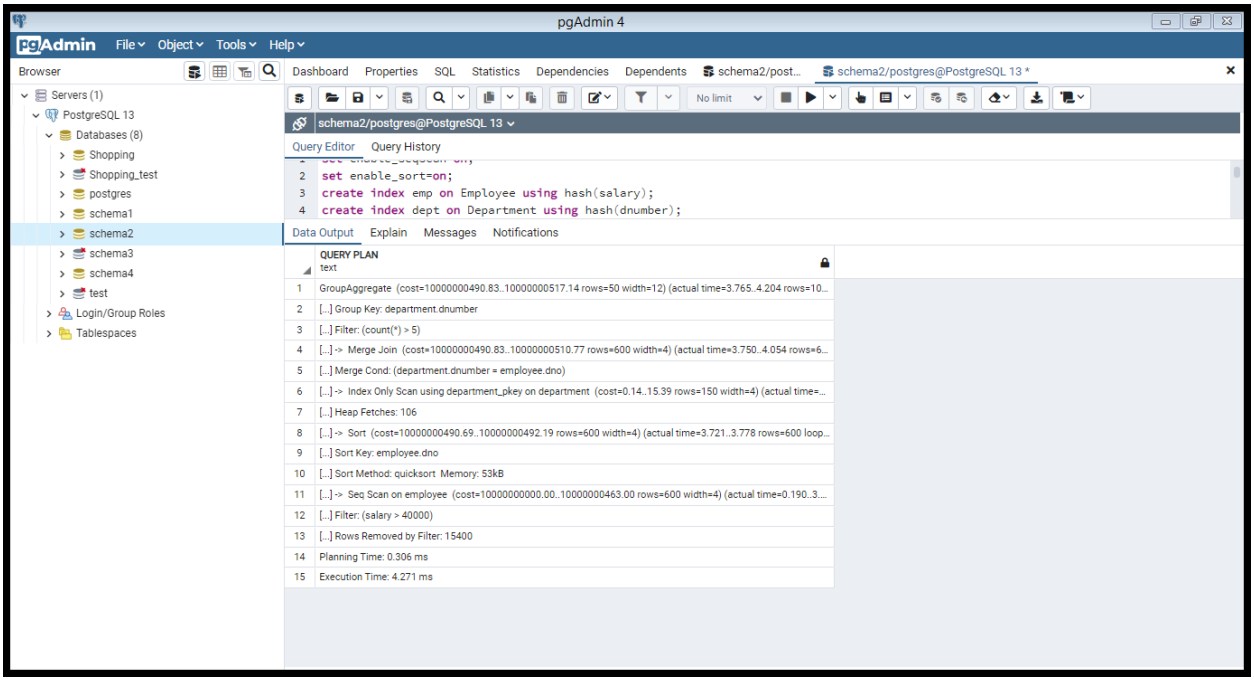
Execution plan and costs:



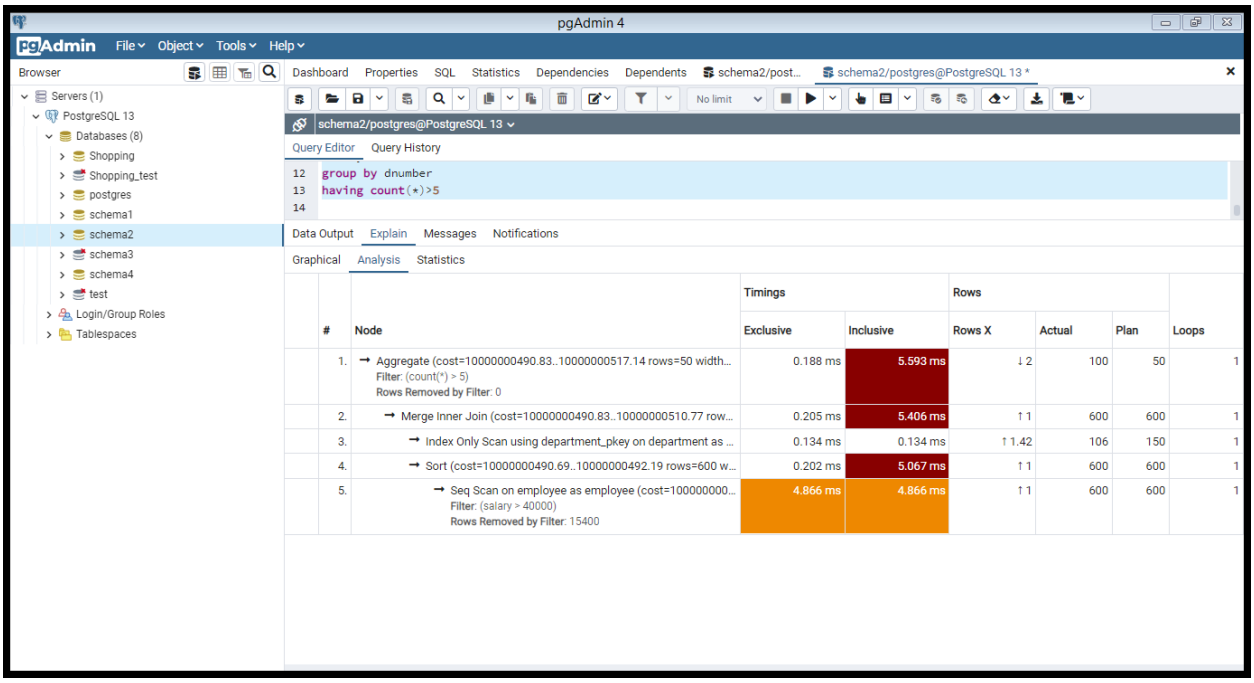
Query6 optimized using hash indecies on : Employee(salary), Department(dnumber)

With seqscan disabled

Planning and execution time:



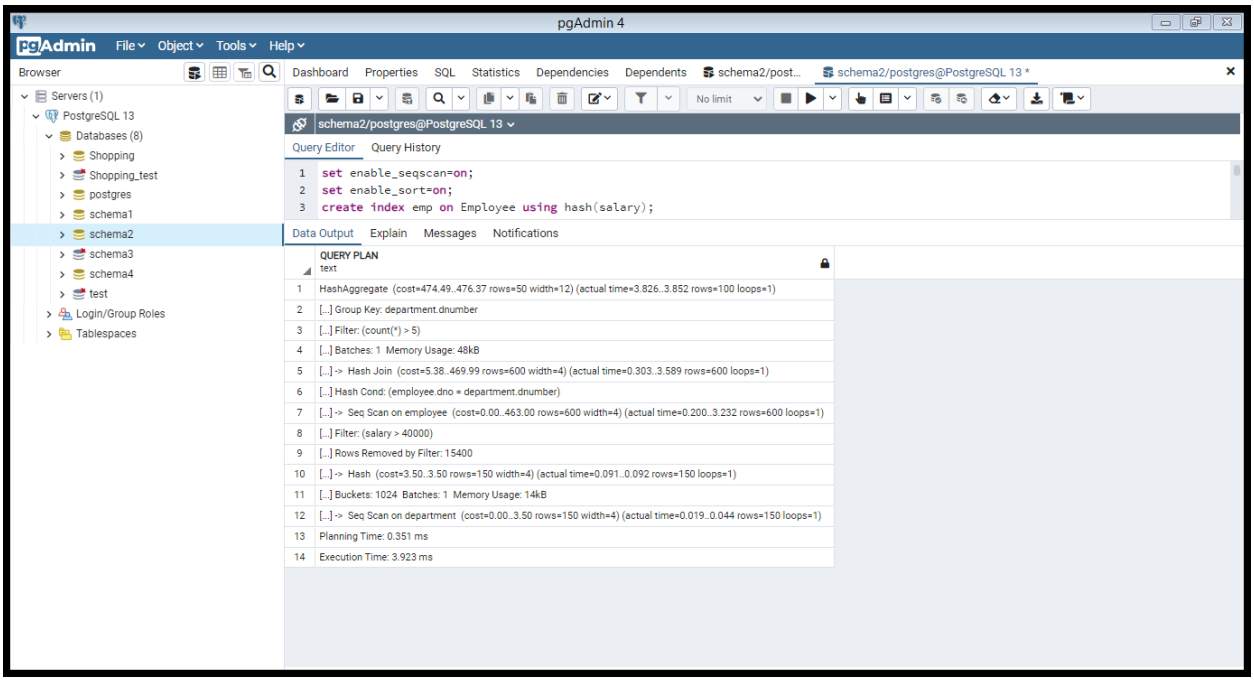
Execution plan and costs:



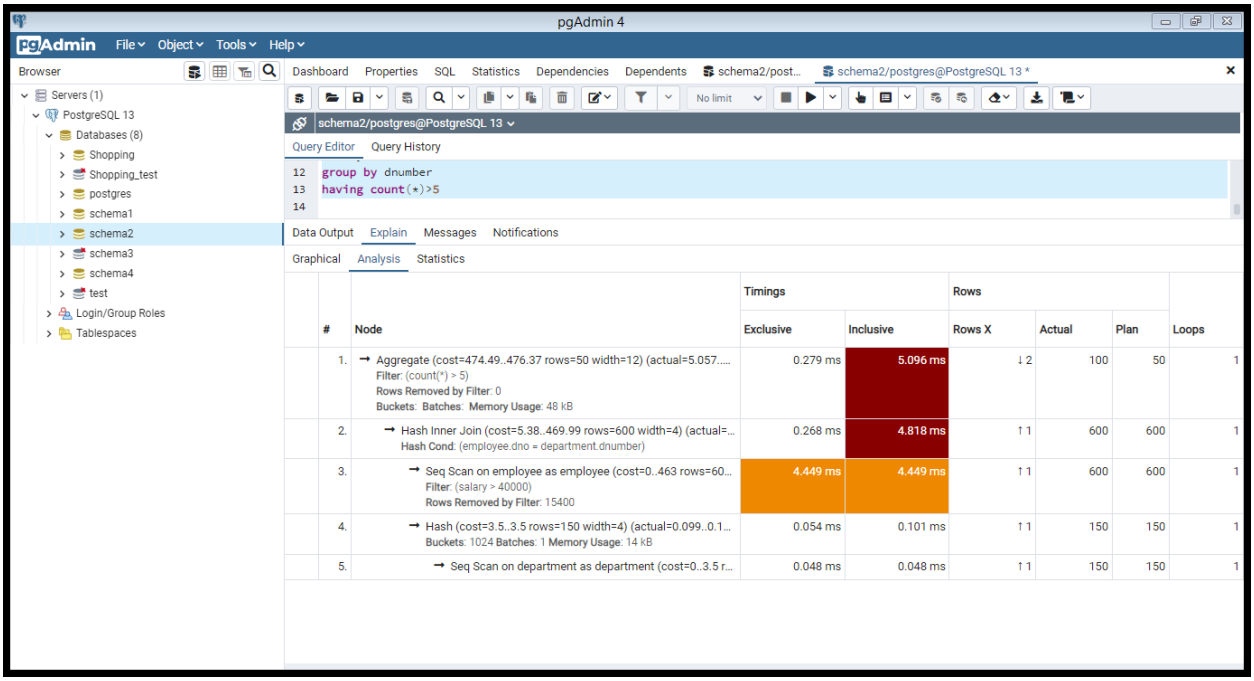
Here the hash did not help and it increased the cost as it is used in exact value queries only in O(1) time so it did not help in the group by and the range query

With seqscan enabled

Planning and execution time:

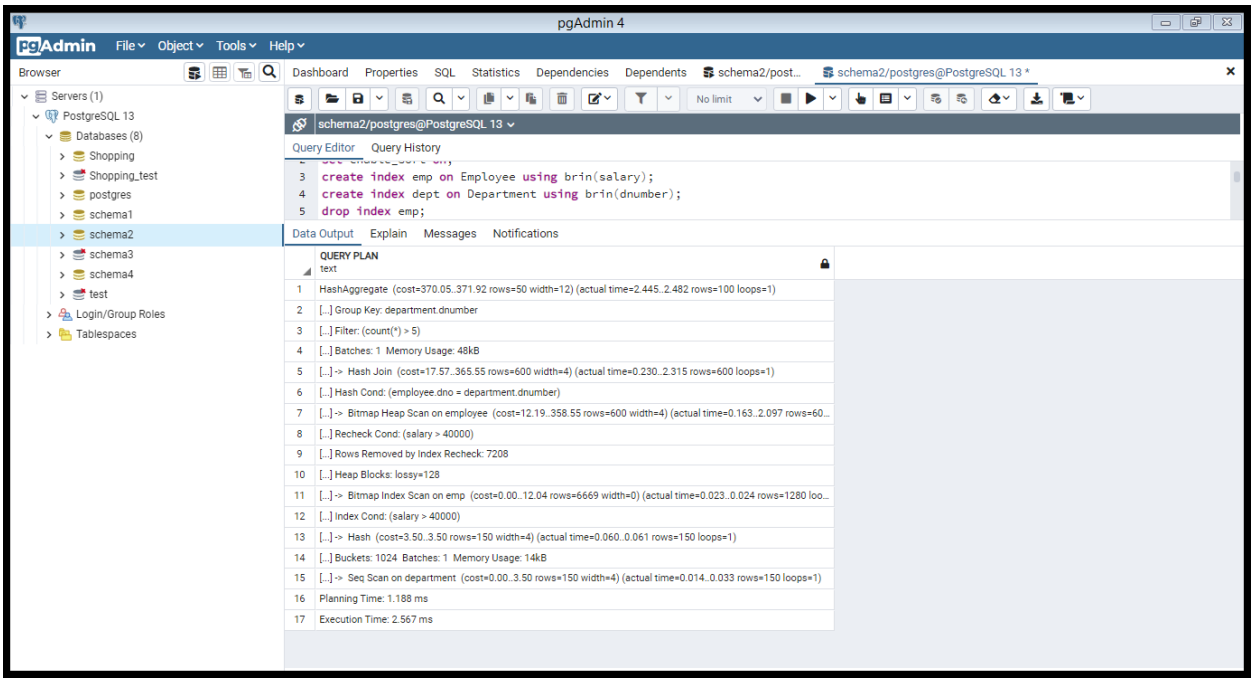


Execution plan and costs:

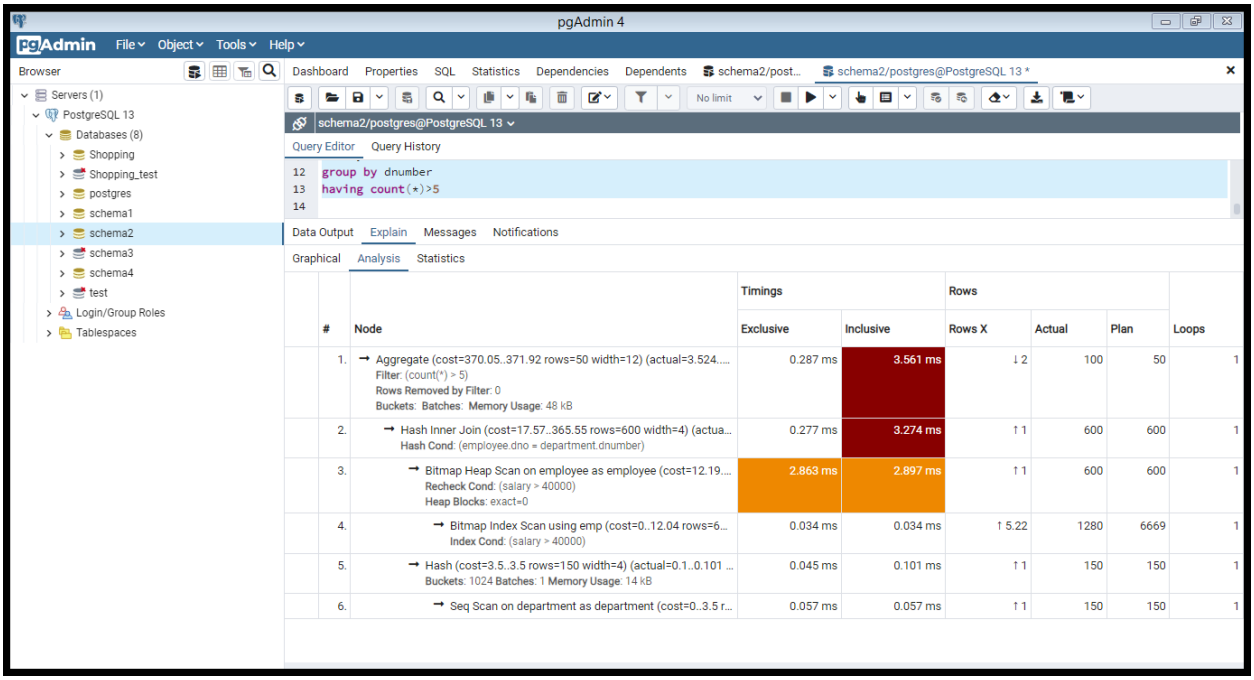


Query6 optimized using BRIN indecies on : Employee(salary), Department(dnumber)

Planning and execution time:

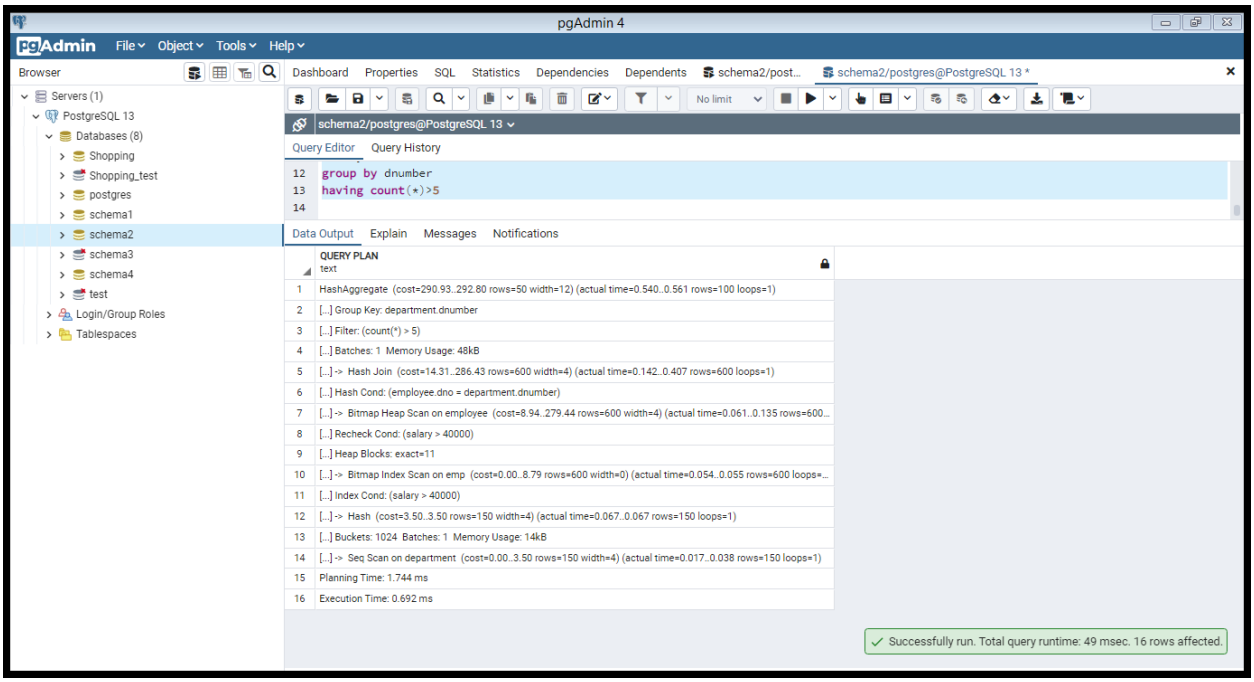


Execution plan and costs:

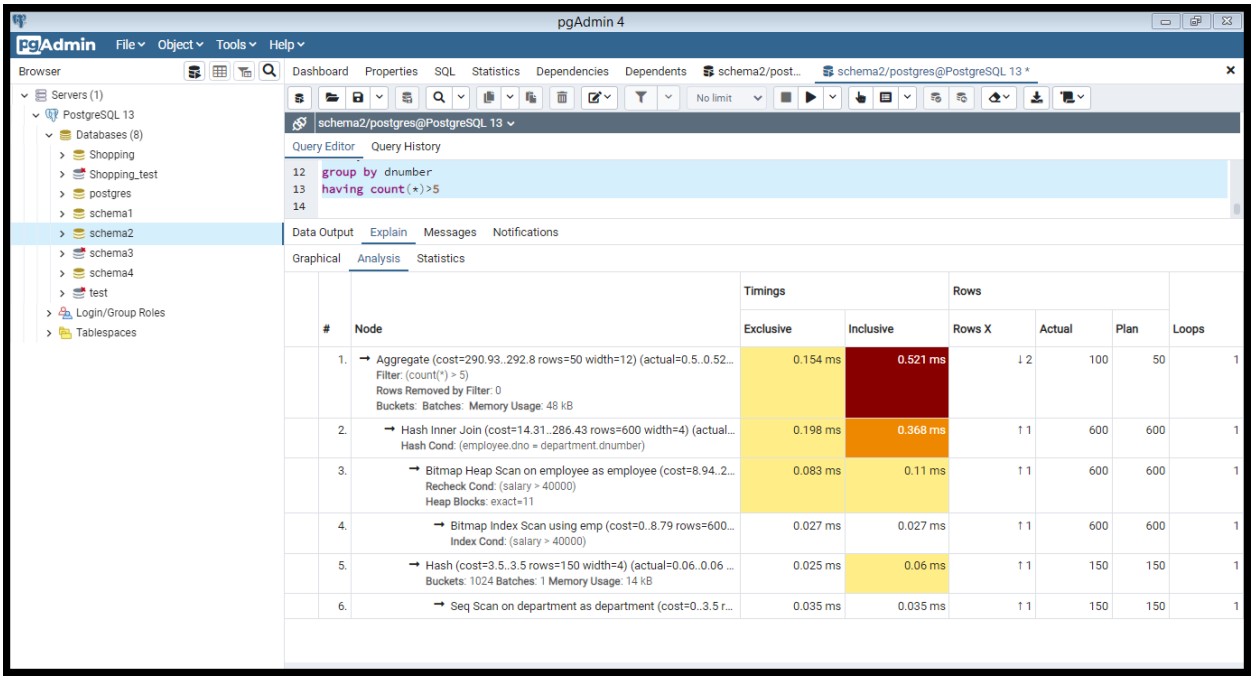


Query6 with mixed indecies on : B+ tree on Employee(salary) , Department(dnumber)

Planning and execution time:

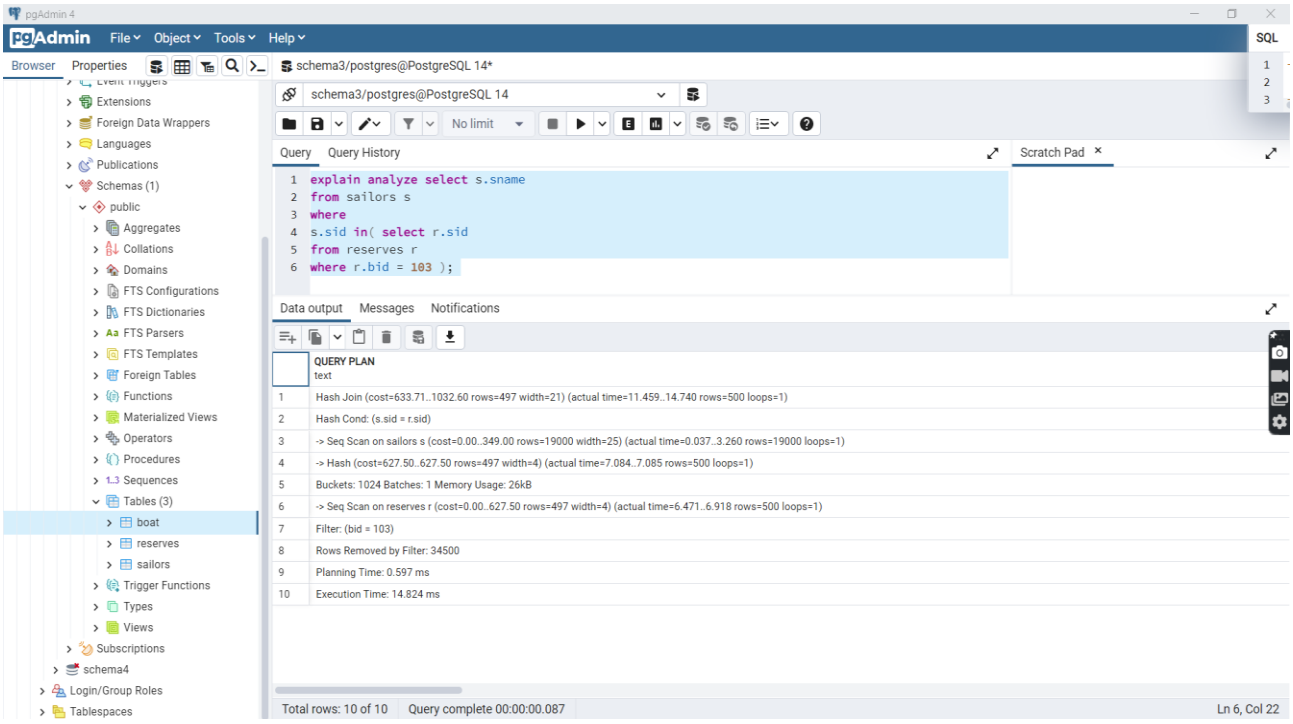


Execution plan and costs:

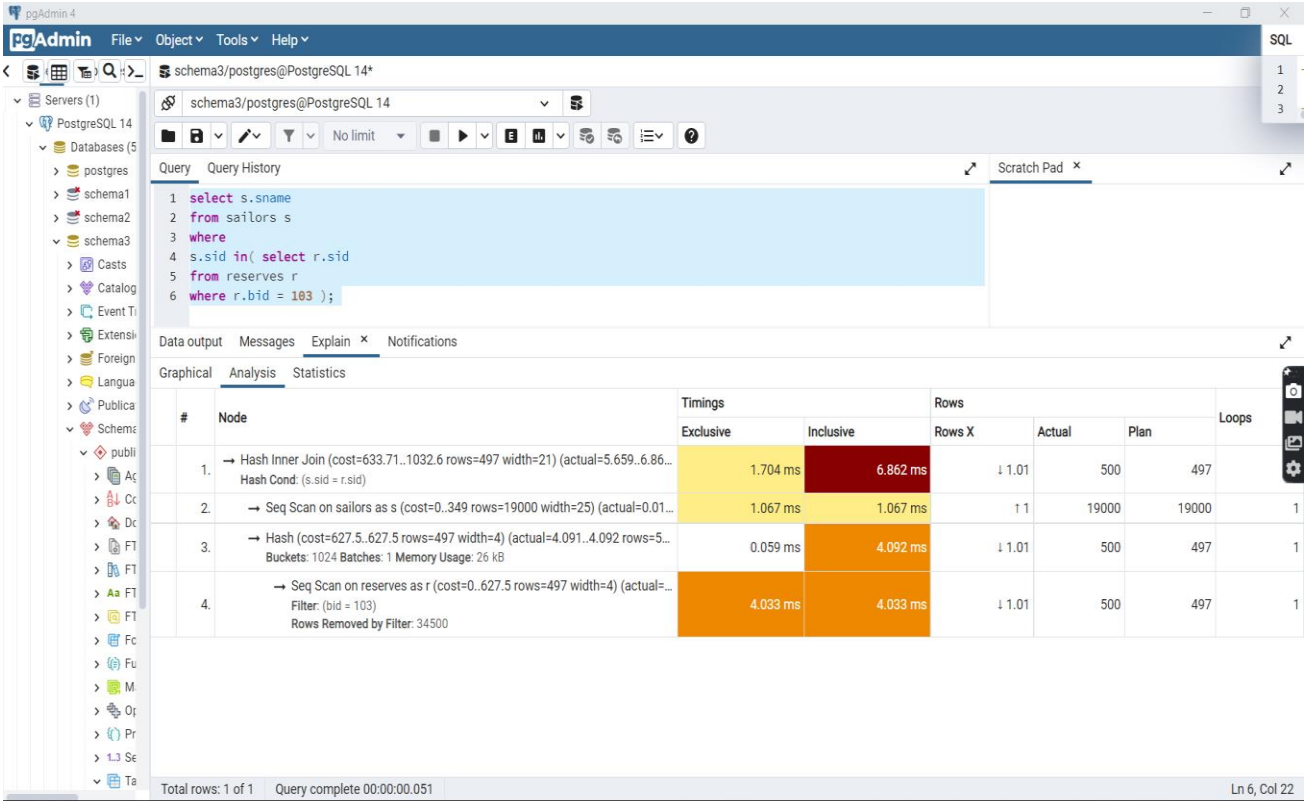


Query 7 with no indices(Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary key indices are indisvalid = true])

Planning and execution time

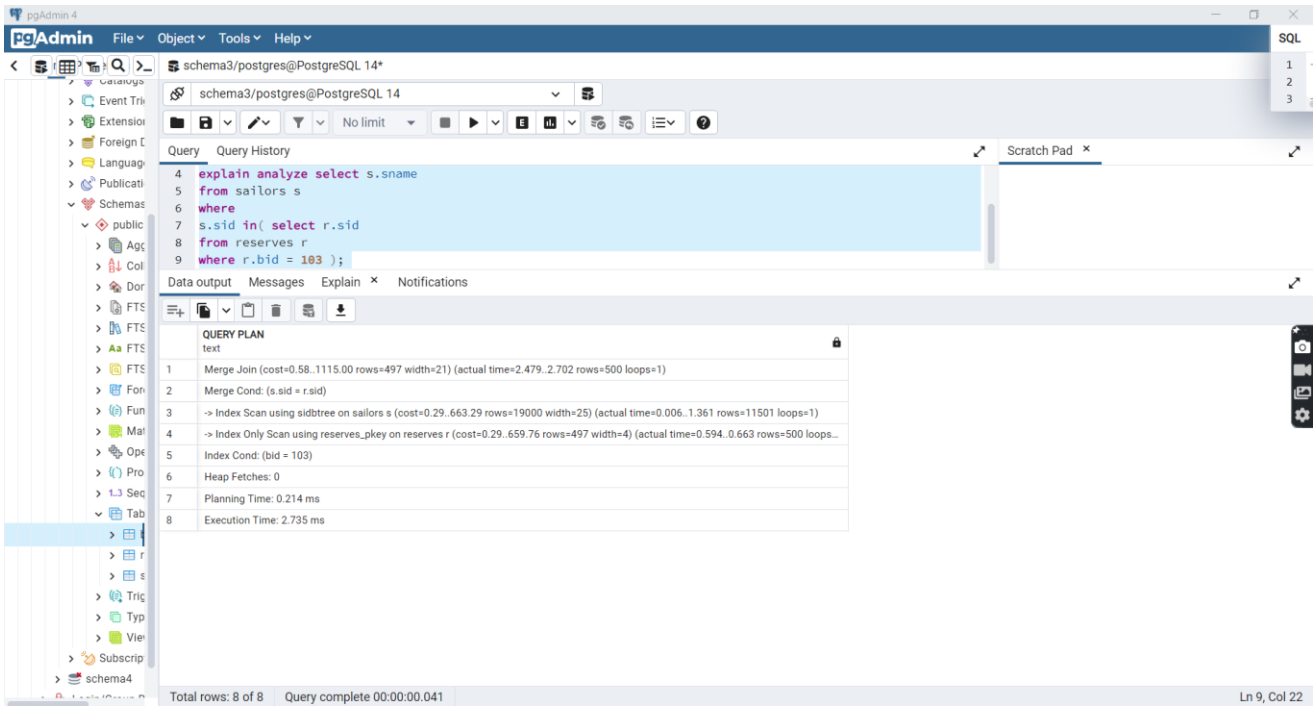


Execution Plan and Costs

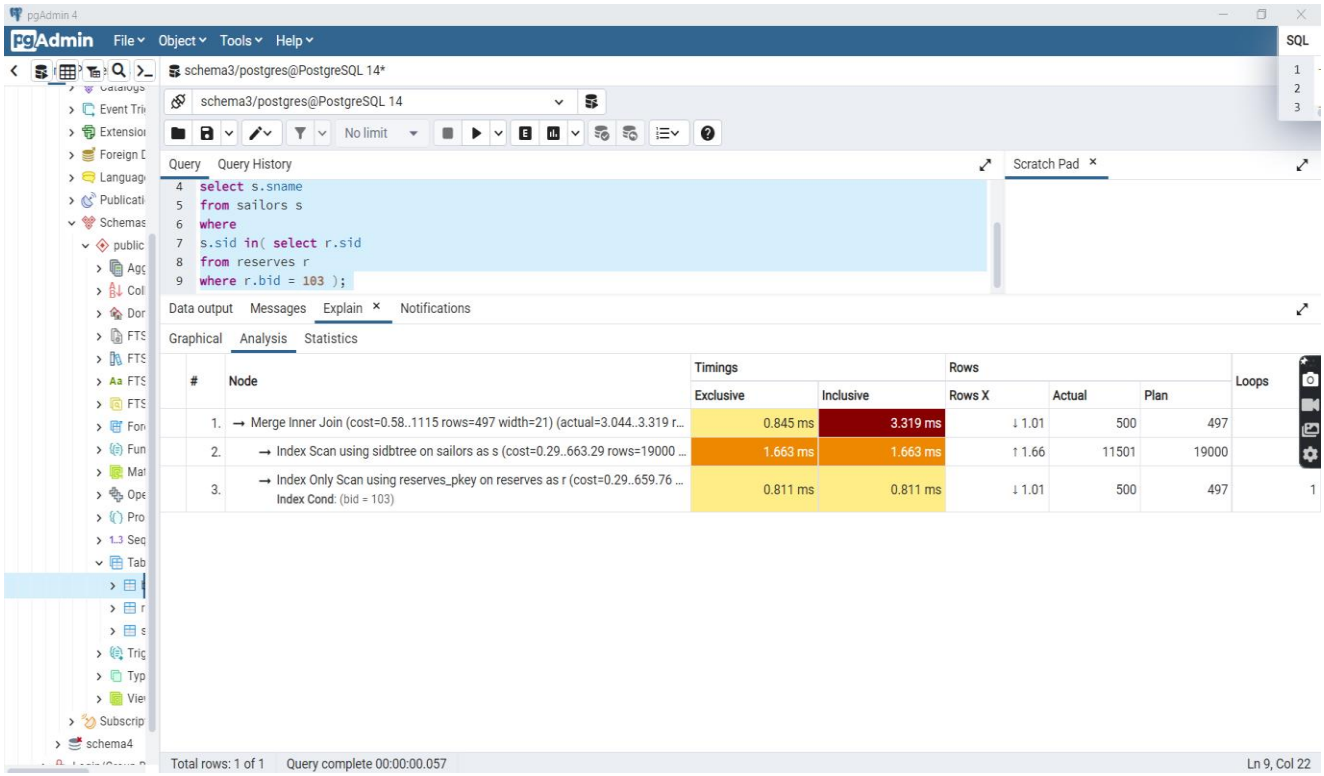


Query 7 with B+ Tree on: sailors(sid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and Execution time

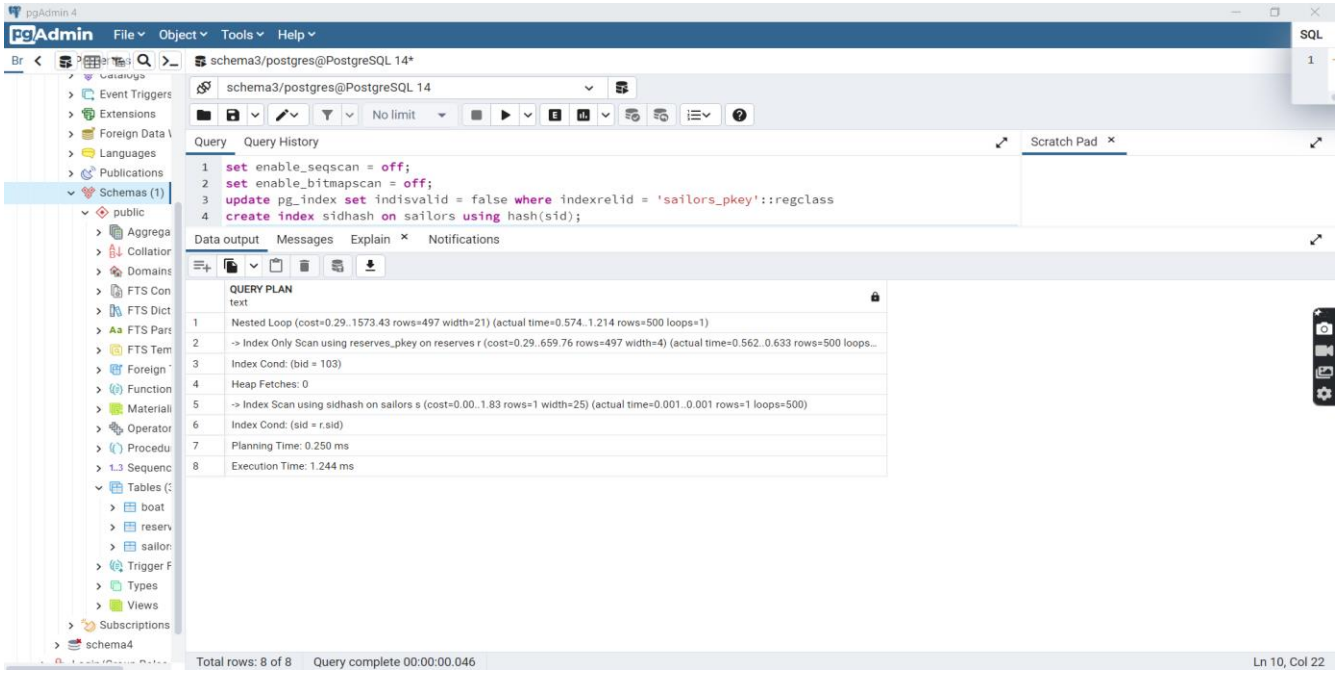


Execution Plan and Costs

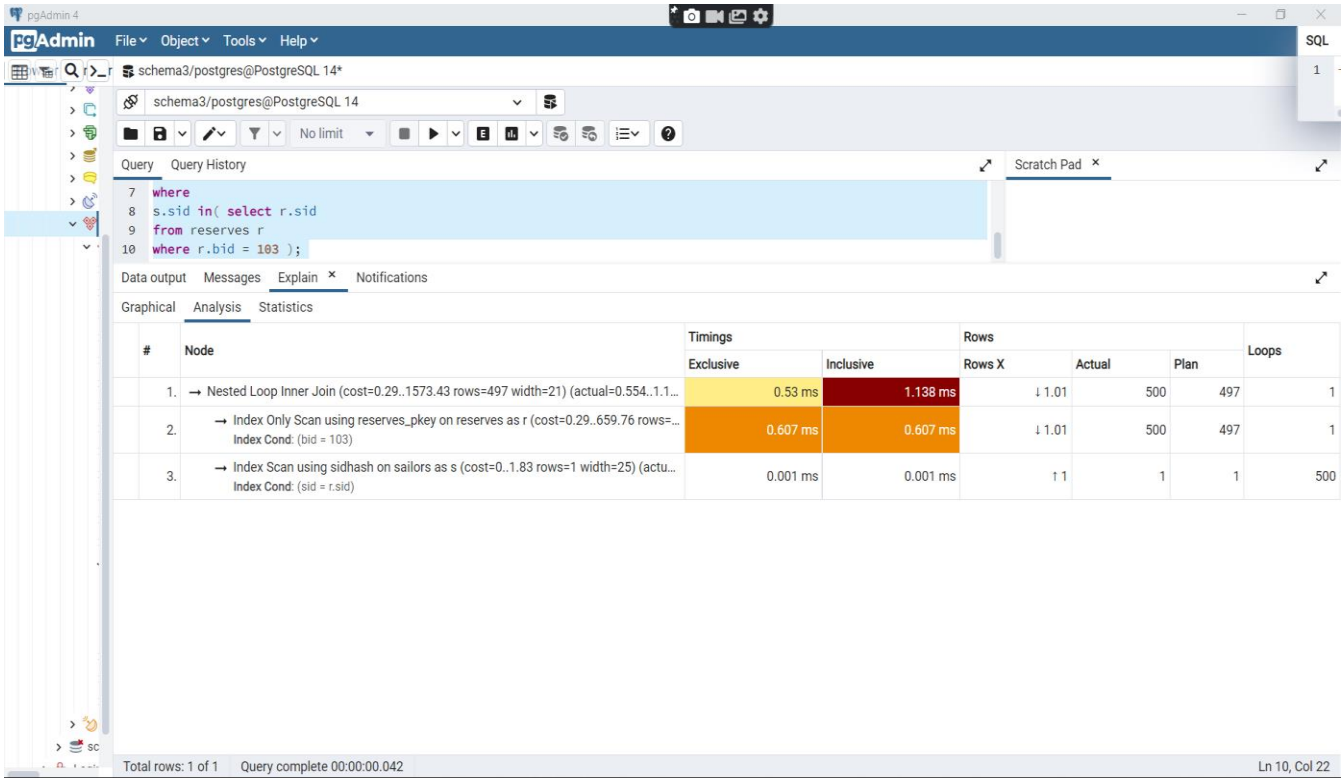


B+ tree index optimised here as it was used to search(in O(logn)) in table sailors for every r.sid generated in the subquery

Query 7 with Hash index on: sailors(sid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])



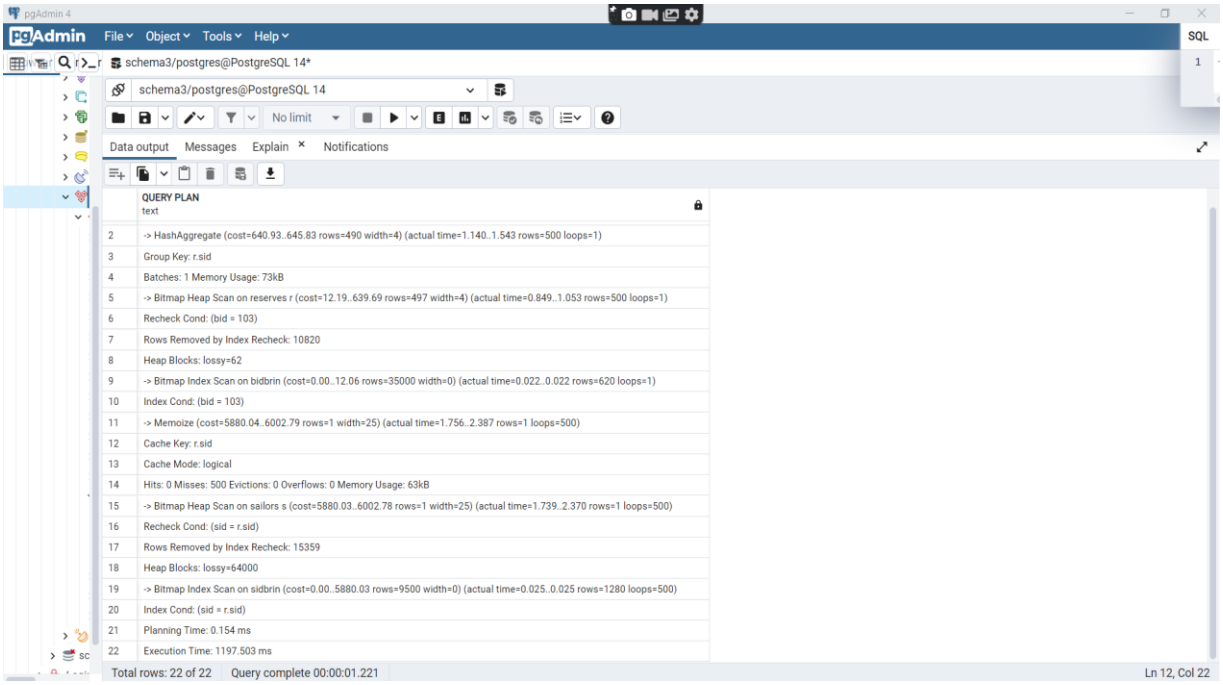
Execution Plan and Costs



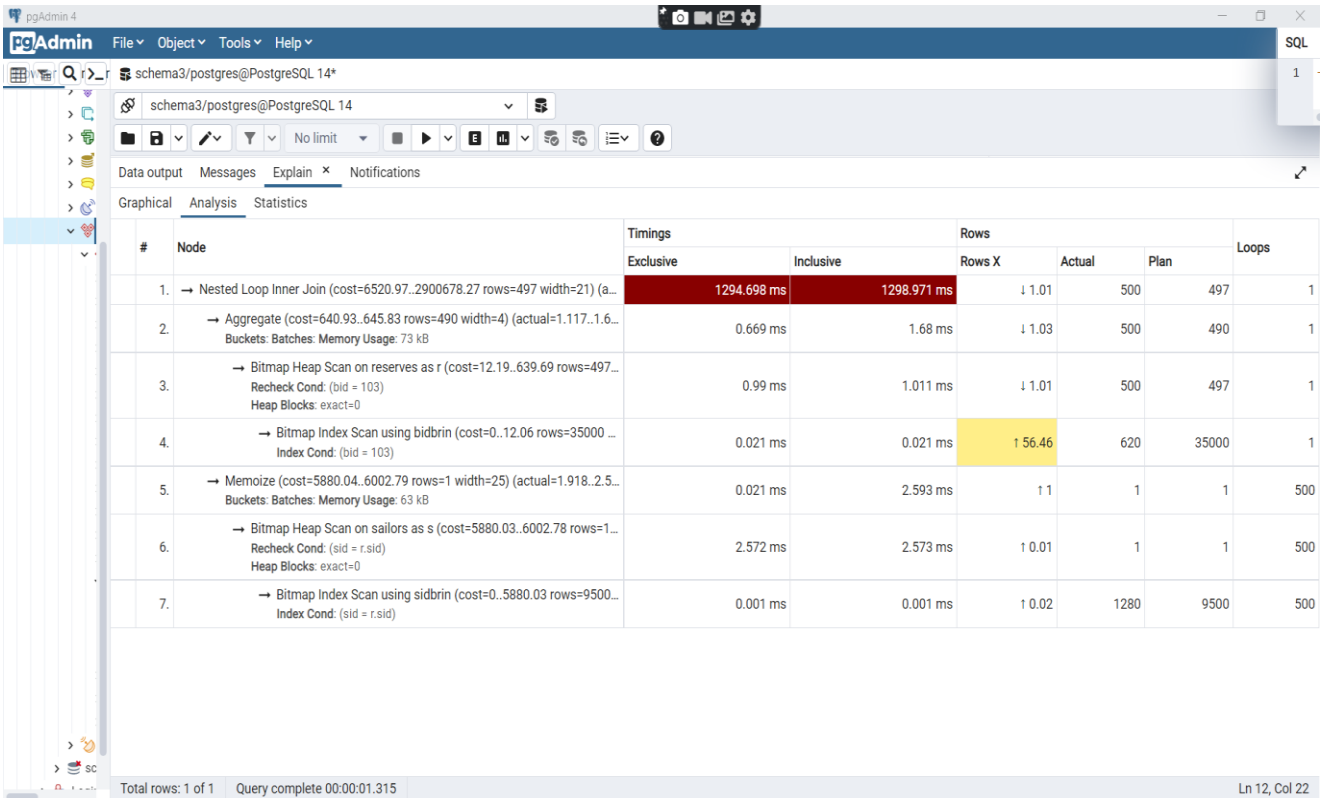
Hash index optimised here as it was used to search(in O(1)) in table sailors for every r.sid generated in the subquery

Query 7 with BRIN on: sailors(sid) and Reserves(bid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and Execution time



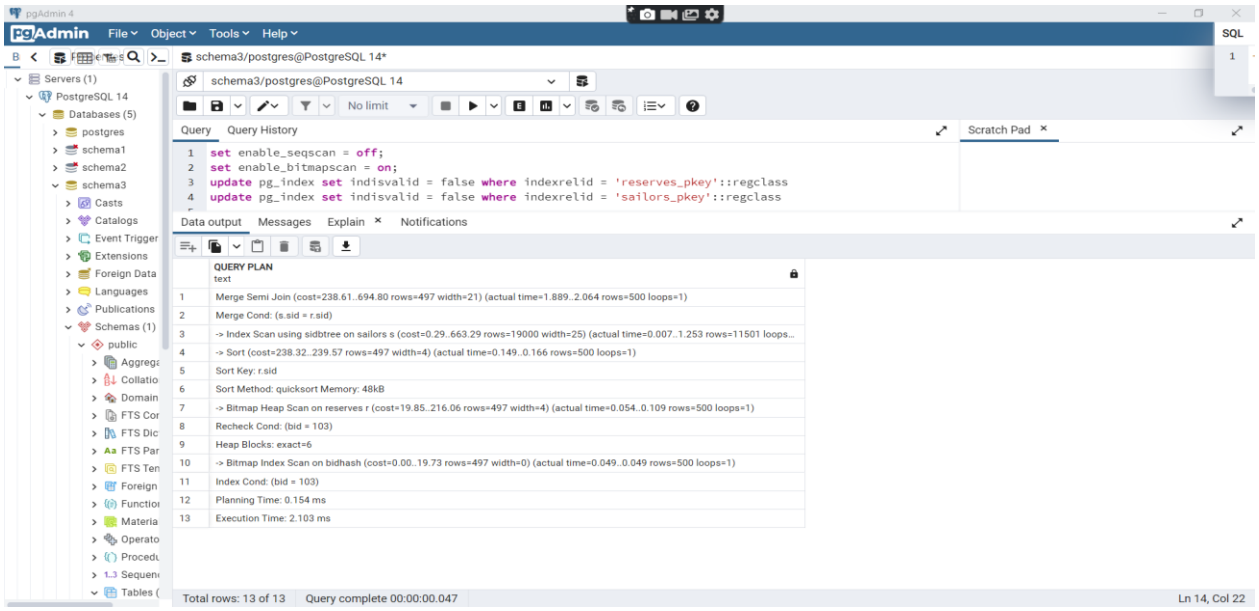
Execution Plan and Costs



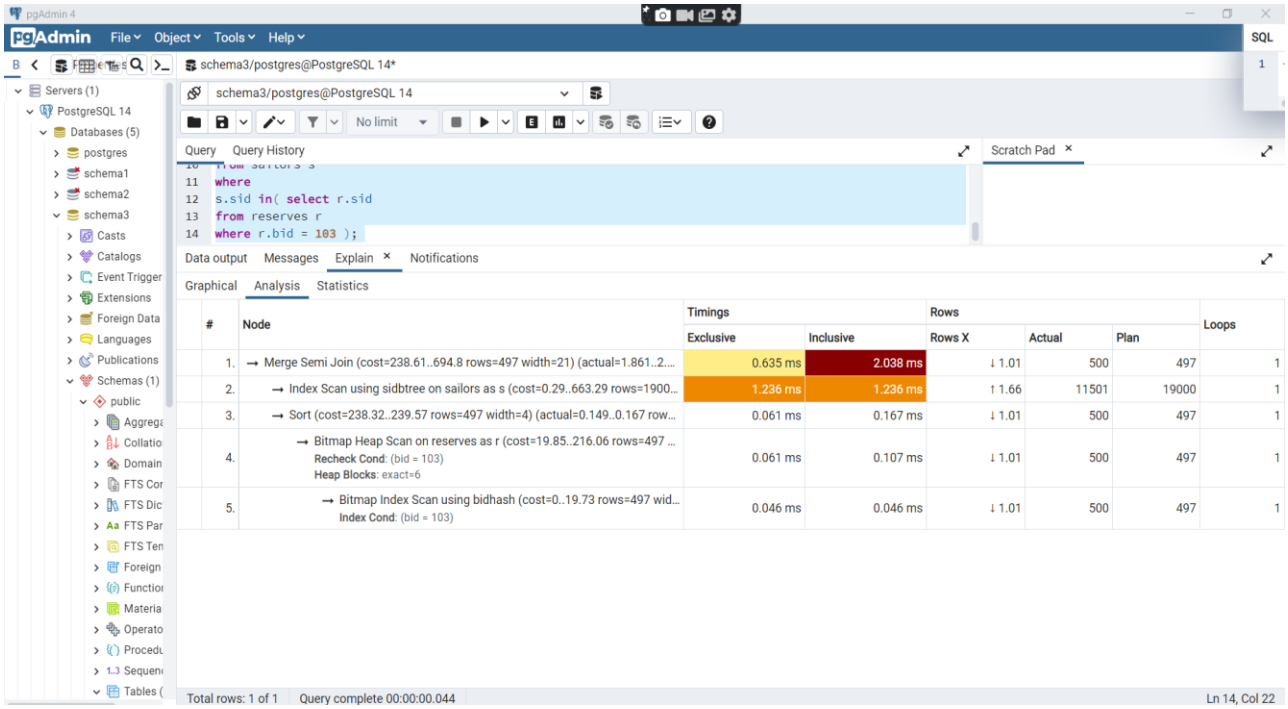
BRIN index here didn't optimize here as the search on r.bid was an exact value and the search on s.sid was also an exact value and BRIN is suitable for small range queries in a table with a very large range

