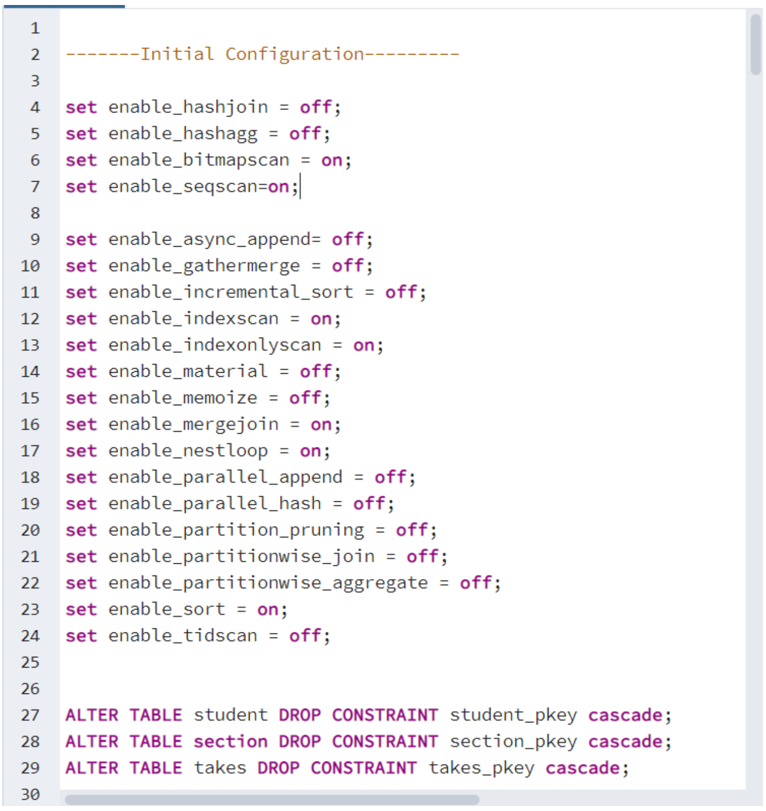
Query 1

- Display a list of all students in the CSEN department, along with the course sections, if any, that they have taken in Semester 1 2019; all course sections from Spring 2019 must be displayed, even if no student from the CSEN department has taken the course section.

Original Query

```
select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester = 1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

Initial Configuration



```

-------Initial Configuration---------

set enable_hashjoin = off;
set enable_hashagg = off;
set enable_bitmapscan = on;
set enable_seqscan=on;

set enable_async_append= off;
set enable_gathermerge = off;
set enable_incremental_sort = off;
set enable_indexscan = on;
set enable_indexonlyscan = on;
set enable_material = off;
set enable_memoize = off;
set enable_mergejoin = on;
set enable_nestloop = on;
set enable_parallel_append = off;
set enable_parallel_hash = off;
set enable_partition_pruning = off;
set enable_partitionwise_join = off;
set enable_partitionwise_aggregate = off;
set enable_sort = on;
set enable_tidscan = off;


ALTER TABLE student DROP CONSTRAINT student_pkey cascade;
ALTER TABLE section DROP CONSTRAINT section_pkey cascade;
ALTER TABLE takes DROP CONSTRAINT takes_pkey cascade;

```

- The 3 primary key constraints from tables student, section and takes were dropped as to be able to test the query cost and plan of any queries with no indexes correctly without the interfence of the default btree index that gets created on the primary keys.

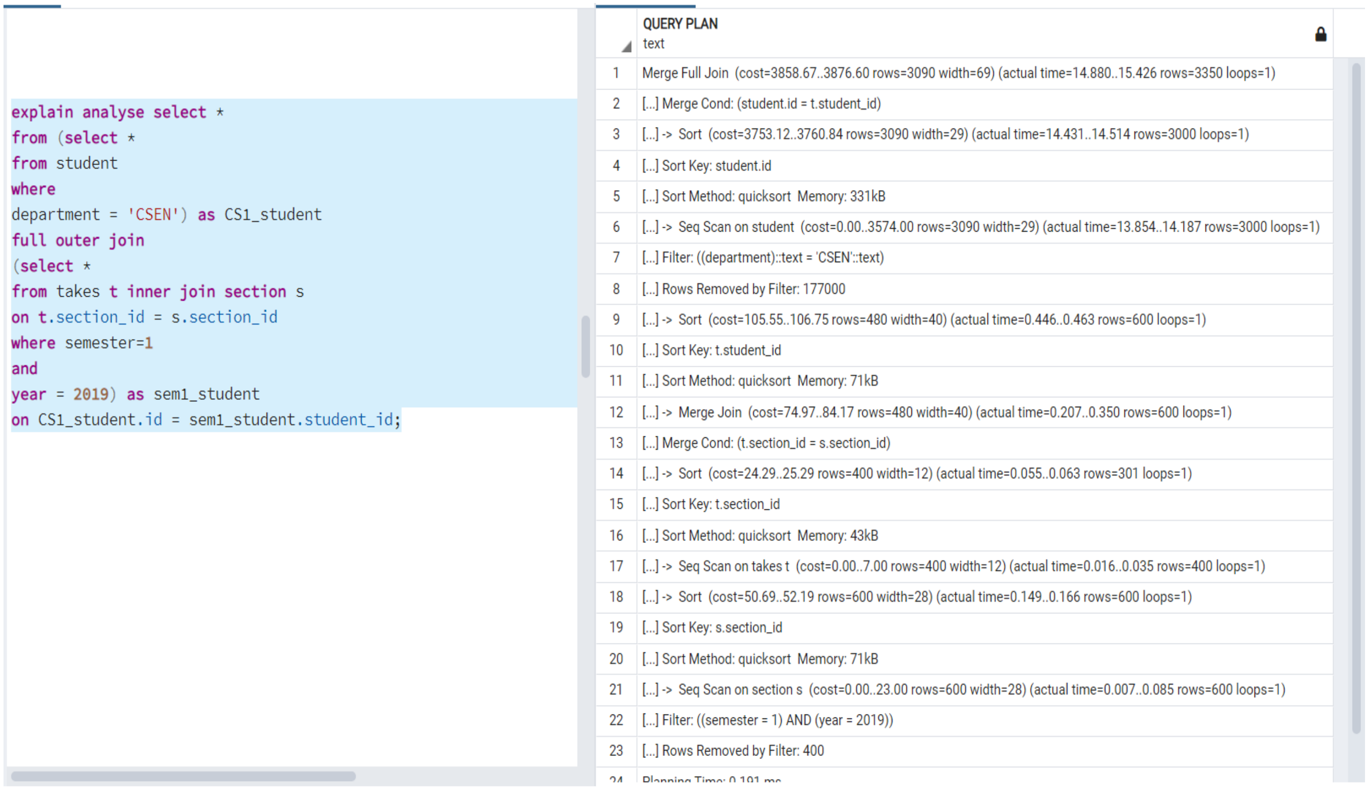
- Bitmapscan was turned on to hint postgres to use index scan and not seq scan. Disabling seqscan wasn't useful in my case as the cost of queries with indices while having seqscan off was approximately 10^6 times worse than the query cost without any index and that may have occured due to my laptop gaming specs and the usage of SSD instead of HDD.
- Also, Hash Join and Hash Aggregate flags were disabled for as after many trials and errors , I've discovered this is best way to show the difference in terms of the cost and to beat the Postgres Query Optimizer Algorithm to be able to show indices effect and cost differences.

Result Set

- 3350 rows

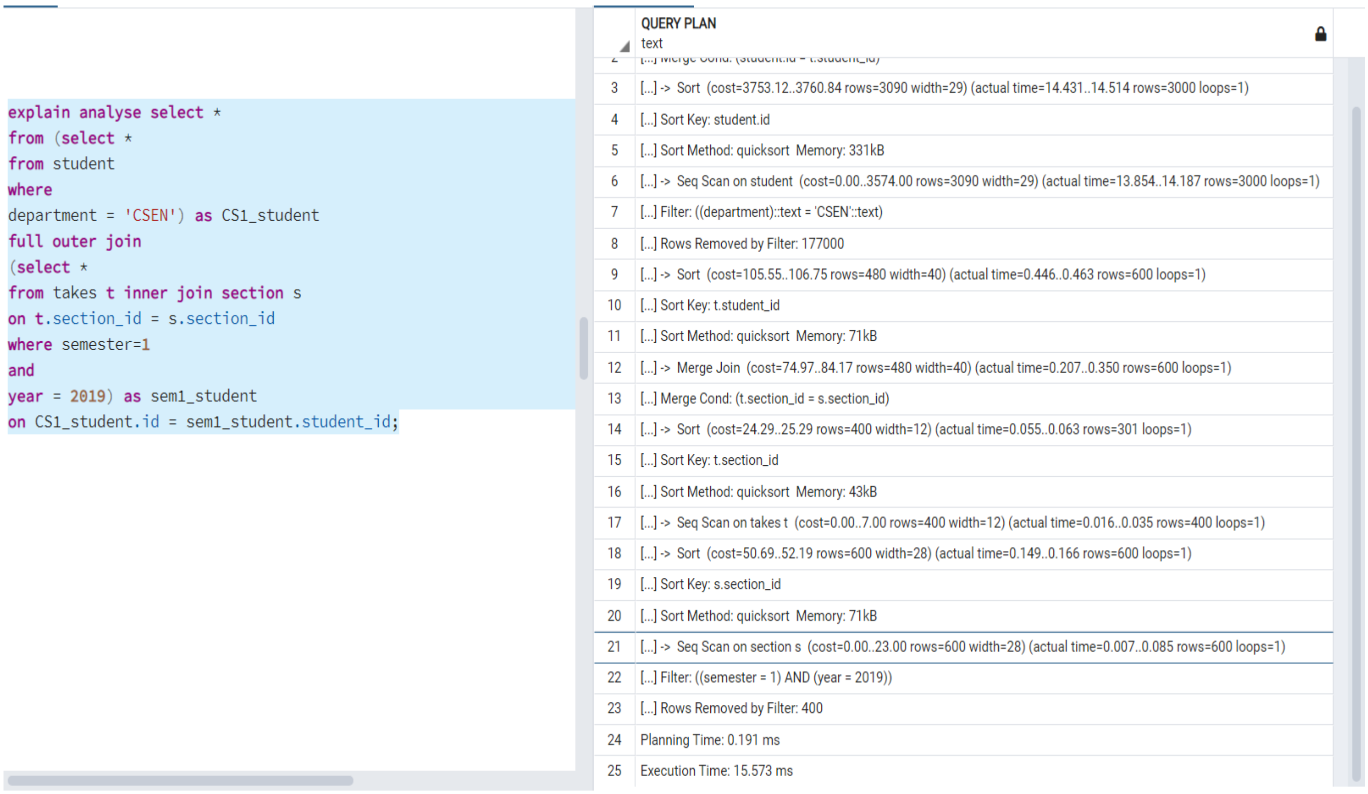
Report

1. given query without an index



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

```
QUERY PLAN
text
Merge Full Join  (cost=3858.67..3876.60 rows=3090 width=69) (actual time=14.880..15.426 rows=3350 loops=1)
[...] Merge Cond: (student.id = t.student_id)
[...] -> Sort  (cost=3753.12..3760.84 rows=3090 width=29) (actual time=14.431..14.514 rows=3000 loops=1)
[...] Sort Key: student.id
[...] Sort Method: quicksort  Memory: 331kB
[...] -> Seq Scan on student  (cost=0.00..3574.00 rows=3090 width=29) (actual time=13.854..14.187 rows=3000 loops=1)
[...] Filter: ((department)::text = 'CSEN'::text)
[...] Rows Removed by Filter: 177000
[...] -> Sort  (cost=105.55..106.75 rows=480 width=40) (actual time=0.446..0.463 rows=600 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] -> Merge Join  (cost=74.97..84.17 rows=480 width=40) (actual time=0.207..0.350 rows=600 loops=1)
[...] Merge Cond: (t.section_id = s.section_id)
[...] -> Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.055..0.063 rows=301 loops=1)
[...] Sort Key: t.section_id
[...] Sort Method: quicksort  Memory: 43kB
[...] -> Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.016..0.035 rows=400 loops=1)
[...] -> Sort  (cost=50.69..52.19 rows=600 width=28) (actual time=0.149..0.166 rows=600 loops=1)
[...] Sort Key: s.section_id
[...] Sort Method: quicksort  Memory: 71kB
[...] -> Seq Scan on section s  (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.085 rows=600 loops=1)
[...] Filter: ((semester = 1) AND (year = 2019))
[...] Rows Removed by Filter: 400
```



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

```
QUERY PLAN
text
[...] -> Sort  (cost=3753.12..3760.84 rows=3090 width=29) (actual time=14.431..14.514 rows=3000 loops=1)
[...] Sort Key: student.id
[...] Sort Method: quicksort  Memory: 331kB
[...] -> Seq Scan on student  (cost=0.00..3574.00 rows=3090 width=29) (actual time=13.854..14.187 rows=3000 loops=1)
[...] Filter: ((department)::text = 'CSEN'::text)
[...] Rows Removed by Filter: 177000
[...] -> Sort  (cost=105.55..106.75 rows=480 width=40) (actual time=0.446..0.463 rows=600 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] -> Merge Join  (cost=74.97..84.17 rows=480 width=40) (actual time=0.207..0.350 rows=600 loops=1)
[...] Merge Cond: (t.section_id = s.section_id)
[...] -> Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.055..0.063 rows=301 loops=1)
[...] Sort Key: t.section_id
[...] Sort Method: quicksort  Memory: 43kB
[...] -> Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.016..0.035 rows=400 loops=1)
[...] -> Sort  (cost=50.69..52.19 rows=600 width=28) (actual time=0.149..0.166 rows=600 loops=1)
[...] Sort Key: s.section_id
[...] Sort Method: quicksort  Memory: 71kB
[...] -> Seq Scan on section s  (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.085 rows=600 loops=1)
[...] Filter: ((semester = 1) AND (year = 2019))
[...] Rows Removed by Filter: 400
```

