# *Schema 2*

***Modifications on Data Insertions:***

As required in the project description, that the number of Employees should be 5000, 30 departments, 20 department locations, and 600 projects.

So in order for the results of the query to be not empty, I have changed the following data population code:

1. For the **populateEmployee** in order to make sure that the 5000 Employee is inserted I’ve changed the for loop condition for the I to be less than 5001 “ for(int i =1 ; i < 5001 ; I++) “.
2. Also in order to satisfy that there are salaries greater than or equal to 40000 I made an int variable called salary which is a random variable (I.e. int salary = new Random().nextInt(40000+1)+10000;).
3. I’ve also changed the if condition in line 270 to be if(i> 500) then result = “F”;
4. I’ve also changed the variable “i” in if condition line 272 that is inserted in dno to be i%30+1 as to make sure of constraints.
5. For the **populateDepartment** in order to make sure that the departments of total 30, I’ve changed the for loop to be (I.e. for (int i =1 ; i< 31 ; i++)).
6. And the if condition in line 288, I’ve changed the value of variable i in MgrSSN to be : i%5000+1, in order to make sure that it respect the constraints of foreign key.
7. For the **populateDeptLocations** same as before I changed the i value in for loop to be less than 21 to make sure that I have 20 department locations as stated in the requirements, And I’ve also changed the if condition in line 297 the value of I in Dnumber to be (i%30+1), in order to make sure that it respect the constraints of foreign key.
8. For the **populateProject** same as before I changed the i value in for loop to be less than 601 to make sure that I have 600 department locations as stated in the requirements, And the if condition in line 307, I’ve changed the “i” value to be i%30+1 in Dnumber, in order to make sure that it respect the constraints of foreign key.
9. For the **populateDependent** I haven’t changed the for loop, it have the same value, however in line 329 I’ve changed the “I” value in Essn to be I%5000+1, in order to make sure that it respect the constraints of foreign key.
10. For the **PopulateWorksOn** I haven’t changed the for loop, it have the same value, however in line 316 I’ve changed the “i” value in Essn to be i%5000+1, and in pNo to be i%600+1, in order to make sure that it respect the constraints of foreign key.

***Query 2:***

Output of Query : 201

**Without an index:**

**Flags :**

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = off;

set enable\_indexonlyscan = off;

set enable\_material = off;

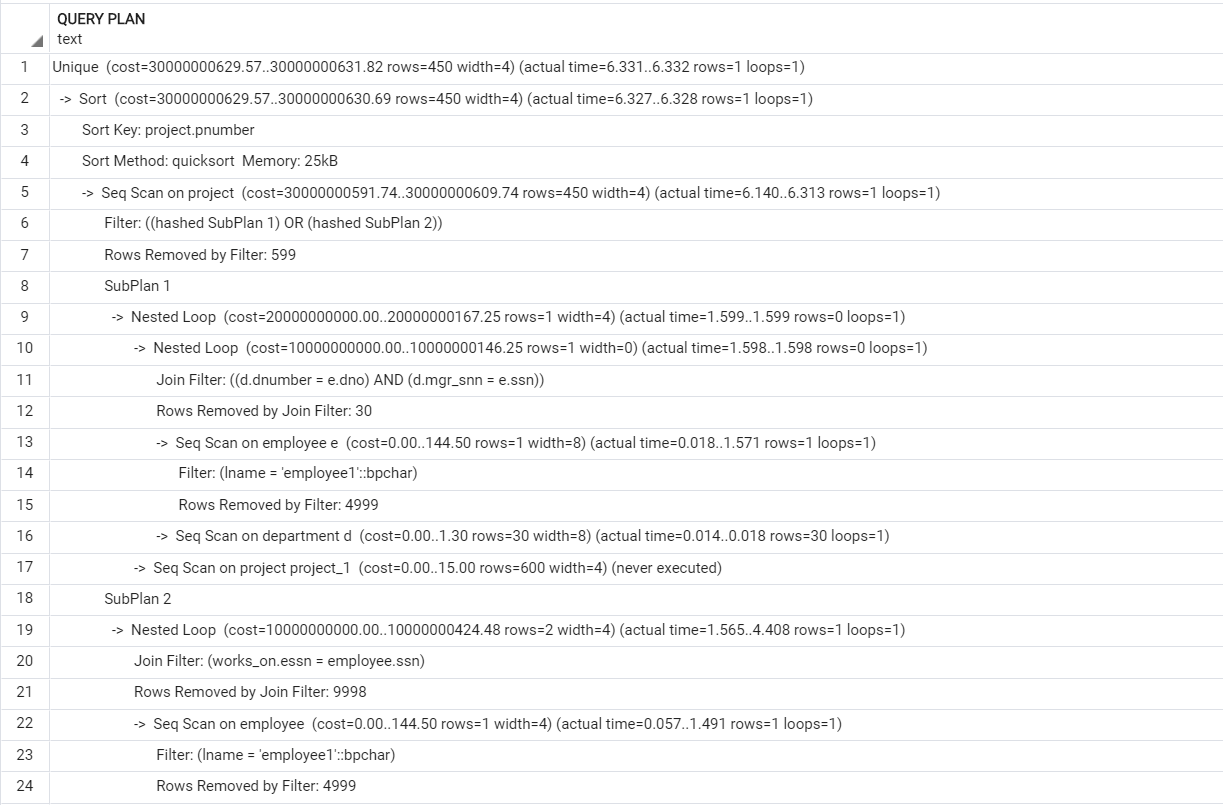
set enable\_mergejoin = off;

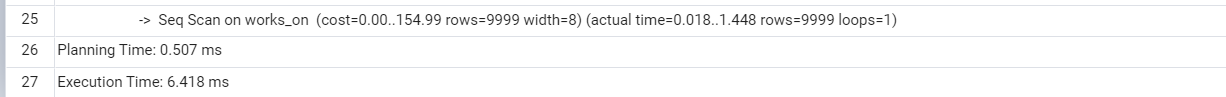
set enable\_nestloop = off;

set enable\_bitmapscan= off;

set enable\_sort = on;

set enable\_tidscan = off;





Estimated cost:



