Query Planning and Optimization

Akshat Karani (170010003)

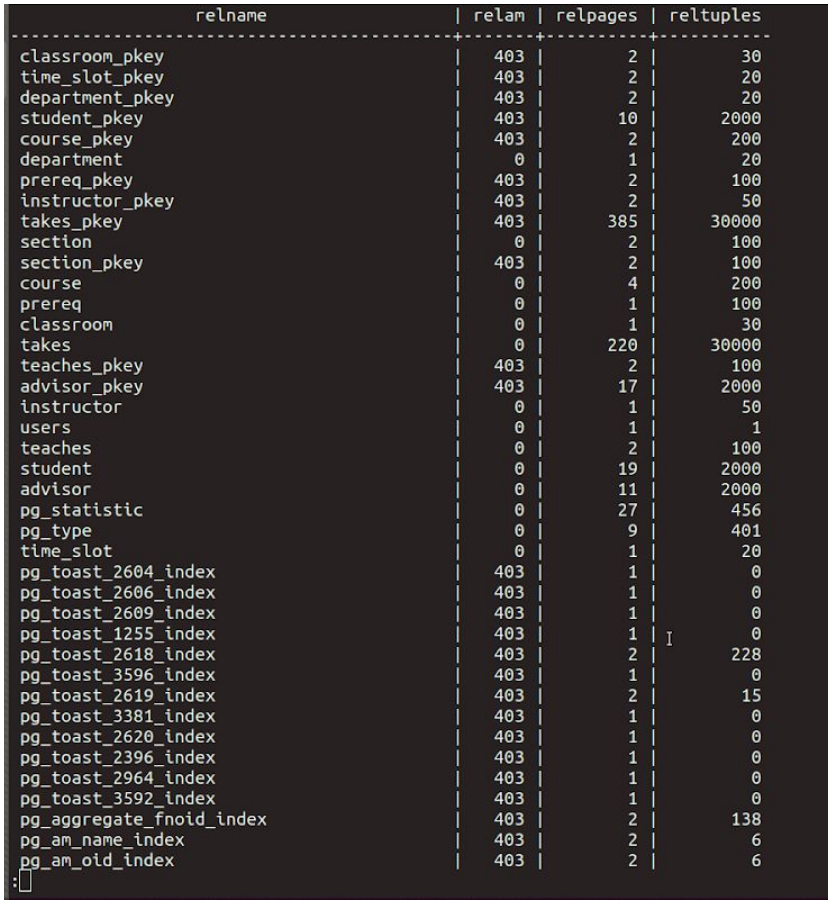
Question 1

To get statistics for tables and indexes, using the following command

```
select relname, relam, relpages, reltuples from pg_class where
relkind='r' or relkind='i';
```

a. relname is the name of the table, index
b. relam: if this is an index, then the access method used 403 means btree
c. relpages: Size of the on-disk representation of this table in pages
d. reltuples: Number of rows in the table
e. relkind = ‘r’ is for ordinary table and ‘i’ is for index

The output of this command is:



relname	relam	relpages	reltuples
classroom_pkey	403	2	30
time_slot_pkey	403	2	20
department_pkey	403	2	20
student_pkey	403	10	2000
course_pkey	403	2	200
department	0	1	20
prereq_pkey	403	2	100
instructor_pkey	403	2	50
takes_pkey	403	385	30000
section	0	2	100
section_pkey	403	2	100
course	0	4	200
prereq	0	1	100
classroom	0	1	30
takes	0	220	30000
teaches_pkey	403	2	100
advisor_pkey	403	17	2000
instructor	0	1	50
users	0	1	1
teaches	0	2	100
student	0	19	2000
advisor	0	11	2000
pg_statistic	0	27	456
pg_type	0	9	401
time_slot	0	1	20
pg_toast_2604_index	403	1	0
pg_toast_2606_index	403	1	0
pg_toast_2609_index	403	1	0
pg_toast_1255_index	403	1	0
pg_toast_2618_index	403	2	228
pg_toast_3596_index	403	1	0
pg_toast_2619_index	403	2	15
pg_toast_3381_index	403	1	0
pg_toast_2620_index	403	1	0
pg_toast_2396_index	403	1	0
pg_toast_2964_index	403	1	0
pg_toast_3592_index	403	1	0
pg_aggregate_fnoid_index	403	2	138
pg_am_name_index	403	2	6
pg_am_oid_index	403	2	6