```
Planning Time: 0.191 ms
Execution Time: 15.573 ms
```

Explanation :

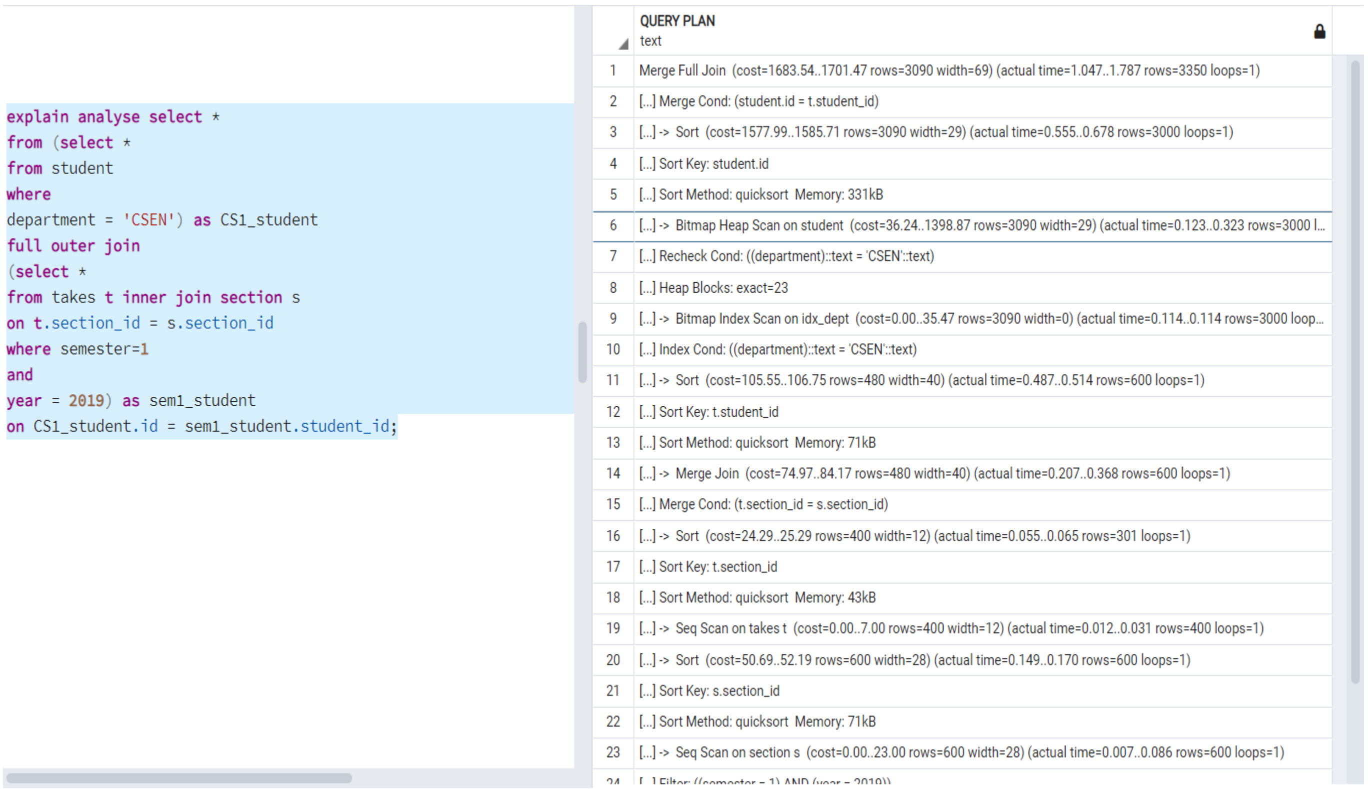
- Metrics :

  Execution Time : 15.573 ms   Total Expected Cost : 3876.60

- The cost reported above was the result of postgres of performing seq scan on all tables (student, section & takes) to get all corresponding and matching tuples from each, in addition to using merge algorithm to perform the join operation between section and takes tables.

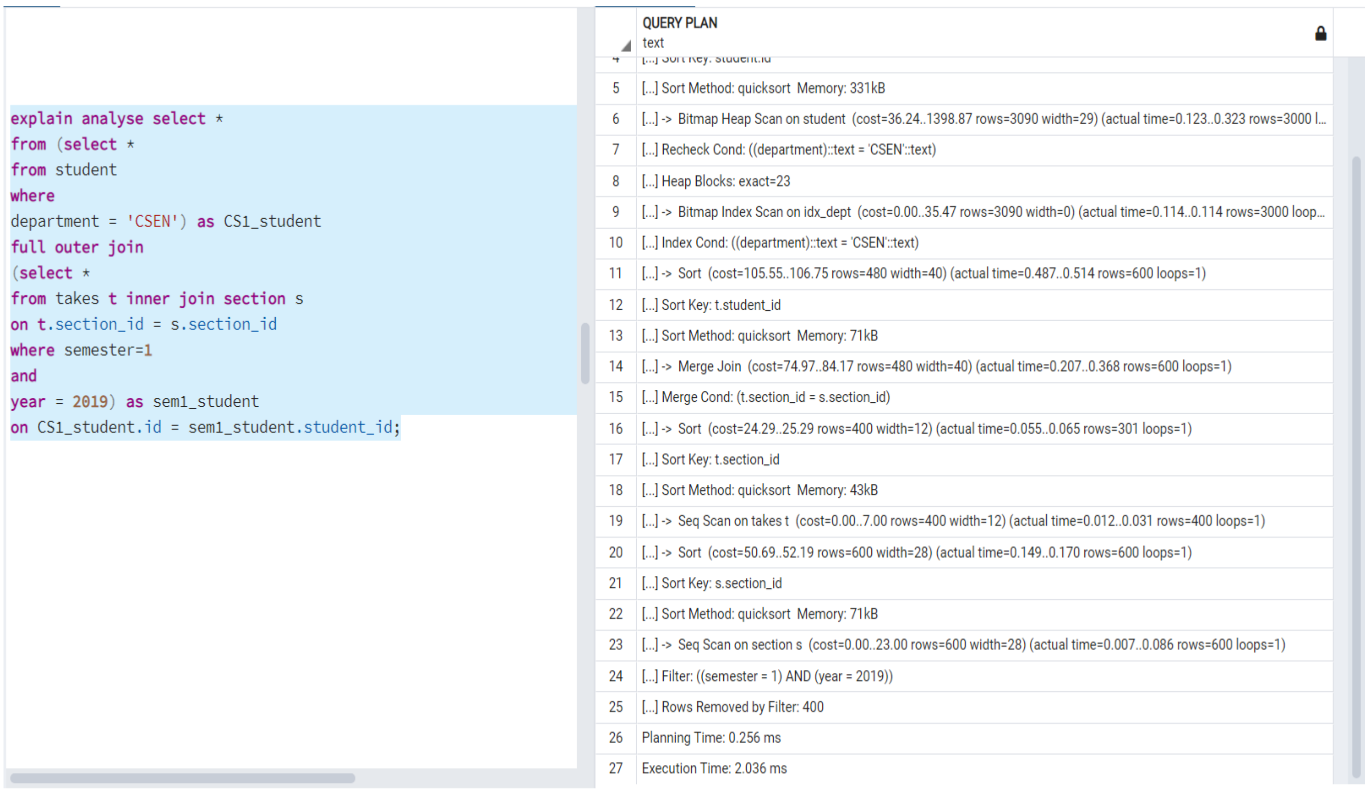
2. given query with B+ trees indices only

```sql
CREATE INDEX idx_dept ON student
USING btree (department);
CREATE INDEX idx_id ON student
USING btree (id);
CREATE INDEX idx_sec ON section
USING btree (section_id);
CREATE INDEX idx_takes ON takes
USING btree (section_id);
CREATE INDEX idx_semester ON section
USING btree (semester);
CREATE INDEX idx_year ON section
USING btree (year);
```



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

```
QUERY PLAN
text
Merge Full Join  (cost=1683.54..1701.47 rows=3090 width=69) (actual time=1.047..1.787 rows=3350 loops=1)
[...] Merge Cond: (student.id = t.student_id)
[...] ->  Sort  (cost=1577.99..1585.71 rows=3090 width=29) (actual time=0.555..0.678 rows=3000 loops=1)
[...] Sort Key: student.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Bitmap Heap Scan on student  (cost=36.24..1398.87 rows=3090 width=29) (actual time=0.123..0.323 rows=3000 l...
[...] Recheck Cond: ((department)::text = 'CSEN'::text)
[...] Heap Blocks: exact=23
[...] ->  Bitmap Index Scan on idx_dept  (cost=0.00..35.47 rows=3090 width=0) (actual time=0.114..0.114 rows=3000 loop...
[...] Index Cond: ((department)::text = 'CSEN'::text)
[...] ->  Sort  (cost=105.55..106.75 rows=480 width=40) (actual time=0.487..0.514 rows=600 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Merge Join  (cost=74.97..84.17 rows=480 width=40) (actual time=0.207..0.368 rows=600 loops=1)
[...] Merge Cond: (t.section_id = s.section_id)
[...] ->  Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.055..0.065 rows=301 loops=1)
[...] Sort Key: t.section_id
[...] Sort Method: quicksort  Memory: 43kB
[...] ->  Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.012..0.031 rows=400 loops=1)
[...] ->  Sort  (cost=50.69..52.19 rows=600 width=28) (actual time=0.149..0.170 rows=600 loops=1)
[...] Sort Key: s.section_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Seq Scan on section s  (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.086 rows=600 loops=1)
[...] Filter: ((semester = 1) AND (year = 2019))
```



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

```
QUERY PLAN
text
[...] Sort Key: student.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Bitmap Heap Scan on student  (cost=36.24..1398.87 rows=3090 width=29) (actual time=0.123..0.323 rows=3000 l...
[...] Recheck Cond: ((department)::text = 'CSEN'::text)
[...] Heap Blocks: exact=23
[...] ->  Bitmap Index Scan on idx_dept  (cost=0.00..35.47 rows=3090 width=0) (actual time=0.114..0.114 rows=3000 loop...
[...] Index Cond: ((department)::text = 'CSEN'::text)
[...] ->  Sort  (cost=105.55..106.75 rows=480 width=40) (actual time=0.487..0.514 rows=600 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Merge Join  (cost=74.97..84.17 rows=480 width=40) (actual time=0.207..0.368 rows=600 loops=1)
[...] Merge Cond: (t.section_id = s.section_id)
[...] ->  Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.055..0.065 rows=301 loops=1)
[...] Sort Key: t.section_id
[...] Sort Method: quicksort  Memory: 43kB
[...] ->  Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.012..0.031 rows=400 loops=1)
[...] ->  Sort  (cost=50.69..52.19 rows=600 width=28) (actual time=0.149..0.170 rows=600 loops=1)
[...] Sort Key: s.section_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Seq Scan on section s  (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.086 rows=600 loops=1)
[...] Filter: ((semester = 1) AND (year = 2019))
[...] Rows Removed by Filter: 400
Planning Time: 0.256 ms
Execution Time: 2.036 ms
```

Explanation :

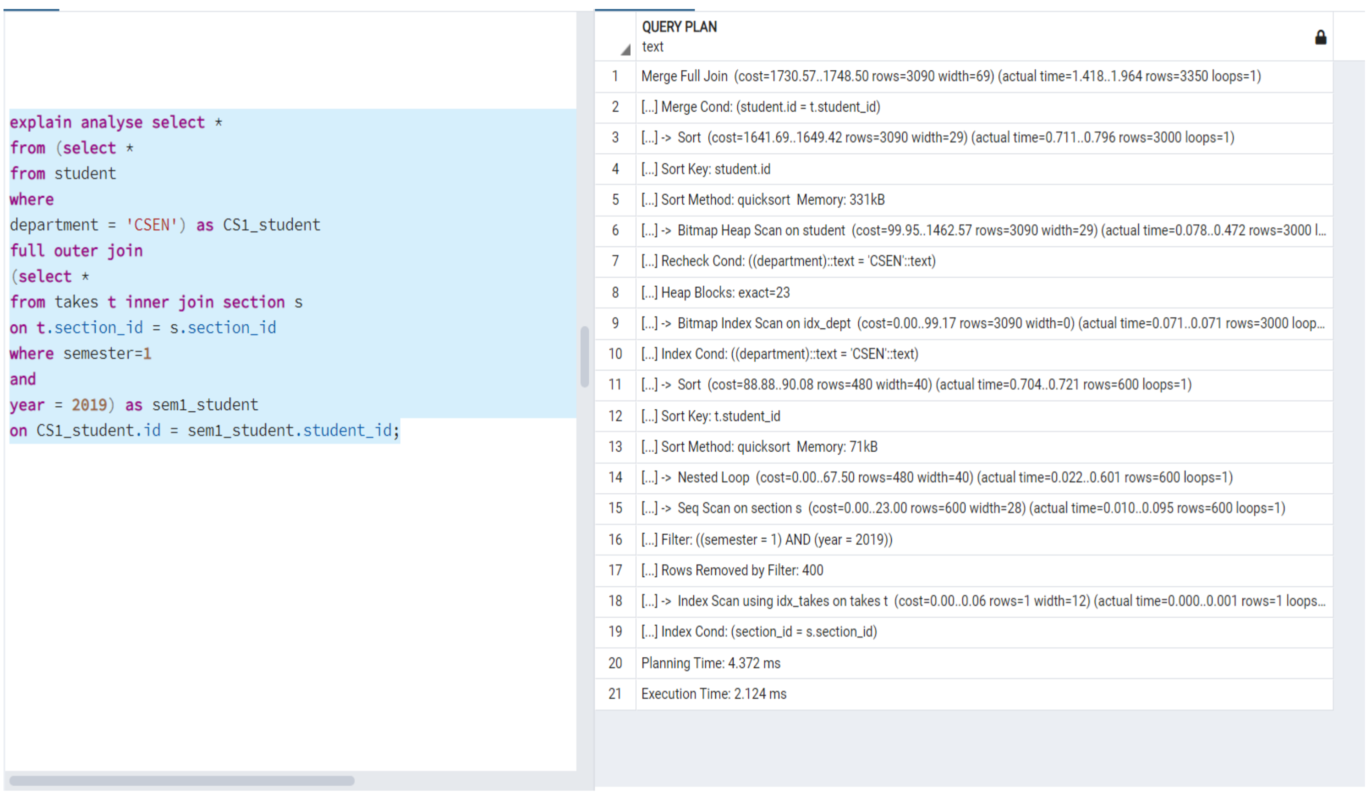
- Metrics :

  Execution Time : 2.036 ms   Total Expected Cost : 1701.47

- Btree indices were created on the following columns: (id , department) in table student, (section_id, semester, year) in table section & (section_id) in table takes.
- We can see here that the cost along with the execution time have reduced significantly compared to no index usage and that's due to the index scan being performed on table student while looking for the CSEN department.
- Searching for the CSEN departments took O(log n) where n is the number of tuples in student table instead of O(n). Table student had 180k rows in it so searching linearly definitely had a set back on performance and searching using Btree helped very much to boost performance and retrieve needed tuples much faster.

3. given query with hash indices only

