German University in Cairo
Faculty of Media Engineering and Technology
Database II - CSEN 604

Dr. Wael Abouelsaadat

Project 2

Due to shortening the term, this has been turned to an optional project. If you do not do it, the 20% of the course mark will be allocated to project 1. If you submit this project, it will count for 7% and project 1 will count for 13%.

Release Date: June 7th, 2021 **Due Date: 11:59PM, July 28th, 2021**

Team Size and Work Estimation

- You can work with the same from Project 1 or change your team. It is up to you to find team partners if you would like to change team.
- This project is to be done in teams of maximum size 5. Smaller team sizes are acceptable but not more than 5.
- Time expected to complete what is required for 1 schema (if you have never done a similar task before) is between 3 and 6 hours. When you read through the document, you will discover that we have provided many things for you to ensure that you do not waste time doing tasks not related to learning objective of this project. **Your approach should be 1) read this doc 2) read through all the references in the resources section at the end of this doc 3) modify given Java code to do data insertion, 4) analyze given queries, 5) document PostgreSQL output in a report.**
- Analyzing SQL queries is the focus of this project. That is why, you are given 4 schemas and queries to analyze. Most of the work has been done for you. You are given a script to create the schema, a java program which you will modify to populate

the data, and you are only required to run, analyze and document the performance of the queries (which are also given to you).

Submission

- Your submission should include:
  - A **teams.txt** including your team member names and student ids.
  - A **PDF file** with answers, and make sure to include the following:
    1) **output from PostgreSQL analyze command**
    2) **Your Justification of the reported result for each query.**
    3) **Details of any special configuration/parameter adjustment you had to make in PostgreSQL to change how it consider plans. This is basically a trace of your work at either the PostgreSQL shell or through pgAdmin.**

Your TAs will provide you with details regarding submission means.

Required Software

In this project, you are going to work with "**PostgreSQL**" (http://www.postgresql.org) (version 13) the most popular open-source SQL database. After downloading and installing PostgreSQL, install the graphical interface **PGAdmin** from https://www.pgadmin.org/download/. You will also need **Eclipse IDE**

Note: last page in this doc is a references page containing pointers to additional resources.

Inputs You Need to Use

You are given three inputs:

1. a shell script **script.sh** which creates 4 different databases and their respective tables.
2. a java project **MP2DataGenerator** which generates 4 different datasets, one for each database in the shell script and inserts these datasets into their respective tables accordingly.
3. All the queries you need to analyze in a text file **sql-queries.txt**

Executing the shell file

In order to create the databases and tables required for this project, you should execute the shell file using the steps below:

1. Open PostgreSQL shell command
2. Type the following command "PATH-TO-FILE-ON-YOUR-PC/script.sh"
3. Enter your PostgreSQL password

Importing JDBC jar

In order for your java code to work properly, the following steps should be conducted:

1. Import the java project MP2DataGenerator into Eclipse IDE
2. Right Click on the project name, choose build path
3. Click on the add external archives option
4. Browse for "postgresql-jdbc.jar ", which is inside the project folder "MP2DataGenerator"

PostgresSQL-Java Connection

At this step, all compile errors should be resolved. You will find 6 classes inside the project. Each class generates a datat set for each of the 6 databases. For each class do the following:

1. Go to the main method
2. Modify the second and third argument of

   ```
   connection = DriverManager .getConnection (
     " jdbc:postgresql ://hostname:port/dbname" , "username " , "password" ) ;
   ```

   with your postgreSQL username and password.

After applying the steps above, when you run eclipse, a dataset will be generated and inserted in the schema having this class name. For example, running class Schema1 will generate and insert a dataset into schema1 that was created in the postgreSQL server. The process with which the dataset is generated will be explained in the sections below.

Insertion Code

In each schema class, there is a method that is responsible for inserting one tuple for each table in the database. For example, in Schema3 class, there are 3 methods whose name begins with "insert". These three methods insert a single tuple in each of the 3 tables that are in schema3 database. Finally, there is a method named "insert[DatabaseName]" that generates the dataset for this database, by inserting records into its respective tables. For example, in Schema3 class, there is a method named insertSchema3 which generates the dataset of schema3 database.