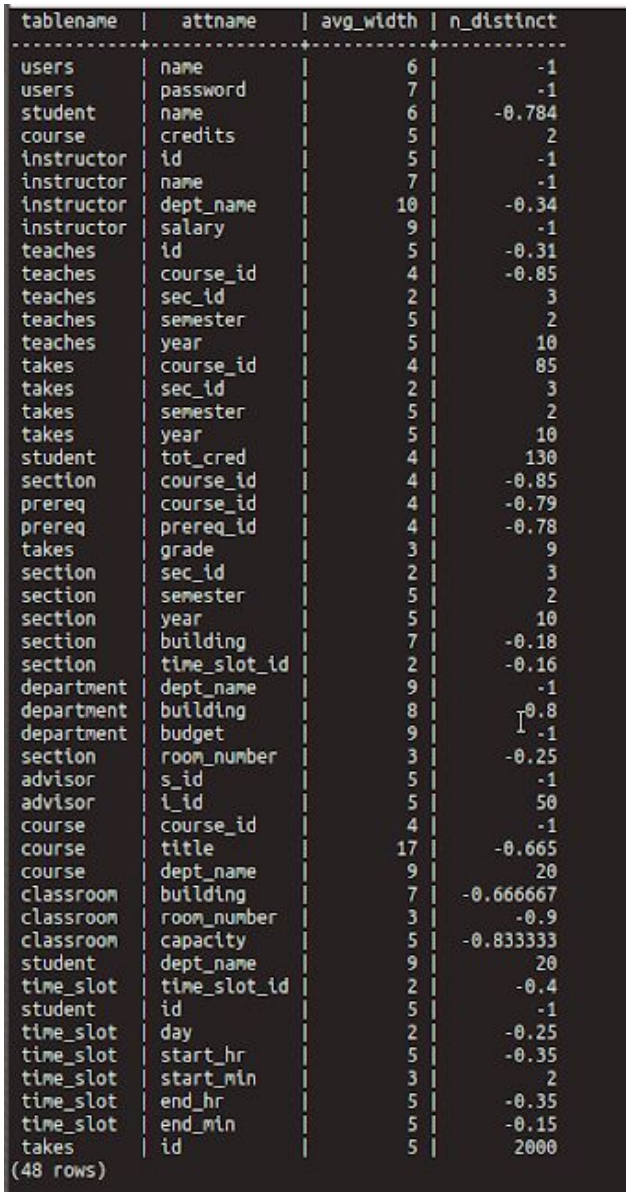
:

To get statistical data about the contents of the database, using the following command

```
select tablename, attname, avg_width, n_distinct from pg_stats where schemaname='public';
```

a. tablename: Name of the table
b. attname: Name of the column described by this row
c. avg_width: Average width in bytes of column's entries
d. n_distinct: If greater than zero, the estimated number of distinct values in a column. If less than zero, the negative of the number of distinct values divided by the number of rows

Output of this command is:



tablename	attname	avg_width	n_distinct
users	name	6	-1
users	password	7	-1
student	name	6	-0.784
course	credits	5	2
instructor	id	5	-1
instructor	name	7	-1
instructor	dept_name	10	-0.34
instructor	salary	9	-1
teaches	id	5	-0.31
teaches	course_id	4	-0.85
teaches	sec_id	2	3
teaches	semester	5	2
teaches	year	5	10
takes	course_id	4	85
takes	sec_id	2	3
takes	semester	5	2
takes	year	5	10
student	tot_cred	4	130
section	course_id	4	-0.85
prereq	course_id	4	-0.79
prereq	prereq_id	4	-0.78
takes	grade	3	9
section	sec_id	2	3
section	semester	5	2
section	year	5	10
section	building	7	-0.18
section	time_slot_id	2	-0.16
department	dept_name	9	-1
department	building	8	-0.8
department	budget	9	-1
section	room_number	3	-0.25
advisor	s_id	5	-1
advisor	i_id	5	50
course	course_id	4	-1
course	title	17	-0.665
course	dept_name	9	20
classroom	building	7	-0.666667
classroom	room_number	3	-0.9
classroom	capacity	5	-0.833333
student	dept_name	9	20
time_slot	time_slot_id	2	-0.4
student	id	5	-1
time_slot	day	2	-0.25
time_slot	start_hr	5	-0.35
time_slot	start_min	3	2
time_slot	end_hr	5	-0.35
time_slot	end_min	5	-0.15
takes	id	5	2000