```sql
CREATE INDEX idx_dept ON student
USING hash (department);
CREATE INDEX idx_id ON student
USING hash (id);
CREATE INDEX idx_sec ON section
USING hash (section_id);
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING hash (semester);
CREATE INDEX idx_year ON section
USING hash (year);
```



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;
```

	QUERY PLAN text
1	Merge Full Join (cost=1730.57..1748.50 rows=3090 width=69) (actual time=1.418..1.964 rows=3350 loops=1)
2	[...] Merge Cond: (student.id = t.student_id)
3	[...] -> Sort (cost=1641.69..1649.42 rows=3090 width=29) (actual time=0.711..0.796 rows=3000 loops=1)
4	[...] Sort Key: student.id
5	[...] Sort Method: quicksort Memory: 331kB
6	[...] -> Bitmap Heap Scan on student (cost=99.95..1462.57 rows=3090 width=29) (actual time=0.078..0.472 rows=3000 l...
7	[...] Recheck Cond: ((department)::text = 'CSEN'::text)
8	[...] Heap Blocks: exact=23
9	[...] -> Bitmap Index Scan on idx_dept (cost=0.00..99.17 rows=3090 width=0) (actual time=0.071..0.071 rows=3000 loop...
10	[...] Index Cond: ((department)::text = 'CSEN'::text)
11	[...] -> Sort (cost=88.88..90.08 rows=480 width=40) (actual time=0.704..0.721 rows=600 loops=1)
12	[...] Sort Key: t.student_id
13	[...] Sort Method: quicksort Memory: 71kB
14	[...] -> Nested Loop (cost=0.00..67.50 rows=480 width=40) (actual time=0.022..0.601 rows=600 loops=1)
15	[...] -> Seq Scan on section s (cost=0.00..23.00 rows=600 width=28) (actual time=0.010..0.095 rows=600 loops=1)
16	[...] Filter: ((semester = 1) AND (year = 2019))
17	[...] Rows Removed by Filter: 400
18	[...] -> Index Scan using idx_takes on takes t (cost=0.00..0.06 rows=1 width=12) (actual time=0.000..0.001 rows=1 loops...
19	[...] Index Cond: (section_id = s.section_id)
20	Planning Time: 4.372 ms
21	Execution Time: 2.124 ms

Explanation :

- Metrics :

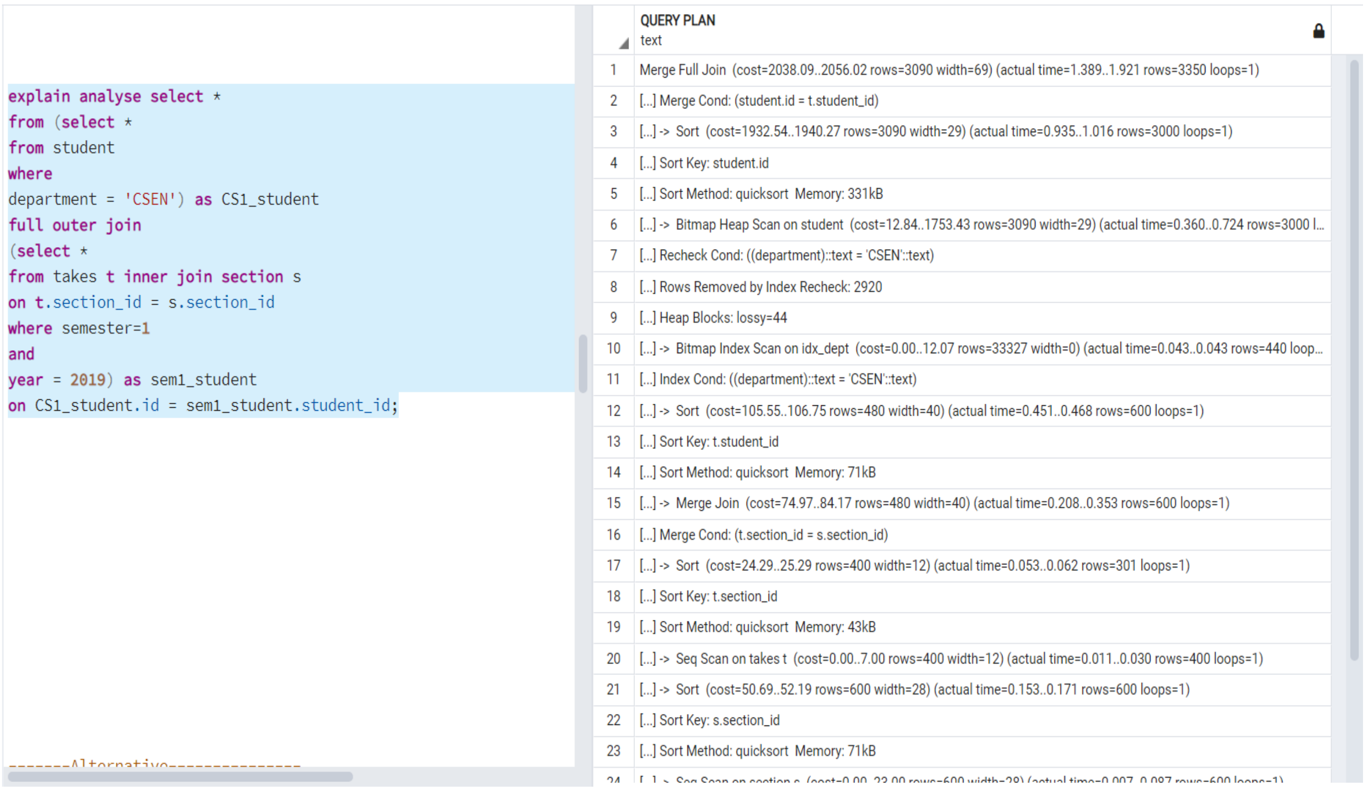
Execution Time : 2.124 ms	**Total Expected Cost : 1748.50**

- Hash indices were created on the following columns: (id , department) in table student, (section_id, semester, year) in table section & (section_id) in table takes.

- We can see here that the cost along with the execution time have reduced significantly compared to no index usage and that's due to the index scan being performed on table student while looking for the CSEN department.
- We can notice that the cost of performing hash indexing was slightly higher than the cost of performing BTree even though hashing is done in O(1) ,which might have been due to the execution of a nested loop while returning the data results at the 14th statement in the image above which wasn't present in Btree's query plan.

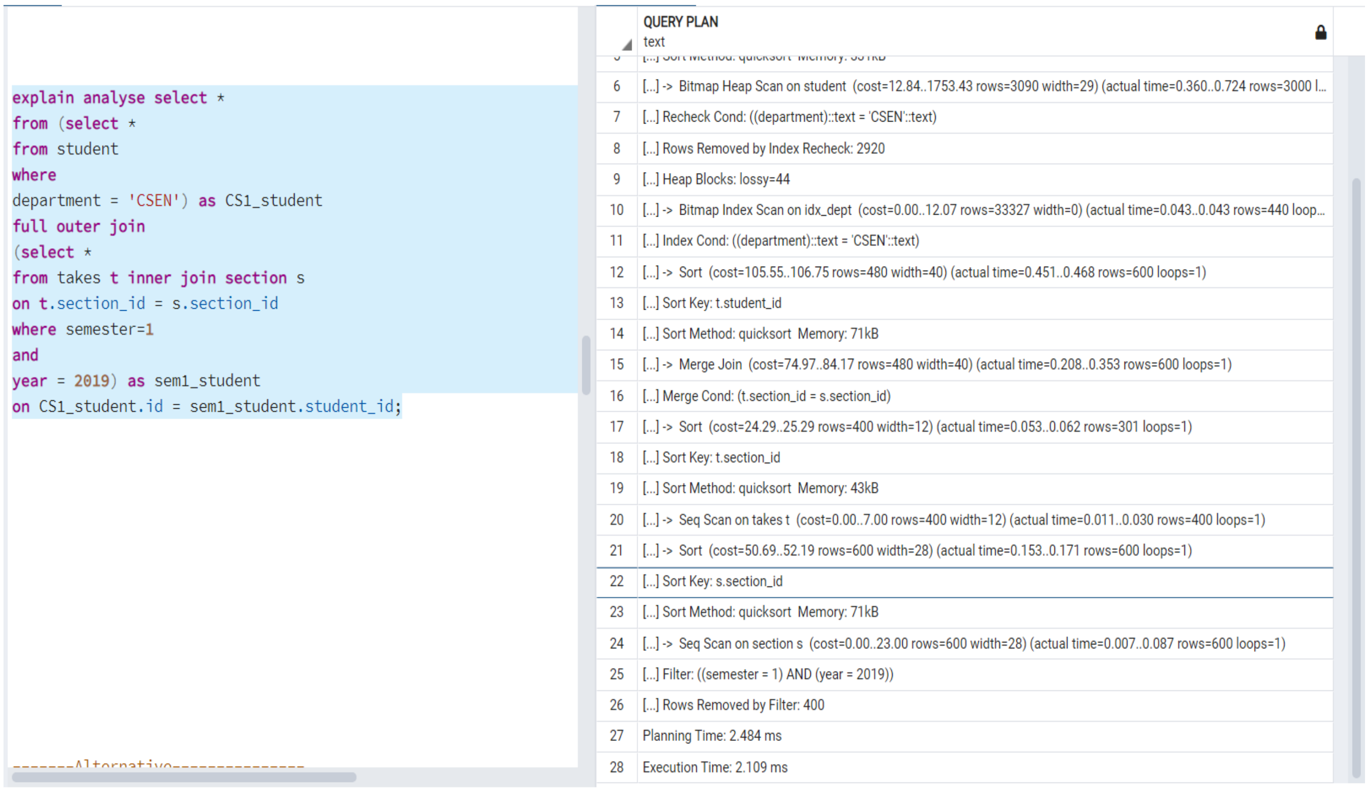
4. given query with BRIN indices only

```
CREATE INDEX idx_dept ON student
USING brin (department);
CREATE INDEX idx_semester ON section
USING brin (semester);
```



```
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;



--------Alternative----------------
```

```
QUERY PLAN
text
1   Merge Full Join  (cost=2038.09..2056.02 rows=3090 width=69) (actual time=1.389..1.921 rows=3350 loops=1)
2   [...] Merge Cond: (student.id = t.student_id)
3   [...] ->  Sort  (cost=1932.54..1940.27 rows=3090 width=29) (actual time=0.935..1.016 rows=3000 loops=1)
4   [...] Sort Key: student.id
5   [...] Sort Method: quicksort  Memory: 331kB
6   [...] ->  Bitmap Heap Scan on student  (cost=12.84..1753.43 rows=3090 width=29) (actual time=0.360..0.724 rows=3000 l...
7   [...] Recheck Cond: ((department)::text = 'CSEN'::text)
8   [...] Rows Removed by Index Recheck: 2920
9   [...] Heap Blocks: lossy=44
10  [...] ->  Bitmap Index Scan on idx_dept  (cost=0.00..12.07 rows=33327 width=0) (actual time=0.043..0.043 rows=440 loop...
11  [...] Index Cond: ((department)::text = 'CSEN'::text)
12  [...] ->  Sort  (cost=105.55..106.75 rows=480 width=40) (actual time=0.451..0.468 rows=600 loops=1)
13  [...] Sort Key: t.student_id
14  [...] Sort Method: quicksort  Memory: 71kB
15  [...] ->  Merge Join  (cost=74.97..84.17 rows=480 width=40) (actual time=0.208..0.353 rows=600 loops=1)
16  [...] Merge Cond: (t.section_id = s.section_id)
17  [...] ->  Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.053..0.062 rows=301 loops=1)
18  [...] Sort Key: t.section_id
19  [...] Sort Method: quicksort  Memory: 43kB
20  [...] ->  Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.011..0.030 rows=400 loops=1)
21  [...] ->  Sort  (cost=50.69..52.19 rows=600 width=28) (actual time=0.153..0.171 rows=600 loops=1)
22  [...] Sort Key: s.section_id
23  [...] Sort Method: quicksort  Memory: 71kB
24  [...] ->  Seq Scan on section s  (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.087 rows=600 loops=1)
```



```
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;



--------Alternative---------------
```