Query 7 with Hash index on: reserves(bid) and B+ tree index on sailors(sid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and execution time



Execution Plan and Costs



Since the search on r.bid = 103 is an exact value and search on s.sid is an exact value having a hash index(search time = O(1)) on reserves.bid and B+tree index(search time = O(logn)) on sailors.sid improved query performance

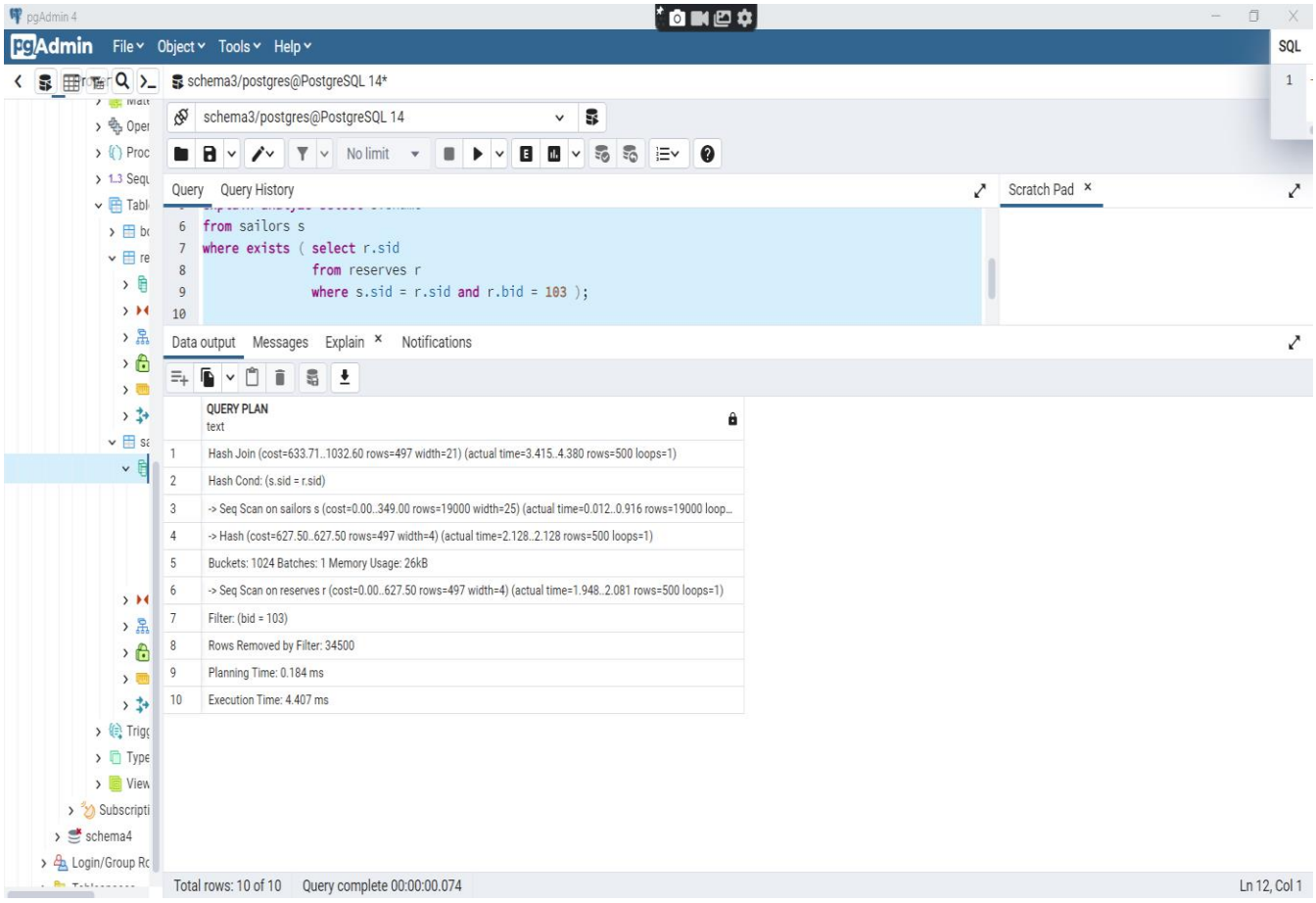
Optimised Query 7 SQL:

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

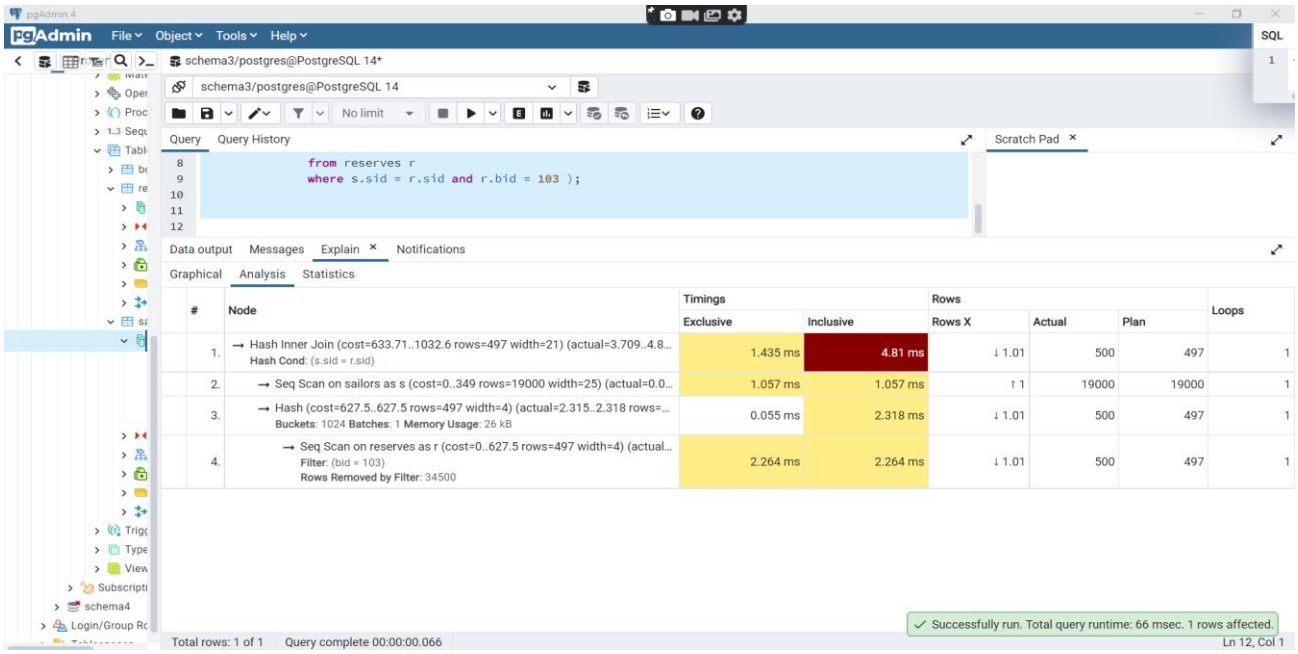
Reason: This query is optimised as it uses exists instead of in which reduces search time as SQL engine will terminate scanning procces once a match is found but in the original query with “in” all records from inner query will be fetched

Optimised Query 7 with no index Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary keyindisvalid = true])

Planning and Execution time

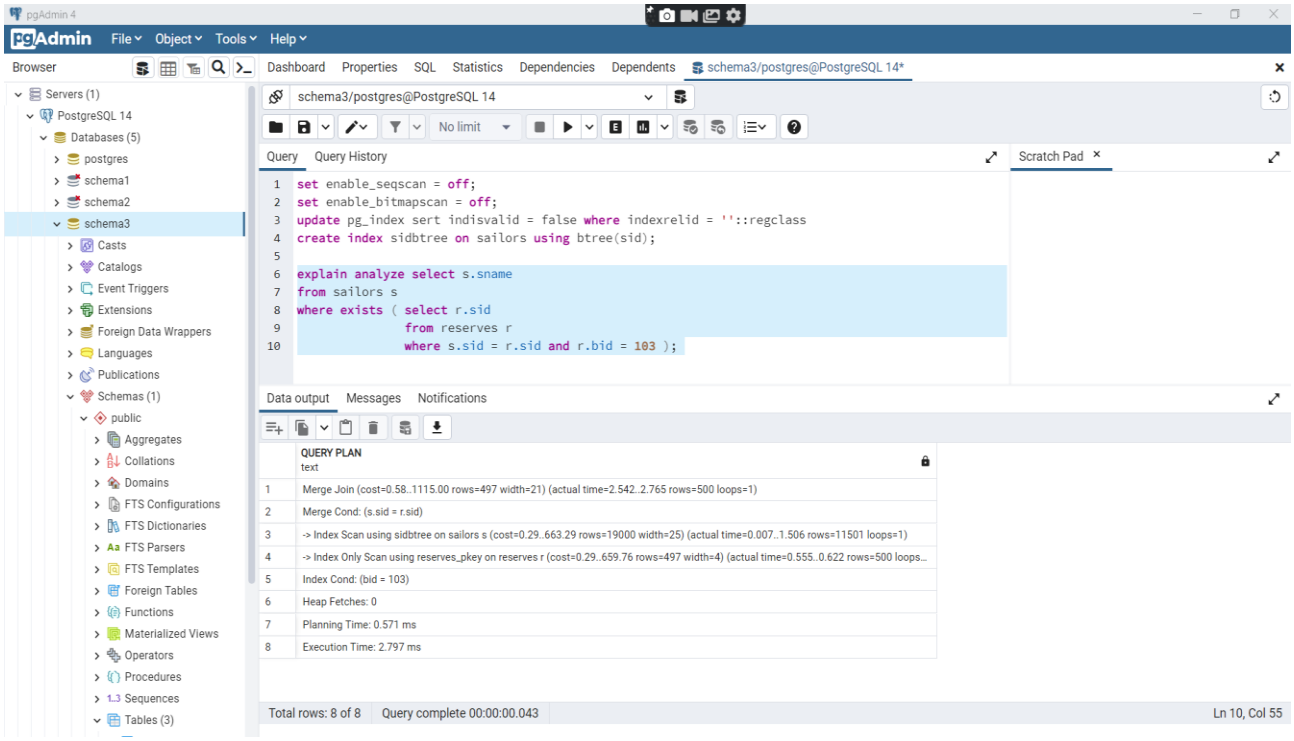

pgAdmin 4
pgAdmin
File
Object
Tools
Help
SQL
schema3/postgres@PostgreSQL 14*
schema3/postgres@PostgreSQL 14
No limit
Query
Query History
Scratch Pad
6 from sailors s
7 where exists (select r.sid
8 from reserves r
9 where s.sid = r.sid and r.bid = 103);
10
Data output
Messages
Explain
Notifications
QUERY PLAN
text
1 Hash Join (cost=633.71..1032.60 rows=497 width=21) (actual time=3.415..4.380 rows=500 loops=1)
2 Hash Cond: (s.sid = r.sid)
3 -> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.012..0.916 rows=19000 loop...
4 -> Hash (cost=627.50..627.50 rows=497 width=4) (actual time=2.128..2.128 rows=500 loops=1)
5 Buckets: 1024 Batches: 1 Memory Usage: 26kB
6 -> Seq Scan on reserves r (cost=0.00..627.50 rows=497 width=4) (actual time=1.948..2.081 rows=500 loops=1)
7 Filter: (bid = 103)
8 Rows Removed by Filter: 34500
9 Planning Time: 0.184 ms
10 Execution Time: 4.407 ms
schema4
Total rows: 10 of 10
Query complete 00:00:00.074
Ln 12, Col 1

Execution Plan and Costs

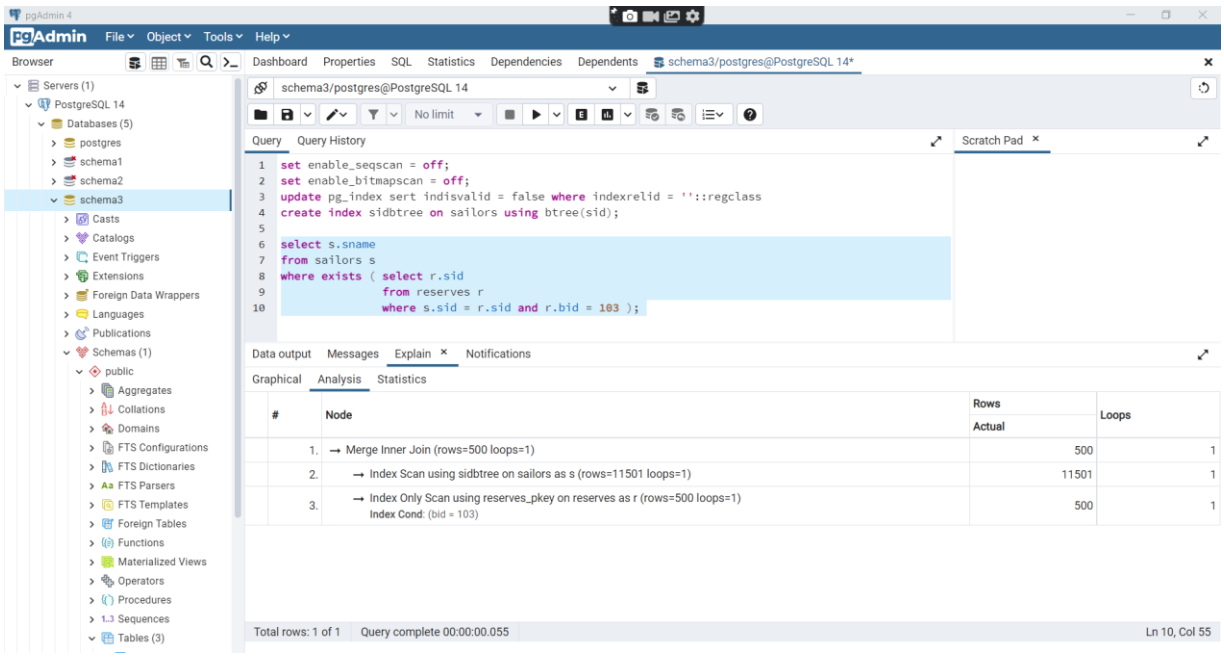


Optimised Query 7 with B+ tree index on: sailors(sid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and Execution time



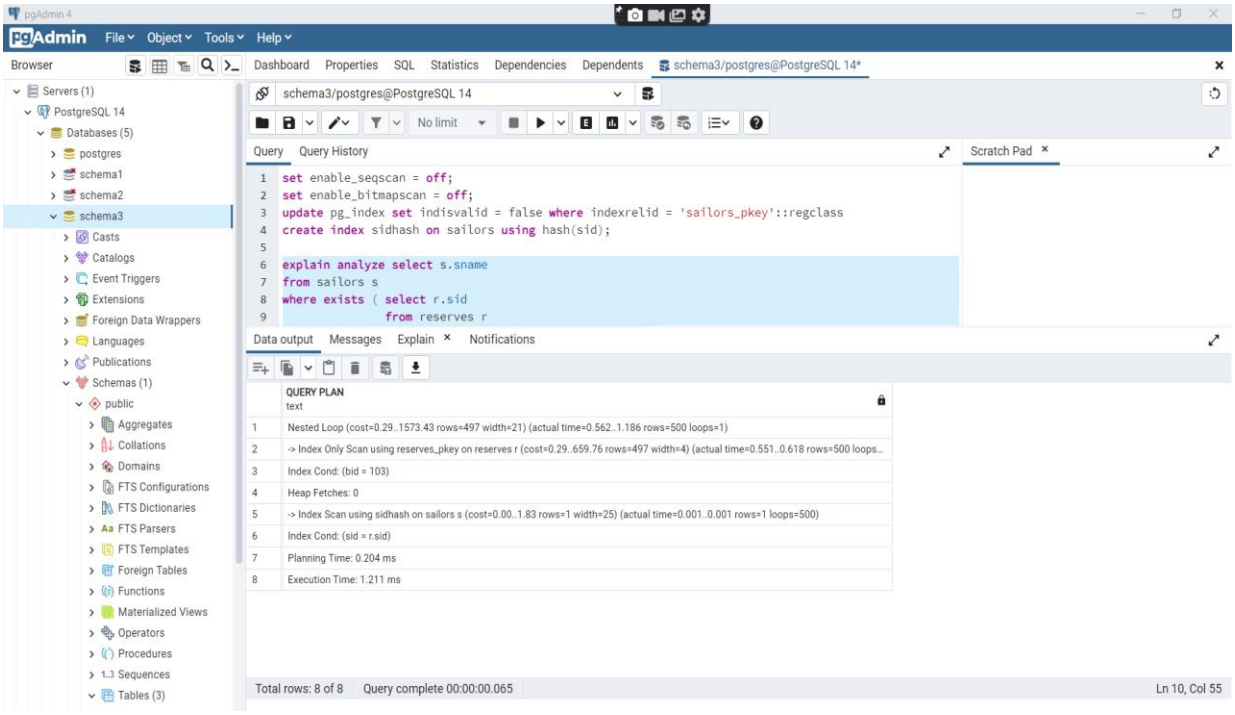
Execution Plan and Costs



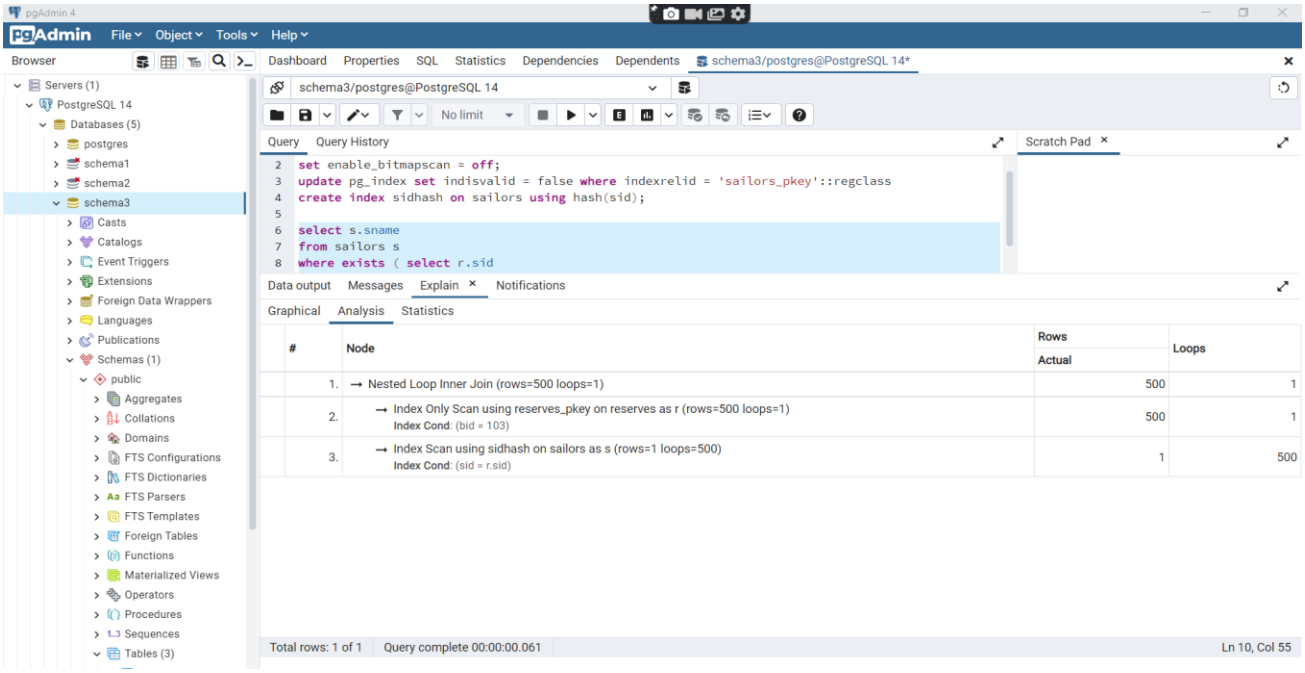
B+ tree index optimised here as it was used to search(in O(logn)) in table sailors for every r.sid = s.sid generated in the subquery

Optimised Query 7 with hash index on: sailors(sid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and Execution time



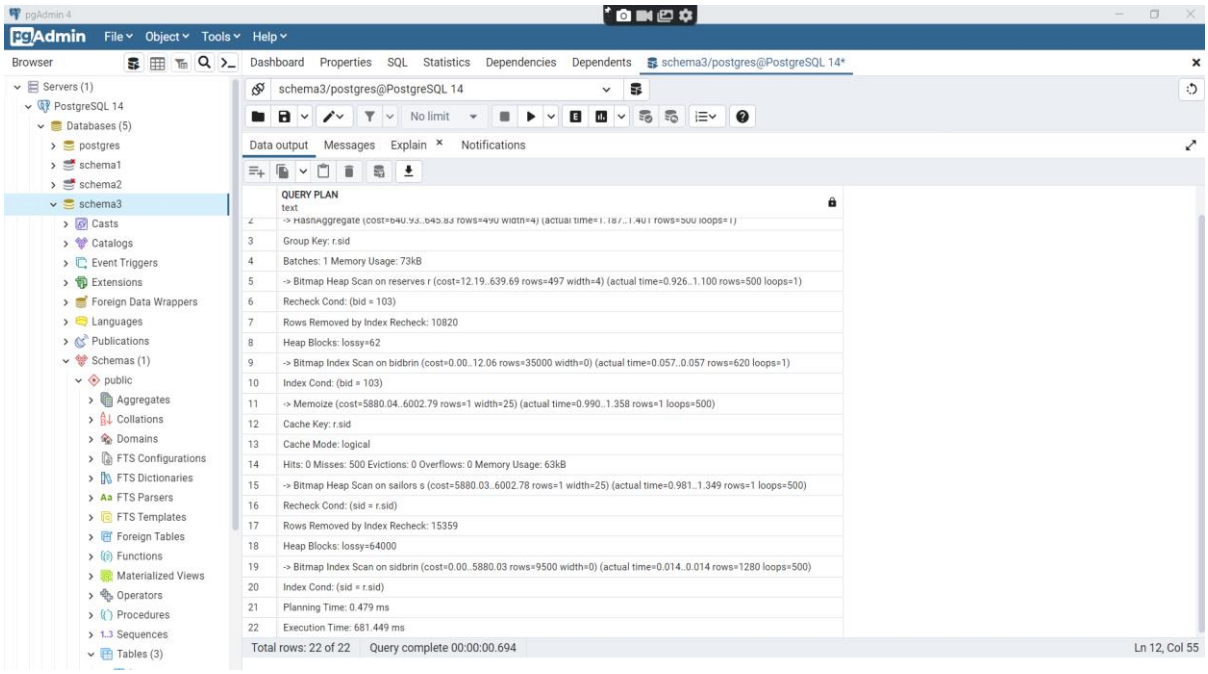
Execution Plan and Costs



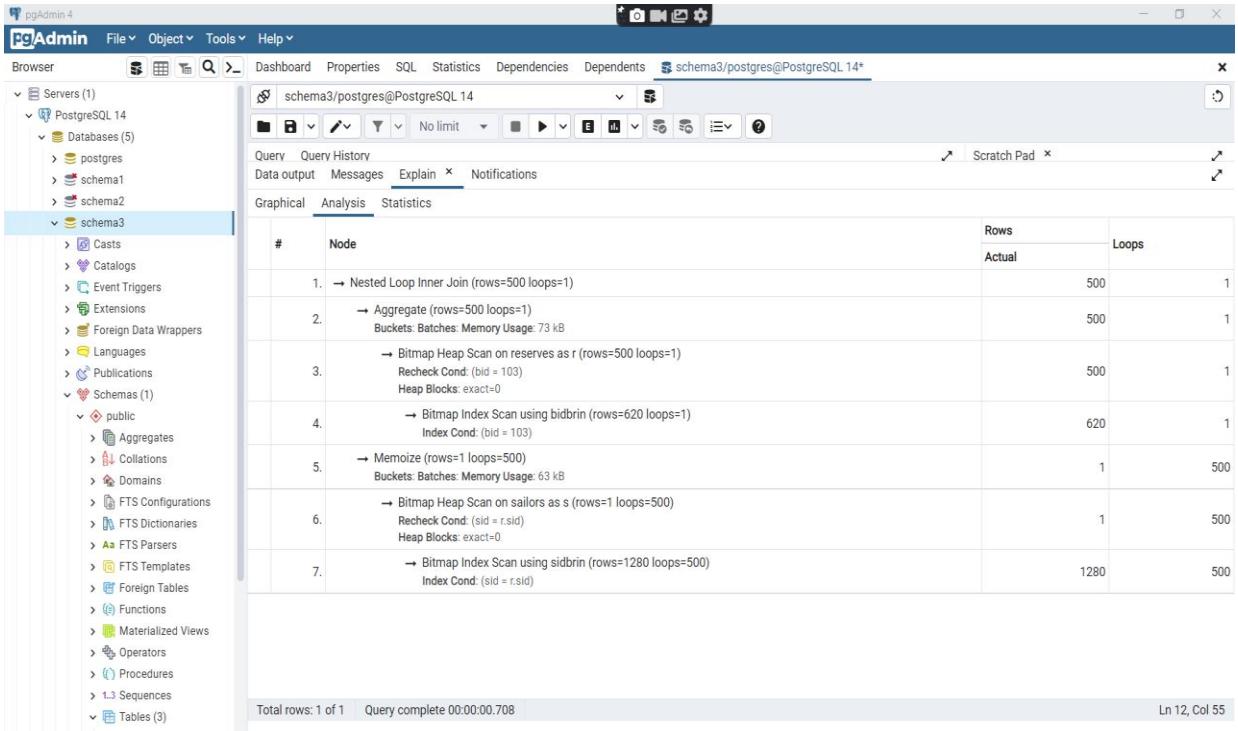
Hash index optimised here as it was used to search(in O(1)) in table sailors for every r.sid = s.sid generated in the subquery