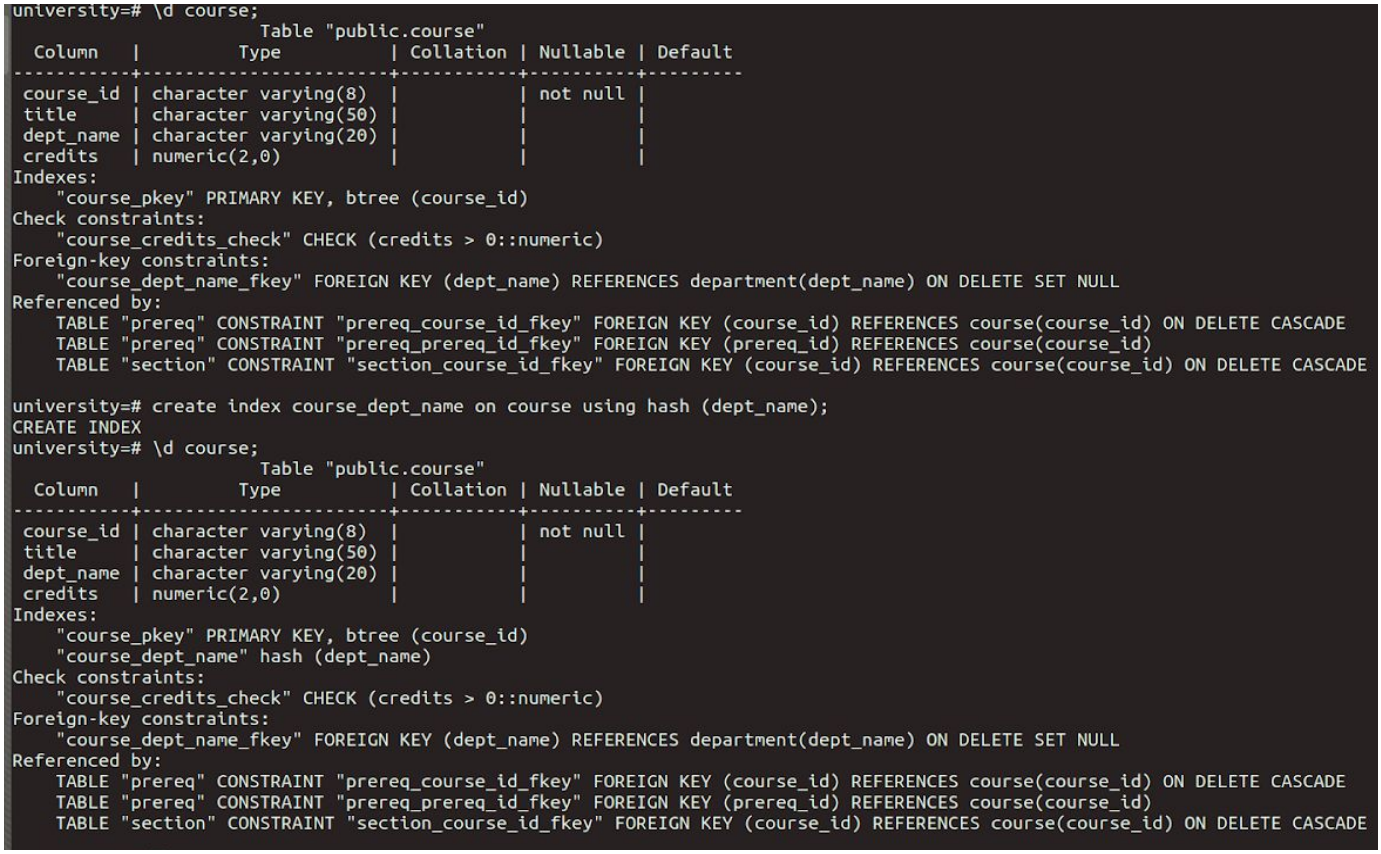
(48 rows)