	QUERY PLAN text
6	[...] -> Bitmap Heap Scan on student (cost=12.84..1753.43 rows=3090 width=29) (actual time=0.360..0.724 rows=3000 l...
7	[...] Recheck Cond: ((department)::text = 'CSEN'::text)
8	[...] Rows Removed by Index Recheck: 2920
9	[...] Heap Blocks: lossy=44
10	[...] -> Bitmap Index Scan on idx_dept (cost=0.00..12.07 rows=33327 width=0) (actual time=0.043..0.043 rows=440 loop...
11	[...] Index Cond: ((department)::text = 'CSEN'::text)
12	[...] -> Sort (cost=105.55..106.75 rows=480 width=40) (actual time=0.451..0.468 rows=600 loops=1)
13	[...] Sort Key: t.student_id
14	[...] Sort Method: quicksort Memory: 71kB
15	[...] -> Merge Join (cost=74.97..84.17 rows=480 width=40) (actual time=0.208..0.353 rows=600 loops=1)
16	[...] Merge Cond: (t.section_id = s.section_id)
17	[...] -> Sort (cost=24.29..25.29 rows=400 width=12) (actual time=0.053..0.062 rows=301 loops=1)
18	[...] Sort Key: t.section_id
19	[...] Sort Method: quicksort Memory: 43kB
20	[...] -> Seq Scan on takes t (cost=0.00..7.00 rows=400 width=12) (actual time=0.011..0.030 rows=400 loops=1)
21	[...] -> Sort (cost=50.69..52.19 rows=600 width=28) (actual time=0.153..0.171 rows=600 loops=1)
22	[...] Sort Key: s.section_id
23	[...] Sort Method: quicksort Memory: 71kB
24	[...] -> Seq Scan on section s (cost=0.00..23.00 rows=600 width=28) (actual time=0.007..0.087 rows=600 loops=1)
25	[...] Filter: ((semester = 1) AND (year = 2019))
26	[...] Rows Removed by Filter: 400
27	Planning Time: 2.484 ms
28	Execution Time: 2.109 ms

Explanation :

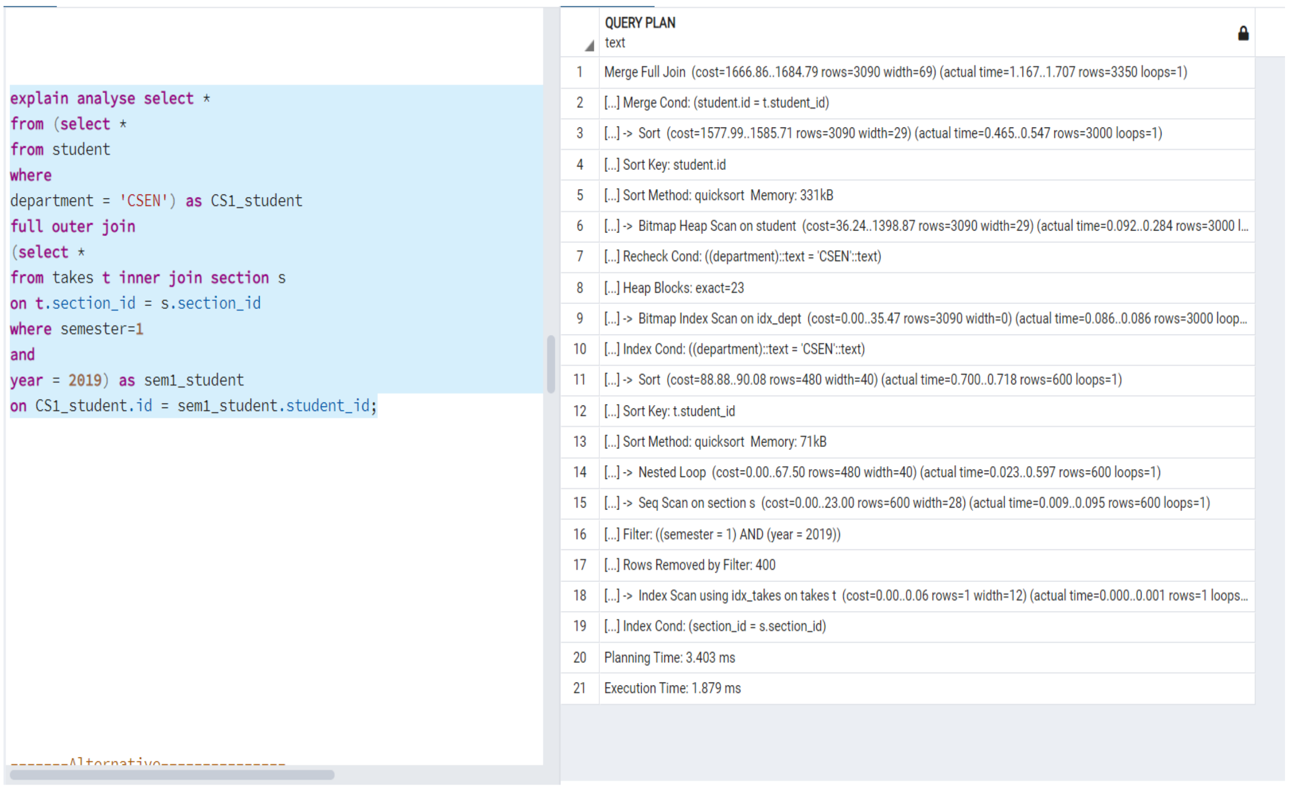
- Metrics :

Execution Time : 2.109 ms   Total Expected Cost : 2056.02

- BRIN indices were created on the following columns: semester in table section and department in table student.
- The cost here is also much smaller than the query without any index as the search time has been reduced due to having not much unique values in both columns since there exists only 60 distinct departments in table student and only 2 values for semester either 1 or 2 and data is sorted on both columns by my insertion code so BRIN knows exactly when to start and end by the min and max that it stores of each page.

5. given query with mixed indices (any mix of your choice).

```sql
CREATE INDEX idx_dept ON student
USING btree (department);
CREATE INDEX idx_id ON student
USING btree (id);
CREATE INDEX idx_sec ON section
USING hash (section_id);
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING brin (semester);
CREATE INDEX idx_year ON section
USING hash (year);
```



```sql
explain analyse select *
from (select *
from student
where
department = 'CSEN') as CS1_student
full outer join
(select *
from takes t inner join section s
on t.section_id = s.section_id
where semester=1
and
year = 2019) as sem1_student
on CS1_student.id = sem1_student.student_id;


-------Alternative---------------
```

	QUERY PLAN text
1	Merge Full Join (cost=1666.86..1684.79 rows=3090 width=69) (actual time=1.167..1.707 rows=3350 loops=1)
2	[...] Merge Cond: (student.id = t.student_id)
3	[...] -> Sort (cost=1577.99..1585.71 rows=3090 width=29) (actual time=0.465..0.547 rows=3000 loops=1)
4	[...] Sort Key: student.id
5	[...] Sort Method: quicksort Memory: 331kB
6	[...] -> Bitmap Heap Scan on student (cost=36.24..1398.87 rows=3090 width=29) (actual time=0.092..0.284 rows=3000 l...
7	[...] Recheck Cond: ((department)::text = 'CSEN'::text)
8	[...] Heap Blocks: exact=23
9	[...] -> Bitmap Index Scan on idx_dept (cost=0.00..35.47 rows=3090 width=0) (actual time=0.086..0.086 rows=3000 loop...
10	[...] Index Cond: ((department)::text = 'CSEN'::text)
11	[...] -> Sort (cost=88.88..90.08 rows=480 width=40) (actual time=0.700..0.718 rows=600 loops=1)
12	[...] Sort Key: t.student_id
13	[...] Sort Method: quicksort Memory: 71kB
14	[...] -> Nested Loop (cost=0.00..67.50 rows=480 width=40) (actual time=0.023..0.597 rows=600 loops=1)
15	[...] -> Seq Scan on section s (cost=0.00..23.00 rows=600 width=28) (actual time=0.009..0.095 rows=600 loops=1)
16	[...] Filter: ((semester = 1) AND (year = 2019))
17	[...] Rows Removed by Filter: 400
18	[...] -> Index Scan using idx_takes on takes t (cost=0.00..0.06 rows=1 width=12) (actual time=0.000..0.001 rows=1 loops...
19	[...] Index Cond: (section_id = s.section_id)
20	Planning Time: 3.403 ms
21	Execution Time: 1.879 ms

Explanation :

- Metrics :

Execution Time : 1.879 ms   Total Expected Cost : 1684.79

- Btree was created on the following columns: (department, id) in table student. Hash indices were created on the following columns: (section_id, year) in table section and (section_id) in table takes. Brin index was created on column semester in section table.

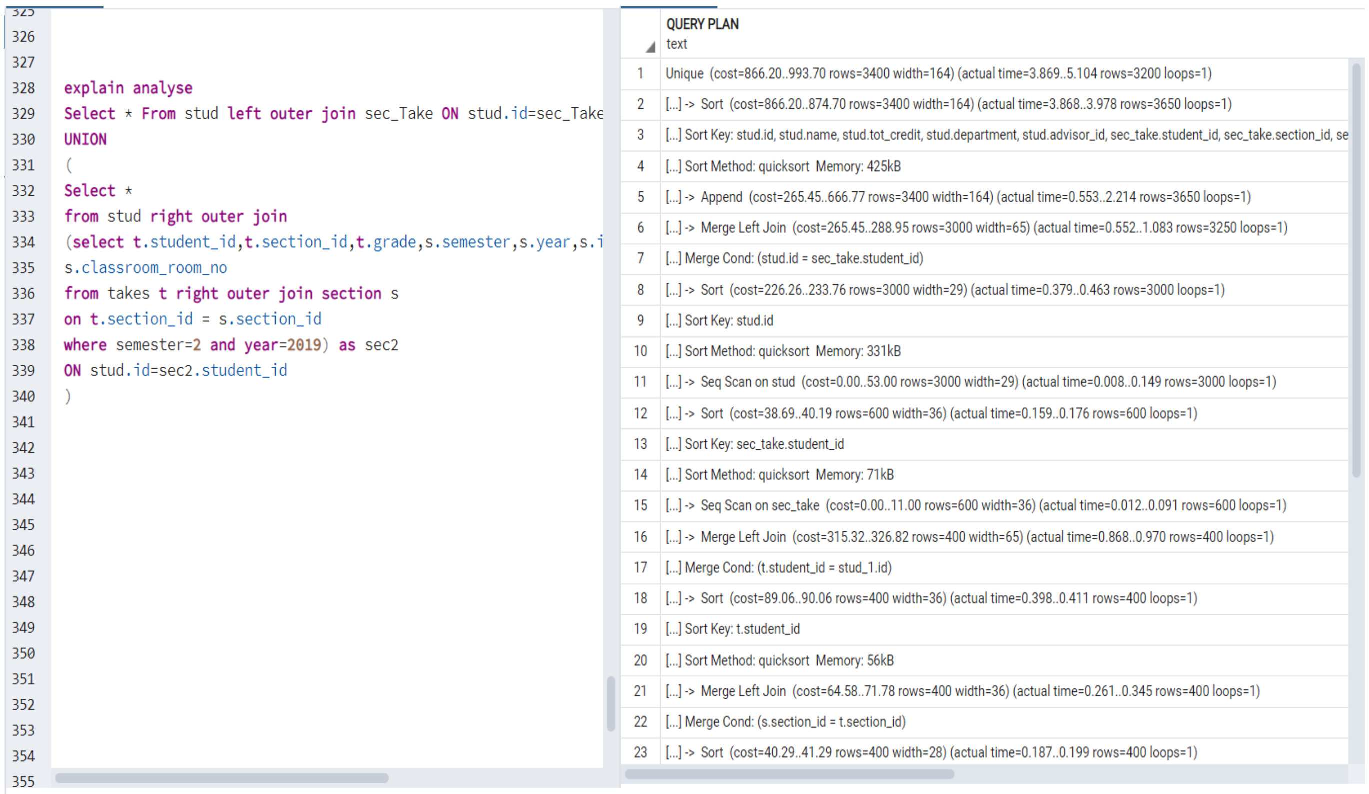
- It's noticed here that the execution time and estimated cost is the BEST so far among all other previous queries and that's what should be expected as taking the best index for each column results in a decent performance boost.

Optimized Query

```
CREATE MATERIALIZED VIEW stud
 AS
 select * from student
 where
 department = 'CSEN'
 WITH DATA;




 create MATERIALIZED VIEW sec_Take
 AS
 select
t.student_id,t.section_id,t.grade,s.semester,s.year,s.instructor_id,s.course_id,s.
classroom_building,
 s.classroom_room_no
 from takes t inner join section s
 on t.section_id = s.section_id
 where s.semester=1 and s.year = 2019
 WITH DATA;




 CREATE MATERIALIZED VIEW sec_Take2
 AS
 select
t.student_id,t.section_id,t.grade,s.semester,s.year,s.instructor_id,s.course_id,s.
classroom_building,
 s.classroom_room_no
 from takes t right outer join section s
 on t.section_id = s.section_id
 where semester=2 and year=2019
 WITH DATA;


 explain analyse
 Select * From stud left outer join sec_Take ON stud.id=sec_Take.student_id
 UNION
 (
 Select *
 from stud right outer join
 (select
t.student_id,t.section_id,t.grade,s.semester,s.year,s.instructor_id,s.course_id,s.
classroom_building,
 s.classroom_room_no
 from takes t right outer join section s
 on t.section_id = s.section_id
 where semester=2 and year=2019) as sec2
 ON stud.id=sec2.student_id
 )
```