1st run execution time: 6.418 ms

2nd run execution time: 7.636 ms

3rd run execution time : 5.685 ms

Average execution time = 6.579 ms

***With an index:***

**B+ Tree Index :**

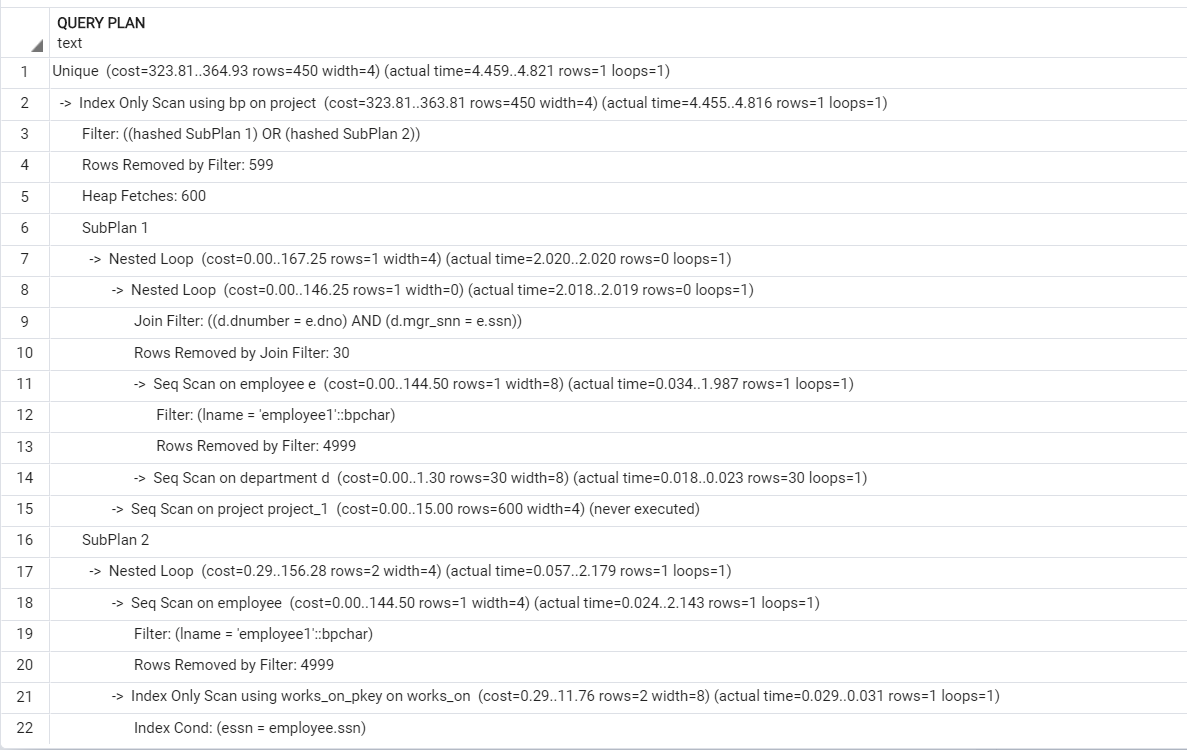
**Flags :**

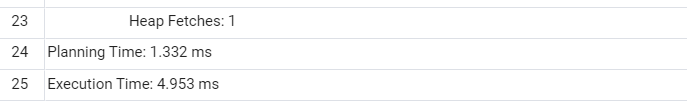
set enable\_seqscan=off;

set enable\_indexscan=on;

**create index be on Employee using btree(Dno);**

**create index bp on Project using btree(Pnumber);**





Estimated cost:



1st run execution time: 4.953 ms

2nd run execution time: 7.357 ms

3rd run execution time : 8.942 ms

Average execution time = 7.084 ms

After creating btree index on last name in table Employee with the previous indexes the Execution time reduced :

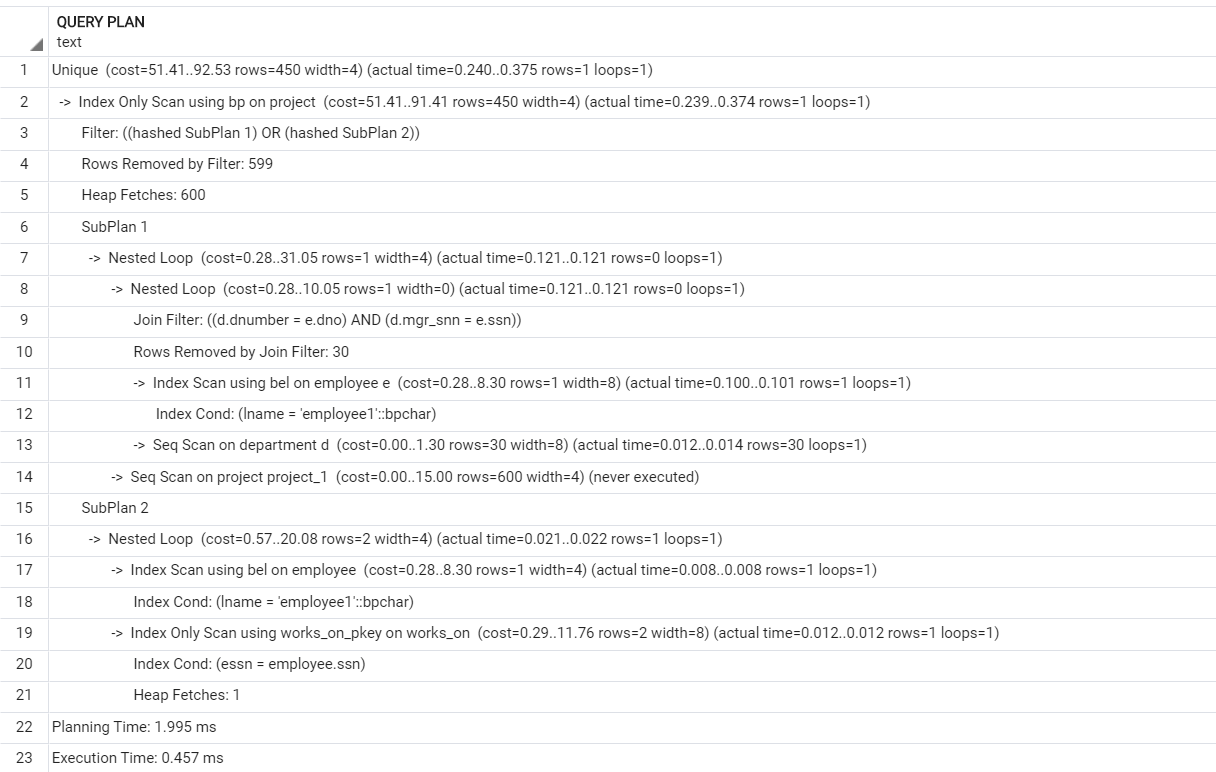
**create index bel on Employee using btree(Lname);**