Dataset Population

Like the insert, in each class there is a method that generates each database's table data. For example, in Schema3 class, there are three methods whose name starts with "populate". Each of these methods, generates data for each table that is in schema3 database.

Schemas & Queries

Following is a description of the 4 schemas used in this project. Each one is a separate schema/database. After each schema, there are one or more SQL statements. In total there are 12 queries you are going to optimize. Your goal will be analyze those queries and try to improve their performance by tuning the database engine.

Performance Tuning and Measurement [what is required!]

After creating each schema, and populating it with specified data, update the database statistics to ensure that PostgreSQL has went through your data and collected all necessary information to do accurate estimation. Next, you need to analyze the execution of each query (check resources section at end of this document for how to update the statistics and how see the plan for a query). The first time you are going to analyze will be done without the existence of any index. Next, you need to inspect each query, and decide which column or columns, you think if you create an index on, it is going to speed the execution of this query. After creating the index, run analyze again to see what the engine does with the existence of an index. You will do this process 3 times using the following indices which are supported in PostgreSQL: first with B+ Tree indices only, second with Hash based indices only, and third with mixed indices where you use any number/type of indices in PostgreSQL at the same time. Thus, you will report than 4 scenarios: 1) without an index, 2) with B+ trees indices only, 3) with hash indices only, and 4) with mixed indices.

The goal behind this exercise is to show you how to discover what plan the query optimizer is using, and whether (or not) an index makes a difference and which index does enhance performance. For each of the above scenarios, you need to answer the following questions:

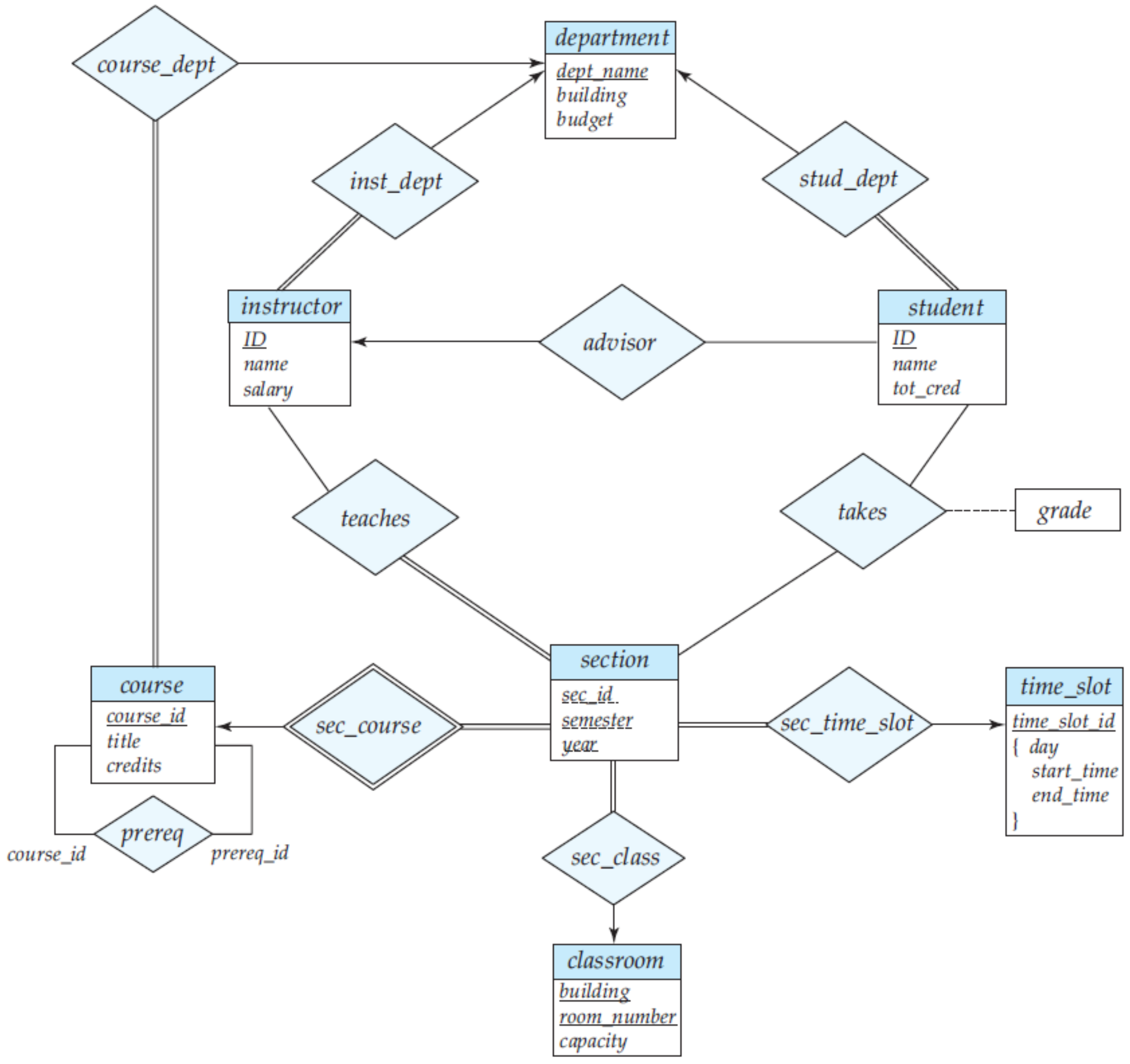
Without an index:

1) What is the execution plan of the query? What is the estimated cost of the plan?

With an index:

2) What is the execution plan of the query? What is the estimated cost of the plan?

Schema 1



Modifications to Insertion Code:

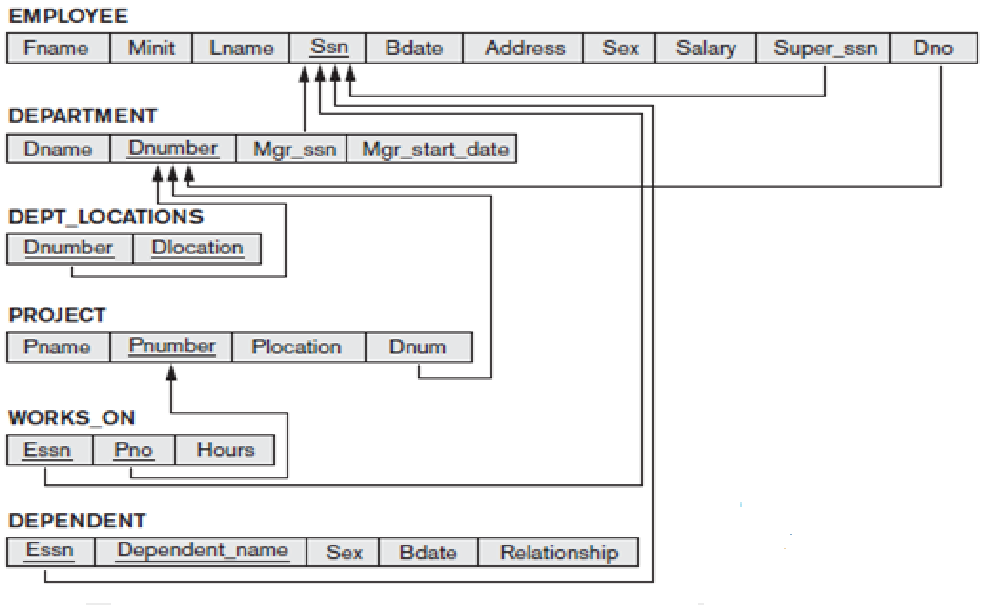
The insertion code provided to you inserts dummy values. You are required to change it to have 15 departments, each offering between 10 and 15 courses. Each department would have between 7 and 12 instructors. Each department would have at least 400 students. Before inserting the data, check query 1 below you are going to analyze, the data that you insert must have values used in those queries. For example, you need to have a CS1 department, and a year 2019 and semester 1.