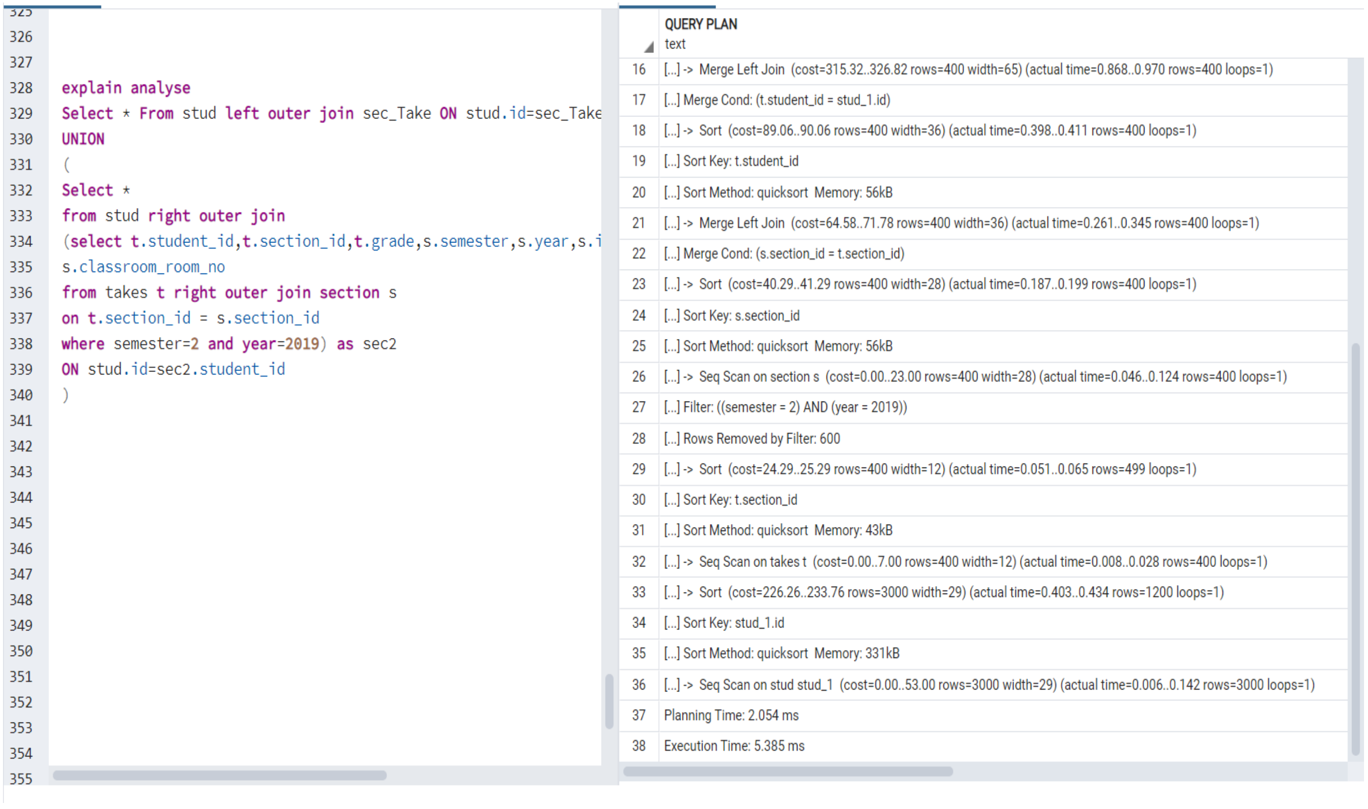
Report

1. given query without an index



```
explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)
```

```
QUERY PLAN
text
1   Unique  (cost=866.20..993.70 rows=3400 width=164) (actual time=3.869..5.104 rows=3200 loops=1)
2   [...] -> Sort  (cost=866.20..874.70 rows=3400 width=164) (actual time=3.868..3.978 rows=3650 loops=1)
3   [...] Sort Key: stud.id, stud.name, stud.tot_credit, stud.department, stud.advisor_id, sec_take.student_id, sec_take.section_id, se
4   [...] Sort Method: quicksort  Memory: 425kB
5   [...] -> Append  (cost=265.45..666.77 rows=3400 width=164) (actual time=0.553..2.214 rows=3650 loops=1)
6   [...] -> Merge Left Join  (cost=265.45..288.95 rows=3000 width=65) (actual time=0.552..1.083 rows=3250 loops=1)
7   [...] Merge Cond: (stud.id = sec_take.student_id)
8   [...] -> Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.379..0.463 rows=3000 loops=1)
9   [...] Sort Key: stud.id
10  [...] Sort Method: quicksort  Memory: 331kB
11  [...] -> Seq Scan on stud  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.008..0.149 rows=3000 loops=1)
12  [...] -> Sort  (cost=38.69..40.19 rows=600 width=36) (actual time=0.159..0.176 rows=600 loops=1)
13  [...] Sort Key: sec_take.student_id
14  [...] Sort Method: quicksort  Memory: 71kB
15  [...] -> Seq Scan on sec_take  (cost=0.00..11.00 rows=600 width=36) (actual time=0.012..0.091 rows=600 loops=1)
16  [...] -> Merge Left Join  (cost=315.32..326.82 rows=400 width=65) (actual time=0.868..0.970 rows=400 loops=1)
17  [...] Merge Cond: (t.student_id = stud_1.id)
18  [...] -> Sort  (cost=89.06..90.06 rows=400 width=36) (actual time=0.398..0.411 rows=400 loops=1)
19  [...] Sort Key: t.student_id
20  [...] Sort Method: quicksort  Memory: 56kB
21  [...] -> Merge Left Join  (cost=64.58..71.78 rows=400 width=36) (actual time=0.261..0.345 rows=400 loops=1)
22  [...] Merge Cond: (s.section_id = t.section_id)
23  [...] -> Sort  (cost=40.29..41.29 rows=400 width=28) (actual time=0.187..0.199 rows=400 loops=1)
```



```
explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)
```

```
QUERY PLAN
text
16  [...] -> Merge Left Join  (cost=315.32..326.82 rows=400 width=65) (actual time=0.868..0.970 rows=400 loops=1)
17  [...] Merge Cond: (t.student_id = stud_1.id)
18  [...] -> Sort  (cost=89.06..90.06 rows=400 width=36) (actual time=0.398..0.411 rows=400 loops=1)
19  [...] Sort Key: t.student_id
20  [...] Sort Method: quicksort  Memory: 56kB
21  [...] -> Merge Left Join  (cost=64.58..71.78 rows=400 width=36) (actual time=0.261..0.345 rows=400 loops=1)
22  [...] Merge Cond: (s.section_id = t.section_id)
23  [...] -> Sort  (cost=40.29..41.29 rows=400 width=28) (actual time=0.187..0.199 rows=400 loops=1)
```

```
24  [...] Sort Key: s.section_id
25  [...] Sort Method: quicksort  Memory: 56kB
26  [...] -> Seq Scan on section s  (cost=0.00..23.00 rows=400 width=28) (actual time=0.046..0.124 rows=400 loops=1)
27  [...] Filter: ((semester = 2) AND (year = 2019))
28  [...] Rows Removed by Filter: 600
29  [...] -> Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.051..0.065 rows=499 loops=1)
30  [...] Sort Key: t.section_id
31  [...] Sort Method: quicksort  Memory: 43kB
32  [...] -> Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.008..0.028 rows=400 loops=1)
33  [...] -> Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.403..0.434 rows=1200 loops=1)
34  [...] Sort Key: stud_1.id
35  [...] Sort Method: quicksort  Memory: 331kB
36  [...] -> Seq Scan on stud stud_1  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.006..0.142 rows=3000 loops=1)
37  Planning Time: 2.054 ms
38  Execution Time: 5.385 ms
```

Explanation :

- Metrics :

Execution Time : 5.385 ms   Total Expected Cost : 993.70

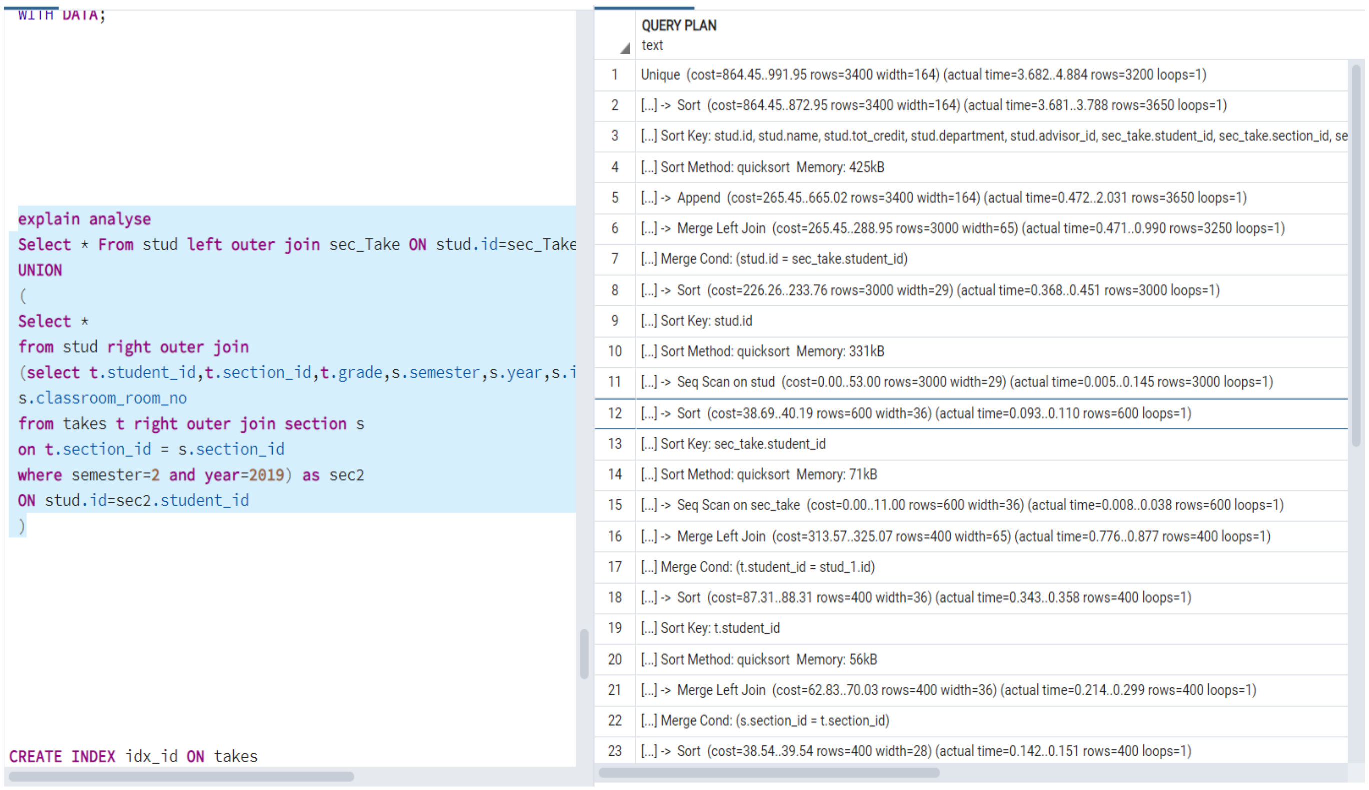
- Please note that the original query was missing course sections from spring 2019, which I've added and that resulted in a bigger result set of course than the original, which makes that comparsion a little

Unfair. Yet, I've managed even with the increase in the result set to reduce the cost size compared to the original query with no indexing.

- Cost have dropped from 3876 to 993 by the use of materialized views whose costs aren't added during the execution of the query as they are only created once and become ready afterwards to be used in the queries, so many of the filtrations and joins were made in them which made the original query much more efficient as the tables it needs are already present.
- The same configurations as the original query still hold.