Optimised Query 7 with BRIN on sailors(sid) and reserves(bid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey indisvalid = false else are true])

Planning and Execution time



Execution Plan and Costs

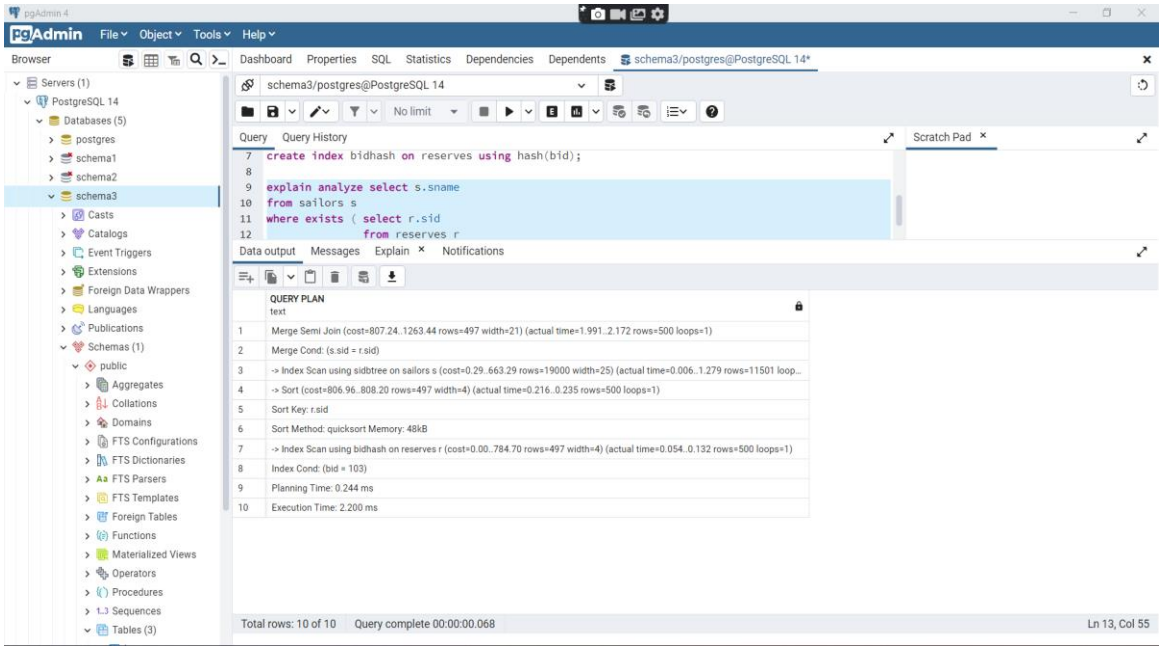


BRIN index here didn’t optimize here as the search on r.bid was an exact value and the

search on s.sid was also an exact value and BRIN is suitable for small range queries in a table with a very large range

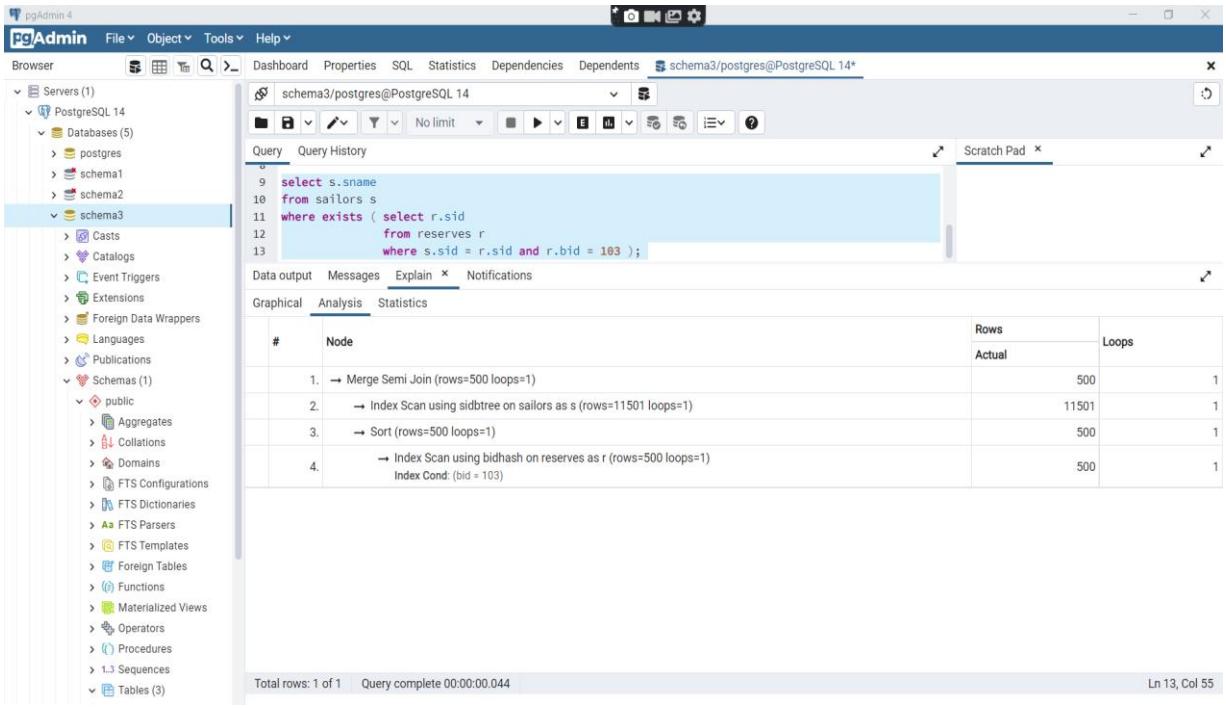
Optimised Query 7 with B+ tree index on sailors(sid) and hash index on reserves(bid) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey and reserves_pkey indisvalid = false else are true])

Planning and Execution time



Execution Plan and Costs

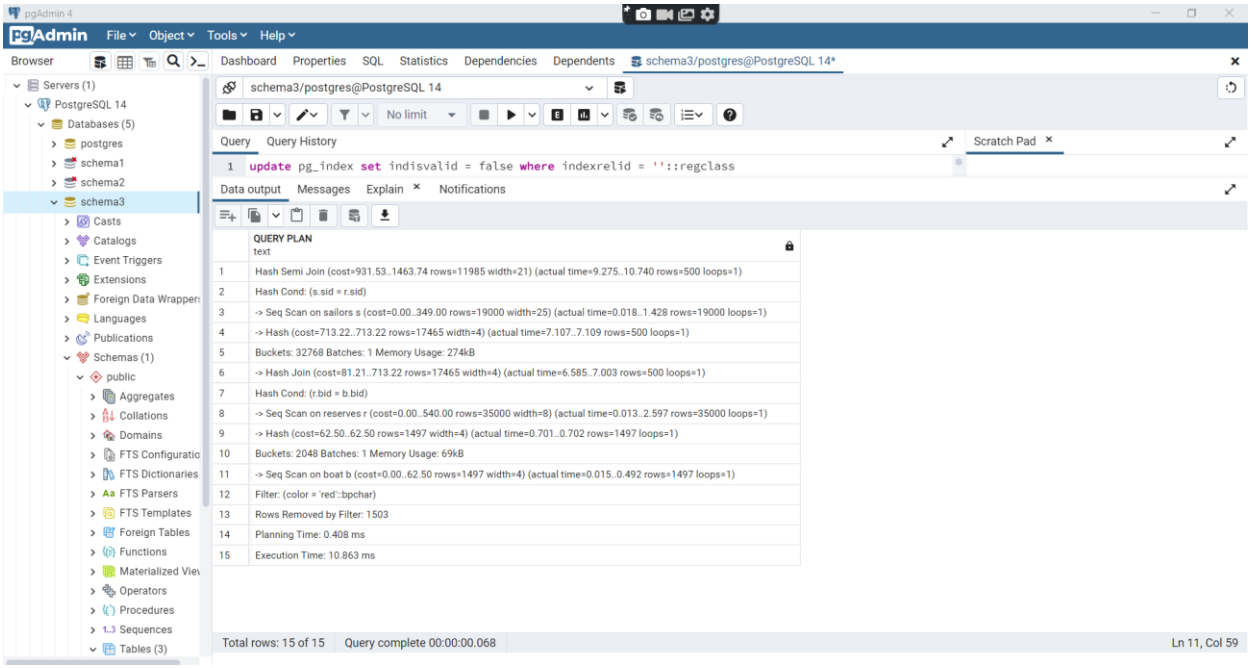
s



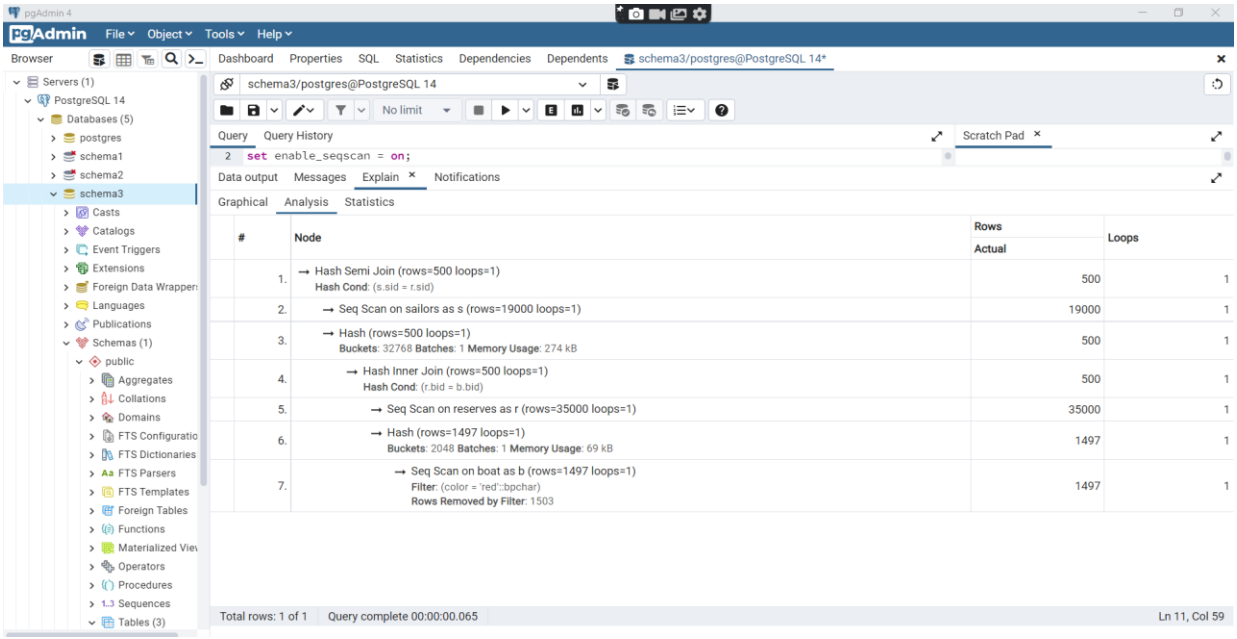
Since the search on r.bid = 103 is an exact value and search on s.sid = r.sid is an exact value having a hash index(search time = O(1)) on reserves.bid and B+tree index(search time = O(logn)) on sailors.sid improved query performance

Query 8 with no index Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary key indisvalid = true])

Planning and Execution time

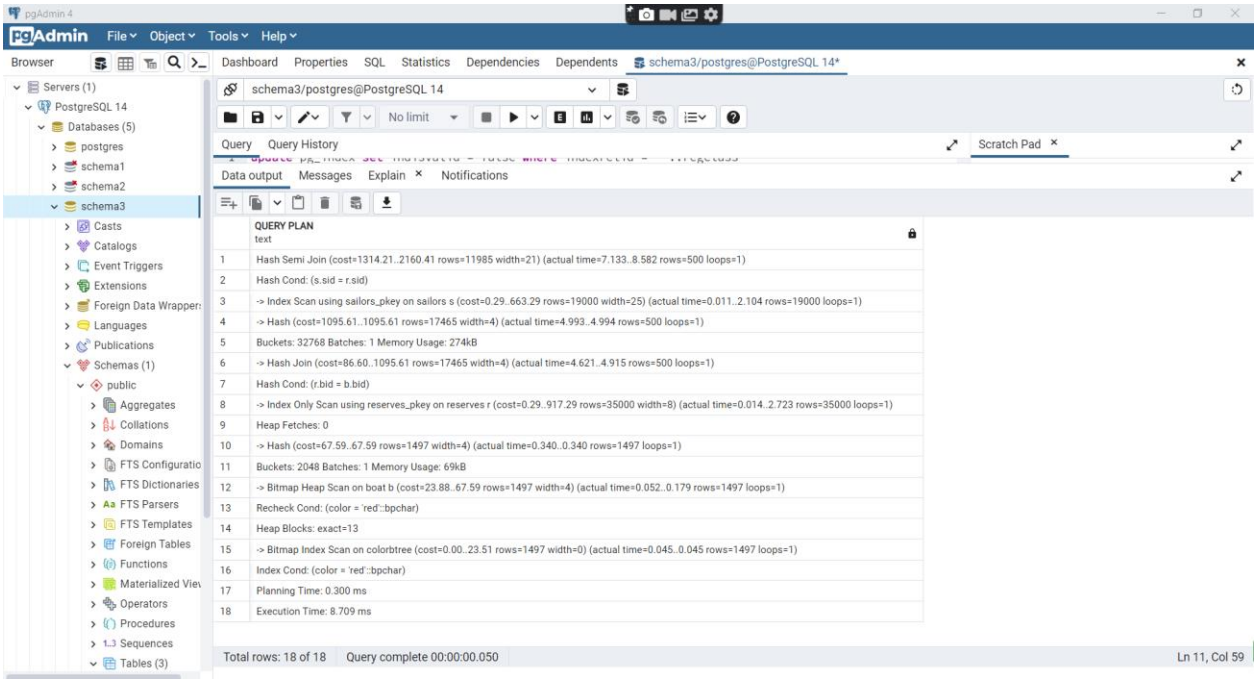


Execution Plan and Costs

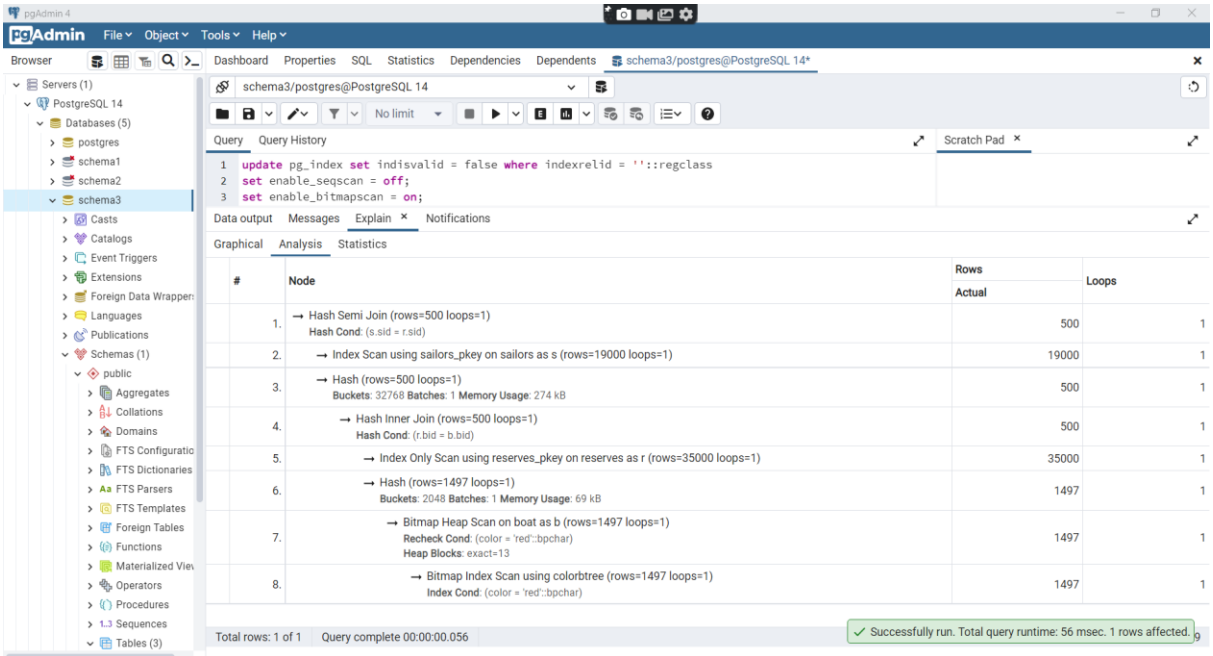


Query 8 with B+ tree index on: boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



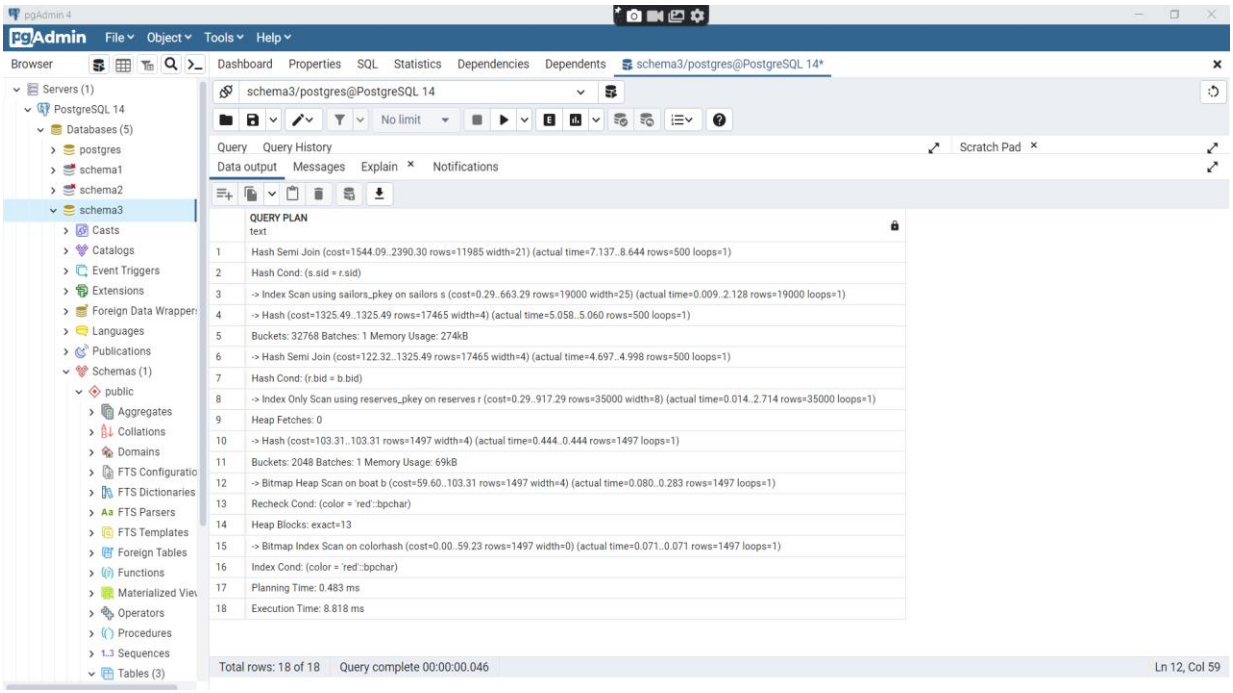
Execution Plan and Costs



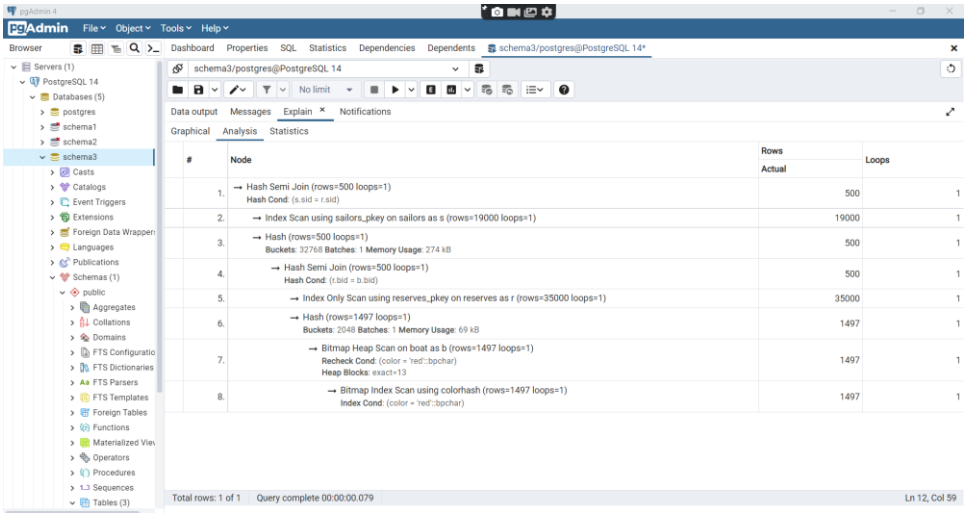
B+tree index optimised here as it was used to search(in O(logn)) in table boatfor every boat.color = ’red’ in the subquery

