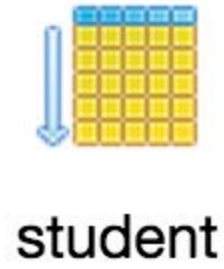
Author Queries 1 & 2: Phillip Griggs

Query 1 Before Tuning:

```
SELECT sname
FROM STUDENT
WHERE id = '545899';
```



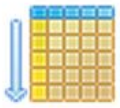
student

QUERY 1 PLAN

```
 Seq Scan on student  (cost=0.00..218.00 rows=1 width=10) (actual time=0.016..1.171 rows=1
loops=1)
   Filter: (id = 545899)
   Rows Removed by Filter: 9999
 Planning time: 0.048 ms
 Execution time: 1.201 ms
(5 rows)
```

Query 1 After Tuning:

```
SELECT sname
FROM STUDENT
WHERE id = '545889';
```



student_pkey

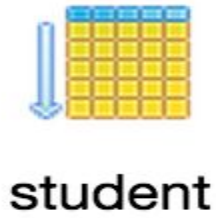
QUERY 1 PLAN

```
 Index Scan using student_pkey on student s  (cost=0.29..8.30 rows=1 width=10) (actual
time=0.026..0.027 rows=1 loops=1)
   Index Cond: (id = 545899)
 Planning time: 0.570 ms
 Execution time: 0.090 ms
(4 rows)
```

Execution Time: 1.201 ms before tune vs 0.090 ms after tune
Observations: The tuned query had to traverse less rows than the untuned query. Also a full table scan was performed on the untuned query while a single row scan was for the tuned query.

Query 2 Before Tuning:

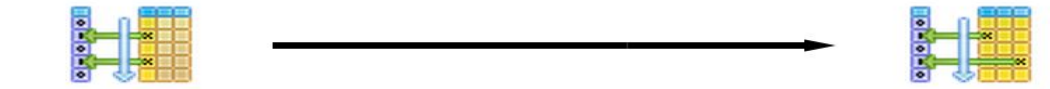
```
SELECT sname
FROM STUDENT
WHERE id BETWEEN '15192' AND '17138';
```




```
                                QUERY 2 PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on student  (cost=0.00..243.00 rows=21 width=10) (actual time=0.122..1.748
rows=18 loops=1)
   Filter: ((id >= 15192) AND (id <= 17138))
   Rows Removed by Filter: 9982
 Planning time: 0.040 ms
 Execution time: 1.763 ms
(5 rows)
```

Query 2 After Tuning:

```
SELECT sname
FROM STUDENT
WHERE id BETWEEN '15192' AND '17138';
```



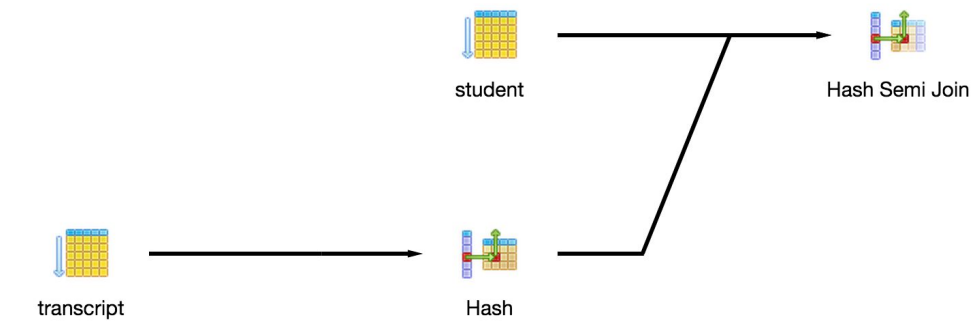
QUERY 2 PLAN

```
 Bitmap Heap Scan on student s  (cost=4.50..55.05 rows=21 width=10) (actual
time=0.031..0.089 rows=18 loops=1)
   Recheck Cond: ((id >= 15192) AND (id <= 17138))
   Heap Blocks: exact=15
   ->  Bitmap Index Scan on student_pkey  (cost=0.00..4.50 rows=21 width=0) (actual
time=0.021..0.021 rows=18 loops=1)
         Index Cond: ((id >= 15192) AND (id <= 17138))
 Planning time: 0.092 ms
 Execution time: 0.120 ms
(7 rows)
```

Execution Time: 1.763 ms before tune vs. 0.120 ms after tune
Observations: The untuned query performed a sequential scan on the table while the tuned query did not and performed a heap scan. This led to significantly faster query result times. Also an index scan was used with the tuned query.