Question 2

Created a new index for course table on dept_name column using the following command

```
create index course_dept_name on course using hash (dept_name);
```

Following is the screenshot of the above command



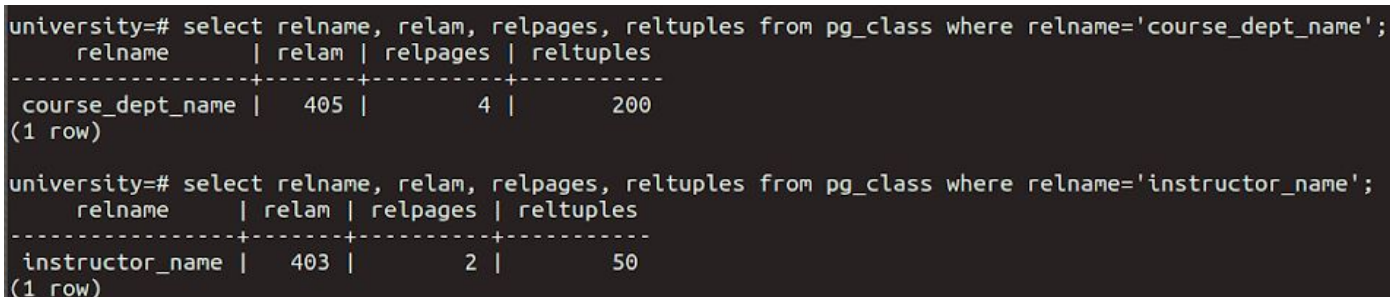
```
university=# \d course;
                    Table "public.course"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 course_id | character varying(8)  |           | not null |
 title     | character varying(50) |           |          |
 dept_name | character varying(20) |           |          |
 credits   | numeric(2,0)          |           |          |
Indexes:
    "course_pkey" PRIMARY KEY, btree (course_id)
Check constraints:
    "course_credits_check" CHECK (credits > 0::numeric)
Foreign-key constraints:
    "course_dept_name_fkey" FOREIGN KEY (dept_name) REFERENCES department(dept_name) ON DELETE SET NULL
Referenced by:
    TABLE "prereq" CONSTRAINT "prereq_course_id_fkey" FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE
    TABLE "prereq" CONSTRAINT "prereq_prereq_id_fkey" FOREIGN KEY (prereq_id) REFERENCES course(course_id)
    TABLE "section" CONSTRAINT "section_course_id_fkey" FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE

university=# create index course_dept_name on course using hash (dept_name);
CREATE INDEX
university=# \d course;
                    Table "public.course"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 course_id | character varying(8)  |           | not null |
 title     | character varying(50) |           |          |
 dept_name | character varying(20) |           |          |
 credits   | numeric(2,0)          |           |          |
Indexes:
    "course_pkey" PRIMARY KEY, btree (course_id)
    "course_dept_name" hash (dept_name)
Check constraints:
    "course_credits_check" CHECK (credits > 0::numeric)
Foreign-key constraints:
    "course_dept_name_fkey" FOREIGN KEY (dept_name) REFERENCES department(dept_name) ON DELETE SET NULL
Referenced by:
    TABLE "prereq" CONSTRAINT "prereq_course_id_fkey" FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE
    TABLE "prereq" CONSTRAINT "prereq_prereq_id_fkey" FOREIGN KEY (prereq_id) REFERENCES course(course_id)
    TABLE "section" CONSTRAINT "section_course_id_fkey" FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE
```

Also created a new index for instructor table instructor name using the following command

```
create index instructor_name on instructor using btree (name);
```

Statistics for both these indexes are



```
university=# select relname, relam, relpages, reltuples from pg_class where relname='course_dept_name';
     relname      | relam | relpages | reltuples
------------------+-------+----------+-----------
 course_dept_name |   405 |        4 |       200
(1 row)

university=# select relname, relam, relpages, reltuples from pg_class where relname='instructor_name';
     relname     | relam | relpages | reltuples
-----------------+-------+----------+-----------
 instructor_name |   403 |        2 |        50
(1 row)
```

Question 3

1. Consider the following exact match predicate query

```
explain analyze select * from takes where (course_id, sec_id) =
(401, 1) and year > 2002;
```

