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

Dr. Wael Abouelsaadat

Project II

Course Weight: 6%

Release Date: April 27th , 2024 **Due Date: 11:59PM, May 22nd, 2024**

Team Size and Work Plan

- It is preferred that you do this project in the same team you did Project 1 with. In either case, include in your submission a file with your team members' name and ID.
- 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 resources section at end of this doc,
  3) modify given Java code to do data insertion,
  4) analyze given queries,
  5) document PostgreSQL output in a report, and
  6) discuss your findings from PostgreSQL output
- Analyzing and writing efficient 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 required to run, analyze and document the performance of the queries (with and without indices, in each of the isolation levels).

Submission

- Your submission should include:
    - A **teams.txt** including your team member names and student ids.
    - 4 PostgreSQL log (*.csv) files, one for each database. The log is proof you have done the work.
    - A **PDF report**, including:
        1) Screen capture of output from querying pg_locks (more about that later in the document).
        2) Screen capture of output from PostgreSQL analyze 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.
        4) Discussion why did PostgreSQL best plan for each query (why is it considered the best? Why did PostgreSQL pick that specific one?)
- 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 15) 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 four inputs:

1. a shell script **script.sh** which creates 4 different databases and their respective tables.
2. a **README.txt** file about how to run the shell file and the file also has few tips about connecting from Java to PostgreSQL.

3. 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.
4. The given queries you need to analyze in a text file **sql-queries.txt**
5. A video showing how to setup the software (thanks to TA Ahmad Hassan for preparing that).

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

Notes:

- running the script is not mandatory, you can open the file using any text editor and copy and paste the table creation statements in PgAdmin (after connecting to installed PostgreSQL from PgAdmin).

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"

Note: Make sure to have only one jdbc driver imported. There is already one imported in the project provided to you. You will need to edit the path to that or remove it and add your own. You can do that in Eclipse IDE from Build Path -> Configure Build Path...

PostgresSQL-Java Connection

At this step, all compile errors should be resolved. You will find 4 classes inside the project. Each class generates a data set for each of the 4 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 analyze. Your goal will be analyze those queries and try to improve their performance by tuning the database engine.

Step 1: Enabling Logging

- To enable us to verify that you did the work, and for you to learn about PostgreSQL logs, you are required to enable logging with the below configuration. You will type the below in the shell before doing anything else for each schema (i.e. before creating the schema, the tables, and filling them with data).
- You will be submitting the log file for each with your submission. You can zip it, before submission, since it might be large in size.
- Log configuration on command line in shell:

```
SET log_destination TO 'csvlog';
SET logging_collector TO on;
SET log_directory TO '/var/log/postgresql';
SET log_filename TO 'postgresql-%Y-%m-%d.csv';
SET log_statement TO 'all';
SET log_min_messages TO 'DEBUG5';
SET log_min_duration_statement TO 10;
SET log_connections TO on;
SET log_disconnections TO on;
SET log_lock_waits TO on;
```

- Note: set an appropriate location in log_directory depending on where you want to have the logs stored on your hard disk.
- Read this to learn about the parameters: https://www.postgresql.org/docs/current/runtime-config-logging.html

Step 2: Creating Tables and Populating Data

- In PostgreSQL, you can query the pg_locks system table view to get information about currently held locks. Each row in pg_locks represents a lock held by a session. Use this query to inspect the currently allocated locks

```
SELECT * FROM pg_locks;
```

- The above query will return currently allocated locks across all sessions in the database. Keep in mind that PostgreSQL uses various types of locks for different purposes, so the count may include locks for rows, tables, transactions, etc.
- You are going to query postgresql locks table to find what locks are allocated in 3 phases: after creating tables with no data, while filling in the table with data, and after filling in the tables with data. **Document and compare the outputs in your report**.
    - Comment in your report about the different lock types you will see: `AccessShareLock`, `RowShareLock`, `RowExclusiveLock`, `ShareUpdateExclusiveLock`, `ShareLock`, `ShareRowExclusiveLock`, `ExclusiveLock`, `AccessExclusiveLock`. Explain what each means and why PostgreSQL was using that for your schema.
- Finally, update the database statistics to ensure that PostgreSQL has gone through your data and collected all necessary information to do accurate estimation.