Author of Query 3: James Griffin
Query 3 Before Tuning:

```
SELECT sname
FROM STUDENT
WHERE id = ANY (
        SELECT studId
        FROM TRANSCRIPT
        WHERE crsCode = 'crsCode902901');
```

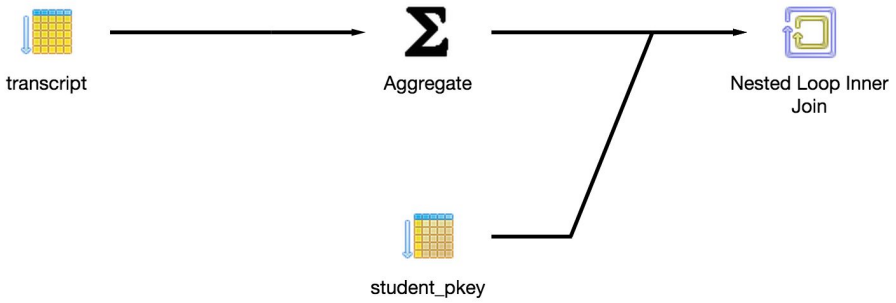


QUERY 3 PLAN

Hash Semi Join (cost=109.54..328.83 rows=3 width=10) (actual time=1.194..2.648 rows=2 loops=1)
Hash Cond: (student.id = transcript.studid)
-> Seq Scan on student (cost=0.00..193.00 rows=10000 width=14) (actual time=0.008..0.925 rows=10000 loops=1)
-> Hash (cost=109.50..109.50 rows=3 width=4) (actual time=0.688..0.688 rows=2 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on transcript (cost=0.00..109.50 rows=3 width=4) (actual time=0.398..0.686 rows=2 loops=1)
Filter: ((crscode)::text = 'crsCode902901'::text)
Rows Removed by Filter: 4998
Planning time: 0.131 ms
Execution time: 2.666 ms
(10 rows)

Query 3 After Tuning:

```
SELECT sname
FROM STUDENT
WHERE id = ANY (
        SELECT stdId
        FROM TRANSCRIPT
        WHERE crsCode = ‘crsCode902901’);
```



QUERY 3 PLAN

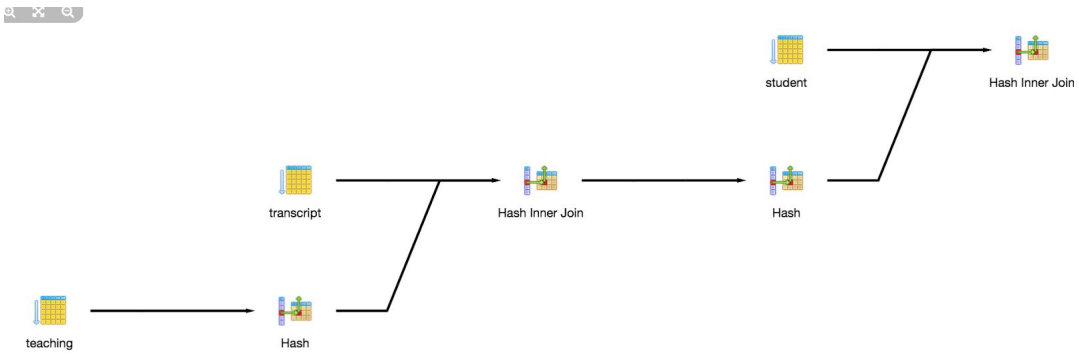
Nested Loop (cost=109.79..134.47 rows=3 width=10) (actual time=0.838..0.850 rows=2 loops=1)
-> HashAggregate (cost=109.51..109.54 rows=3 width=4) (actual time=0.826..0.827 rows=2 loops=1)
Group Key: sid.studid
-> Seq Scan on transcript sid (cost=0.00..109.50 rows=3 width=4) (actual time=0.481..0.821 rows=2 loops=1)
Filter: ((crscode)::text = 'crsCode902901'::text)
Rows Removed by Filter: 4998
-> Index Scan using student_pkey on student s (cost=0.29..8.30 rows=1 width=14) (actual time=0.008..0.009 rows=1 loops=2)
Index Cond: (id = sid.studid)
Planning time: 0.540 ms
Execution time: 0.917 ms
(10 rows)

Execution Time: 2.666 ms before tune vs. 0.917 ms after tune
Observations: The untuned query had to perform a hashed semi join. The tuned query did not have to perform a join which led to faster querying. The tuned query also used a nested loop to perform even faster than the untuned.