Query 8 with hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



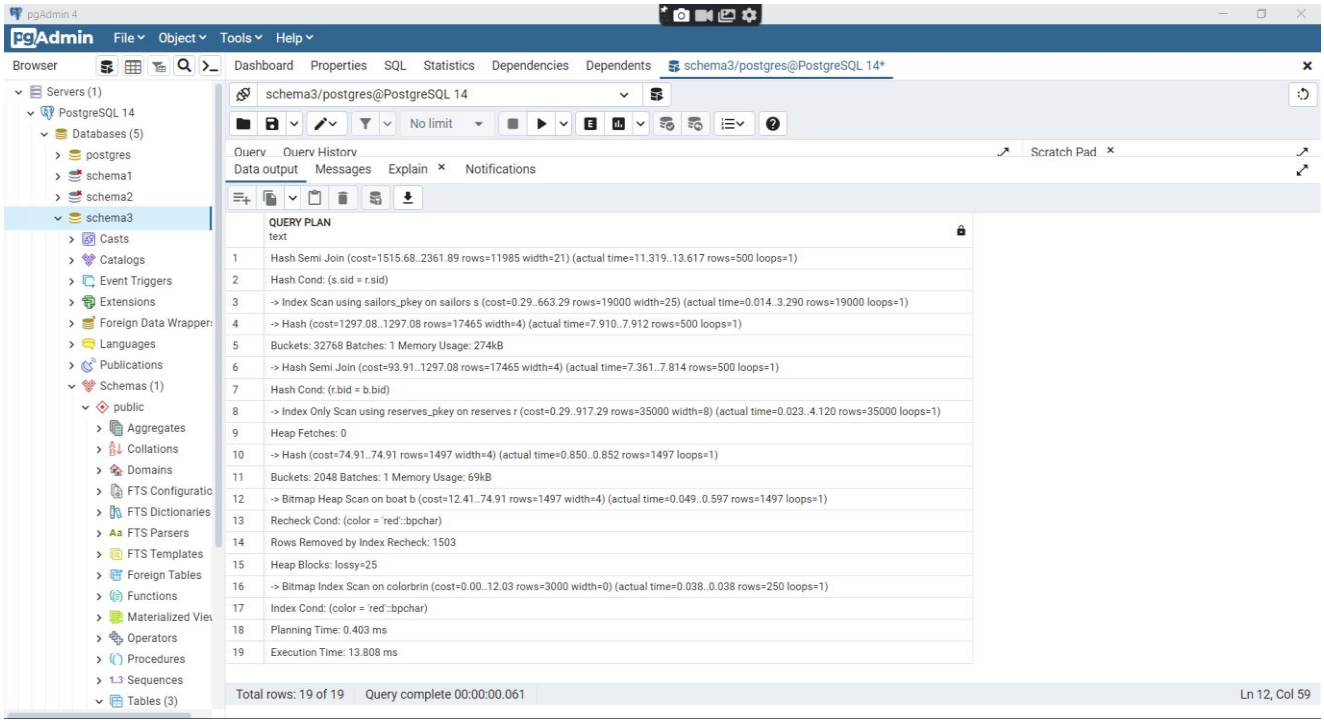
Execution Plan and Costs



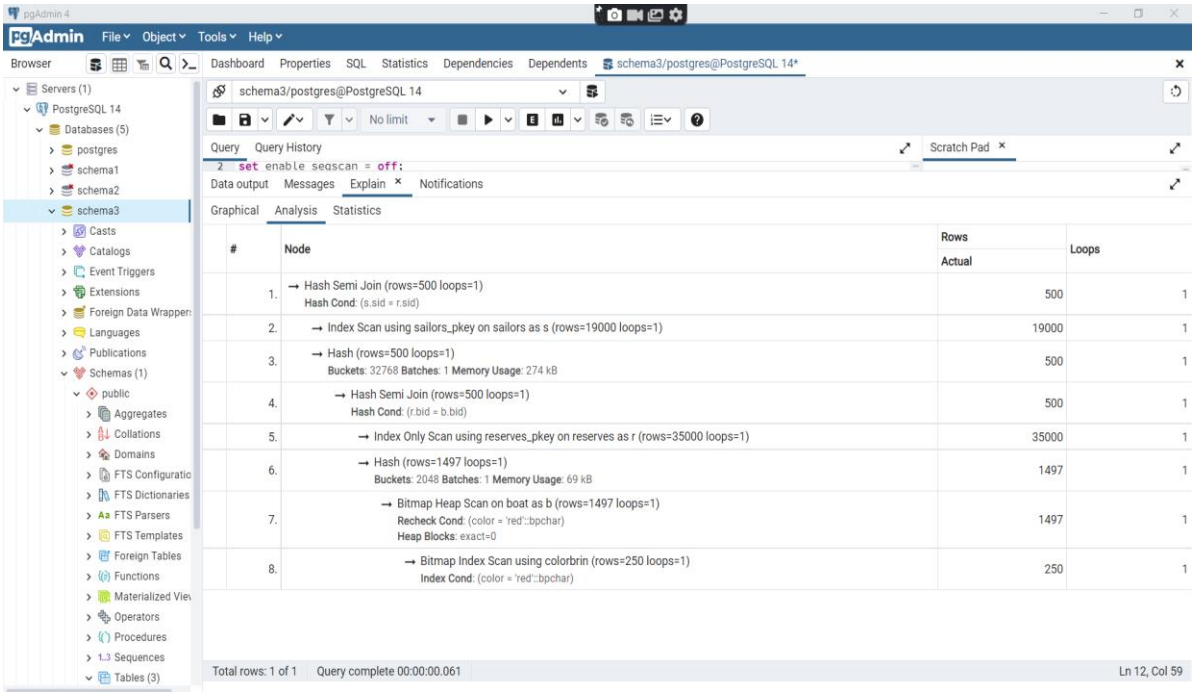
Hash index optimised here as it was used to search(in O(1)) in table boat for every boat.color = ’red’ in the subquery

Query 8 with BRIN on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



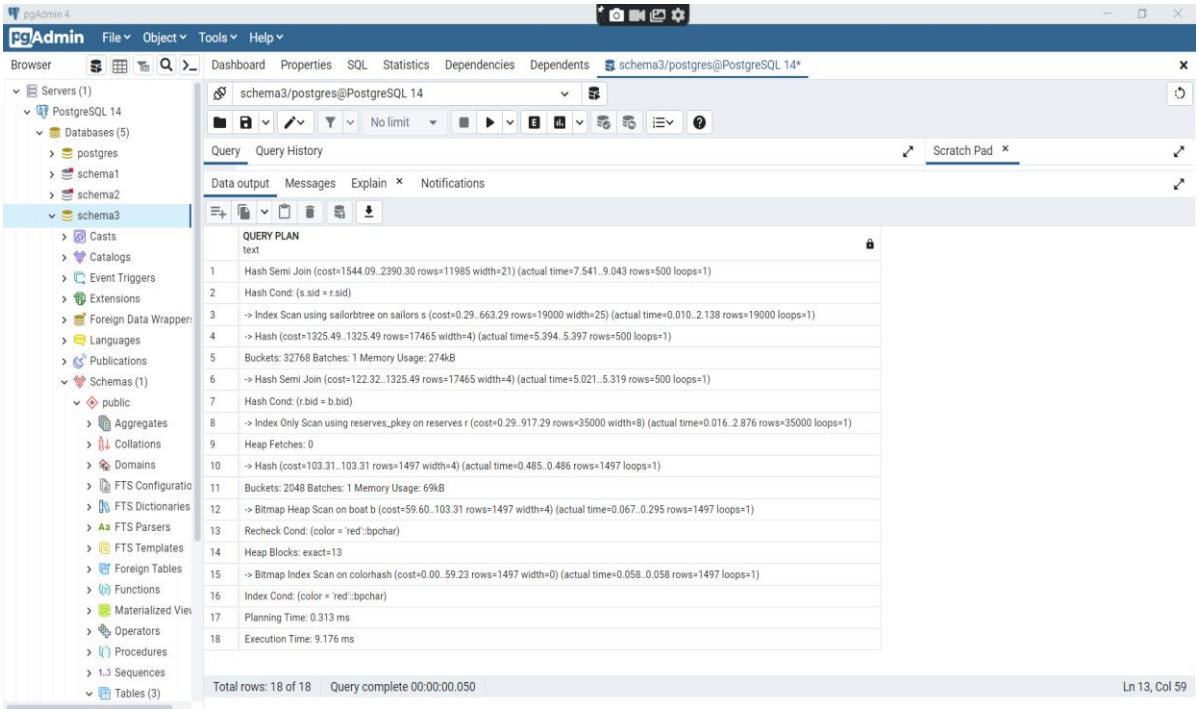
Execution Plan and Costs



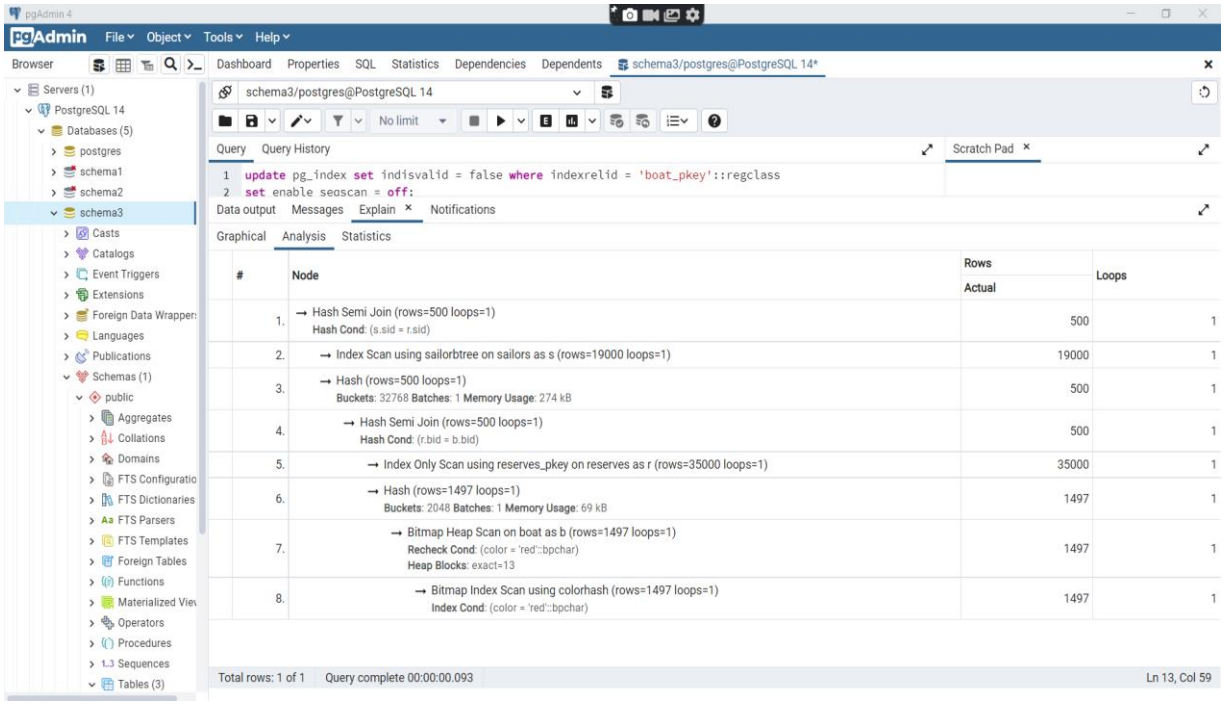
BRIN index here didn't optimize here as the search on boat.color was an exact value and the search BRIN is suitable for small range queries in a table with a very large range

Query 8 with hash index on boat(color) and B+tree index on sailors(sid) Flags["seqscan = off"," hashjoin=on", "mergejoin = on", sailors_pkey and reserves_pkey indisvalid = false else are true])

Planning and Execution time



Execution Plan and Costs



Since the search on b.color= 'red' is an exact value and search on s.sid is an exact value having a hash index(search time = O(1)) on boat.color and B+tree index(search time = O(logn)) on sailors.sid improved query performance

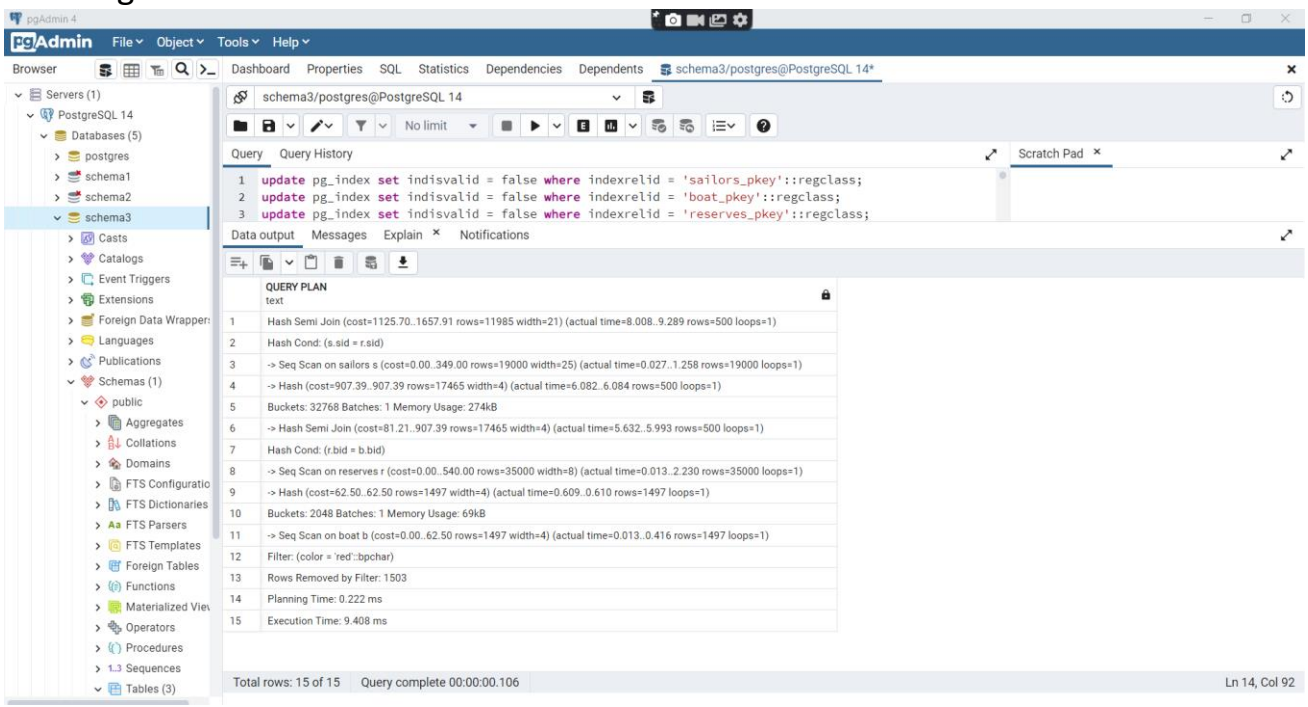
Optimised Query 8 SQL:

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

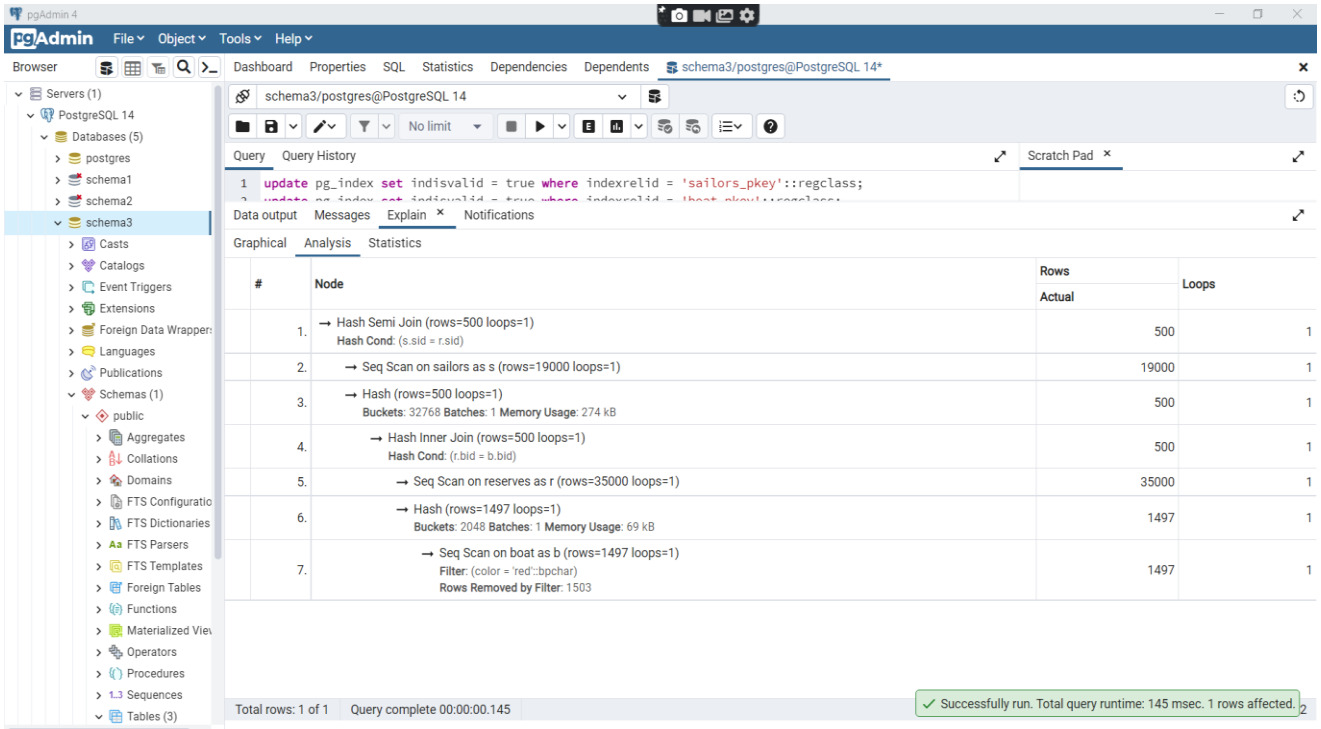
Reason: This query is optimised as it uses exists instead of in which reduces search time as SQL engine will terminate scanning procces once a match is found but in the original query with "in" all records from inner query will be fetched

Optimised Query 8 with no index Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary key indisvalid = true])

Planning and Execution time

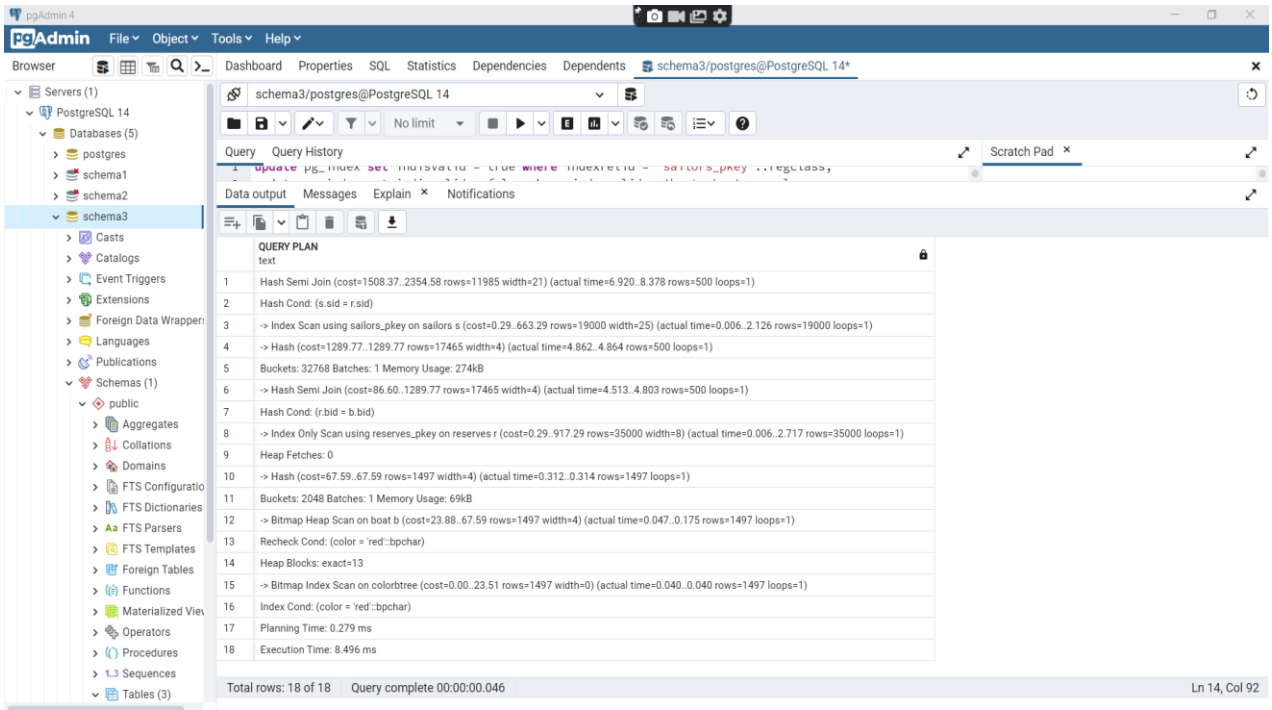


Execution Plan and Costs

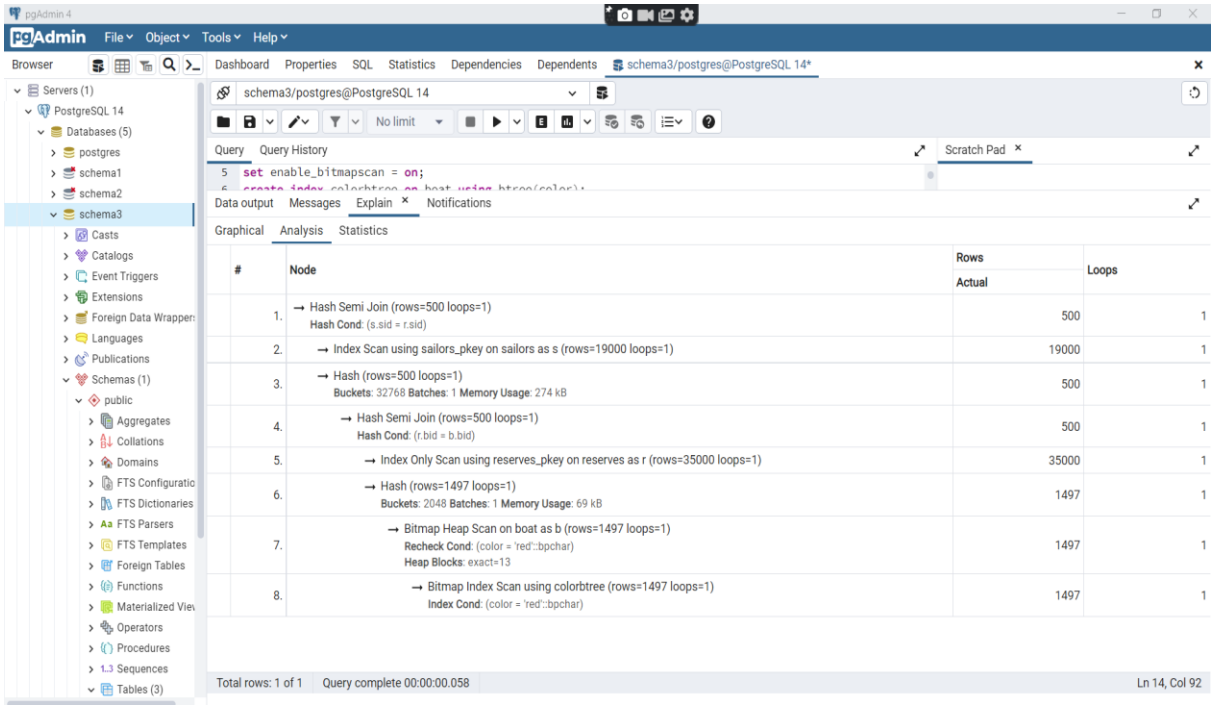


Optimised Query 8 with B+tree on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



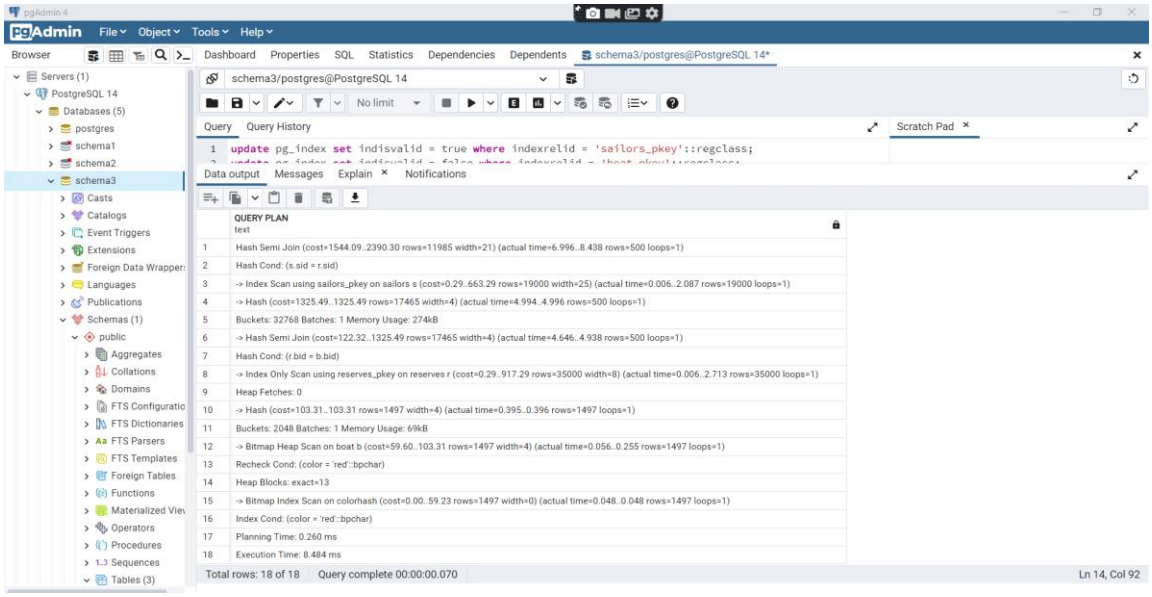
Execution Plan and Costs



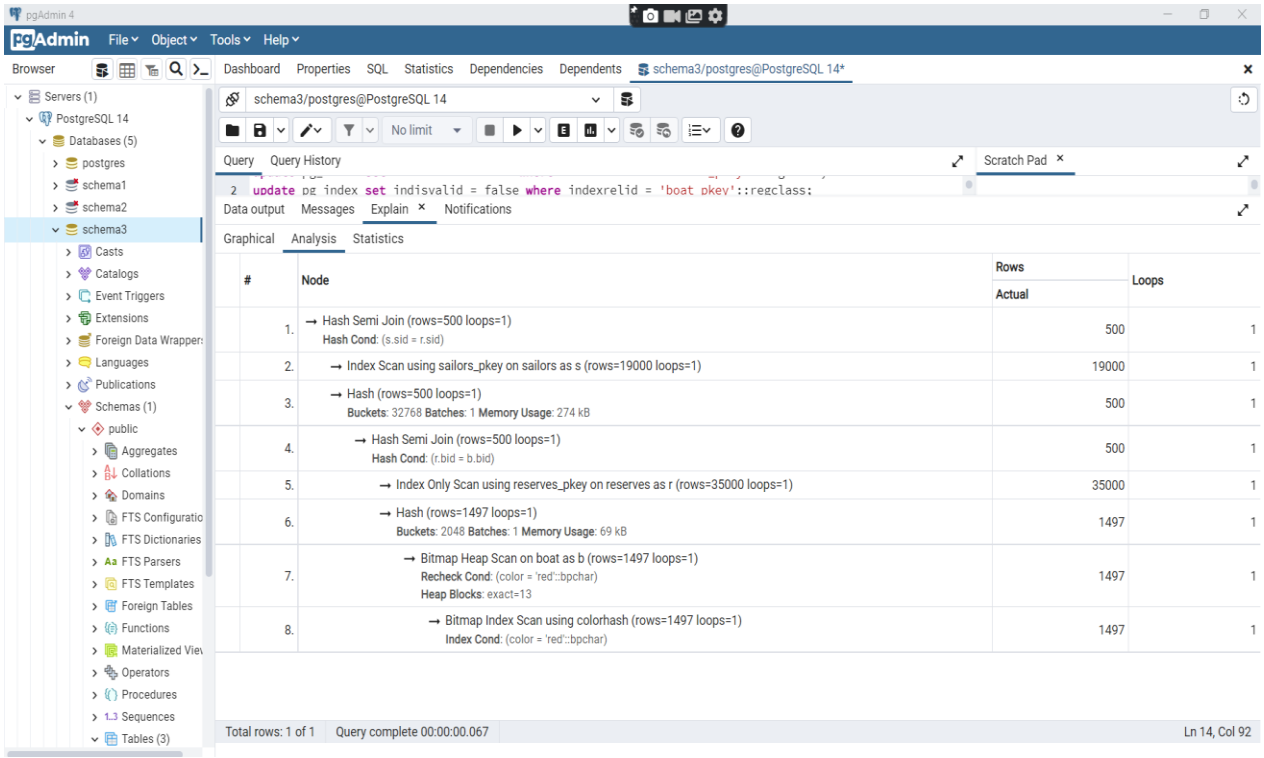
B+tree index optimised here as it was used to search(in O(logn)) in table boat for every boat.color = ’red’ in the subquery