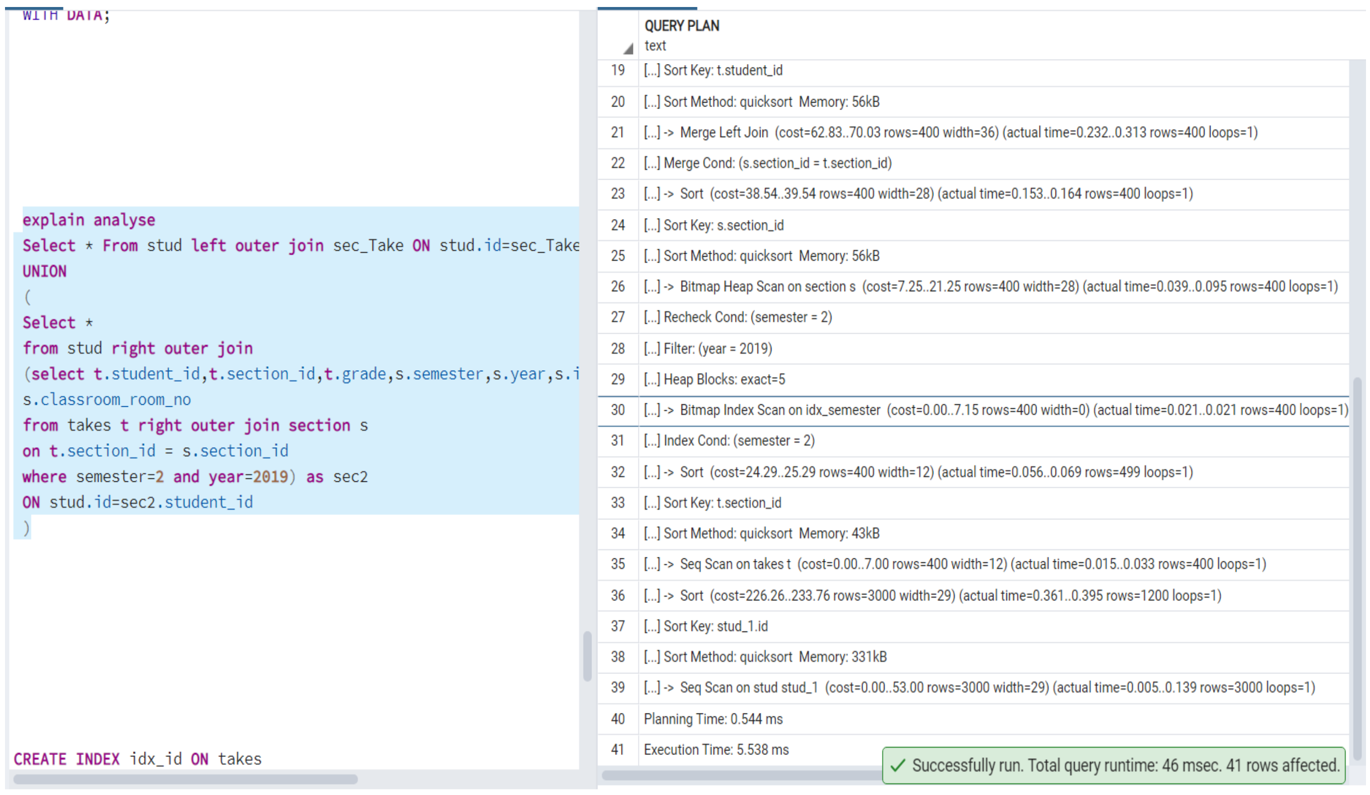
1. given query with B+ trees indices only,

```
CREATE INDEX idx_dept ON stud
USING btree (department);
CREATE INDEX idx_id ON student
USING btree (id);
CREATE INDEX idx_sec ON section
USING btree (section_id);
CREATE INDEX idx_takes ON takes
USING btree (section_id);
CREATE INDEX idx_semester ON section
USING btree (semester);
CREATE INDEX idx_year ON section
USING btree (year);
```



```
WITH DATA;




explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)



CREATE INDEX idx_id ON takes
```

```
QUERY PLAN
text
Unique  (cost=864.45..991.95 rows=3400 width=164) (actual time=3.682..4.884 rows=3200 loops=1)
[...] ->  Sort  (cost=864.45..872.95 rows=3400 width=164) (actual time=3.681..3.788 rows=3650 loops=1)
[...] Sort Key: stud.id, stud.name, stud.tot_credit, stud.department, stud.advisor_id, sec_take.student_id, sec_take.section_id, se
[...] Sort Method: quicksort  Memory: 425kB
[...] ->  Append  (cost=265.45..665.02 rows=3400 width=164) (actual time=0.472..2.031 rows=3650 loops=1)
[...] ->  Merge Left Join  (cost=265.45..288.95 rows=3000 width=65) (actual time=0.471..0.990 rows=3250 loops=1)
[...] Merge Cond: (stud.id = sec_take.student_id)
[...] ->  Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.368..0.451 rows=3000 loops=1)
[...] Sort Key: stud.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Seq Scan on stud  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.005..0.145 rows=3000 loops=1)
[...] ->  Sort  (cost=38.69..40.19 rows=600 width=36) (actual time=0.093..0.110 rows=600 loops=1)
[...] Sort Key: sec_take.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Seq Scan on sec_take  (cost=0.00..11.00 rows=600 width=36) (actual time=0.008..0.038 rows=600 loops=1)
[...] ->  Merge Left Join  (cost=313.57..325.07 rows=400 width=65) (actual time=0.776..0.877 rows=400 loops=1)
[...] Merge Cond: (t.student_id = stud_1.id)
[...] ->  Sort  (cost=87.31..88.31 rows=400 width=36) (actual time=0.343..0.358 rows=400 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 56kB
[...] ->  Merge Left Join  (cost=62.83..70.03 rows=400 width=36) (actual time=0.214..0.299 rows=400 loops=1)
[...] Merge Cond: (s.section_id = t.section_id)
[...] ->  Sort  (cost=38.54..39.54 rows=400 width=28) (actual time=0.142..0.151 rows=400 loops=1)
```



```sql
WITH DATA;

explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)

CREATE INDEX idx_id ON takes
```

```
QUERY PLAN
text
19  [...] Sort Key: t.student_id
20  [...] Sort Method: quicksort  Memory: 56kB
21  [...] -> Merge Left Join  (cost=62.83..70.03 rows=400 width=36) (actual time=0.232..0.313 rows=400 loops=1)
22  [...] Merge Cond: (s.section_id = t.section_id)
23  [...] -> Sort  (cost=38.54..39.54 rows=400 width=28) (actual time=0.153..0.164 rows=400 loops=1)
24  [...] Sort Key: s.section_id
25  [...] Sort Method: quicksort  Memory: 56kB
26  [...] -> Bitmap Heap Scan on section s  (cost=7.25..21.25 rows=400 width=28) (actual time=0.039..0.095 rows=400 loops=1)
27  [...] Recheck Cond: (semester = 2)
28  [...] Filter: (year = 2019)
29  [...] Heap Blocks: exact=5
30  [...] -> Bitmap Index Scan on idx_semester  (cost=0.00..7.15 rows=400 width=0) (actual time=0.021..0.021 rows=400 loops=1)
31  [...] Index Cond: (semester = 2)
32  [...] -> Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.056..0.069 rows=499 loops=1)
33  [...] Sort Key: t.section_id
34  [...] Sort Method: quicksort  Memory: 43kB
35  [...] -> Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.015..0.033 rows=400 loops=1)
36  [...] -> Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.361..0.395 rows=1200 loops=1)
37  [...] Sort Key: stud_1.id
38  [...] Sort Method: quicksort  Memory: 331kB
39  [...] -> Seq Scan on stud stud_1  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.005..0.139 rows=3000 loops=1)
40  Planning Time: 0.544 ms
41  Execution Time: 5.538 ms
✓ Successfully run. Total query runtime: 46 msec. 41 rows affected.
```

Explanation :

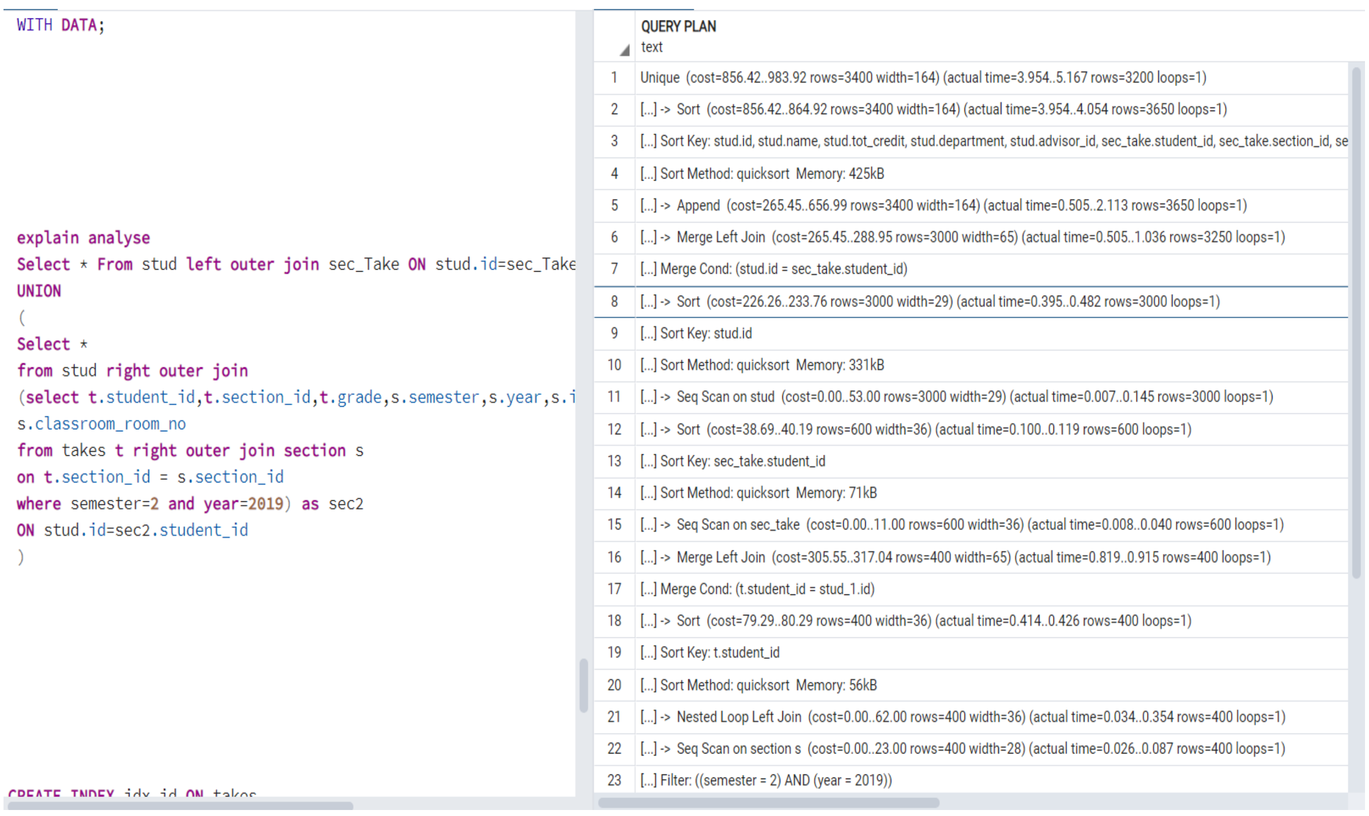
- Metrics :

Execution Time : 5.538 ms Total Expected Cost : 991.95

- The cost here have changed slightly as compared to no indices dropping from 993 to 991.
- The small drop in cost that happened was due to the Bitmap Heap Scan that was done on table section and the bitmap index scan on the index on the semester column instead of traversing and searching in table section sequentially. Improvement was somehow limited as postgres chose not to use index on department column as well as section_id on takes table and did a seq scan instead.

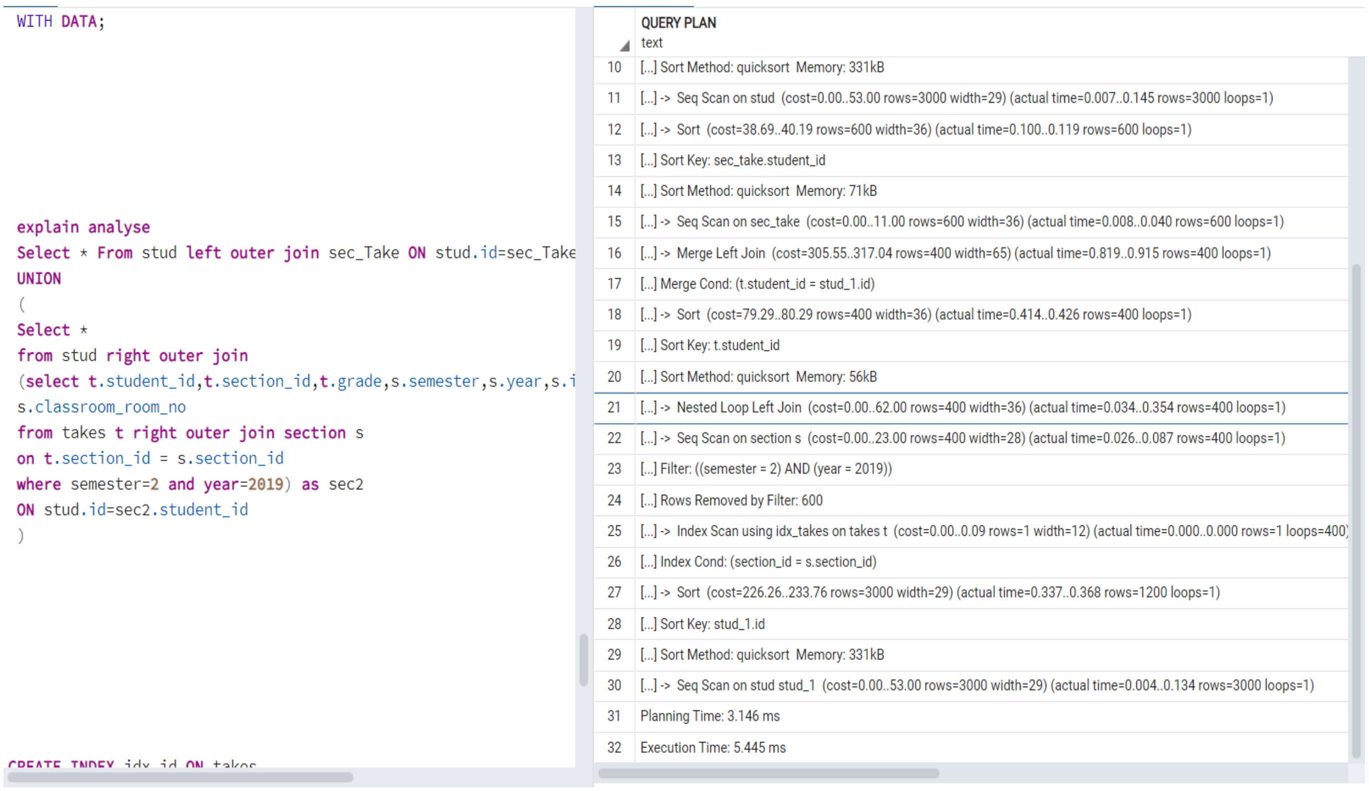
3. given query with hash indices only

```sql
CREATE INDEX idx_id ON takes
USING hash (student_id);
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_year ON section
USING hash (year);
CREATE INDEX idx_sec ON section
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING hash (semester);
```



```sql
WITH DATA;



explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)


CREATE INDEX idx_id ON takes
```

	QUERY PLAN text
1	Unique (cost=856.42..983.92 rows=3400 width=164) (actual time=3.954..5.167 rows=3200 loops=1)
2	[...] -> Sort (cost=856.42..864.92 rows=3400 width=164) (actual time=3.954..4.054 rows=3650 loops=1)
3	[...] Sort Key: stud.id, stud.name, stud.tot_credit, stud.department, stud.advisor_id, sec_take.student_id, sec_take.section_id, se
4	[...] Sort Method: quicksort Memory: 425kB
5	[...] -> Append (cost=265.45..656.99 rows=3400 width=164) (actual time=0.505..2.113 rows=3650 loops=1)
6	[...] -> Merge Left Join (cost=265.45..288.95 rows=3000 width=65) (actual time=0.505..1.036 rows=3250 loops=1)
7	[...] Merge Cond: (stud.id = sec_take.student_id)
8	[...] -> Sort (cost=226.26..233.76 rows=3000 width=29) (actual time=0.395..0.482 rows=3000 loops=1)
9	[...] Sort Key: stud.id
10	[...] Sort Method: quicksort Memory: 331kB
11	[...] -> Seq Scan on stud (cost=0.00..53.00 rows=3000 width=29) (actual time=0.007..0.145 rows=3000 loops=1)
12	[...] -> Sort (cost=38.69..40.19 rows=600 width=36) (actual time=0.100..0.119 rows=600 loops=1)
13	[...] Sort Key: sec_take.student_id
14	[...] Sort Method: quicksort Memory: 71kB
15	[...] -> Seq Scan on sec_take (cost=0.00..11.00 rows=600 width=36) (actual time=0.008..0.040 rows=600 loops=1)
16	[...] -> Merge Left Join (cost=305.55..317.04 rows=400 width=65) (actual time=0.819..0.915 rows=400 loops=1)
17	[...] Merge Cond: (t.student_id = stud_1.id)
18	[...] -> Sort (cost=79.29..80.29 rows=400 width=36) (actual time=0.414..0.426 rows=400 loops=1)
19	[...] Sort Key: t.student_id
20	[...] Sort Method: quicksort Memory: 56kB
21	[...] -> Nested Loop Left Join (cost=0.00..62.00 rows=400 width=36) (actual time=0.034..0.354 rows=400 loops=1)
22	[...] -> Seq Scan on section s (cost=0.00..23.00 rows=400 width=28) (actual time=0.026..0.087 rows=400 loops=1)
23	[...] Filter: ((semester = 2) AND (year = 2019))



```
WITH DATA;




explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)




CREATE INDEX idx_id ON takes
```