AUTHOR OF QUERIES 4-6: ASHETON HARRELL

Q4:: List the names of students who have taken a course taught by professor v5 (491584)
Query before optimization:

```
EXPLAIN ANALYZE
SELECT s.sname
FROM STUDENT AS s, TRANSCRIPT AS r, TEACHING AS t
WHERE t.profId=491584 AND r.crsCode=t.crsCode AND r.studId=s.id;
```

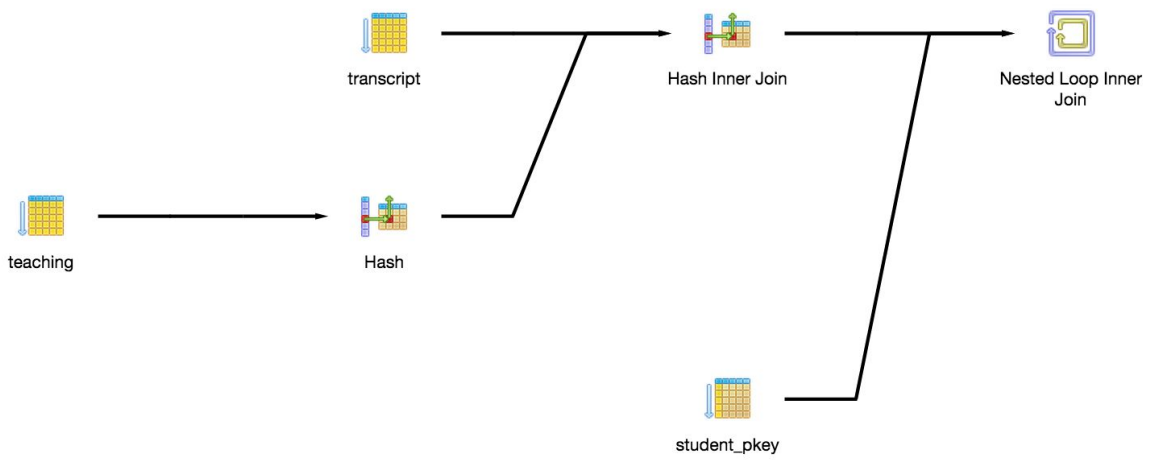


```
 Hash Join  (cost=244.65..475.30 rows=15 width=10) (actual time=2.024..3.699 rows=11
loops=1)
   Hash Cond: (s.id = r.studid)
   ->  Seq Scan on student s  (cost=0.00..193.00 rows=10000 width=14) (actual
time=0.008..0.937 rows=10000 loops=1)
   ->  Hash  (cost=244.46..244.46 rows=15 width=4) (actual time=1.728..1.728 rows=11
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Hash Join  (cost=103.56..244.46 rows=15 width=4) (actual time=0.538..1.726 rows=11
loops=1)
               Hash Cond: ((r.crscode)::text = (t.crscode)::text)
               ->  Seq Scan on transcript r  (cost=0.00..97.00 rows=5000 width=17) (actual
time=0.005..0.494 rows=5000 loops=1)
               ->  Hash  (cost=103.50..103.50 rows=5 width=13) (actual time=0.518..0.518 rows=2
loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on teaching t  (cost=0.00..103.50 rows=5 width=13) (actual
time=0.297..0.516 rows=2 loops=1)
                           Filter: (profid = 491584)
                           Rows Removed by Filter: 4998
 Planning time: 0.420 ms
 Execution time: 3.724 ms
(15 rows)
```

Observations: Execution was a little slower than I expected. Joining full tables and then parsing through all the rows in the full tables gives us a very long execution time.