Step 3: Performance Tuning and Measurement

- Next, you need to analyze the execution of the given queries (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 the 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 that index. You will report 7 scenarios for a given query:

    1) given query without an index,

    2) given query with B+ trees indices only (any number of B+ trees)

    3) given query with hash indices only (any number of hashes)

    4) given query with BRIN indices only (any number of BRINs)

    5) given query with GIN index (if applicable).

    6) given query with mixed indices on all columns (every column must have an index created. More than one index type must be used).

7) given query with your opinionated best mix of indices (i.e. you decide that the best combination is to use B+ tree for one column, hash for a second and no other index is going to enhance the performance further – even if there are many other columns).

Note: you will need in several cases to **set flags** related to indices to force PostgreSQL to use your desired index in the execution plan. Here is a list of most used flags:

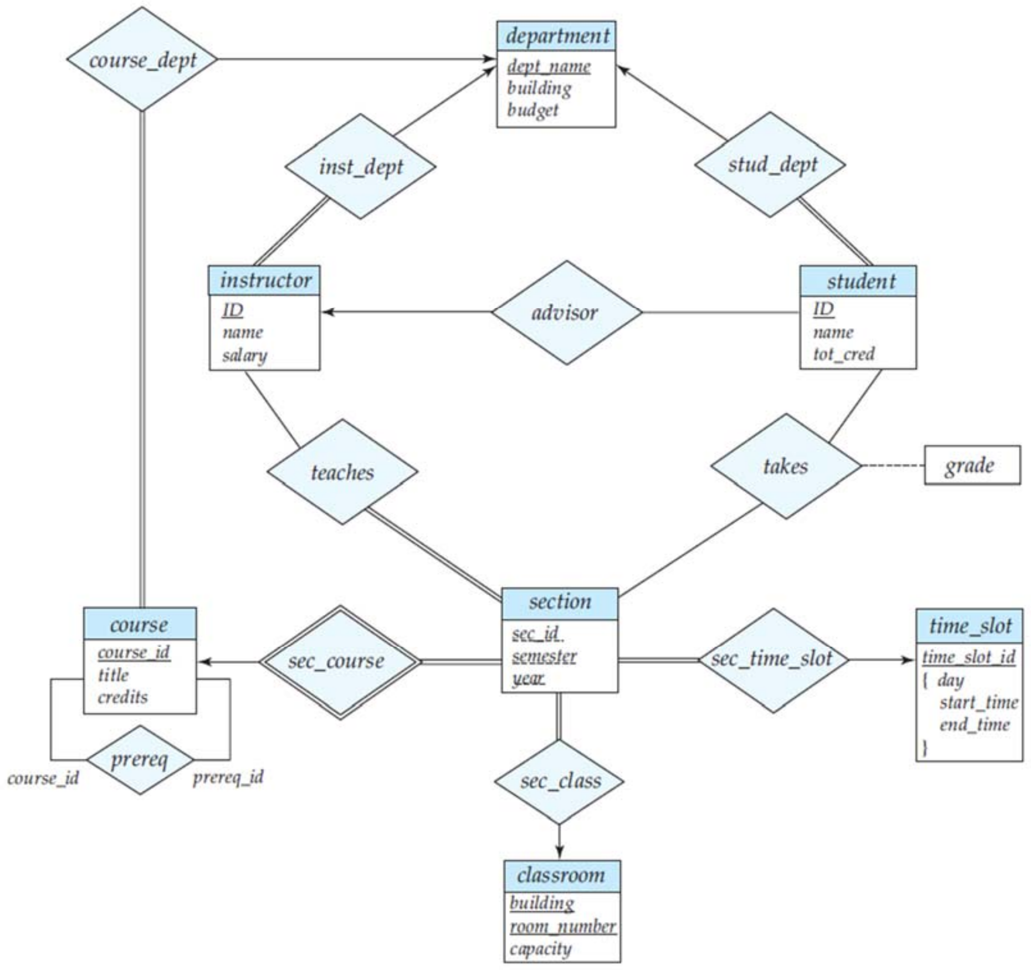
`enable_indexscan`:	enables/disables the use of index scans in query plans.
`enable_bitmapscan`:	enables/disables the use of bitmap index scans in query plans.
`enable_indexonlyscan`:	enables/disables the use of index-only scans in query plans.
`enable_seqscan`:	enables/disables the use of sequential scans in query plans.
`enable_nestloop`:	enables/disables use of nested loop join algorithms in query plans.
`enable_mergejoin`:	enables/disables the use of merge join algorithms in query plans.
`enable_hashjoin`:	enables/disables the use of hash join algorithms in query plans.