Optimised Query 8 with Hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



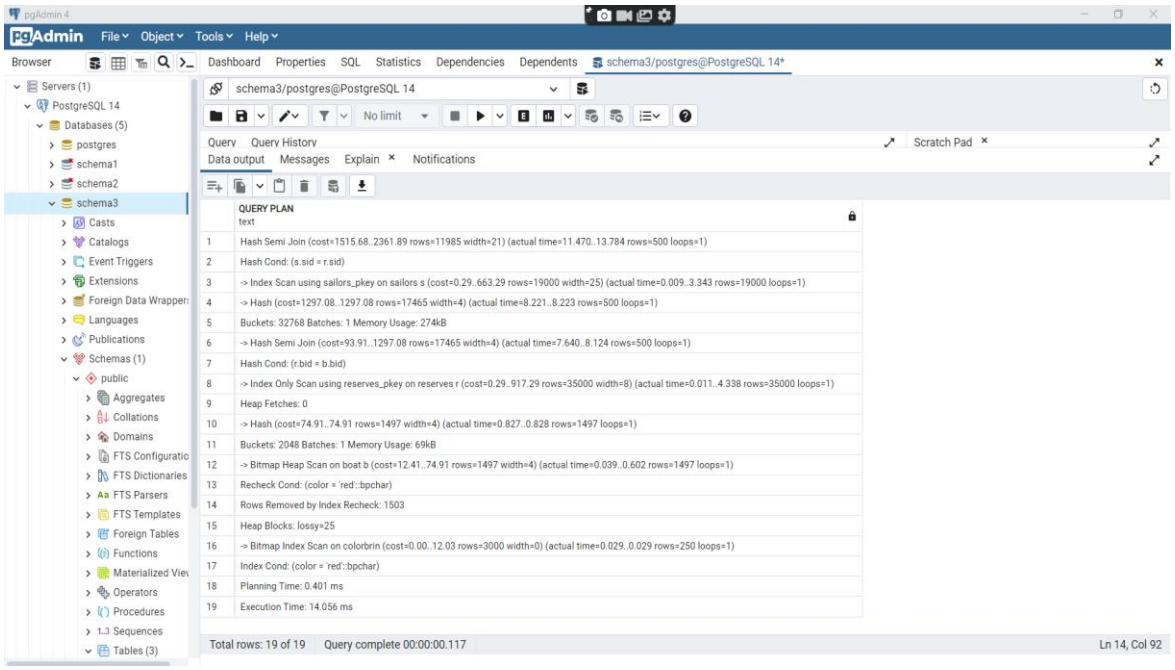
Execution Plan and Costs



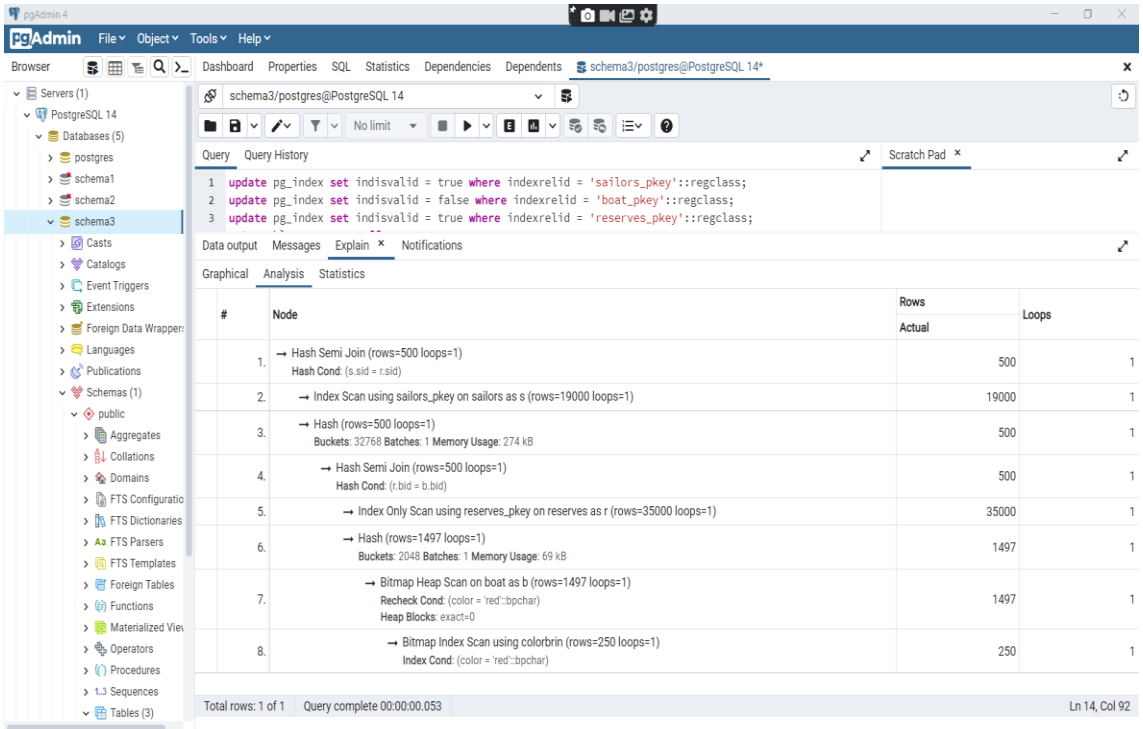
Hash index optimised here as it was used to search(in O(1)) in table boat for every boat.color = ’red’ in the subquery

Optimised Query 8 with BRIN on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



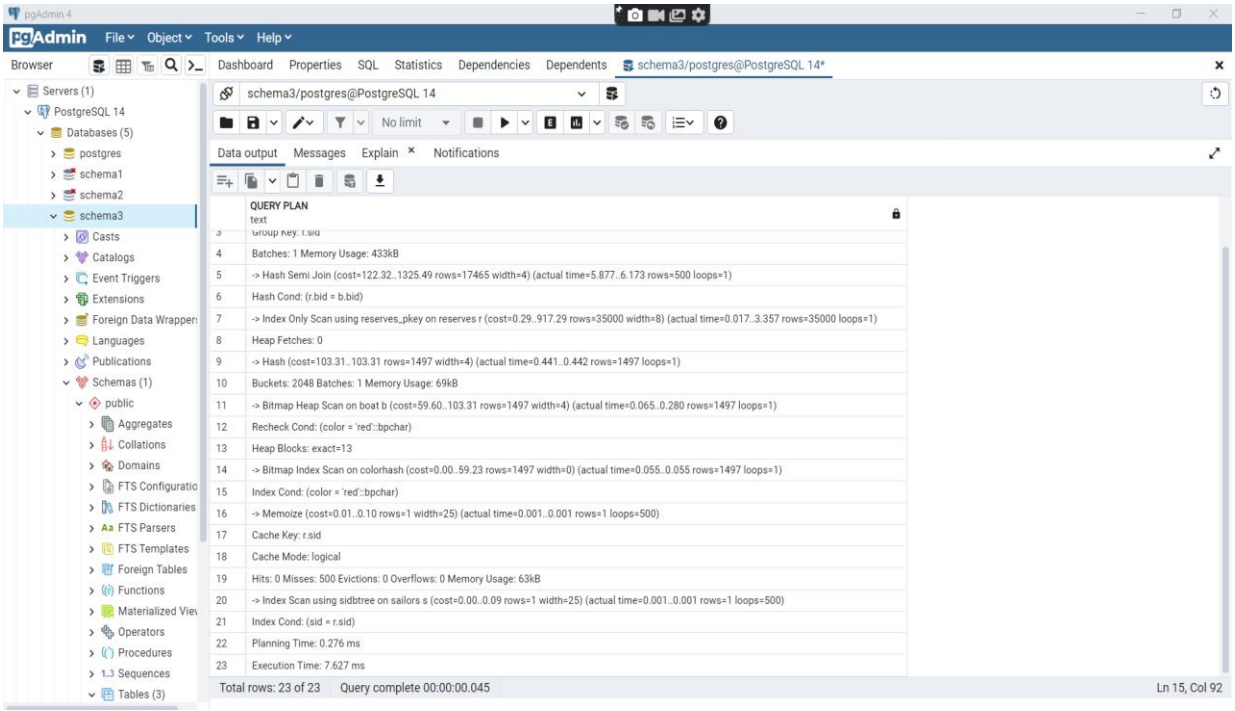
Execution plan and Costs



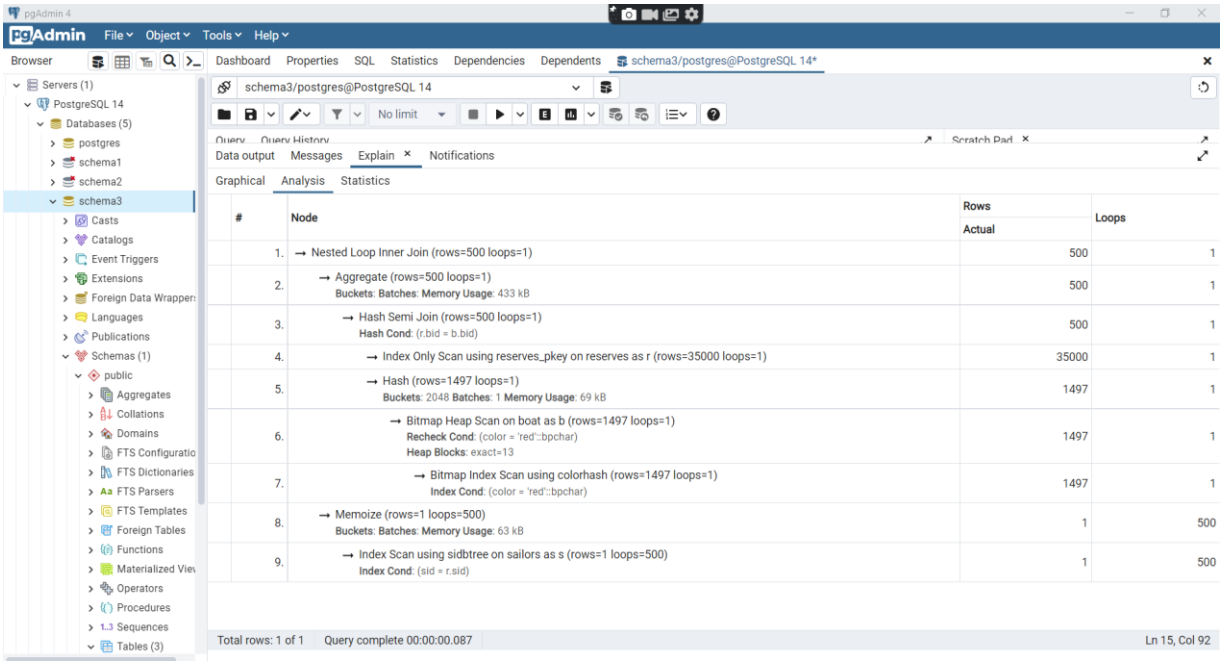
BRIN index here didn’t optimize here as the search on boat.color was an exact value and the search BRIN is suitable for small range queries in a table with a very large range

Optimised Query 8 with B+ tree index on sailor(sid) and hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey and boat_pkey indisvalid = false else are true])

Planning and Execution time



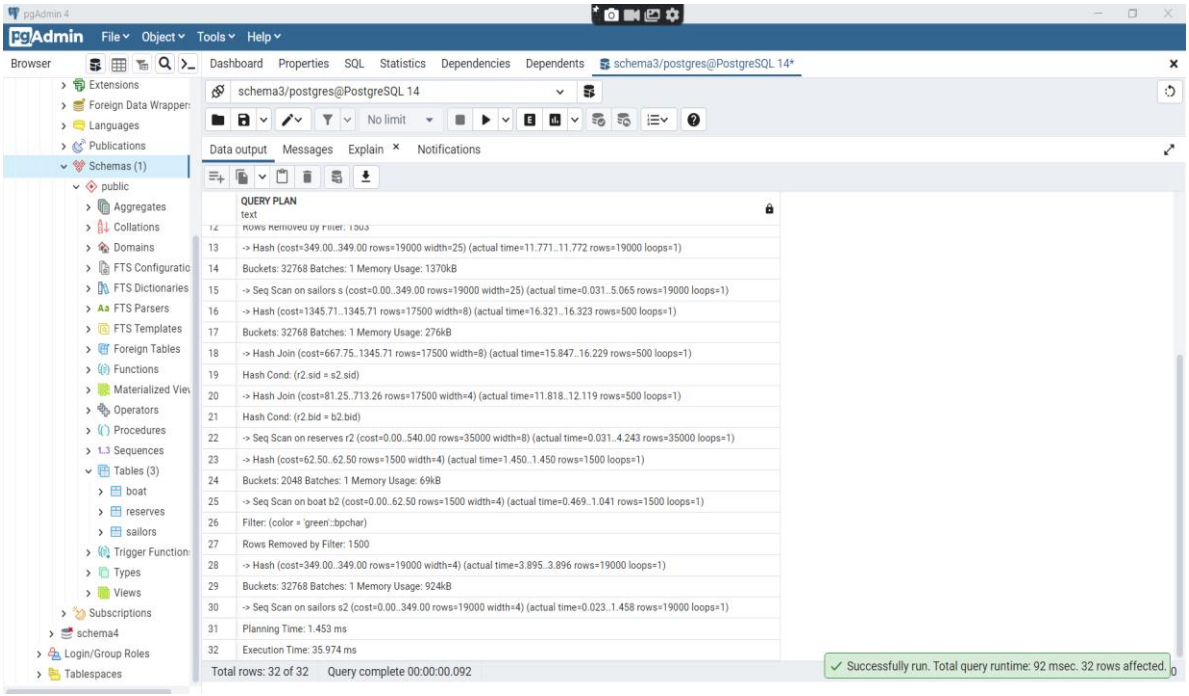
Execution Plan and Costs



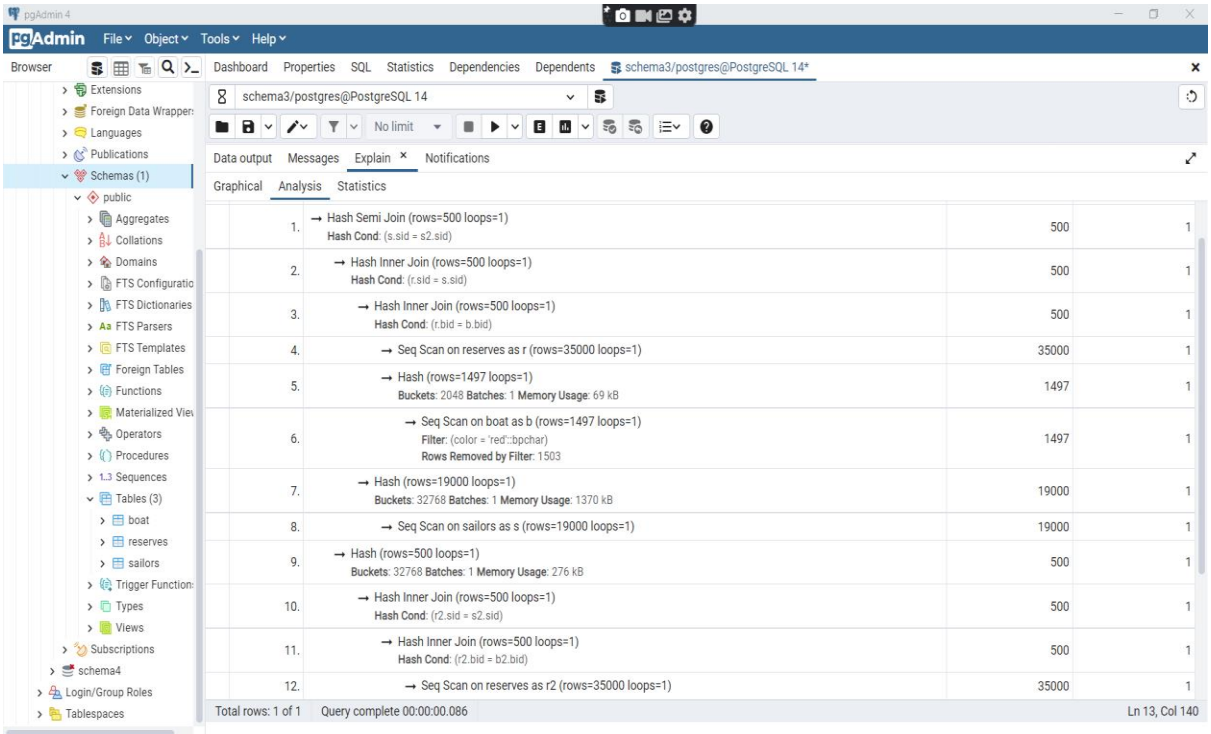
Since the search on b.color= 'red’ is an exact value and search on s.sid is an exact value having a hash index(search time = O(1)) on boat.color and B+tree index(search time = O(logn)) on sailors.sid improved query performance

Query 9 with no index Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary key indisvalid = false else are true])

Planning and Execution time

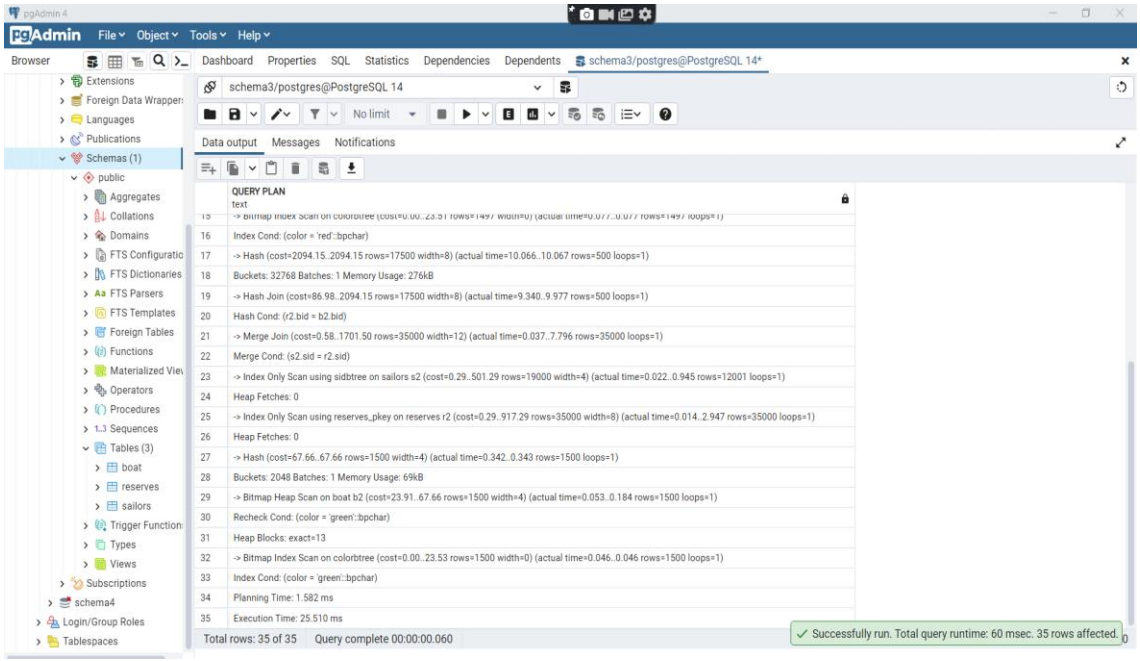


Execution Plan and Costs

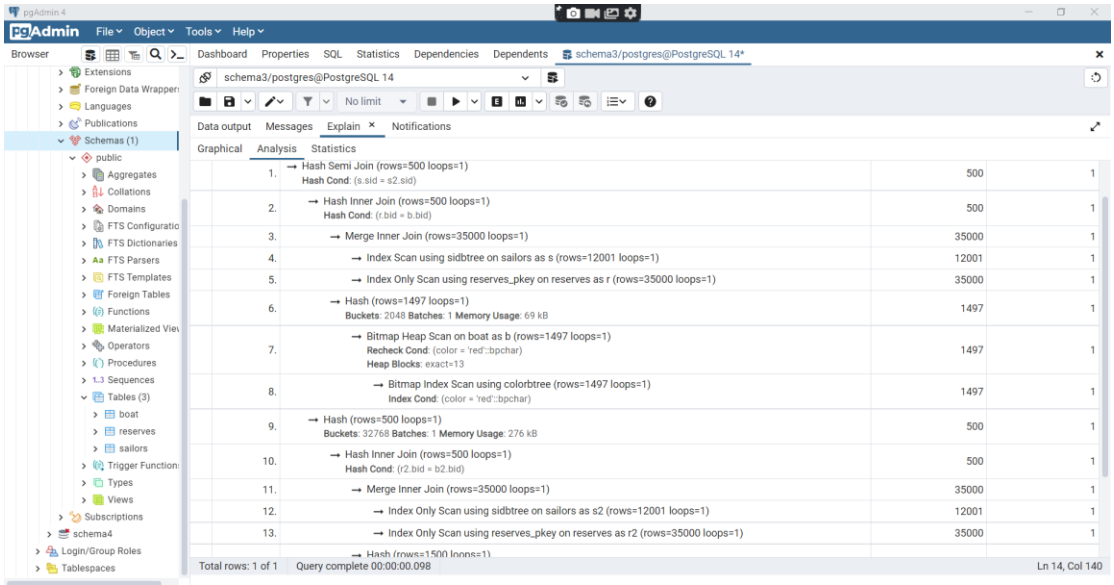


Query 9 with B+tree index on sailors(sid) and boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey and boat_pkey indisvalid = false else are true])