Query-Optimized:

```
SELECT s.sname
FROM STUDENT AS s INNER JOIN TRANSCRIPT AS r ON r.studId=s.id INNER JOIN
TEACHING AS t ON r.crsCode=t.crsCode
WHERE t.profId=491584;
```



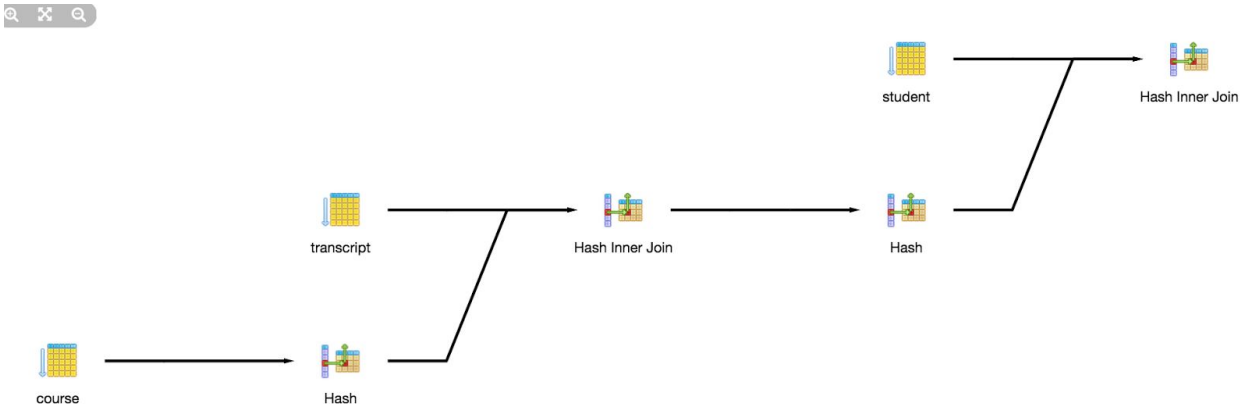

```
 Nested Loop  (cost=103.85..250.63 rows=15 width=10) (actual time=0.813..2.830 rows=11
loops=1)
   ->  Hash Join  (cost=103.56..244.46 rows=15 width=4) (actual time=0.799..2.667 rows=11
loops=1)
         Hash Cond: ((r.crscode)::text = (t.crscode)::text)
         ->  Seq Scan on transcript r  (cost=0.00..97.00 rows=5000 width=17) (actual
time=0.008..0.798 rows=5000 loops=1)
         ->  Hash  (cost=103.50..103.50 rows=5 width=13) (actual time=0.751..0.751 rows=2
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on teaching t  (cost=0.00..103.50 rows=5 width=13) (actual
time=0.369..0.745 rows=2 loops=1)
                     Filter: (profid = 491584)
                     Rows Removed by Filter: 4998
   ->  Index Scan using student_pkey on student s  (cost=0.29..0.40 rows=1 width=14) (actual
time=0.013..0.013 rows=1 loops=11)
         Index Cond: (id = r.studid)
 Planning time: 0.921 ms
 Execution time: 2.866 ms
(13 rows)
```

Observations: This is a full ms less than the before optimized query. Indexing and cross-products/inner joins helps reduce execution time, as there are less rows being searched.

Q5:: List the names of students who have taken a course from department v6 (deptId), but not v7

Query before optimization:

```
EXPLAIN ANALYZE
SELECT s.sname
FROM STUDENT AS s, TRANSCRIPT AS t, COURSE AS c
WHERE t.crsCode=c.crsCode AND c.deptId='deptId664077' AND t.studId=s.id
    AND c.deptId!='deptId424969';
```