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 3 main questions:

What is the execution plan of the query? What is the estimated cost of the plan? Why?

Schema 1



Modifications to Insertion Code:

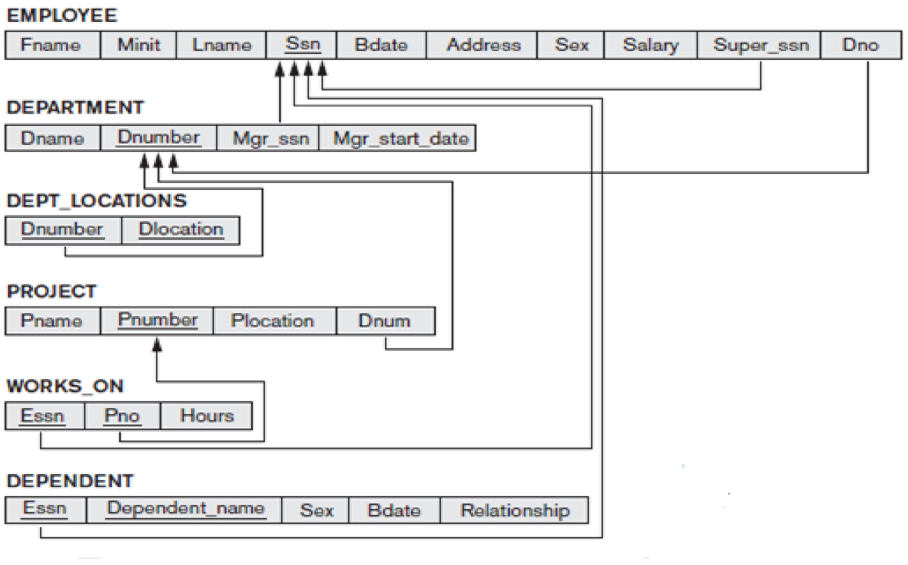
The insertion code provided to you inserts dummy values. You are required to change it to have 60 departments, each offering between 20 and 45 courses. Each department would have between 10 and 35 instructors. Each department would have at least 1100 students. Before inserting the data, check query 1 below you are going to analyze. **The inner result set (semester = 1 and year = 2024) should have 250 rows** at least.

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 2024; all course sections from Spring 2024 must be displayed, even if no student from the CSEN department has taken the course section.” This query can be written as:

```
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 = 2024) as sem1_student
          on CS1_student.id = sem1_student.student_id;
```

Schema 2



Modifications to Insertion Code:

The insertion code provided to you inserts dummy values. You are required to change it to have 16000 employees, 150 departments, 150 department locations, and 9200 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. **Each query must return 600 rows in the result set** (an empty result set is not acceptable) – with the exception of query 6 whose result set size is 100 rows.