Planning and Execution time



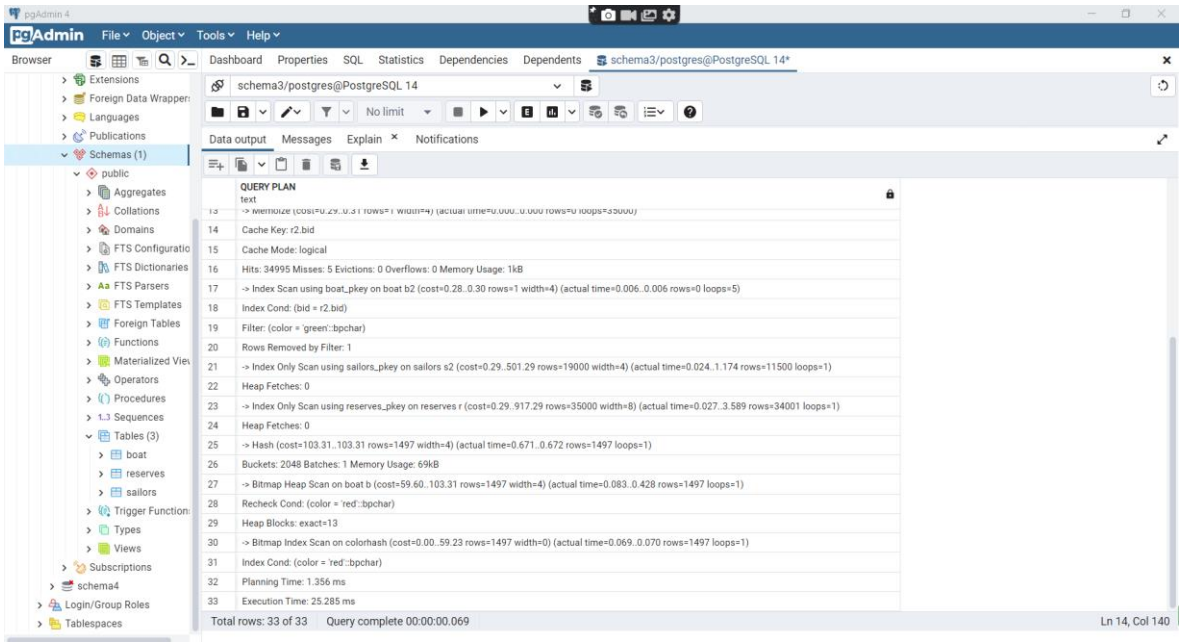
Execution Plan and Costs



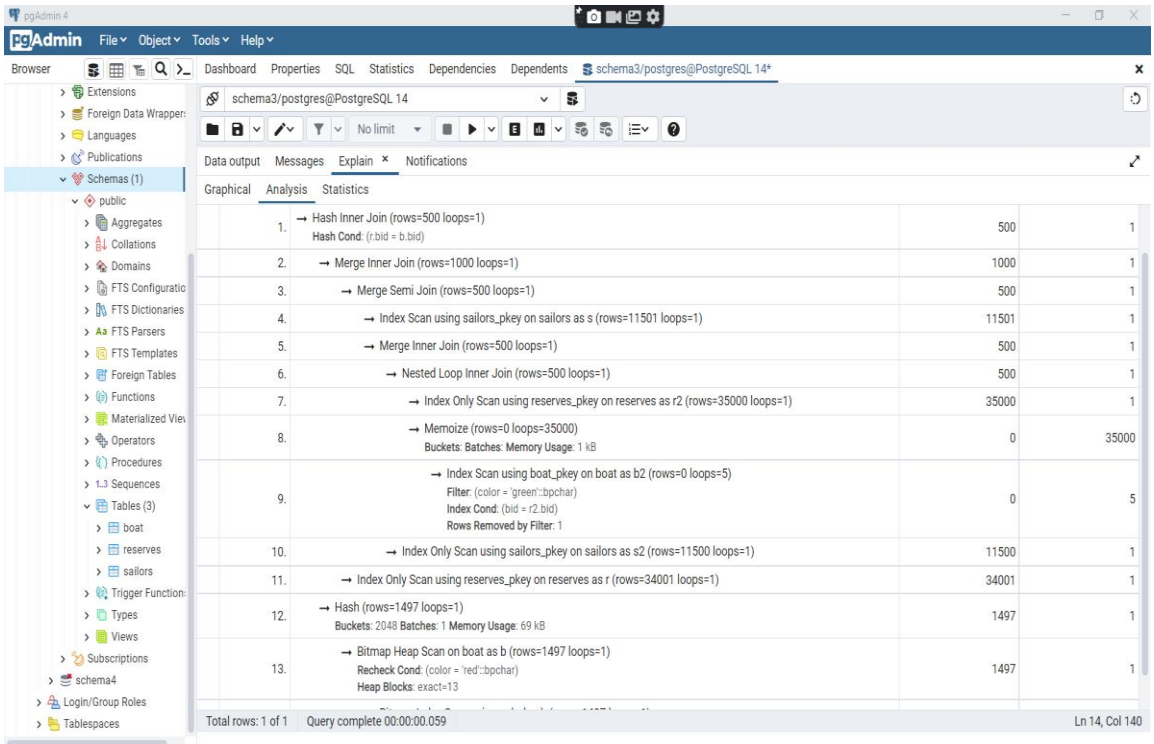
B+tree index optimised here as it was used to search(in O(logn)) in table sailors for every reserves.sid=sailors.sid and was used for boat.color = ‘red’ and boat.color = ‘green’

Query 9 with hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



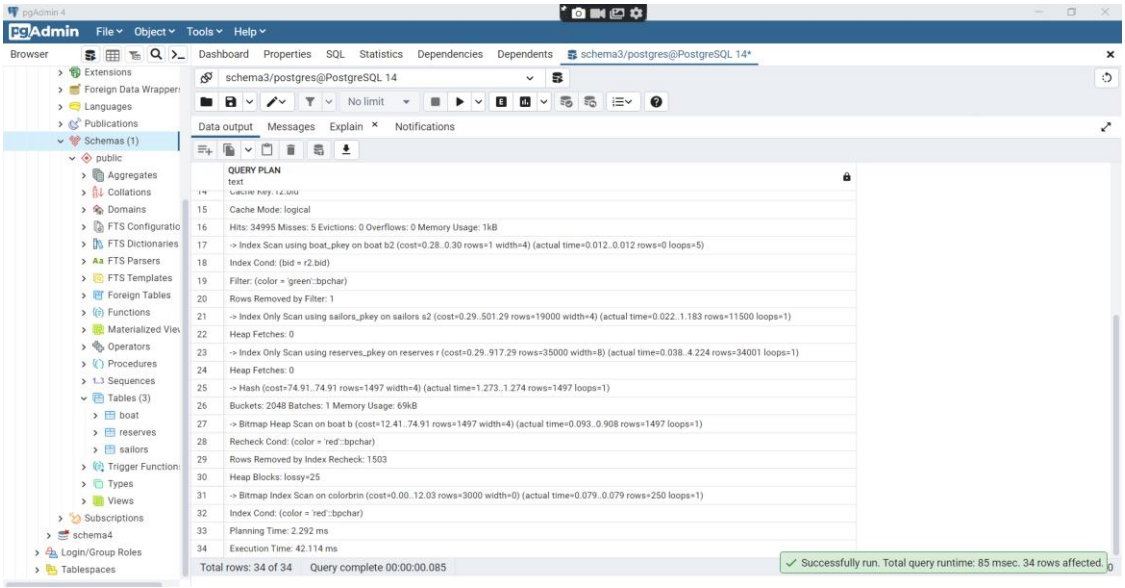
Execution Plan and Costs



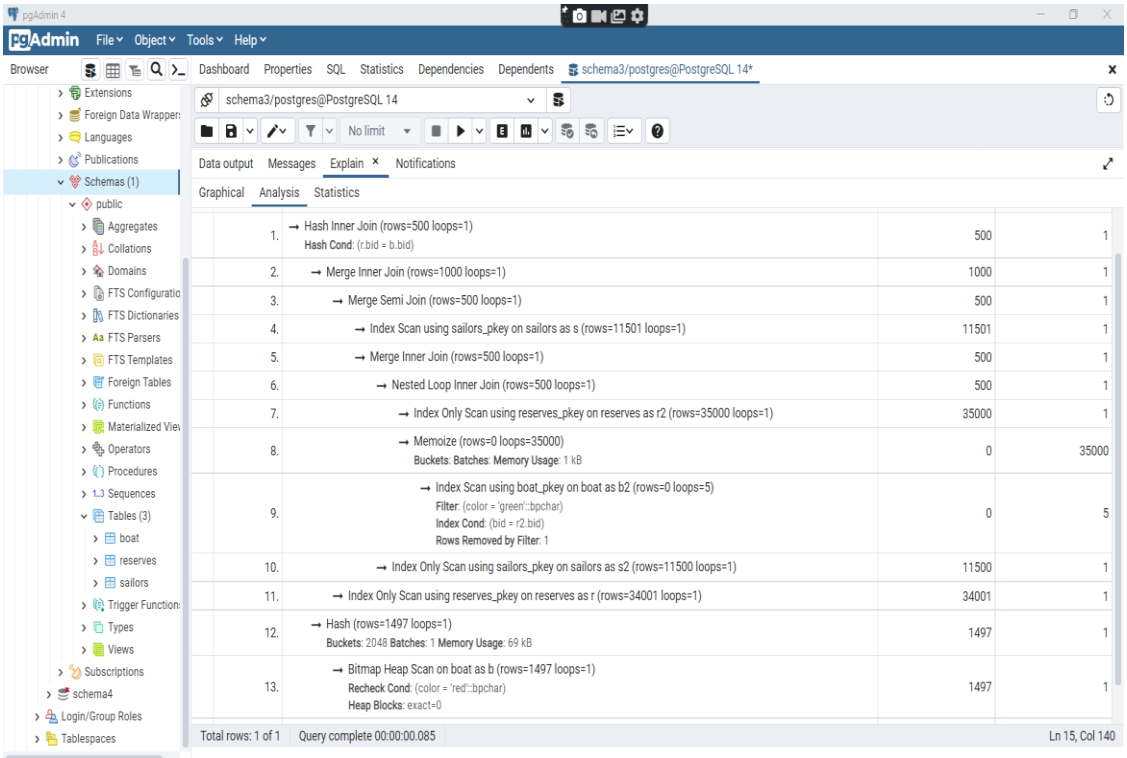
Hash index optimised here as it was used to search(in O(1)) for boat.color = ‘red’ and boat.color = ‘green’

Query 9 with BRIN index on boat(Color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



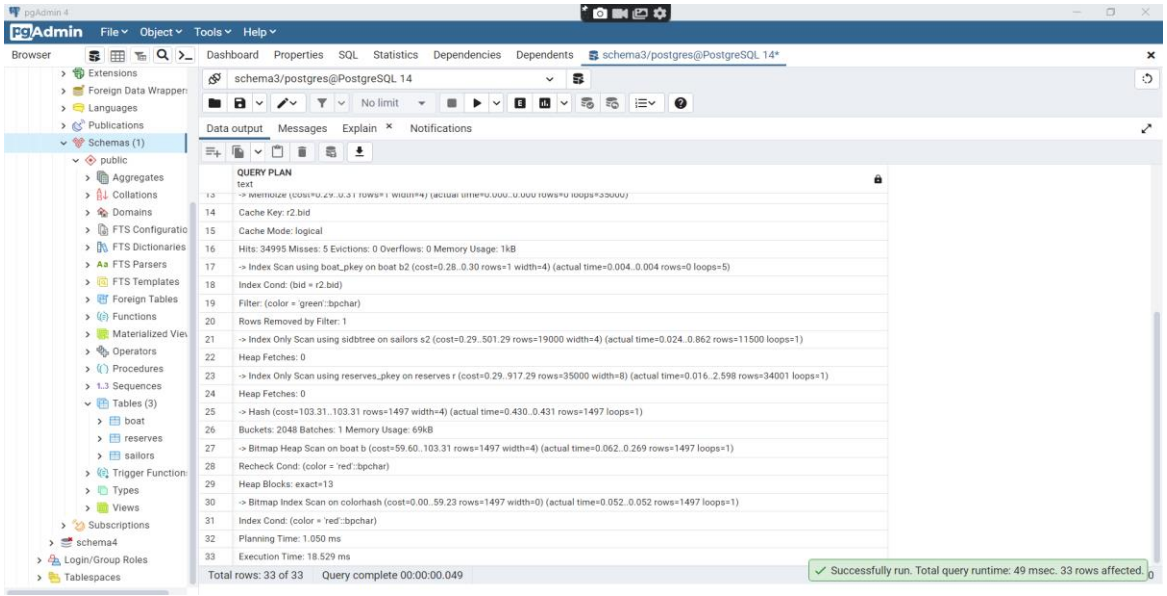
Execution Plan and Costs



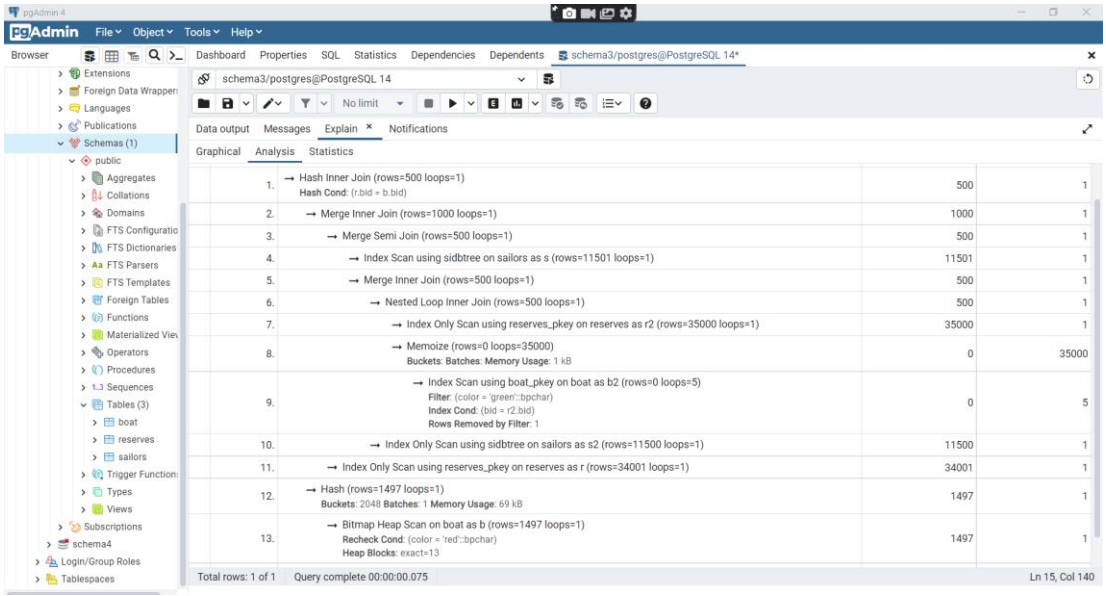
BRIN index here didn’t optimize here as the search on boat.color was an exact value and the search BRIN is suitable for small range queries in a table with a very large range

Query 9 with B+tree index on sailors(sid) and hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, sailors_pkey and boat_pkey indisvalid = false else are true])

Planning and Execution time



Execution Plan and Costs



Since the search on b.color= 'red’ and b.color = ‘green’is an exact value and search on sailors.sid = reserves.sed is an exact value having a hash index(search time = O(1)) on boat.color and B+tree index(search time = O(logn)) on sailors.sid improved query performance

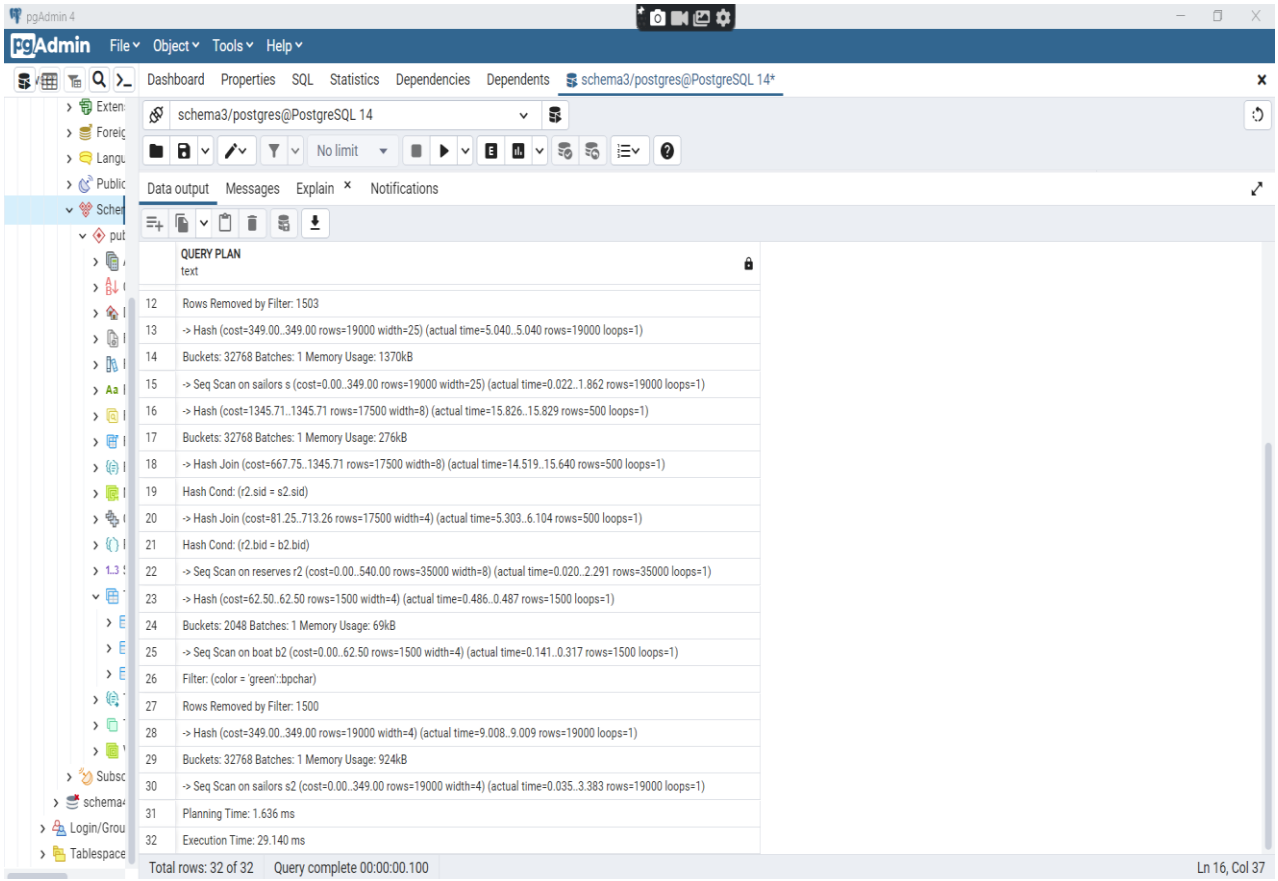
Optimised Query 9 SQL:

```
select s.sname

from sailors s inner join reserves r on s.sid = r.sid inner join boat b on b.bid =
r.bid

where b.color = 'red' and exists ( select s2.sid

                                   from sailors s2 inner join reserves r2 on s2.sid = r2.sid
                                    inner join boat b2 on b2.bid=r2.bid

                                   where s.sid = s2.sid and b2.color = 'green');
```

Reason: This query is optimised as it uses inner join instead of cartesian product so less number of rows is propagated upwards in inner nodes of relational algebra tree also t uses exists instead of in which reduces search time as SQL engine will terminate scanning procces once a match is found but in the original query with “in” all records from inner query will be fetched

Optimised Query 9 with no index Flags[“seqscan = on”,” hashjoin=on”, “mergejoin = on”, all primary key indisvalid = true])

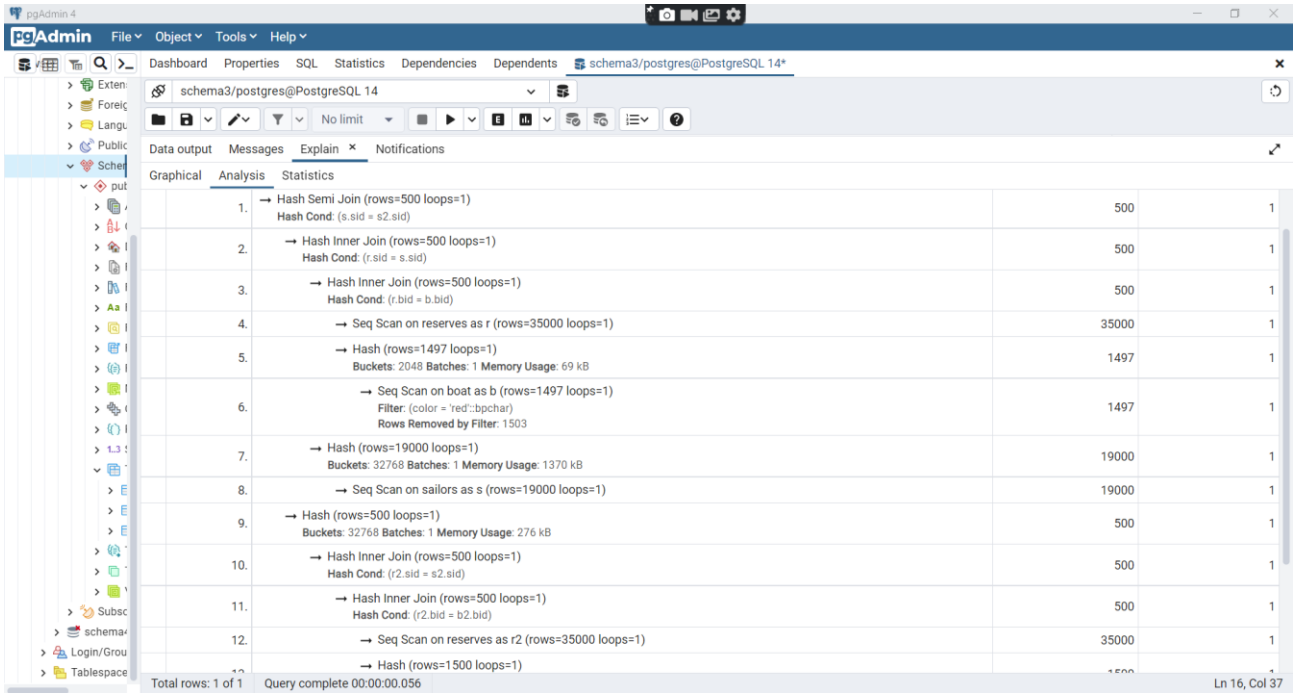
Planning and Execution time



	QUERY PLAN text
12	Rows Removed by Filter: 1503
13	-> Hash (cost=349.00..349.00 rows=19000 width=25) (actual time=5.040..5.040 rows=19000 loops=1)
14	Buckets: 32768 Batches: 1 Memory Usage: 1370kB
15	-> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.022..1.862 rows=19000 loops=1)
16	-> Hash (cost=1345.71..1345.71 rows=17500 width=8) (actual time=15.826..15.829 rows=500 loops=1)
17	Buckets: 32768 Batches: 1 Memory Usage: 276kB
18	-> Hash Join (cost=667.75..1345.71 rows=17500 width=8) (actual time=14.519..15.640 rows=500 loops=1)
19	Hash Cond: (r2.sid = s2.sid)
20	-> Hash Join (cost=81.25..713.26 rows=17500 width=4) (actual time=5.303..6.104 rows=500 loops=1)
21	Hash Cond: (r2.bid = b2.bid)
22	-> Seq Scan on reserves r2 (cost=0.00..540.00 rows=35000 width=8) (actual time=0.020..2.291 rows=35000 loops=1)
23	-> Hash (cost=62.50..62.50 rows=1500 width=4) (actual time=0.486..0.487 rows=1500 loops=1)
24	Buckets: 2048 Batches: 1 Memory Usage: 69kB
25	-> Seq Scan on boat b2 (cost=0.00..62.50 rows=1500 width=4) (actual time=0.141..0.317 rows=1500 loops=1)
26	Filter: (color = 'green'::bpchar)
27	Rows Removed by Filter: 1500
28	-> Hash (cost=349.00..349.00 rows=19000 width=4) (actual time=9.008..9.009 rows=19000 loops=1)
29	Buckets: 32768 Batches: 1 Memory Usage: 924kB
30	-> Seq Scan on sailors s2 (cost=0.00..349.00 rows=19000 width=4) (actual time=0.035..3.383 rows=19000 loops=1)
31	Planning Time: 1.636 ms
32	Execution Time: 29.140 ms

Total rows: 32 of 32 | Query complete 00:00:00.100 | Ln 16, Col 37

Execution Plan and Costs

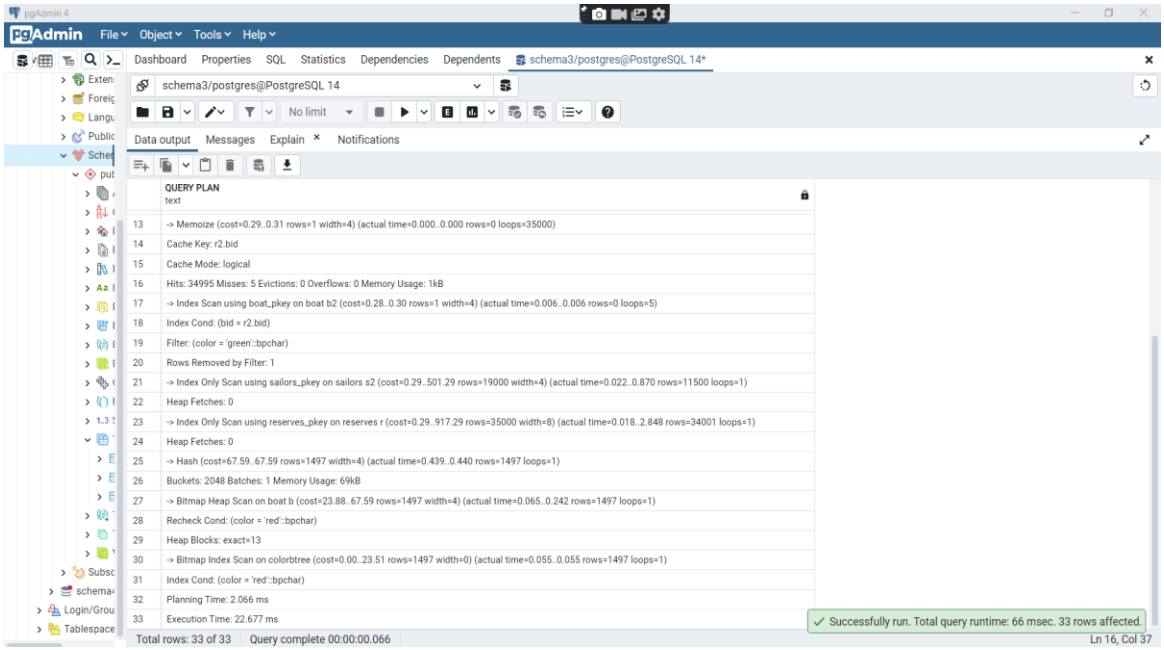


1.	→ Hash Semi Join (rows=500 loops=1) **Hash Cond**: (s.sid = s2.sid)	500	1
2.	→ Hash Inner Join (rows=500 loops=1) **Hash Cond**: (r.sid = s.sid)	500	1
3.	→ Hash Inner Join (rows=500 loops=1) **Hash Cond**: (r.bid = b.bid)	500	1
4.	→ Seq Scan on reserves as r (rows=35000 loops=1)	35000	1
5.	→ Hash (rows=1497 loops=1) **Buckets**: 2048 **Batches**: 1 **Memory Usage**: 69 kB	1497	1
6.	→ Seq Scan on boat as b (rows=1497 loops=1) **Filter**: (color = 'red'::bpchar) **Rows Removed by Filter**: 1503	1497	1
7.	→ Hash (rows=19000 loops=1) **Buckets**: 32768 **Batches**: 1 **Memory Usage**: 1370 kB	19000	1
8.	→ Seq Scan on sailors as s (rows=19000 loops=1)	19000	1
9.	→ Hash (rows=500 loops=1) **Buckets**: 32768 **Batches**: 1 **Memory Usage**: 276 kB	500	1
10.	→ Hash Inner Join (rows=500 loops=1) **Hash Cond**: (r2.sid = s2.sid)	500	1
11.	→ Hash Inner Join (rows=500 loops=1) **Hash Cond**: (r2.bid = b2.bid)	500	1
12.	→ Seq Scan on reserves as r2 (rows=35000 loops=1)	35000	1
	→ Hash (rows=1500 loops=1)		