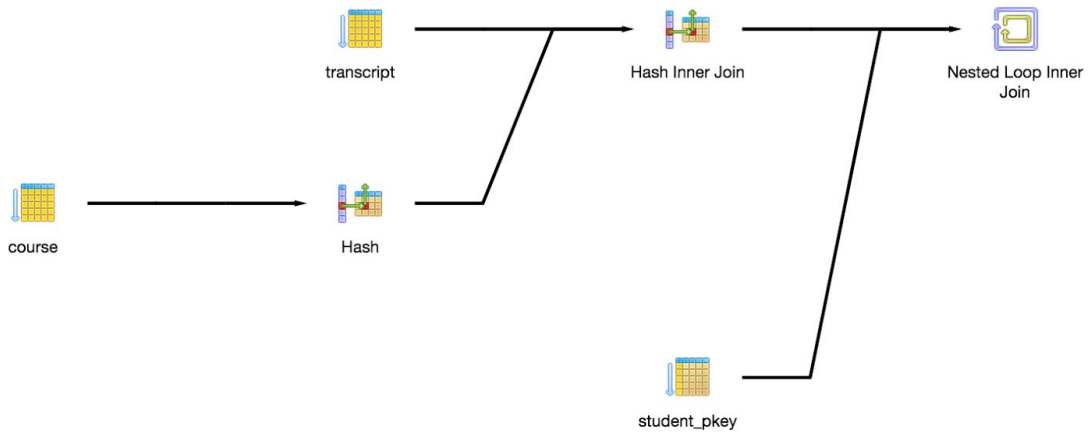



```
 Hash Join  (cost=166.82..397.34 rows=2 width=10) (actual time=1.696..3.458 rows=2 loops=1)
   Hash Cond: (s.id = t.studid)
   ->  Seq Scan on student s  (cost=0.00..193.00 rows=10000 width=14) (actual
time=0.008..0.914 rows=10000 loops=1)
   ->  Hash  (cost=166.80..166.80 rows=2 width=4) (actual time=1.547..1.547 rows=2 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Hash Join  (cost=51.03..166.80 rows=2 width=4) (actual time=0.704..1.534 rows=2
loops=1)
               Hash Cond: ((t.crscode)::text = (c.crscode)::text)
               ->  Seq Scan on transcript t  (cost=0.00..97.00 rows=5000 width=17) (actual
time=0.004..0.475 rows=5000 loops=1)
               ->  Hash  (cost=51.02..51.02 rows=1 width=13) (actual time=0.435..0.435 rows=1
loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on course c  (cost=0.00..51.02 rows=1 width=13) (actual
time=0.169..0.433 rows=1 loops=1)
                           Filter: (((deptid)::text <> 'deptId424969'::text) AND ((deptid)::text =
'deptId664077'::text))
                           Rows Removed by Filter: 2000
 Planning time: 0.240 ms
 Execution time: 3.487 ms
(15 rows)
```

Observations: Execution time is quite long due to the combination of multiple full tables and searching through multiple where conditions. No indexing makes this query take quite long.

Query after optimization:

```
EXPLAIN ANALYZE
SELECT s.sname
FROM STUDENT AS s INNER JOIN TRANSCRIPT AS t ON s.id=t.studId INNER JOIN
COURSE AS c ON t.crsCode=c.crsCode
WHERE c.deptId='deptId664077' AND c.deptId!='deptId424969';
```

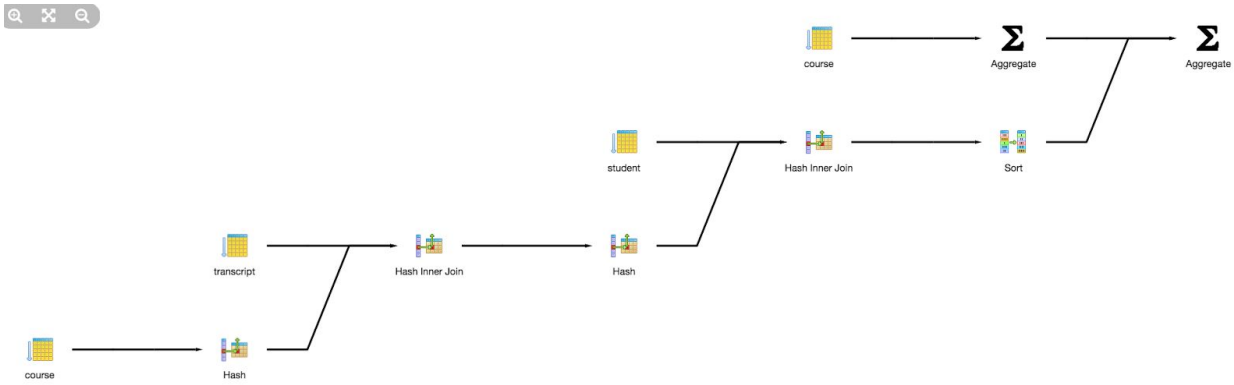



```
 Nested Loop  (cost=103.85..250.63 rows=15 width=10) (actual time=0.813..2.830 rows=11
loops=1)
   ->  Hash Join  (cost=103.56..244.46 rows=15 width=4) (actual time=0.799..2.667 rows=11
loops=1)
         Hash Cond: ((r.crscode)::text = (t.crscode)::text)
         ->  Seq Scan on transcript r  (cost=0.00..97.00 rows=5000 width=17) (actual
time=0.008..0.798 rows=5000 loops=1)
         ->  Hash  (cost=103.50..103.50 rows=5 width=13) (actual time=0.751..0.751 rows=2
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on teaching t  (cost=0.00..103.50 rows=5 width=13) (actual
time=0.369..0.745 rows=2 loops=1)
                     Filter: (profid = 491584)
                     Rows Removed by Filter: 4998
   ->  Index Scan using student_pkey on student s  (cost=0.29..0.40 rows=1 width=14) (actual
time=0.013..0.013 rows=1 loops=11)
         Index Cond: (id = r.studid)
 Planning time: 0.921 ms
 Execution time: 2.866 ms
(13 rows)
```

