Exercises Set 3

Indexes

1. 4 queries in this exercise
   1. EXPLAIN ANALYZE SELECT academic.famname as family\_name, academic.givename as given\_name  
      FROM academic INNER JOIN interest ON academic.acnum = interest.acnum  
      WHERE interest.fieldnum = 292;
   2. EXPLAIN ANALYZE SELECT \* FROM author ORDER BY acnum DESC;
   3. EXPLAIN ANALYZE SELECT paper.panum as paper\_number, paper.title as paper\_title  
      FROM paper  
      WHERE paper.title ILIKE '%database%';
   4. EXPLAIN ANALYZE SELECT COUNT(academic.acnum), academic.deptnum, department.deptname

FROM academic LEFT JOIN department ON academic.deptnum = department.deptnum

WHERE department.deptnum IN (SELECT academic.deptnum FROM academic)

GROUP BY academic.deptnum, department.deptname

ORDER BY COUNT(academic.acnum) ASC;

1. When running the VACUUM ANALYZE it just only influences slightly to the “actual time”. Therefore, the VACUUM ANALYZE statement is only necessary when updating or deleting rows in the non-temporary tables.

## Query 1:

1. **Without index**

['Hash Join (cost=42.96..53.81 rows=3 width=42) (actual time=0.251..0.318 rows=1 loops=1)']  
[' Hash Cond: (academic.acnum = interest.acnum)']  
[' -> Seq Scan on academic (cost=0.00..8.58 rows=358 width=46) (actual time=0.002..0.023 rows=358 loops=1)']  
[' -> Hash (cost=42.92..42.92 rows=3 width=4) (actual time=0.231..0.231 rows=1 loops=1)']  
[' Buckets: 1024 Batches: 1 Memory Usage: 9kB']  
[' -> Seq Scan on interest (cost=0.00..42.92 rows=3 width=4) (actual time=0.006..0.229 rows=1 loops=1)']  
[' Filter: (fieldnum = 292)']  
[' Rows Removed by Filter: 1673']  
['Planning time: 0.597 ms']  
['Execution time: 0.367 ms']

1. **With indexes**

Indexes in order:

* CREATE INDEX ON interest(fieldnum)

['Hash Join (cost=8.50..19.34 rows=3 width=42) (actual time=0.054..0.172 rows=1 loops=1)']  
[' Hash Cond: (academic.acnum = interest.acnum)']  
[' -> Seq Scan on academic (cost=0.00..8.58 rows=358 width=46) (actual time=0.002..0.062 rows=358 loops=1)']  
[' -> Hash (cost=8.46..8.46 rows=3 width=4) (actual time=0.026..0.026 rows=1 loops=1)']  
[' Buckets: 1024 Batches: 1 Memory Usage: 9kB']  
[' -> Index Scan using interest\_fieldnum\_idx on interest (cost=0.28..8.46 rows=3 width=4) (actual time=0.022..0.022 rows=1 loops=1)']  
[' Index Cond: (fieldnum = 292)']  
['Planning time: 0.662 ms']  
['Execution time: 0.223 ms']

**Comparison**:

In this query, i generated an index on the interest.fieldnum in the WHERE statement. Therefore, with the index, there is a decrement in the execution time. And the significant changes are the Filter:(fieldnum = 292) and Rows Removed by Filter: 1673 in the non-index scan which do not exist in the index scan anymore. Also, the cost for the Hash Join decrease dramatically from 42.96..53.81 to 8.50..19.34, almost 4 times. It's the same with  
the actual time, inclining from 0.251..0.318 to 0.054..0.172.

### Query 2:

1. **Without indexes**

['Sort (cost=656.09..676.71 rows=8249 width=8) (actual time=4.896..5.366 rows=8249 loops=1)']  
[' Sort Key: acnum DESC']  
[' Sort Method: quicksort Memory: 771kB']  
[' -> Seq Scan on author (cost=0.00..119.49 rows=8249 width=8) (actual time=0.012..1.805 rows=8249 loops=1)']  
['Planning time: 0.263 ms']  
['Execution time: 5.772 ms']

1. **With indexes**

Indexes in order:

* CREATE INDEX ON author(acnum)

['Index Scan Backward using author\_acnum\_idx on author (cost=0.29..371.95 rows=8249 width=8) (actual time=0.023..2.762 rows=8249 loops=1)']  
['Planning time: 0.328 ms']  
['Execution time: 3.154 ms']

**Comparison:**

In this case, without creating an index on the the acnum column, the execution time will be 5.772 ms which is much more slower than the one having index.In the sequence scan, it needs to sort through the primary key which squander lots of time. In another hand, with the index scan Backward, the ORDER BY statement will check from the right leaf nodes of the B-trees to sort the acnum number.