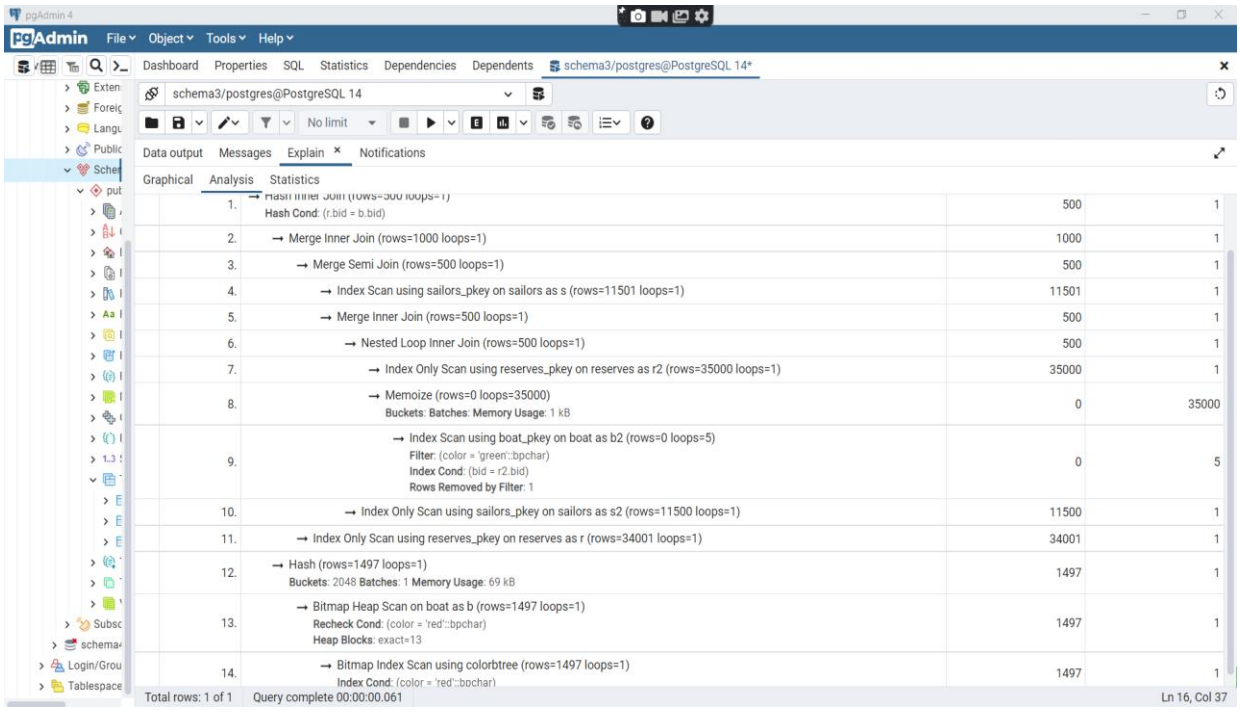
Total rows: 1 of 1 | Query complete 00:00:00.056 | Ln 16, Col 37

Optimised Query 9 with B+tree index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



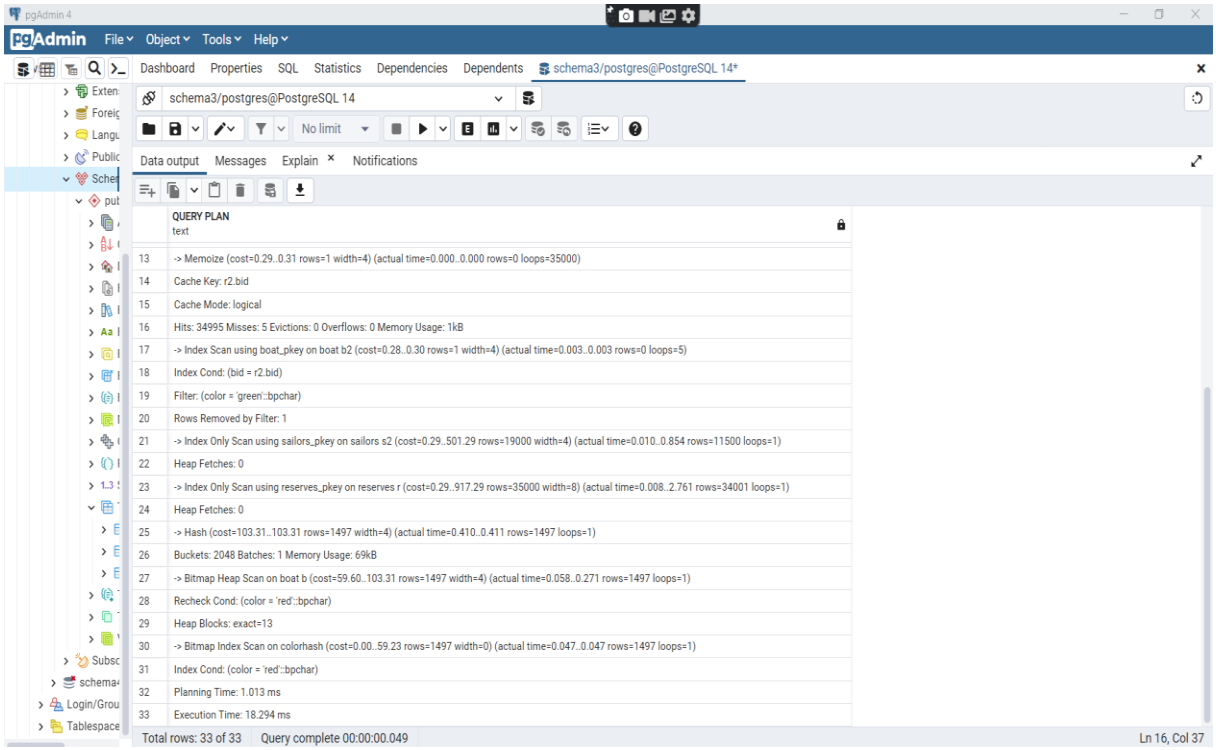
Execution Plan and Costs



B+tree index optimised here as it was used to search(in O(logn)) for boat.color = ‘red’ and boat.color = ‘green’

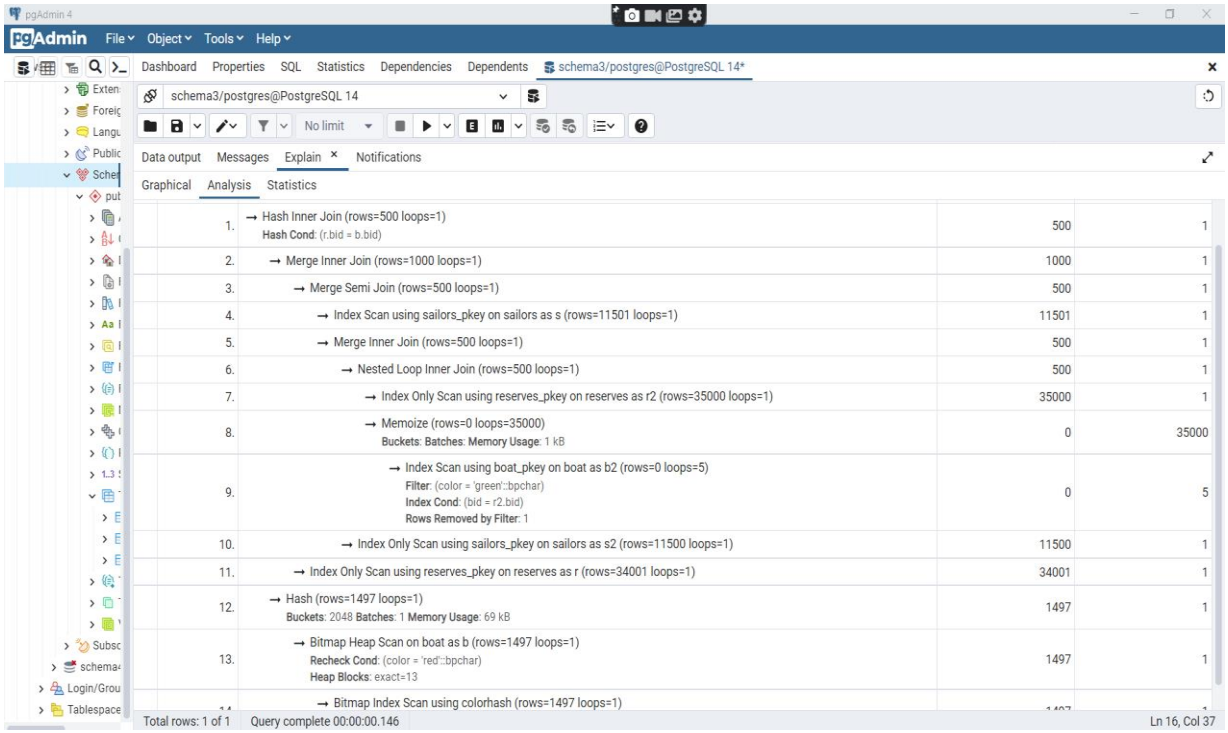
Optimised Query 9 with Hash index on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



	QUERY PLAN text
13	-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)
14	Cache Key: r2.bid
15	Cache Mode: logical
16	Hits: 34995 Misses: 5 Evictions: 0 Overflows: 0 Memory Usage: 1kB
17	-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=5)
18	Index Cond: (bid = r2.bid)
19	Filter: (color = 'green'::bpchar)
20	Rows Removed by Filter: 1
21	-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..501.29 rows=19000 width=4) (actual time=0.010..0.854 rows=11500 loops=1)
22	Heap Fetches: 0
23	-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.008..2.761 rows=34001 loops=1)
24	Heap Fetches: 0
25	-> Hash (cost=103.31..103.31 rows=1497 width=4) (actual time=0.410..0.411 rows=1497 loops=1)
26	Buckets: 2048 Batches: 1 Memory Usage: 69kB
27	-> Bitmap Heap Scan on boat b (cost=59.60..103.31 rows=1497 width=4) (actual time=0.058..0.271 rows=1497 loops=1)
28	Recheck Cond: (color = 'red'::bpchar)
29	Heap Blocks: exact=13
30	-> Bitmap Index Scan on colorhash (cost=0.00..59.23 rows=1497 width=0) (actual time=0.047..0.047 rows=1497 loops=1)
31	Index Cond: (color = 'red'::bpchar)
32	Planning Time: 1.013 ms
33	Execution Time: 18.294 ms

Execution Plan and Costs

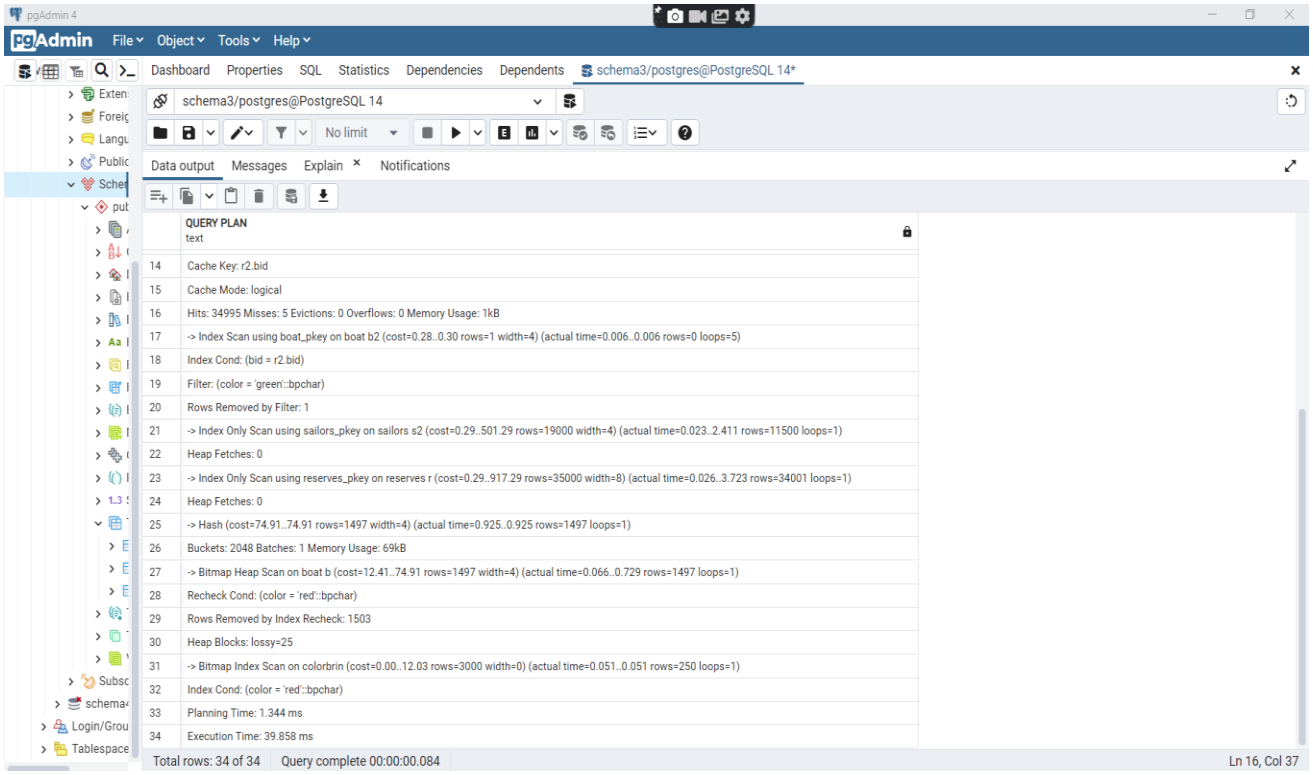


1.	→ Hash Inner Join (rows=500 loops=1) **Hash Cond**: (r.bid = b.bid)	500	1
2.	→ Merge Inner Join (rows=1000 loops=1)	1000	1
3.	→ Merge Semi Join (rows=500 loops=1)	500	1
4.	→ Index Scan using sailors_pkey on sailors as s (rows=11501 loops=1)	11501	1
5.	→ Merge Inner Join (rows=500 loops=1)	500	1
6.	→ Nested Loop Inner Join (rows=500 loops=1)	500	1
7.	→ Index Only Scan using reserves_pkey on reserves as r2 (rows=35000 loops=1)	35000	1
8.	→ Memoize (rows=0 loops=35000) **Buckets**: **Batches**: **Memory Usage**: 1 kB	0	35000
9.	→ Index Scan using boat_pkey on boat as b2 (rows=0 loops=5) **Filter**: (color = 'green'::bpchar) **Index Cond**: (bid = r2.bid) **Rows Removed by Filter**: 1	0	5
10.	→ Index Only Scan using sailors_pkey on sailors as s2 (rows=11500 loops=1)	11500	1
11.	→ Index Only Scan using reserves_pkey on reserves as r (rows=34001 loops=1)	34001	1
12.	→ Hash (rows=1497 loops=1) **Buckets**: 2048 **Batches**: 1 **Memory Usage**: 69 kB	1497	1
13.	→ Bitmap Heap Scan on boat as b (rows=1497 loops=1) **Recheck Cond**: (color = 'red'::bpchar) **Heap Blocks**: exact=13	1497	1
14.	→ Bitmap Index Scan using colorhash (rows=1497 loops=1)	1497	1

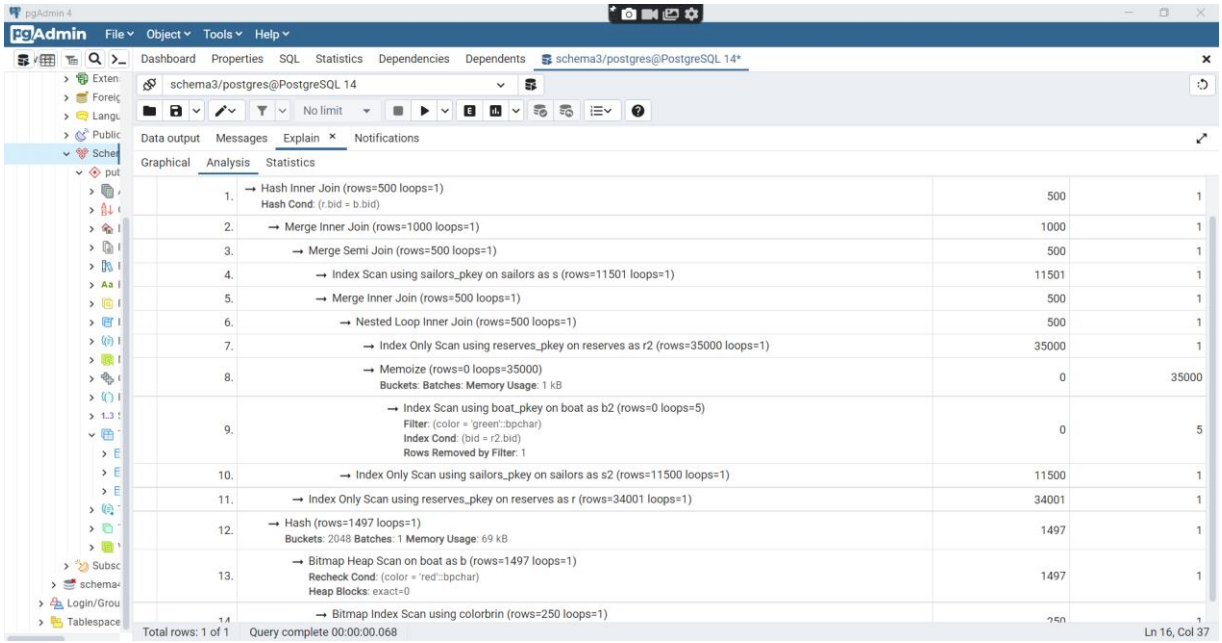
Hash index optimised here as it was used to search(in O(1)) for boat.color = ‘red’ and boat.color = ‘green’

Optimised Query 9 with BRIN on boat(color) Flags[“seqscan = off”,” hashjoin=on”, “mergejoin = on”, boat_pkey indisvalid = false else are true])

Planning and Execution time



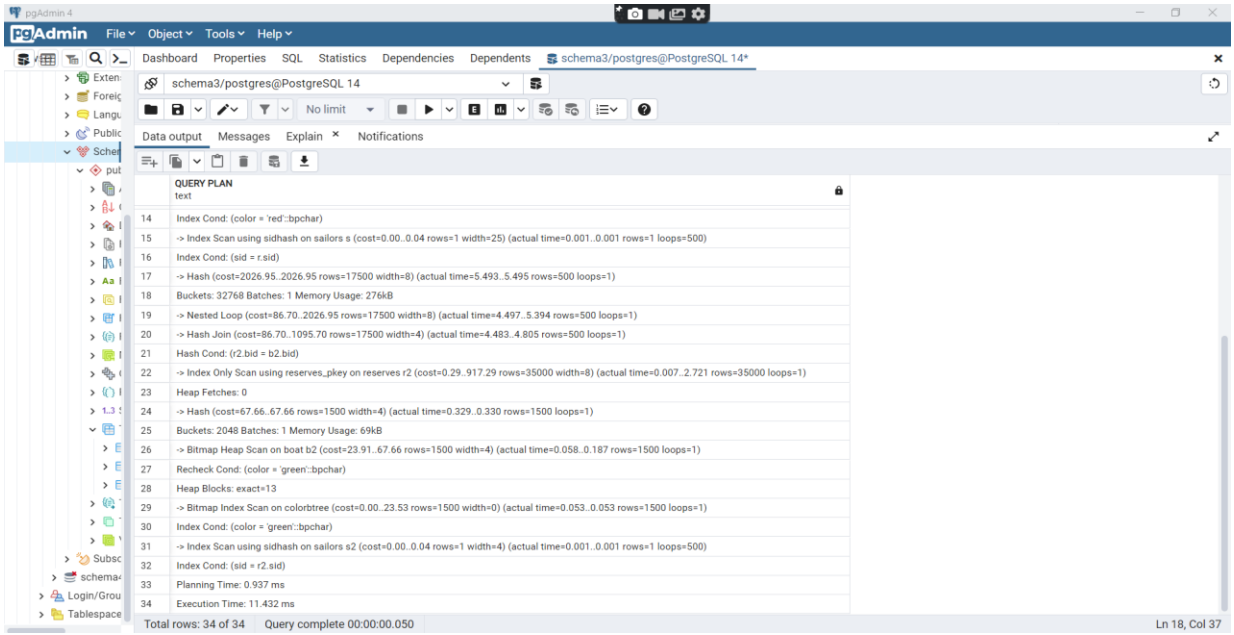
Execution Plan and Costs



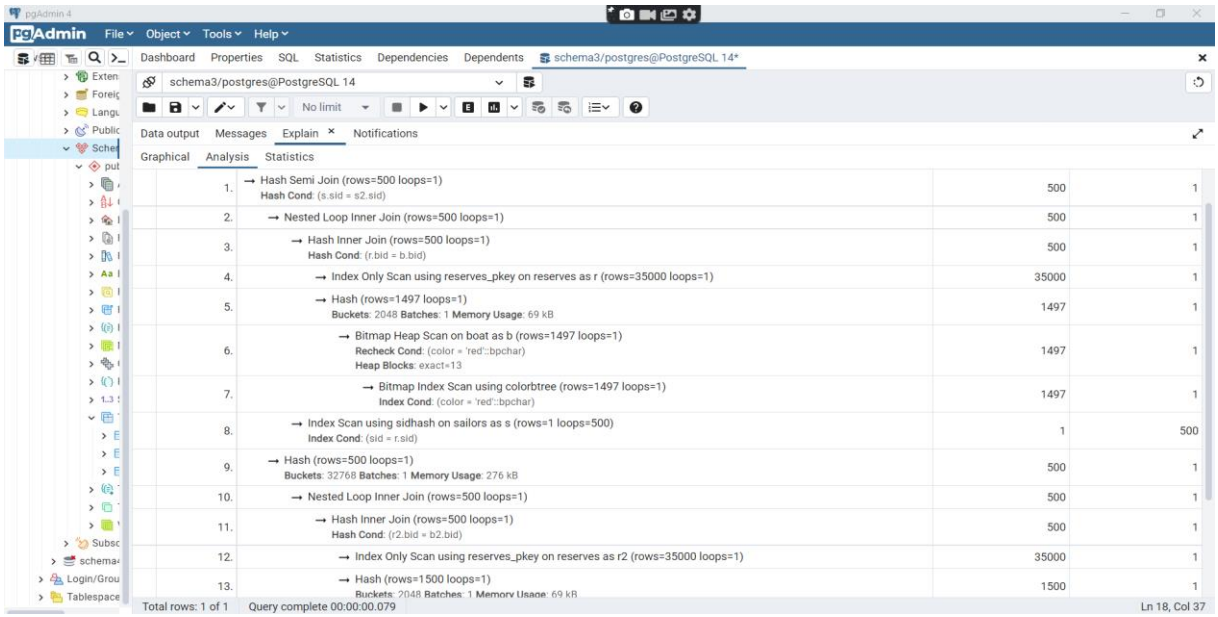
BRIN index here didn't optimize here as the search on boat.color was an exact value and the search BRIN is suitable for small range queries in a table with a very large range

Optimised Query 9 with B+tree index on boat(color) and Hash index on sailors(sid) Flags["seqscan = off"," hashjoin=on", "mergejoin = on", sailors_pkey and boat_pkey indisvalid = false else are true])

Planning and Execution time



Execution Plan and Costs



Since the search on b.color= 'red' and b.color = 'green'is an exact value and search on sailors.sid = reserves.sid is an exact value having a hash index(search time = O(1)) on sailors.sid and B+tree index(search time = O(logn)) on boat.color improved query performance