Observations: Execution is faster because we are using smaller rows based on indexing and inner joining rules. You can tell that the hashing join is helping reduce runtime in both versions of this query, but indexing helps a bit in turning it down. From the visualizations of the query plan you can see that there is a level of complexity that gets taken out when indexing.

Q6:: List the names of students who have taken all courses offered by department v8 (deptId)
Query 6 before-optimization:

```
EXPLAIN ANALYZE
SELECT s.sname
FROM STUDENT AS s, COURSE AS c, TRANSCRIPT AS t
WHERE t.crsCode=c.crsCode AND t.studId=s.id AND c.deptId='deptId424969'
GROUP BY s.sname
HAVING COUNT(*) = (SELECT COUNT(*) FROM COURSE WHERE
COURSE.deptId='deptId424969');
```




```
 GroupAggregate  (cost=438.38..438.41 rows=2 width=10) (actual time=5.305..5.308 rows=3
loops=1)
   Group Key: s.sname
   Filter: (count(*) = $0)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=46.02..46.02 rows=1 width=8) (actual time=0.368..0.368 rows=1
loops=1)
           ->  Seq Scan on course  (cost=0.00..46.01 rows=1 width=0) (actual time=0.028..0.363
rows=1 loops=1)
                 Filter: ((deptid)::text = 'deptId424969'::text)
                 Rows Removed by Filter: 2000
   ->  Sort  (cost=392.35..392.36 rows=2 width=10) (actual time=4.922..4.923 rows=3 loops=1)
         Sort Key: s.sname
         Sort Method: quicksort  Memory: 25kB
```

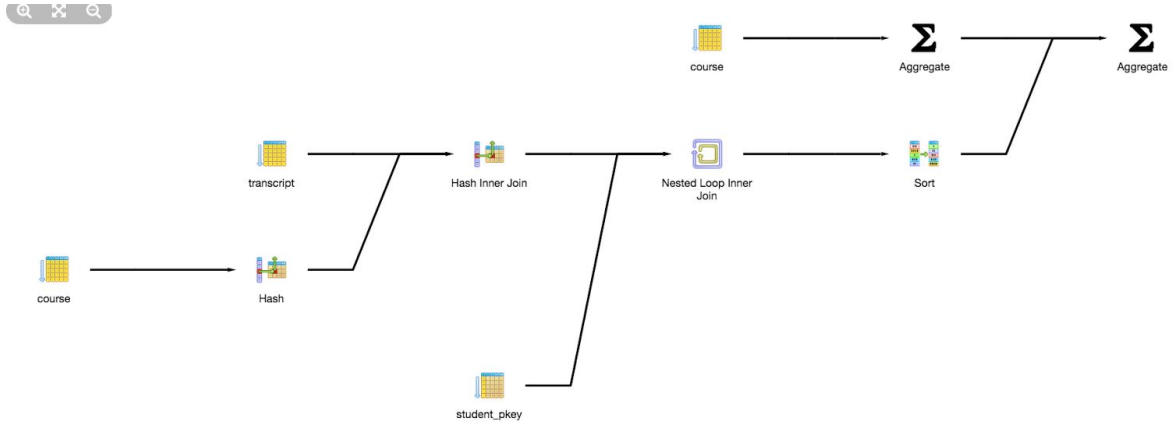
```
      ->  Hash Join  (cost=161.82..392.34 rows=2 width=10) (actual time=2.706..4.900 rows=3
loops=1)
            Hash Cond: (s.id = t.studid)
            ->  Seq Scan on student s  (cost=0.00..193.00 rows=10000 width=14) (actual
time=0.013..1.462 rows=10000 loops=1)
            ->  Hash  (cost=161.80..161.80 rows=2 width=4) (actual time=2.079..2.079 rows=3
loops=1)
                  Buckets: 1024  Batches: 1  Memory Usage: 9kB
                  ->  Hash Join  (cost=46.03..161.80 rows=2 width=4) (actual time=0.711..2.065
rows=3 loops=1)
                        Hash Cond: ((t.crscode)::text = (c.crscode)::text)
                        ->  Seq Scan on transcript t  (cost=0.00..97.00 rows=5000 width=17) (actual
time=0.006..0.728 rows=5000 loops=1)
                        ->  Hash  (cost=46.01..46.01 rows=1 width=13) (actual time=0.402..0.402
rows=1 loops=1)
                              Buckets: 1024  Batches: 1  Memory Usage: 9kB
                              ->  Seq Scan on course c  (cost=0.00..46.01 rows=1 width=13) (actual
time=0.015..0.399 rows=1 loops=1)
                                    Filter: ((deptid)::text = 'deptId424969'::text)
                                    Rows Removed by Filter: 2000
 Planning time: 0.455 ms
 Execution time: 5.366 ms
(26 rows)
```