1st run execution time: 0.457 ms

2nd run execution time:0.879 ms

3rd run execution time : 0.726 ms

Average execution time = 0.687 ms

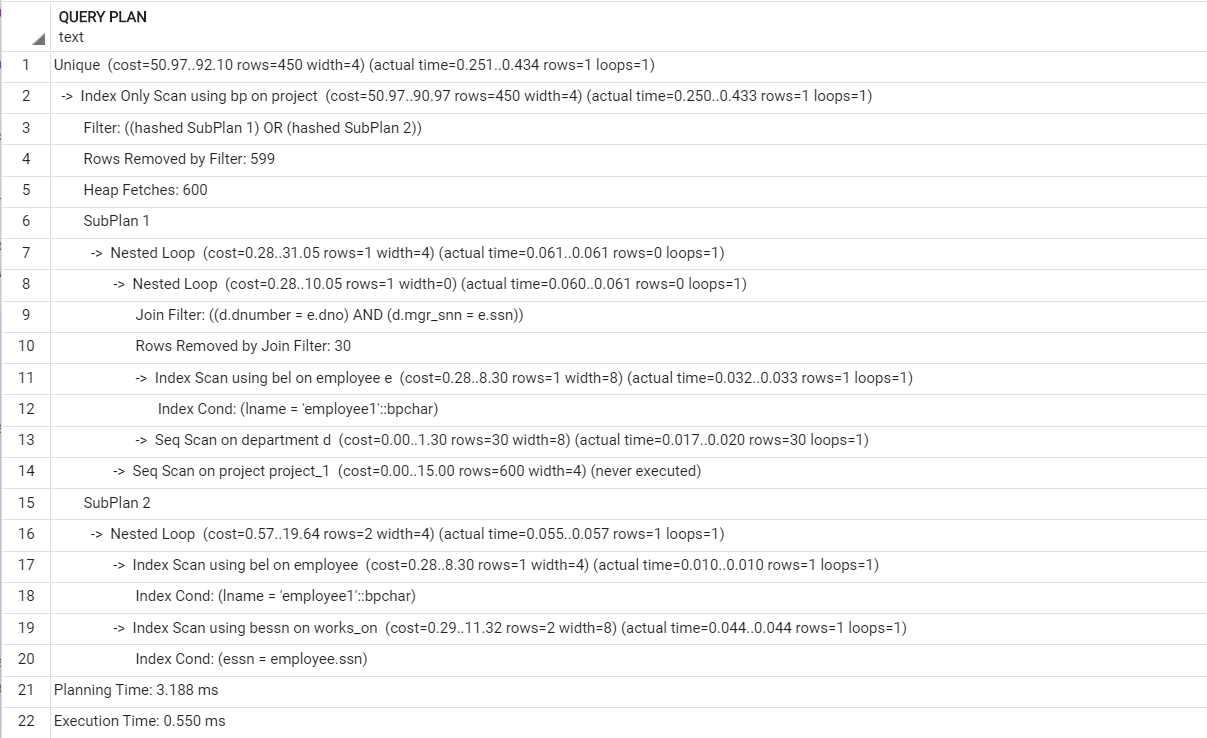


After creating an index on Essn in Table Works\_on

Estimated cost:



**create index bessn on Works\_on using btree(Essn);**



Estimated cost:



1st run execution time: 0.550 ms

2nd run execution time:0.435 ms

3rd run execution time : 0.350 ms

Average execution time = 0.445 ms

Explanation :

The cost has decreased due to the usage of the B-tree index that has time complexity of O(log n) instead of the linear searching of O (n). And the execution time has decreased as well.

drop index be;

drop index bp;

drop index bel;

drop index bessn;

**Bitmap Index :**

**Flags:**

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

set enable\_bitmapscan= on;

set enable\_sort = off;

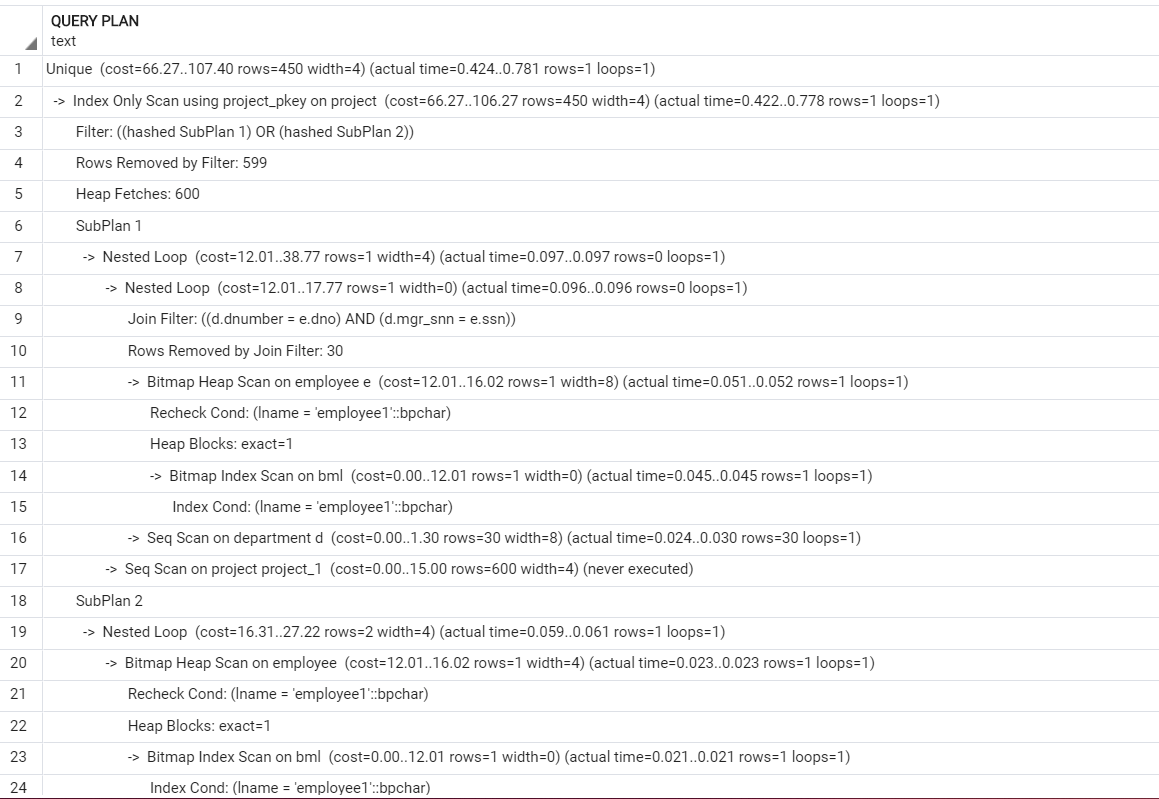
**create extension btree\_gin;**

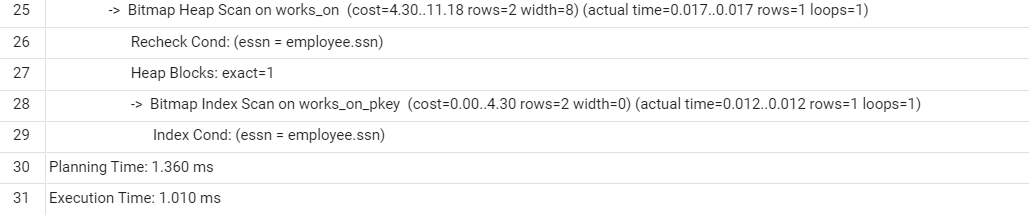
**create index bmE on Employee using gin(ssn,dno);**