Query 1:

"Display a list of all students in the CS1 department, along with the course sections, if any, that they have taken in Semester 1 2019; all course sections from Spring 2009 must be displayed, even if no student from the CS1 department has taken the course section." This query can be written as:

```
select *
from (select *
      from student
      where
      department = 'CS1') as CS1_student
      natural 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;
```

Schema 2



Modifications to Insertion Code:

The insertion code provided to you inserts dummy values. You are required to change it to have 5000 employees, 30 departments, 20 department locations, and 600 projects. Before inserting the data, check queries 2-6 below you are going to analyze, the data that you insert must have values used in those queries. For example, you need to have an employee with name employee1 and department 5 and several who make a salary greater than 40,000. All queries must return something (an empty result set is not acceptable).

Query 2:

```
select distinct pnumber
from project
where pnumber in
                (select pnumber
                 from project, department d, employee e
                 where e.dno=d.dnumber
                      and
                      d.mgr_snn=ssn
                      and
                      e.lname='employee1' )
                or
                pnumber in
                      (select pno
                       from works_on, employee
                       where essn=ssn and lname='employee1' );
```

Query 3:

Select the names of employees whose salary is greater than the salary of all the employees in department 5

```
select lname, fname
from employee
where salary > all (
      select salary
      from employee
      where dno=5 );
```

Query 4:

Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
select e.fname, e.lname
from employee as e
where e.ssn in (
           select essn
           from dependent as d
           where e.fname != d.dependent_name
           and
           e.sex!=d.sex );
```

Query 5:

Retrieve the names of employees who have no dependents.

```
select fname, lname
from employee
where exists ( select *
                from dependent
               where ssn=essn );
```

Query 6:

For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
and
dno =  (
       select dno
       from employee
       group by dno
       having count (*) > 2)
group by dnumber;
```

Schema 3

Sailors(<u>*sid:*integer</u>, *sname:* string, *rating:* integer, *age:* real)

Boats(<u>*bid:*integer</u>, *bname:* string, *color:* string)

Reserves (<u>*sid:* integer, *bid:* integer</u>, *day:* date)

Modifications to Insertion Code:

The insertion code provided to you inserts dummy values. You are required to change it to have 9000 sailors, 3000 boats, and 15000 reserves. Before inserting the data, check queries 7-9 below you are going to analyze, the data that you insert must have values used in those queries (an empty result set is not acceptable).

Query 7:

Find the names of sailors who have reserved boat 103.

```
select s.sname
from sailors s
where
s.sid in(  select r.sid
           from reserves r
           where r.bid = 103 );
```

Query 8:

Find the names of sailors 'who ha'ue reserved a red boat.

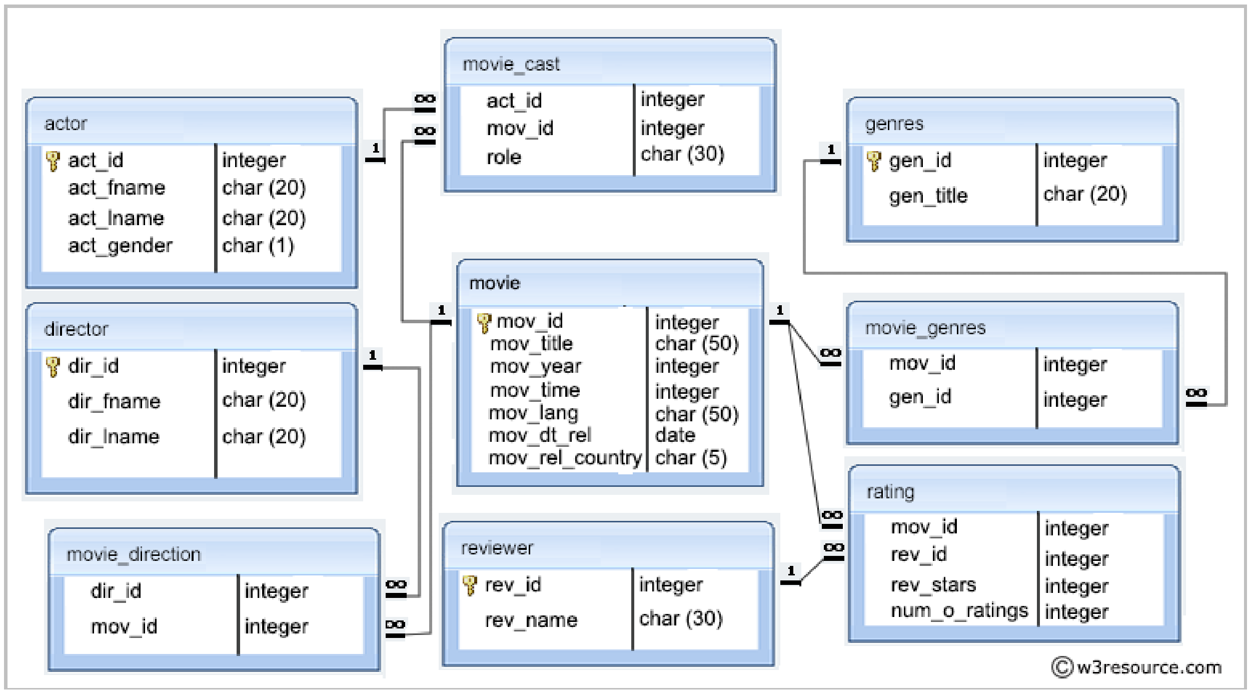
```
select s.sname
from sailors s
where s.sid in ( select r.sid
                 from reserves r
                 where r. bid in (select b.bid
                                  from boat b
                                  where b.color = 'red'));
```