Observations: This query is very unoptimized, as shown by the execution time. The joining of multiple full tables with multiple where clause searching and no indexing makes for a very very long execution phase.

Query 6 optimized:

```
SELECT s.sname
FROM STUDENT AS s INNER JOIN TRANSCRIPT AS t ON t.studId=s.id INNER JOIN
COURSE AS c ON t.crsCode=c.crsCode
WHERE c.deptId='deptId424969'
GROUP BY s.sname
```

HAVING COUNT(*) = (SELECT COUNT(crsCode) FROM COURSE WHERE deptId='deptId424969');




```
 GroupAggregate  (cost=208.65..208.69 rows=2 width=10) (actual time=3.581..3.588 rows=3 loops=1)
   Group Key: s.sname
   Filter: (count(*) = $0)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=46.02..46.02 rows=1 width=8) (actual time=0.406..0.406 rows=1 loops=1)
           ->  Seq Scan on course  (cost=0.00..46.01 rows=1 width=13) (actual time=0.029..0.400 rows=1 loops=1)
                 Filter: ((deptid)::text = 'deptId424969'::text)
                 Rows Removed by Filter: 2000
   ->  Sort  (cost=162.63..162.63 rows=2 width=10) (actual time=3.146..3.149 rows=3 loops=1)
         Sort Key: s.sname
         Sort Method: quicksort  Memory: 25kB
         ->  Nested Loop  (cost=46.31..162.62 rows=2 width=10) (actual time=1.221..2.995 rows=3 loops=1)
               ->  Hash Join  (cost=46.03..161.80 rows=2 width=4) (actual time=1.151..2.837 rows=3 loops=1)
                     Hash Cond: ((t.crscode)::text = (c.crscode)::text)
                     ->  Seq Scan on transcript t  (cost=0.00..97.00 rows=5000 width=17) (actual time=0.027..0.950 rows=5000 loops=1)
                     ->  Hash  (cost=46.01..46.01 rows=1 width=13) (actual time=0.666..0.666 rows=1 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on course c  (cost=0.00..46.01 rows=1 width=13) (actual time=0.045..0.645 rows=1 loops=1)
                                 Filter: ((deptid)::text = 'deptId424969'::text)
                                 Rows Removed by Filter: 2000
```

```
            ->  Index Scan using student_pkey on student s  (cost=0.29..0.40 rows=1 width=14)
(actual time=0.043..0.044 rows=1 loops=3)
                  Index Cond: (id = t.studid)
 Planning time: 1.082 ms
 Execution time: 3.770 ms
(24 rows)
```

Observations: We shave off almost 2ms of execution time by removing where clause conditions, adding indexing to the tables, and inner joining tables to reduce the amount of rows searched. It seems the aggregate functions cost a good amount on the scheduler. I think this execution would be faster had postgres not keep a buffer of previously run queries.