	QUERY PLAN text
10	[...] Sort Method: quicksort Memory: 331kB
11	[...] -> Seq Scan on stud (cost=0.00..53.00 rows=3000 width=29) (actual time=0.007..0.145 rows=3000 loops=1)
12	[...] -> Sort (cost=38.69..40.19 rows=600 width=36) (actual time=0.100..0.119 rows=600 loops=1)
13	[...] Sort Key: sec_take.student_id
14	[...] Sort Method: quicksort Memory: 71kB
15	[...] -> Seq Scan on sec_take (cost=0.00..11.00 rows=600 width=36) (actual time=0.008..0.040 rows=600 loops=1)
16	[...] -> Merge Left Join (cost=305.55..317.04 rows=400 width=65) (actual time=0.819..0.915 rows=400 loops=1)
17	[...] Merge Cond: (t.student_id = stud_1.id)
18	[...] -> Sort (cost=79.29..80.29 rows=400 width=36) (actual time=0.414..0.426 rows=400 loops=1)
19	[...] Sort Key: t.student_id
20	[...] Sort Method: quicksort Memory: 56kB
21	[...] -> Nested Loop Left Join (cost=0.00..62.00 rows=400 width=36) (actual time=0.034..0.354 rows=400 loops=1)
22	[...] -> Seq Scan on section s (cost=0.00..23.00 rows=400 width=28) (actual time=0.026..0.087 rows=400 loops=1)
23	[...] Filter: ((semester = 2) AND (year = 2019))
24	[...] Rows Removed by Filter: 600
25	[...] -> Index Scan using idx_takes on takes t (cost=0.00..0.09 rows=1 width=12) (actual time=0.000..0.000 rows=1 loops=400)
26	[...] Index Cond: (section_id = s.section_id)
27	[...] -> Sort (cost=226.26..233.76 rows=3000 width=29) (actual time=0.337..0.368 rows=1200 loops=1)
28	[...] Sort Key: stud_1.id
29	[...] Sort Method: quicksort Memory: 331kB
30	[...] -> Seq Scan on stud stud_1 (cost=0.00..53.00 rows=3000 width=29) (actual time=0.004..0.134 rows=3000 loops=1)
31	Planning Time: 3.146 ms
32	Execution Time: 5.445 ms

Explanation :

- Metrics :

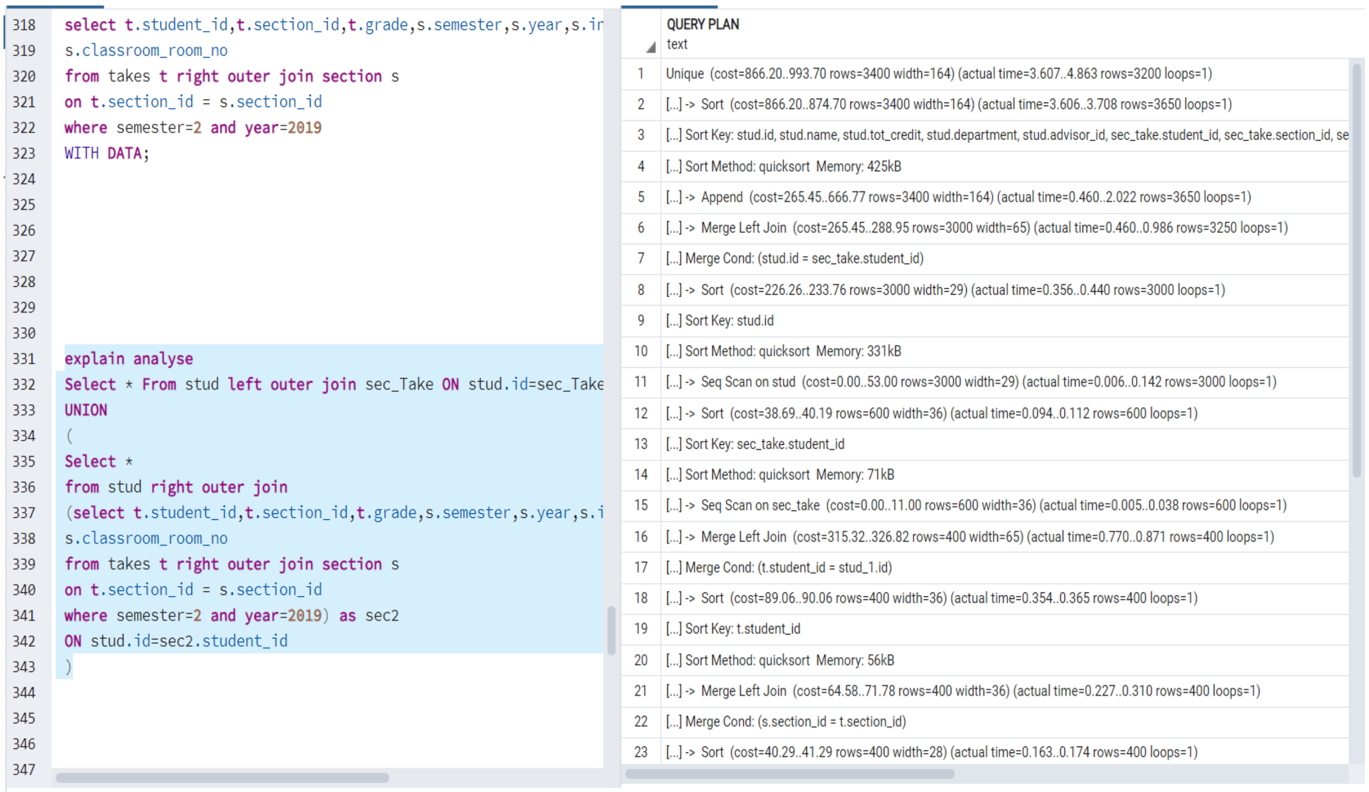
Execution Time : 5.445 ms	Total Expected Cost : 983.92

- It is noticed here that the cost have been slightly reduced by the usage of hash indices compared to no indices at all as instead of doing a seq scan to find a particular section id while doing the join, an index scan on table takes was used which caused that boost.

- Here as expected, hash index with its O(1) performance beats Btree indices, as hashing is best for exact queries.

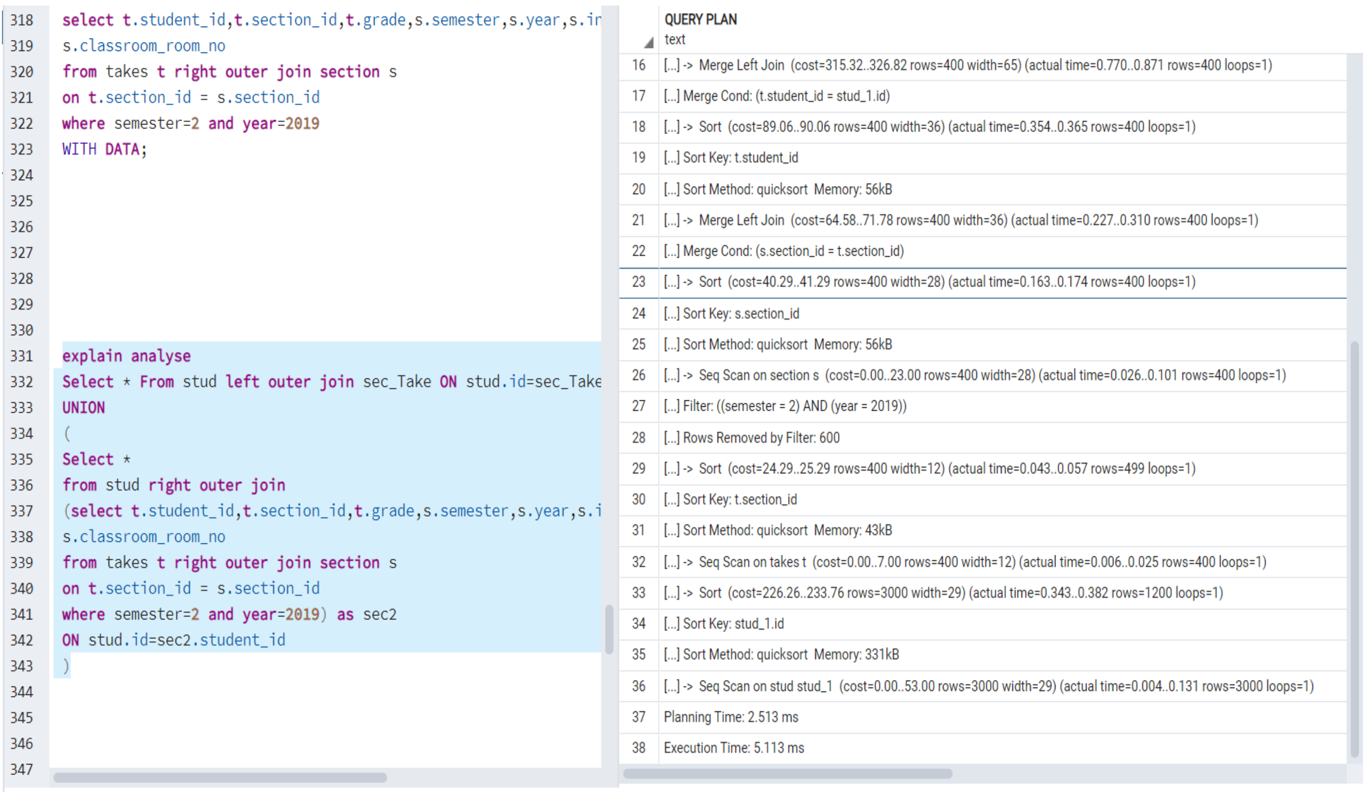
4. given query with BRIN indices only

```sql
CREATE INDEX idx_dept ON stud
USING brin (department);
CREATE INDEX idx_semester ON sec_Take
USING brin (semester);
```



```sql
select t.student_id,t.section_id,t.grade,s.semester,s.year,s.ir
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019
WITH DATA;







explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)
```

```
QUERY PLAN
text
Unique  (cost=866.20..993.70 rows=3400 width=164) (actual time=3.607..4.863 rows=3200 loops=1)
[...] ->  Sort  (cost=866.20..874.70 rows=3400 width=164) (actual time=3.606..3.708 rows=3650 loops=1)
[...] Sort Key: stud.id, stud.name, stud.tot_credit, stud.department, stud.advisor_id, sec_take.student_id, sec_take.section_id, se
[...] Sort Method: quicksort  Memory: 425kB
[...] ->  Append  (cost=265.45..666.77 rows=3400 width=164) (actual time=0.460..2.022 rows=3650 loops=1)
[...] ->  Merge Left Join  (cost=265.45..288.95 rows=3000 width=65) (actual time=0.460..0.986 rows=3250 loops=1)
[...] Merge Cond: (stud.id = sec_take.student_id)
[...] ->  Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.356..0.440 rows=3000 loops=1)
[...] Sort Key: stud.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Seq Scan on stud  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.006..0.142 rows=3000 loops=1)
[...] ->  Sort  (cost=38.69..40.19 rows=600 width=36) (actual time=0.094..0.112 rows=600 loops=1)
[...] Sort Key: sec_take.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Seq Scan on sec_take  (cost=0.00..11.00 rows=600 width=36) (actual time=0.005..0.038 rows=600 loops=1)
[...] ->  Merge Left Join  (cost=315.32..326.82 rows=400 width=65) (actual time=0.770..0.871 rows=400 loops=1)
[...] Merge Cond: (t.student_id = stud_1.id)
[...] ->  Sort  (cost=89.06..90.06 rows=400 width=36) (actual time=0.354..0.365 rows=400 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 56kB
[...] ->  Merge Left Join  (cost=64.58..71.78 rows=400 width=36) (actual time=0.227..0.310 rows=400 loops=1)
[...] Merge Cond: (s.section_id = t.section_id)
[...] ->  Sort  (cost=40.29..41.29 rows=400 width=28) (actual time=0.163..0.174 rows=400 loops=1)
```



```sql
select t.student_id,t.section_id,t.grade,s.semester,s.year,s.ir
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019
WITH DATA;







explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)
```

```
QUERY PLAN
text
[...] ->  Merge Left Join  (cost=315.32..326.82 rows=400 width=65) (actual time=0.770..0.871 rows=400 loops=1)
[...] Merge Cond: (t.student_id = stud_1.id)
[...] ->  Sort  (cost=89.06..90.06 rows=400 width=36) (actual time=0.354..0.365 rows=400 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 56kB
[...] ->  Merge Left Join  (cost=64.58..71.78 rows=400 width=36) (actual time=0.227..0.310 rows=400 loops=1)
[...] Merge Cond: (s.section_id = t.section_id)
[...] ->  Sort  (cost=40.29..41.29 rows=400 width=28) (actual time=0.163..0.174 rows=400 loops=1)
```

```
[...] Sort Key: s.section_id
[...] Sort Method: quicksort  Memory: 56kB
[...] ->  Seq Scan on section s  (cost=0.00..23.00 rows=400 width=28) (actual time=0.026..0.101 rows=400 loops=1)
[...] Filter: ((semester = 2) AND (year = 2019))
[...] Rows Removed by Filter: 600
[...] ->  Sort  (cost=24.29..25.29 rows=400 width=12) (actual time=0.043..0.057 rows=499 loops=1)
[...] Sort Key: t.section_id
[...] Sort Method: quicksort  Memory: 43kB
[...] ->  Seq Scan on takes t  (cost=0.00..7.00 rows=400 width=12) (actual time=0.006..0.025 rows=400 loops=1)
[...] ->  Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.343..0.382 rows=1200 loops=1)
[...] Sort Key: stud_1.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Seq Scan on stud stud_1  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.004..0.131 rows=3000 loops=1)
Planning Time: 2.513 ms
Execution Time: 5.113 ms
```