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 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 (*) > 5)
group by dnumber;
```

Schema 3

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

Boats(<u>*bid:*</u>integer, *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 19000 sailors, 3000 boats, and 35000 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 Note: in order to see different decisions made internally by the database engine in the query plan and index selection, you must return a non-trivial number of rows in the result set (in the hundreds): 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.

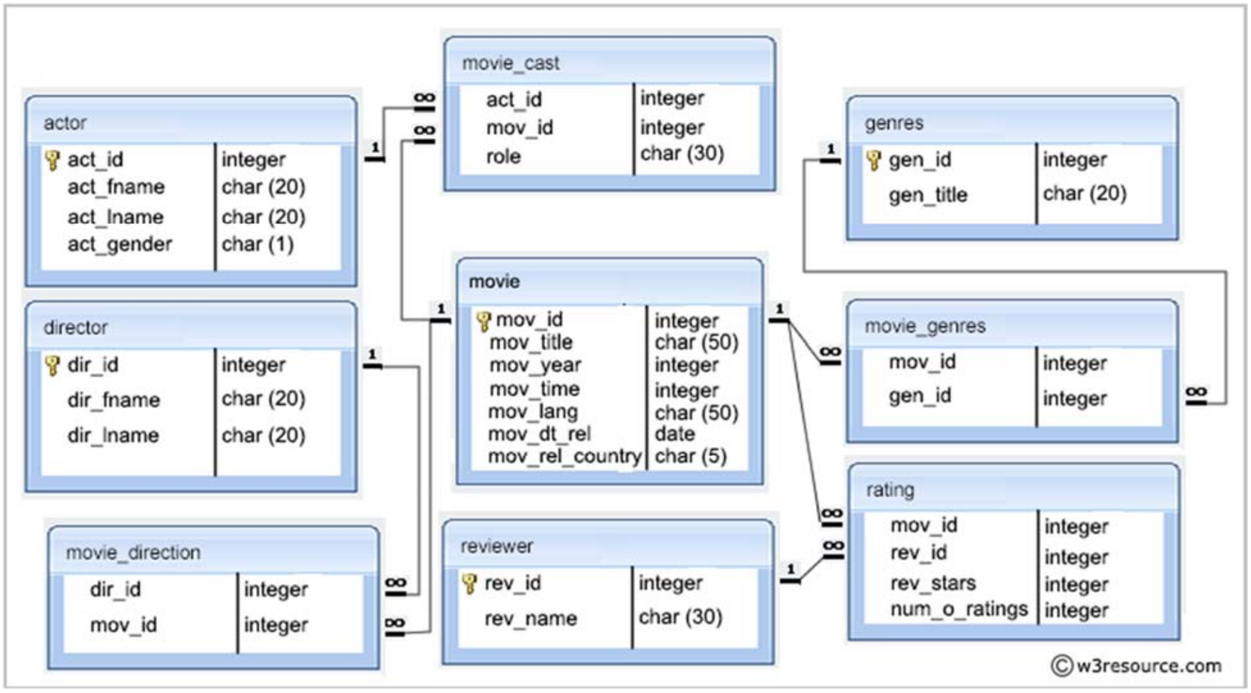
```
select s.sname
from sailors s
where s.sid in ( select r.sid
                from reserves r
                where r. bid in (select b.bid
                                from boat b
                                where b.color = 'red'));
```

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 = 'green');
```

Schema 4



Modifications to Insertion Code:

The insertion code provided to you inserts dummy values. You are required to change it to have 100,000 movies, 120000 actors, and 6000 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'. **Your result set must have 222 rows.**

```
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 ='Annie Hall'));
```

Query 11:

Find the name of the director (first and last names) who directed a movie that casted a role for 'Eyes Wide Shut'. Empty result set not acceptable.

```
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='Eyes
                                           Wide Shut'))));
```

Query 12:

Find the titles of all movies directed by the director whose first and last name are Woddy Allen. **Your result set must contain 350 rows.**

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

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/15/static/runtime-config.html

- *Create Statistics*

  https://www.postgresql.org/docs/15/sql-createstatistics.html

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

  https://www.postgresql.org/docs/15/performance-tips.html

  http://www.postgresql.org/docs/15/static/sql-explain.html

- *Creating an Index for a table:*

  http://www.postgresql.org/docs/15/static/sql-createindex.html

- *Statistics collected by PostgreSQL:*

  https://www.postgresql.org/docs/15/static/planner-stats.html

- *Trouble shooting an Index:*

  https://www.gojek.io/blog/the-case-s-of-postgres-not-using-index

  https://www.pgmustard.com/blog/why-isnt-postgres-using-my-index