**create index bmW on Works\_on using gin(Essn);**

**create index bml on Employee using gin(Lname);**

**create index Pnobitmap on Project using gin(Pnumber);**





Estimated cost:



1st run execution time: 1.010 ms

2nd run execution time: 0.515 ms

3rd run execution time : 0.337 ms

Average execution time = 0.287 ms

***Explanation:***The cost has increased but the execution time has decreased because it uses bit-wise operations to filter the columns, which is really fast.

drop index bmE;

drop index bml;

drop index bmW;

drop index Pnobitmap;

**Hash index:**

Flags:

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

set enable\_bitmapscan= off;

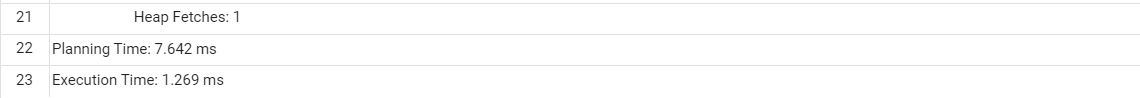
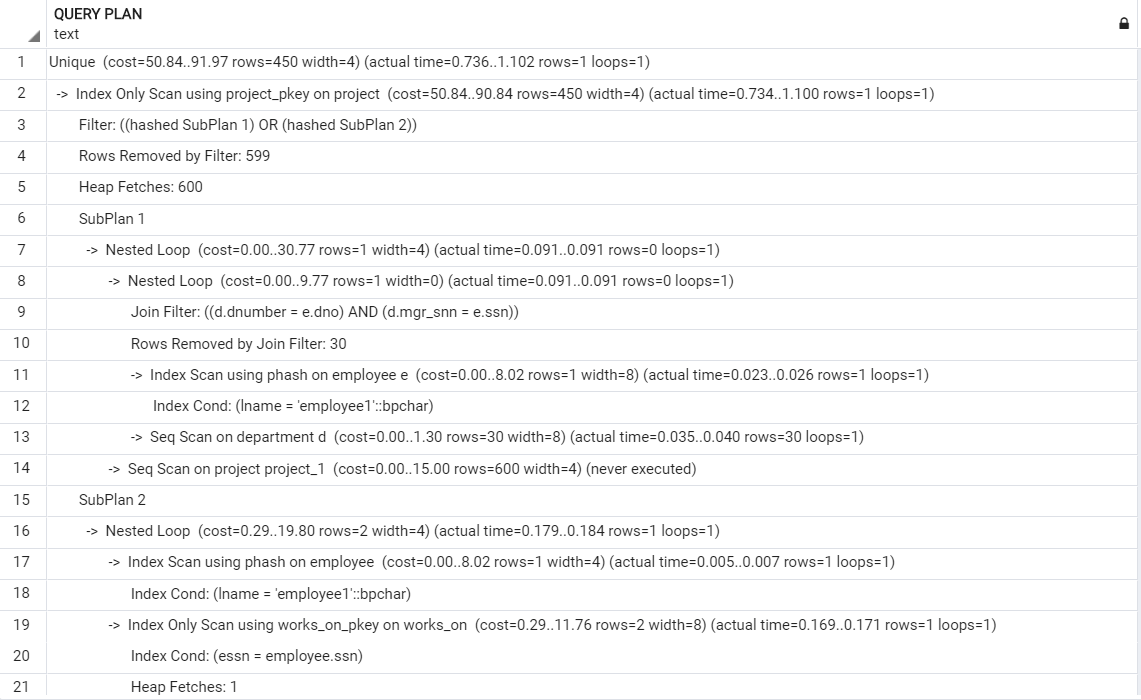
set enable\_sort = off;

set enable\_tidscan = off;

**create index PHash on Employee using hash(Lname);**

**create index EHash on Employee using hash(ssn);**

**create index dHash on Employee using hash(dno);**



Estimated cost:



1st run: 1.269

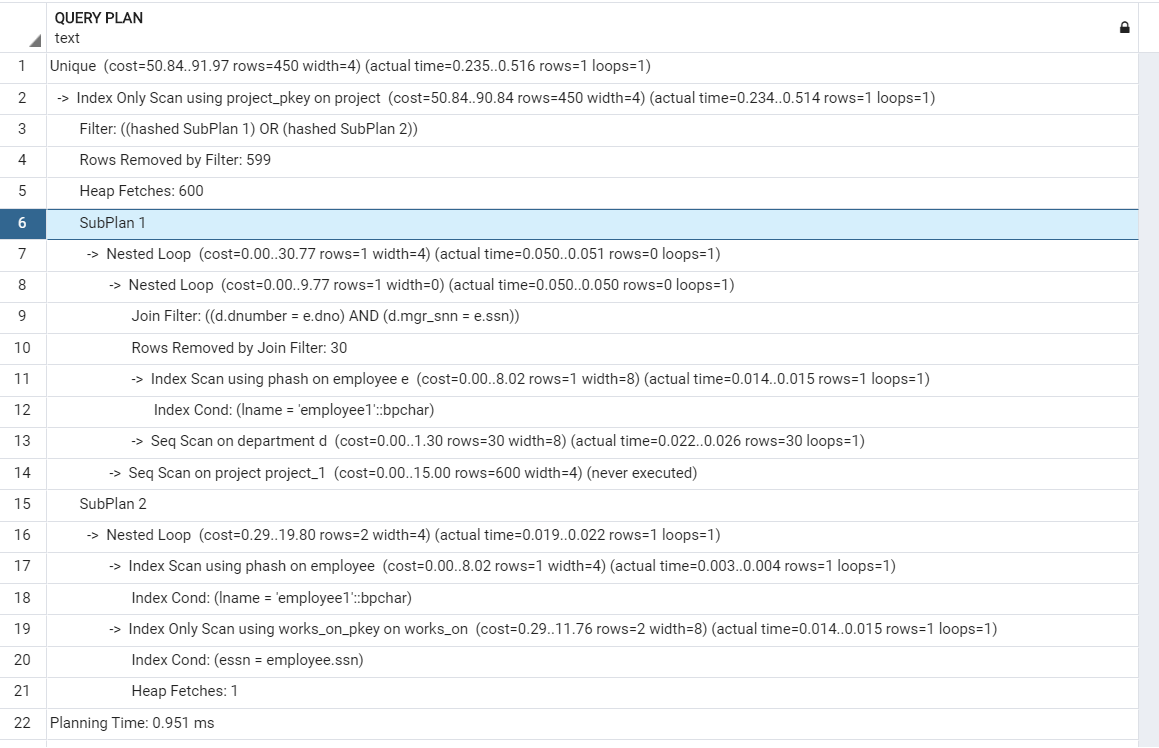
2nd run: 0.829

3rd run: 0.314

Average execution time : 0.804

create index dehash on Department using hash(Mgr\_snn);

create index Pnohash on Project using hash(Pnumber);





Estimated cost:



1st run execution time: 1.269 ms

2nd run execution time: 0.694 ms

3rd run execution time : 0.431 ms

Average execution time = 0.798 ms

***Explanation:***The cost has decreased since we used hash-based index.

drop index PHash;

drop index EHash;

drop index dHash;

drop index dehash;

drop index Pnohash;

***Best Scenario for Query 2:***

Referring to the execution time, and the cost of every case, we can infer that the best scenario is using **Bitmap index**, since it has the least execution time

**Query 3:**

**Without index:**

**Flags:**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = off;

set enable\_indexonlyscan = off;

set enable\_material = off;

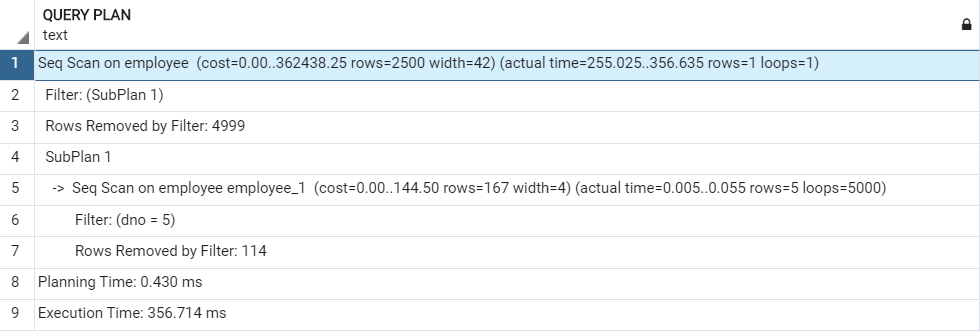
set enable\_mergejoin = off;

set enable\_nestloop = on;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;



Estimated cost:



1st run : execution time : 140.324 ms

2nd run : execution time : 124.582 ms

3rd run : execution time ; 126.978 ms

Average execution time : 130.628ms

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**With Index:**

**B+ tree index:**

Flags:

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

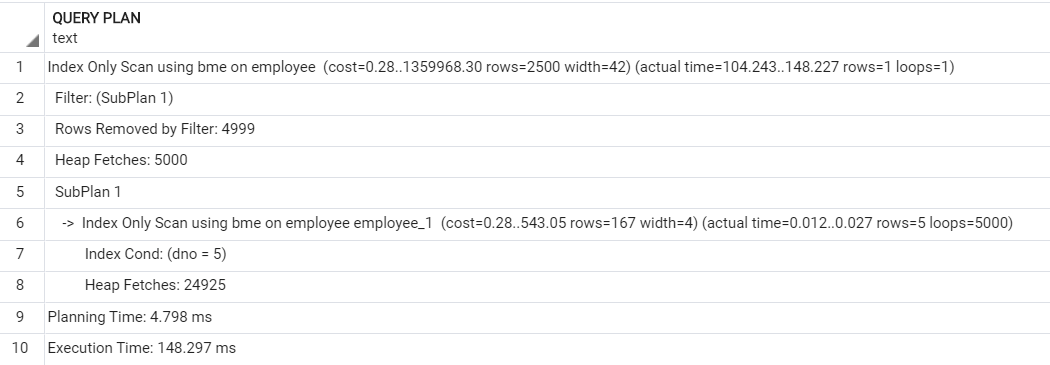
set enable\_nestloop = on;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index bmE on Employee using btree(lname,fname,salary,dno);**



Estimated time:



1st run : execution time : 148.297 ms

2nd run : execution time : 74.242 ms

3rd run : execution time : 78.617 ms

Average execution time : 100.385 ms

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Drop index bmE;

*Explanation:*The cost has decreased due to the usage of the B+ tree index that has time complexity of O (log n) instead of the linear searching of O (n), and we can see that it affects in the execution time.

**With Bitmap index:**

**Flags:**

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

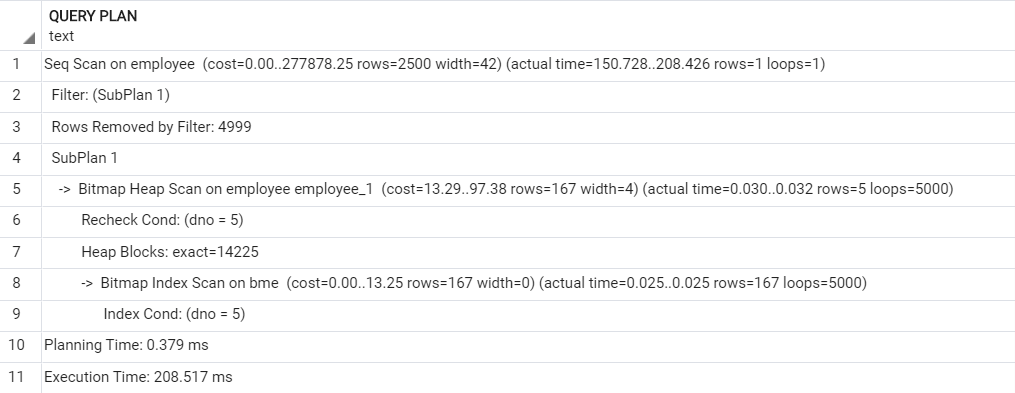
set enable\_bitmapscan= on;

set enable\_sort = off;

set enable\_tidscan = off;

**create extension btree\_gin;**

**create index bmE on Employee using gin(lname,fname,salary,dno);**



Estimated cost:



1st run : execution time : 208.517 ms

2nd run : execution time : 166.775 ms

3rd run : execution time : 156.318 ms

Average execution time : 177.203 ms

*Explanation:* The cost has decreased since we used bitmap index that uses bit-wise operations to filter the columns.

**Hash index :**

**Flags:**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

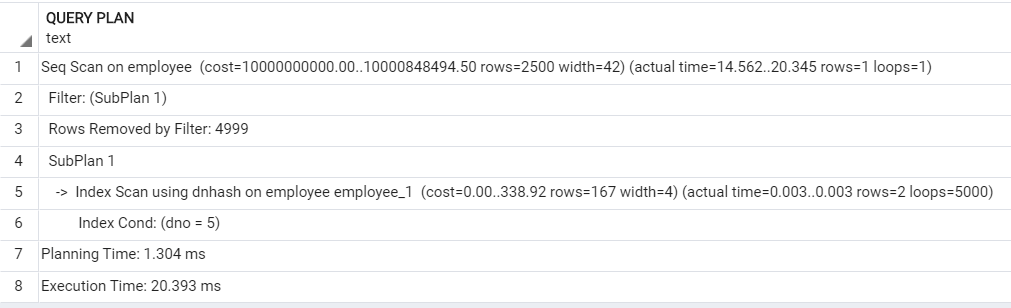
set enable\_nestloop = on;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index dnHash on Employee using hash(dno);**



Estimated cost:



1st Execution time : 20.393 ms

2nd Execution time : 21.759 ms

3rd Execution time : 22.992 ms

Average Execution time : 21.714 ms

*Explanation:* The cost has increased since the seqScan flag is off however the time complexity is reduced due to the use of hash index which has a time complexity of O(1).