Explanation :

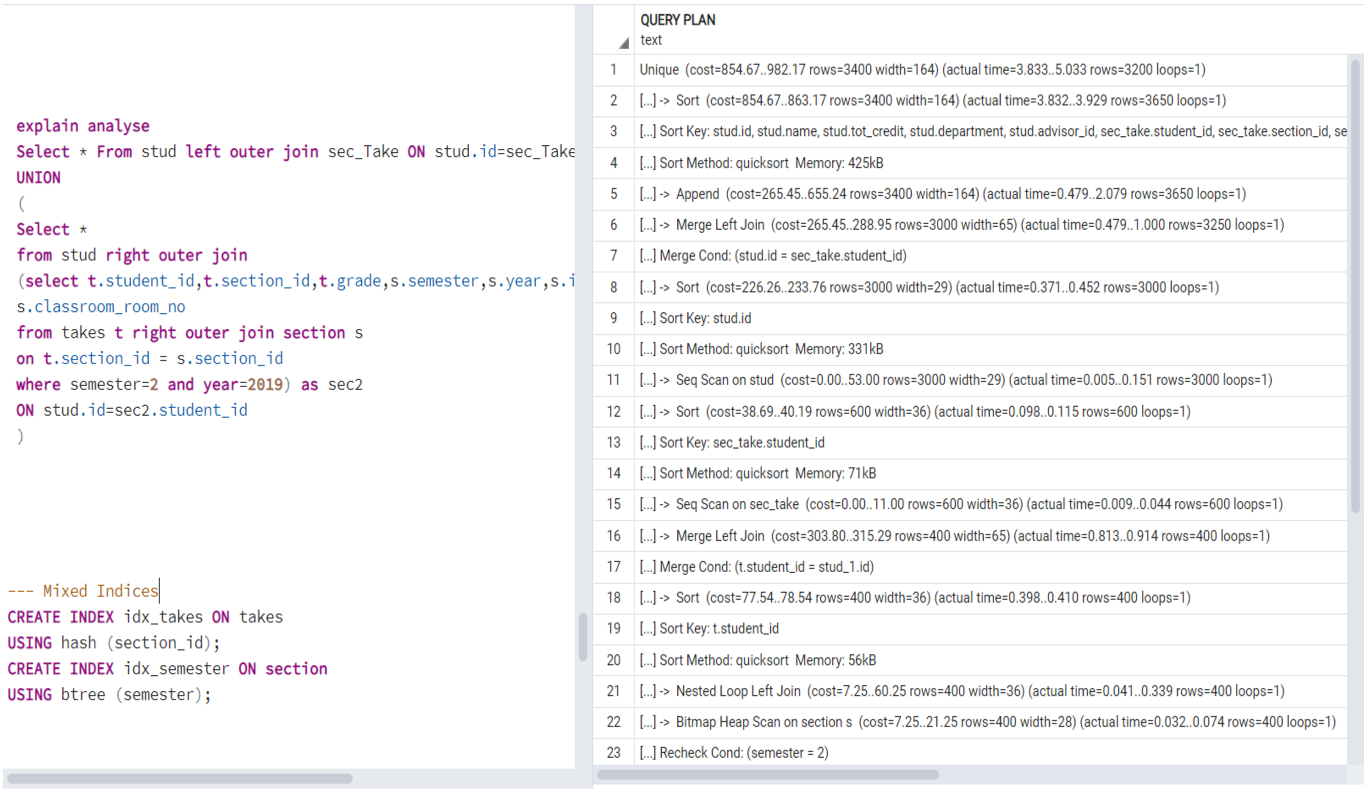
- Metrics :

Execution Time : 5.113 ms   Total Expected Cost : 993.70

- The cost here is almost the same and the brin didn't have any effect on the performance of the query.
- This is due to the bitmap index scan not being chosen by postgres but a seq scan instead even though bitmap scan is enabled as it doesn't detect the query as a low selectivity one so brin is not favoured.

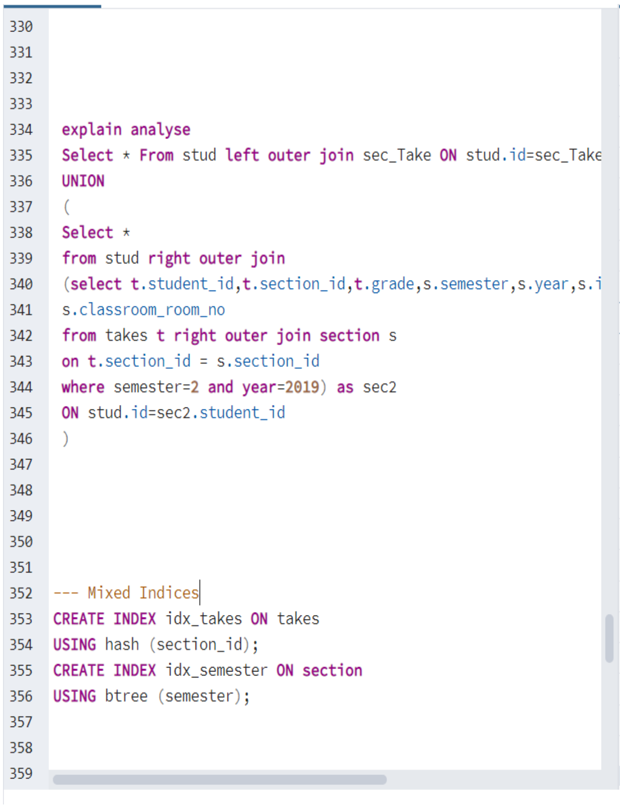
5. given query with mixed indices (any mix of your choice)

```sql
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING btree (semester);
```

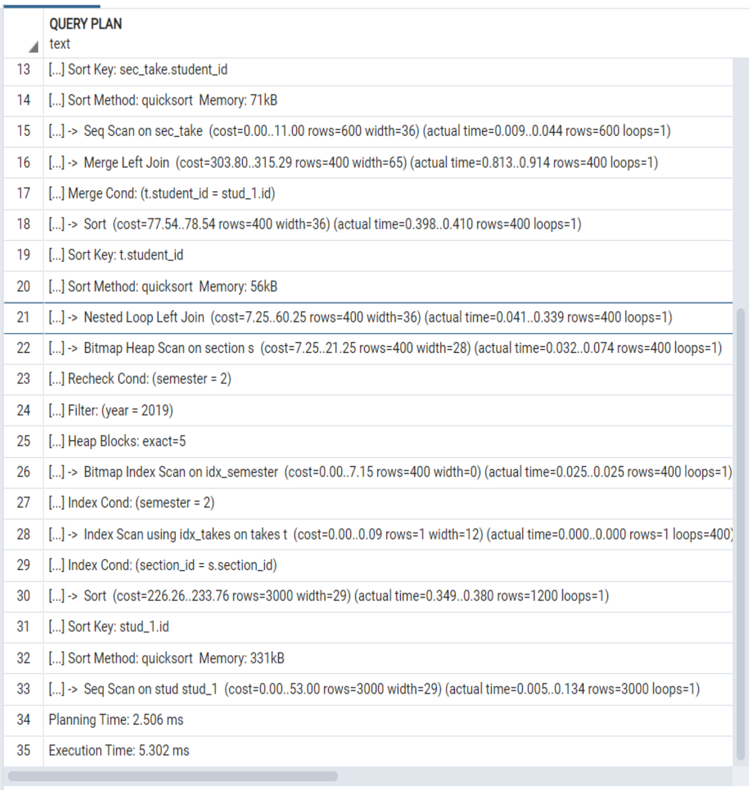


```sql
explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)



--- Mixed Indices
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING btree (semester);
```

	QUERY PLAN text
1	Unique (cost=854.67..982.17 rows=3400 width=164) (actual time=3.833..5.033 rows=3200 loops=1)
2	[...] -> Sort (cost=854.67..863.17 rows=3400 width=164) (actual time=3.832..3.929 rows=3650 loops=1)
3	[...] Sort Key: stud.id, stud.name, stud.tot_credit, stud.department, stud.advisor_id, sec_take.student_id, sec_take.section_id, se
4	[...] Sort Method: quicksort Memory: 425kB
5	[...] -> Append (cost=265.45..655.24 rows=3400 width=164) (actual time=0.479..2.079 rows=3650 loops=1)
6	[...] -> Merge Left Join (cost=265.45..288.95 rows=3000 width=65) (actual time=0.479..1.000 rows=3250 loops=1)
7	[...] Merge Cond: (stud.id = sec_take.student_id)
8	[...] -> Sort (cost=226.26..233.76 rows=3000 width=29) (actual time=0.371..0.452 rows=3000 loops=1)
9	[...] Sort Key: stud.id
10	[...] Sort Method: quicksort Memory: 331kB
11	[...] -> Seq Scan on stud (cost=0.00..53.00 rows=3000 width=29) (actual time=0.005..0.151 rows=3000 loops=1)
12	[...] -> Sort (cost=38.69..40.19 rows=600 width=36) (actual time=0.098..0.115 rows=600 loops=1)
13	[...] Sort Key: sec_take.student_id
14	[...] Sort Method: quicksort Memory: 71kB
15	[...] -> Seq Scan on sec_take (cost=0.00..11.00 rows=600 width=36) (actual time=0.009..0.044 rows=600 loops=1)
16	[...] -> Merge Left Join (cost=303.80..315.29 rows=400 width=65) (actual time=0.813..0.914 rows=400 loops=1)
17	[...] Merge Cond: (t.student_id = stud_1.id)
18	[...] -> Sort (cost=77.54..78.54 rows=400 width=36) (actual time=0.398..0.410 rows=400 loops=1)
19	[...] Sort Key: t.student_id
20	[...] Sort Method: quicksort Memory: 56kB
21	[...] -> Nested Loop Left Join (cost=7.25..60.25 rows=400 width=36) (actual time=0.041..0.339 rows=400 loops=1)
22	[...] -> Bitmap Heap Scan on section s (cost=7.25..21.25 rows=400 width=28) (actual time=0.032..0.074 rows=400 loops=1)
23	[...] Recheck Cond: (semester = 2)



```sql
explain analyse
Select * From stud left outer join sec_Take ON stud.id=sec_Take
UNION
(
Select *
from stud right outer join
(select t.student_id,t.section_id,t.grade,s.semester,s.year,s.i
s.classroom_room_no
from takes t right outer join section s
on t.section_id = s.section_id
where semester=2 and year=2019) as sec2
ON stud.id=sec2.student_id
)




--- Mixed Indices
CREATE INDEX idx_takes ON takes
USING hash (section_id);
CREATE INDEX idx_semester ON section
USING btree (semester);
```



```
QUERY PLAN
text
[...] Sort Key: sec_take.student_id
[...] Sort Method: quicksort  Memory: 71kB
[...] ->  Seq Scan on sec_take  (cost=0.00..11.00 rows=600 width=36) (actual time=0.009..0.044 rows=600 loops=1)
[...] ->  Merge Left Join  (cost=303.80..315.29 rows=400 width=65) (actual time=0.813..0.914 rows=400 loops=1)
[...] Merge Cond: (t.student_id = stud_1.id)
[...] ->  Sort  (cost=77.54..78.54 rows=400 width=36) (actual time=0.398..0.410 rows=400 loops=1)
[...] Sort Key: t.student_id
[...] Sort Method: quicksort  Memory: 56kB
[...] ->  Nested Loop Left Join  (cost=7.25..60.25 rows=400 width=36) (actual time=0.041..0.339 rows=400 loops=1)
[...] ->  Bitmap Heap Scan on section s  (cost=7.25..21.25 rows=400 width=28) (actual time=0.032..0.074 rows=400 loops=1)
[...] Recheck Cond: (semester = 2)
[...] Filter: (year = 2019)
[...] Heap Blocks: exact=5
[...] ->  Bitmap Index Scan on idx_semester  (cost=0.00..7.15 rows=400 width=0) (actual time=0.025..0.025 rows=400 loops=1)
[...] Index Cond: (semester = 2)
[...] ->  Index Scan using idx_takes on takes t  (cost=0.00..0.09 rows=1 width=12) (actual time=0.000..0.000 rows=1 loops=400)
[...] Index Cond: (section_id = s.section_id)
[...] ->  Sort  (cost=226.26..233.76 rows=3000 width=29) (actual time=0.349..0.380 rows=1200 loops=1)
[...] Sort Key: stud_1.id
[...] Sort Method: quicksort  Memory: 331kB
[...] ->  Seq Scan on stud stud_1  (cost=0.00..53.00 rows=3000 width=29) (actual time=0.005..0.134 rows=3000 loops=1)
Planning Time: 2.506 ms
Execution Time: 5.302 ms
```

Explanation :

- Metrics :

Execution Time : 5.302 ms   Total Expected Cost : 982.17

- The cost here is the best among all other previous queries due to using both hash and btree indices on the columns where they fit best.
- The btree index didn't enhance it as much as hash index enhanced it for the same reason mentioned above about btree not enhancing the cost quite as much.