### Query 3:

1. **Without indexes**

['Seq Scan on paper (cost=0.00..149.50 rows=102 width=105) (actual time=0.016..3.423 rows=167 loops=1)']  
[" Filter: (title ~~\* '%database%'::text)"]  
[' Rows Removed by Filter: 4833']  
['Planning time: 0.500 ms']  
['Execution time: 3.468 ms']

1. **With indexes**

Indexes in order:

* CREATE INDEX ON paper(title)

['Seq Scan on paper (cost=0.00..149.50 rows=102 width=105) (actual time=0.017..3.450 rows=167 loops=1)']  
[" Filter: (title ~~\* '%database%'::text)"]  
[' Rows Removed by Filter: 4833']  
['Planning time: 0.626 ms']  
['Execution time: 3.496 ms']

**Comparison:**

In this circumstance, i tried to create index for the paper.title, but nothing change much. The execution time is almost similar to each other.  
Because postgreSQL does not support the index for the 'LIKE' or 'ILIKE' statements. Furthermore, the B-tree index scan just only match with the prefixes and the wildcard at the start of query.

### Query 4:

1. **Without indexes**

['Sort (cost=48.82..49.72 rows=358 width=89) (actual time=0.682..0.686 rows=44 loops=1)']  
[' Sort Key: (count(academic.acnum))']  
[' Sort Method: quicksort Memory: 31kB']  
[' -> HashAggregate (cost=30.06..33.64 rows=358 width=89) (actual time=0.626..0.635 rows=44 loops=1)']  
[' Group Key: academic.deptnum, department.deptname']  
[' -> Hash Join (cost=14.10..27.37 rows=358 width=89) (actual time=0.262..0.433 rows=358 loops=1)']  
[' Hash Cond: (academic.deptnum = department.deptnum)']  
[' -> Seq Scan on academic (cost=0.00..8.58 rows=358 width=8) (actual time=0.007..0.047 rows=358 loops=1)']  
[' -> Hash (cost=13.55..13.55 rows=44 width=89) (actual time=0.239..0.239 rows=44 loops=1)']  
[' Buckets: 1024 Batches: 1 Memory Usage: 14kB']  
[' -> Hash Join (cost=10.46..13.55 rows=44 width=89) (actual time=0.194..0.223 rows=44 loops=1)']  
[' Hash Cond: (department.deptnum = academic\_1.deptnum)']  
[' -> Seq Scan on department (cost=0.00..2.47 rows=47 width=85) (actual time=0.001..0.007 rows=47 loops=1)']  
[' -> Hash (cost=9.91..9.91 rows=44 width=4) (actual time=0.182..0.182 rows=44 loops=1)']  
[' Buckets: 1024 Batches: 1 Memory Usage: 10kB']  
[' -> HashAggregate (cost=9.47..9.91 rows=44 width=4) (actual time=0.160..0.167 rows=44 loops=1)']  
[' Group Key: academic\_1.deptnum']  
[' -> Seq Scan on academic academic\_1 (cost=0.00..8.58 rows=358 width=4) (actual time=0.001..0.046 rows=358 loops=1)']  
['Planning time: 1.064 ms']  
['Execution time: 0.867 ms']

1. **With indexes**

Indexes in order:

* CREATE INDEX ON academic(deptname)
* CREATE INDEX ON academic(deptnum)

['Sort (cost=48.55..49.45 rows=358 width=89) (actual time=0.474..0.474 rows=44 loops=1)']  
[' Sort Key: (count(academic.acnum))']  
[' Sort Method: quicksort Memory: 31kB']  
[' -> HashAggregate (cost=29.78..33.36 rows=358 width=89) (actual time=0.430..0.440 rows=44 loops=1)']  
[' Group Key: academic.deptnum, department.deptname']  
[' -> Hash Join (cost=13.83..27.10 rows=358 width=89) (actual time=0.123..0.263 rows=358 loops=1)']  
[' Hash Cond: (academic.deptnum = department.deptnum)']  
[' -> Seq Scan on academic (cost=0.00..8.58 rows=358 width=8) (actual time=0.002..0.026 rows=358 loops=1)']  
[' -> Hash (cost=13.28..13.28 rows=44 width=89) (actual time=0.112..0.112 rows=44 loops=1)']  
[' Buckets: 1024 Batches: 1 Memory Usage: 14kB']  
[' -> Nested Loop Semi Join (cost=0.15..13.28 rows=44 width=89) (actual time=0.039..0.091 rows=44 loops=1)']  
[' -> Seq Scan on department (cost=0.00..2.47 rows=47 width=85) (actual time=0.002..0.007 rows=47 loops=1)']  
[' -> Index Only Scan using academic\_deptnum\_idx on academic academic\_1 (cost=0.15..0.46 rows=8 width=4) (actual time=0.001..0.001 rows=1 loops=47)']  
[' Index Cond: (deptnum = department.deptnum)']  
[' Heap Fetches: 0']  
['Planning time: 1.332 ms']  
['Execution time: 0.581 ms']

**Comparison:**

In this query, i tried to generate index for academic.deptname, academic.deptnum one by one. However, in the first two index, nothing is gonna change and when creating the index for academic.deptnum, the query plans have changed. To be more specific, without the indexes, the (WHERE department.deptnum IN (SELECT academic.deptnum FROM academic)) statement will run on the seq scan and the cost will be much larger than the one that using indexes (0.00..8.58 compared with 0.15..0.46). And the "HashAggregate" disappeared in the query using indexes.