Query 9:

Find the names of sailors who have reserved both a red and a green boat.

```
select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in ( select s2.sid
           from sailors s2, boat b2, reserves r2
           where s2.sid = r2.sid
           and
           r2.bid = b2.bid
           and
           b2.color = 'red');
```

Schema 4



Modifications to Insertion Code:

The insertion code provided to you inserts dummy values. You are required to change it to have 1000 movies, 12000 actors, and 2000 directors. Before inserting the data, check queries 10-12 below you are going to analyze, the data that you insert must have values used in those queries (an empty result set is not acceptable).

Query 10:

List all the information of the actors who played a role in the movie 'Annie Hall'.

```
select *
from actor
where act_id in(
           select act_id
           from movie_cast
           where mov_id in(
                           select mov_id
                           from movie
                           where mov_title ='movie1'));
```

Query 11:

Find the name of the director (first and last names) who directed a movie that casted a role for 'Eyes Wide Shut'.

```
select dir_fname, dir_lname
from  director
where dir_id in(
           select dir_id
           from movie_direction
           where mov_id in(
                select mov_id
                from movie_cast
                where role =any( select role
                                from movie_cast
                                where mov_id in(
                                           select  mov_id
                                           from movie
                                           where
                                           mov_title='movie
                                           2'))));
```

Query 12:

Find the titles of all movies directed by the director whose first and last name are Woddy Allen.

```
select mov_title
from   movie
where  mov_id=(
                select mov_id
                from movie_direction
                where dir_id=
                     (select dir_id
                     from director
                     where dir_fname='actor1'
                     and
                     dir_lname='actor1'));
```

Resources

- Introduction to SQL Queries performance measurement in PostgreSQL

  https://www.citusdata.com/blog/2018/03/06/postgres-planner-and-its-usage-of-statistics/

  https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

- Advanced Tuning:

  https://www.postgresql.org/docs/13/static/runtime-config.html

- Create Statistics:

  https://www.postgresql.org/docs/13/sql-createstatistics.html

- How to use EXPLAIN command and understand its output & other stuff:

  https://www.postgresql.org/docs/13/performance-tips.html

  http://www.postgresql.org/docs/13/static/sql-explain.html

- Creating an Index for a table:

  http://www.postgresql.org/docs/13/static/sql-createindex.html

- Statistics collected by PostgreSQL:

  https://www.postgresql.org/docs/13/static/planner-stats.html

- Query Planner Configuration hint in PostgreSQL:

  The SET command can be used to change the behavior of the query planner.

  For example, “SET enable_nestloop=false;” command in PostgreSQL disables the query planner’s use of nested loop join plans.