***Best Scenario for Query 3:***

Referring to the execution time, and the cost of every case, we can infer that the best scenario is using **Hash Index**, since it has the least execution time.

***Query 4:***

*Flags:*

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = off;

set enable\_indexonlyscan = off;

set enable\_material = off;

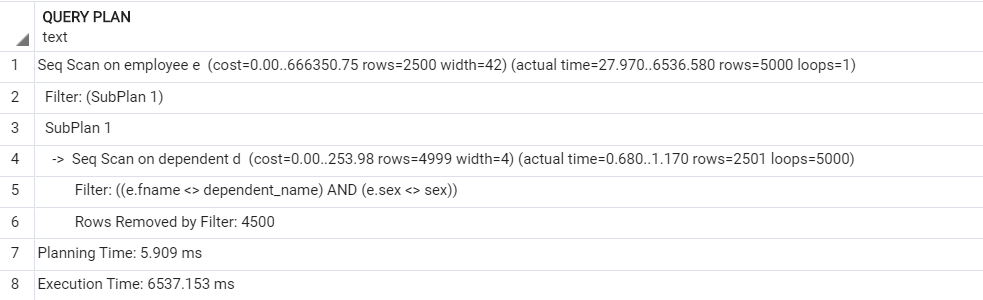
set enable\_mergejoin = off;

set enable\_nestloop = off;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;



Estimated cost:



1st Execution time : 6537.153 ms

2nd Execution time : 6356.290 ms

3rd Execution time : 6339.351 ms

Average Execution time : 6410.931 ms

**With Index :**

**Explanation:**

The behavior of the “!= ” in the query makes it almost impossible to create an index on it, as it will always chooses to do sequential scan and sets the boolean seqScan on automatically although even if it’s set off.

***Best Scenario for Query 4:***

Referring to the previous Explanation the best scenario is without using an index, As the Query does Linear Scan of O(n) complexity due to the “!=” operator.

***Query 5:***

***Without Index:***

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = off;

set enable\_indexonlyscan = off;

set enable\_material = off;

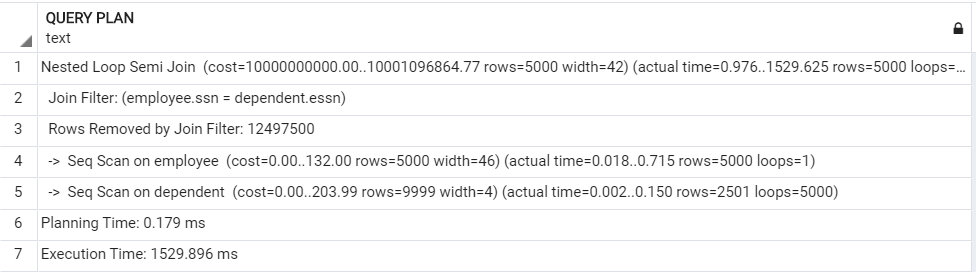
set enable\_mergejoin = off;

set enable\_nestloop = off;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;



Estimated cost:



1st Execution time : 1529.896 ms

2nd Execution time : 1593.303 ms ms

3rd Execution time : 1562.468 ms

Average Execution time : 1561.889 ms

***With Index :***

**B+ tree index :**

**Flags:**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

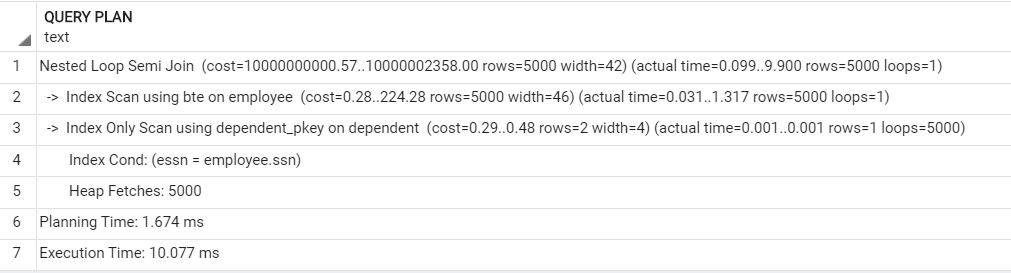
set enable\_nestloop = off;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index bte on Employee using btree(ssn);**



Estimated Cost:



1st Exec time : 10.077 ms

2nd exec time : 10.276 ms

3rd exec time : 14.971 ms

Average Execution time : 11.774 ms

*Explanation:*B-Tree has a time complexity of O (log n), and the column has no duplicates which makes the operation faster and we can notice that the cost has slightly decreased and the execution time also has decreased a lot more than that without index.

drop index bte;

**Bitmap index :**

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

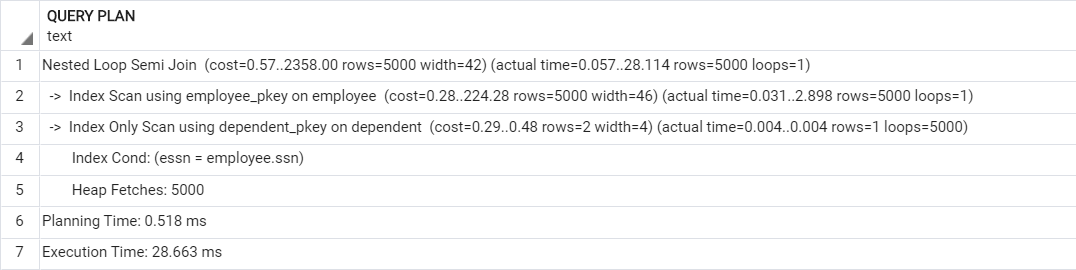
set enable\_bitmapscan= on;

set enable\_sort = off;

create index bmd on dependent using gin(essn);

create index bmE on Employee using gin(ssn,fname);

drop index bmE;



Estimated Cost:



1st Execution Time: 28.663 ms

2nd Execution Time: 35.427 ms

3rd Execution Time: 15.095 ms

Average Execution time: 26.395ms

*Explanation:* The difference is that applying the bitmap index makes the Query faster the cost has reduced a lot and also the execution time.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Hash index :**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

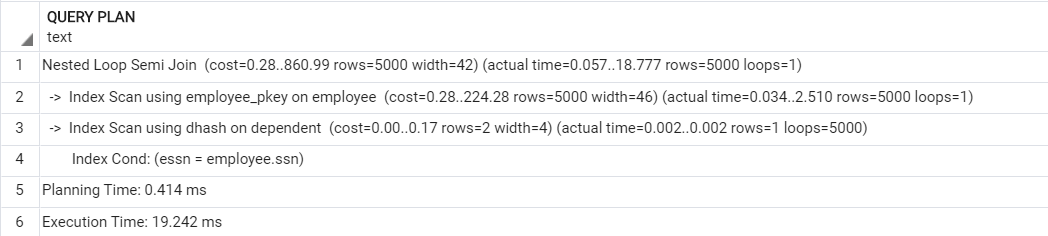
set enable\_nestloop = on;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index dhash on dependent using hash(essn);**



Estimated Cost:



1st Execution Time: 19.242 ms

2nd Execution Time: 7.574 ms

3rd Execution Time: 15.767 ms

Average Execution time: 14.194 ms

drop index dhash;

*Explanation:* The hash-based index , it gives a better performance than without applying any indices, especially with having no duplicates on the column and also it appears in the cost that has reduced and also the execution time.