The query plan for this is

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
Seq Scan on takes  (cost=0.00..745.00 rows=206 width=24) (actual time=0.014..7.023 rows=295 loops=1)
  Filter: ((year > '2002'::numeric) AND ((course_id)::text = '401'::text) AND ((sec_id)::text = '1'::text))
  Rows Removed by Filter: 29705
Planning time: 0.228 ms
Execution time: 7.074 ms
(5 rows)
```

After indexing the takes table using

```
create index takes_index on takes(course_id, sec_id);
```

The new query plan is

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
Bitmap Heap Scan on takes  (cost=6.85..241.90 rows=206 width=24) (actual time=0.235..0.738 rows=295 loops=1)
  Recheck Cond: (((course_id)::text = '401'::text) AND ((sec_id)::text = '1'::text))
  Filter: (year > '2002'::numeric)
  Heap Blocks: exact=164
  ->  Bitmap Index Scan on takes_index  (cost=0.00..6.80 rows=251 width=0) (actual time=0.170..0.170 rows=295 loops=1)
        Index Cond: (((course_id)::text = '401'::text) AND ((sec_id)::text = '1'::text))
Planning time: 0.440 ms
Execution time: 0.823 ms
(8 rows)
```

As we can that with indexing execution is almost 10 times faster. Indexing the takes table on course_id and sec_id makes it faster to find an entry in the table having a particular value of course_id and sec_id.

2. Consider the following query to list all the students who have gotten B+ grade in the course 802

```
select * from student where (ID) in
(select ID from takes where course_id = '802' and grade='B+');
```

The query plan for this is

```
                                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=670.92..715.59 rows=37 width=24) (actual time=5.452..5.662 rows=26 loops=1)
   Hash Cond: ((student.id)::text = (takes.id)::text)
   ->  Seq Scan on student  (cost=0.00..39.00 rows=2000 width=24) (actual time=0.010..0.098 rows=2000 loops=1)
   ->  Hash  (cost=670.46..670.46 rows=37 width=5) (actual time=5.410..5.410 rows=26 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  HashAggregate  (cost=670.09..670.46 rows=37 width=5) (actual time=5.403..5.406 rows=26 loops=1)
               Group Key: (takes.id)::text
               ->  Seq Scan on takes  (cost=0.00..670.00 rows=37 width=5) (actual time=0.560..5.384 rows=26 loops=1)
                     Filter: (((course_id)::text = '802'::text) AND ((grade)::text = 'B+'::text))
                     Rows Removed by Filter: 29974
 Planning time: 1.120 ms
 Execution time: 5.723 ms
```

After indexing the takes table using

```
create index takes_index2 on takes(course_id, grade);
```

The new query plan is

```
                                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
Hash Join  (cost=104.27..148.93 rows=37 width=24) (actual time=0.339..1.292 rows=26 loops=1)
  Hash Cond: ((student.id)::text = (takes.id)::text)
  ->  Seq Scan on student  (cost=0.00..39.00 rows=2000 width=24) (actual time=0.008..0.437 rows=2000 loops=1)
  ->  Hash  (cost=103.80..103.80 rows=37 width=5) (actual time=0.221..0.222 rows=26 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 9kB
        ->  HashAggregate  (cost=103.43..103.80 rows=37 width=5) (actual time=0.166..0.179 rows=26 loops=1)
              Group Key: (takes.id)::text
              ->  Bitmap Heap Scan on takes  (cost=4.67..103.34 rows=37 width=5) (actual time=0.078..0.141 rows=26 loops=1)
                    Recheck Cond: (((course_id)::text = '802'::text) AND ((grade)::text = 'B+'::text))
                    Heap Blocks: exact=26
                    ->  Bitmap Index Scan on takes_index  (cost=0.00..4.66 rows=37 width=0) (actual time=0.059..0.060 rows=26 loops=1)
                          Index Cond: (((course_id)::text = '802'::text) AND ((grade)::text = 'B+'::text))
Planning time: 1.069 ms
Execution time: 1.381 ms
(14 rows)
```

As we can see that with indexing execution time is almost 5 times faster. Indexing the takes table with course_id and grade makes it easier to find an entry in the takes table with particular course_id and grade. Hence execution time is faster.