***Best Scenario for Query 5:***

Referring to the execution time, and the cost of every case, we can infer that the best scenario is using **B-Tree index**, since it has the least execution time.

**Query 6:**

*Without index:*

set enable\_seqscan = on;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = off;

set enable\_indexonlyscan = off;

set enable\_material = off;

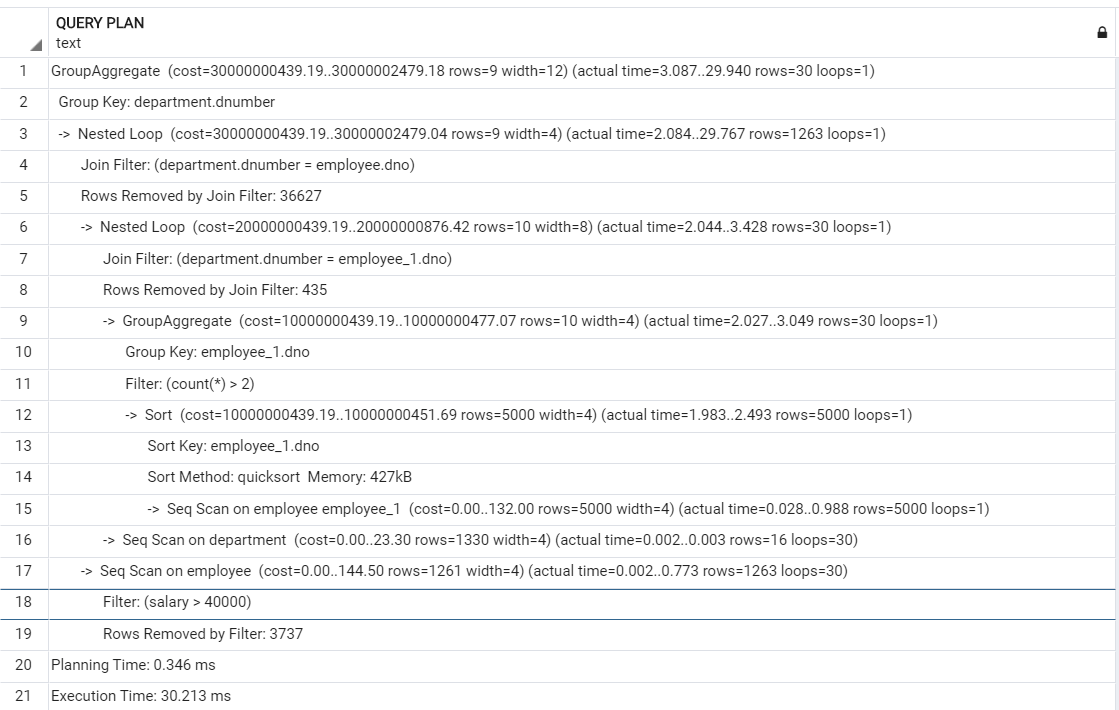
set enable\_mergejoin = off;

set enable\_nestloop = off;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;



Estimated Cost:



1st Execution Time: 30.213 ms

2nd Execution Time: 47.425 ms

3rd Execution Time: 47.810 ms

Average Execution time: 41.816 ms

***With Index:***

**B+ Tree index:**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

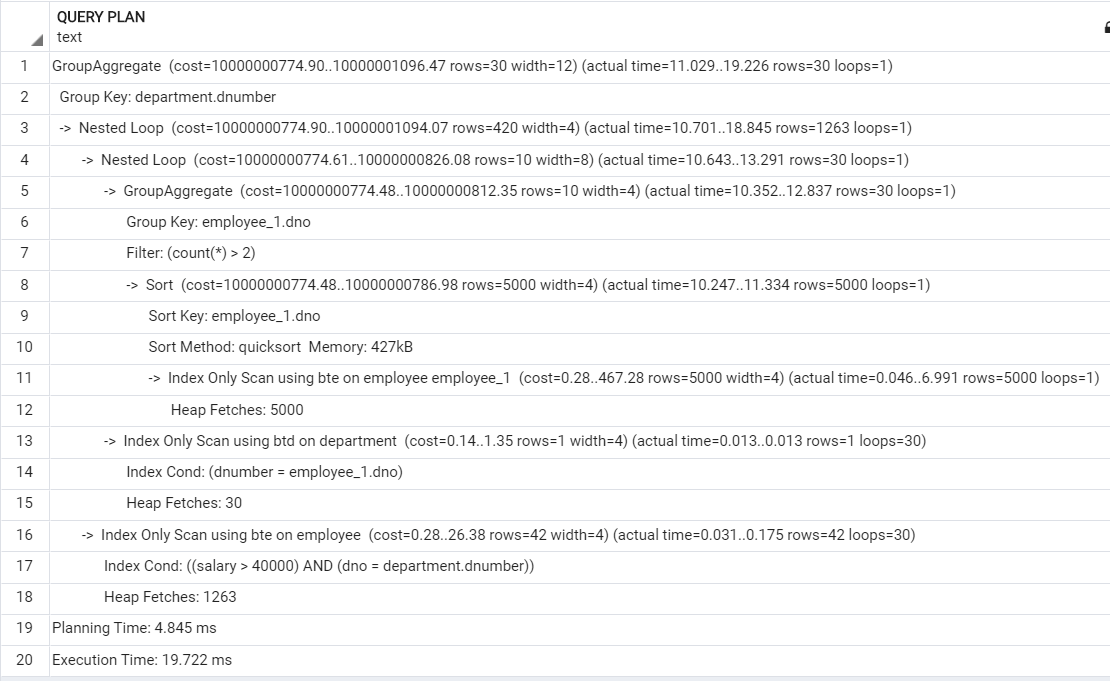
set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index btE on Employee using btree(salary,dno);**

**create index btd on department using btree(dnumber);**



Estimated Cost:



1st Execution Time: 19.722 ms

2nd Execution Time: 20.302 ms

3rd Execution Time: 15.178 ms

Average Execution time: 18.400 ms

drop index btE;

drop index btd;

*Explanation:*The reason of the difference before and after applying the index is that the B-Tree has a time complexity of O (log n), which makes the Query faster and we can notice that in the cost and the execution time that’s reduced a lot.

Bitmap index:

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

set enable\_nestloop = on;

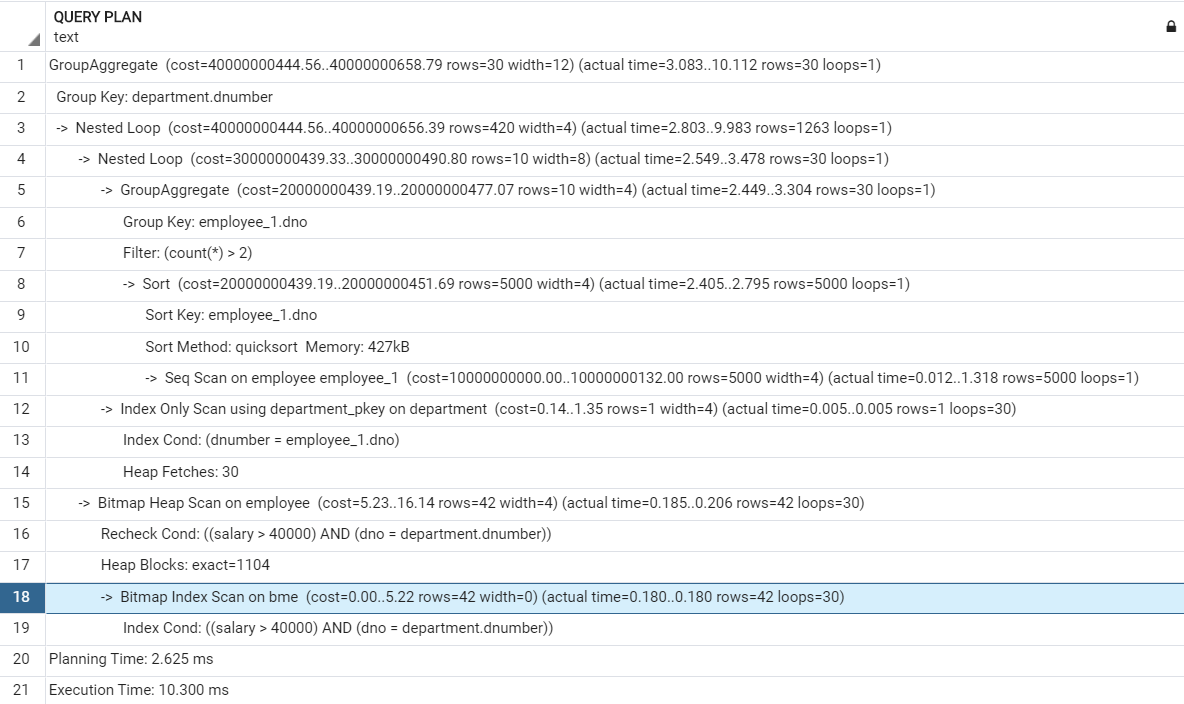
set enable\_bitmapscan= on;

set enable\_sort = off;

set enable\_tidscan = off;

**create index bmE on Employee using gin(salary,dno);**

**create index bmd on department using gin(dnumber);**



Estimated Cost:



1st Execution Time: 10.300 ms

2nd Execution Time: 18.326 ms

3rd Execution Time: 19.594 ms

Average Execution Time: 16.073 ms

*Explanation:* The difference is that applying the bitmap index makes the Query fun faster and we can notice that in the execution time that has reduced slightly however due to turning the seqScan flag off the cost has increased.

**Hash Index:**

set enable\_seqscan = off;

set enable\_gathermerge= off;

set enable\_hashagg = off;

set enable\_hashjoin = off;

set enable\_indexscan = on;

set enable\_indexonlyscan = on;

set enable\_material = off;

set enable\_mergejoin = off;

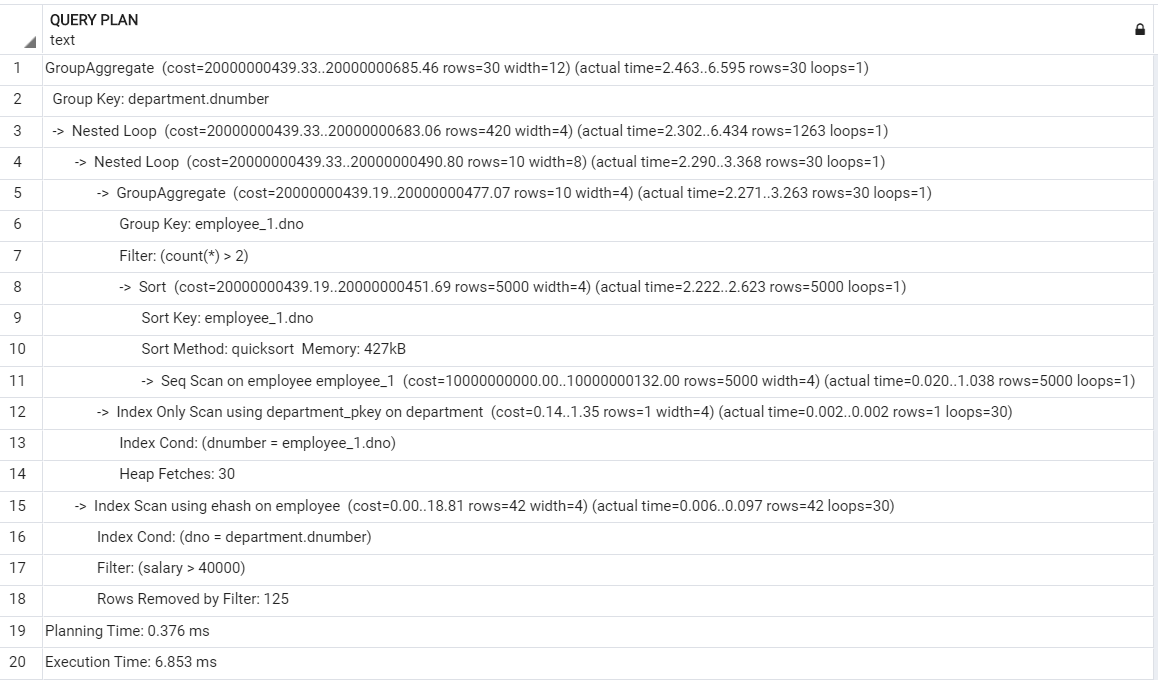
set enable\_nestloop = on;

set enable\_bitmapscan= off;

set enable\_sort = off;

set enable\_tidscan = off;

**create index ehash on Employee using hash(dno);**



Estimated Cost:



1st Execution Time: 6.853 ms

2nd Execution Time: 7.683 ms

3rd Execution Time: 8.710 ms

Average Execution Time: 7.748 ms

*Explanation:* The reason behind this is that the hash-based index runs with time complexity O (1), which is much faster than O (n) without any indices and we can notice that in the execution time.

***Best Scenario for Query 6:***

Referring to the execution time, and the cost of every case, we can infer that the best scenario is using **Hash Index**, since it has the least execution time.