Experiment No.1

Title: Execution of Parallel Database queries.

Batch:SY-IT(B3) **Roll No.: 16010423076** **Experiment No.: 1**

Aim: To execute Parallel Database queries.

Resources needed: PostgreSQL 9.3

Theory

A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. Centralized and client–server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

Types of parallelism :

- Interquery parallelism: Execution of multiple queries in parallel

- Interoperation parallelism: Execution of single queries that may consist of more than one operations to be performed.

  Independent Parallelism - Execution of each operation individually in different processors only if they can be executed independent of each other. For example, if we need to join four tables, then two can be joined at one processor and the other two can be joined at another processor. Final join can be done later.

  Pipe-lined parallelism - Execution of different operations in pipe-lined fashion. For example, if we need to join three tables, one processor may join two tables and send the result set records as and when they are produced to the other processor. In the other processor the third table can be joined with the incoming records and the final result can be produced.

- Intraoperation parallelism Execution of single complex or large operations in parallel in multiple processors. For example, ORDER BY clause of a query that tries to execute on millions of records can be parallelized on multiple processors.

Procedure:

Parallel queries provide parallel execution of sequential scans, joins, and aggregates etc.

Parallel queries provide parallel execution of sequential scans, joins, and aggregates. To make the performance gains need a lot of data.

```
create table ledger (

  id serial primary key,

  date date not null,

  amount decimal(12,2)  not null

);

insert into ledger (date, amount)

  select current_date - (random() * 3650)::integer,

  (random() * 1000000)::decimal(12,2) - 50000

  from generate_series(1,500000);
```

```
explain analyze select sum(amount)  from ledger;
```

Reading the output, we can see that Postgres has chosen to run this query sequentially.

Parallel queries are not enabled by default. To turn them on, we need to increase a config param called max_parallel_workers_per_gather.

```
show max_parallel_workers_per_gather;
```

Let's raise it to four, which happens to be the number of cores on this workstation.

```
set max_parallel_workers_per_gather to 4;
```

Explaining the query again, we can see that Postgres is now choosing a parallel query.
And it's about four times faster.

```
explain analyze select sum(amount)  from ledger;
```

The planner does not always consider a parallel sequential scan to be the best option. If a query is not selective enough and there are many tuples to transfer from worker to worker, it may prefer a "classic" sequential scan.PostgreSQL optimises the number of workers according to size of the table and the min_parallel_relation_size.

Similar ways we can execute join operation and check parallel execution of sequential join.

Create two tables with names lilbrary1 and library2 as follows

```
create table library1 (

  id serial primary key,

  quantity int  not null,

  location varchar(50)  not null

);

create table library2 (

  id serial primary key,

  quantity int  not null,

  location varchar(50)  not null

);
```

```
explain analyse select library1.id,library1.quantity,library2.location from
library2,library1 where library1.id=library2.id;

SET max_parallel_workers_per_gather TO 3;

explain analyse select library1.id,library1.quantity,library2.location from
library2,library1 where library1.id=library2.id;
```

Questions:

1. **Explain the parallelism achieved in the experiment you performed.**

   In this experiment, parallelism was achieved by enabling multiple processors to work together on database queries.

- Initially, queries were executed one step at a time by a single processor.
- After enabling parallelism in PostgreSQL (by increasing max_parallel_workers_per_gather), the workload was distributed among multiple processors. This allowed tasks like calculating the SUM of values or performing joins between tables to be executed simultaneously, significantly speeding up the process.

2. **With comparison of the results explain how degree of parallelism (no of parallel processors) affect the operation conducted.**

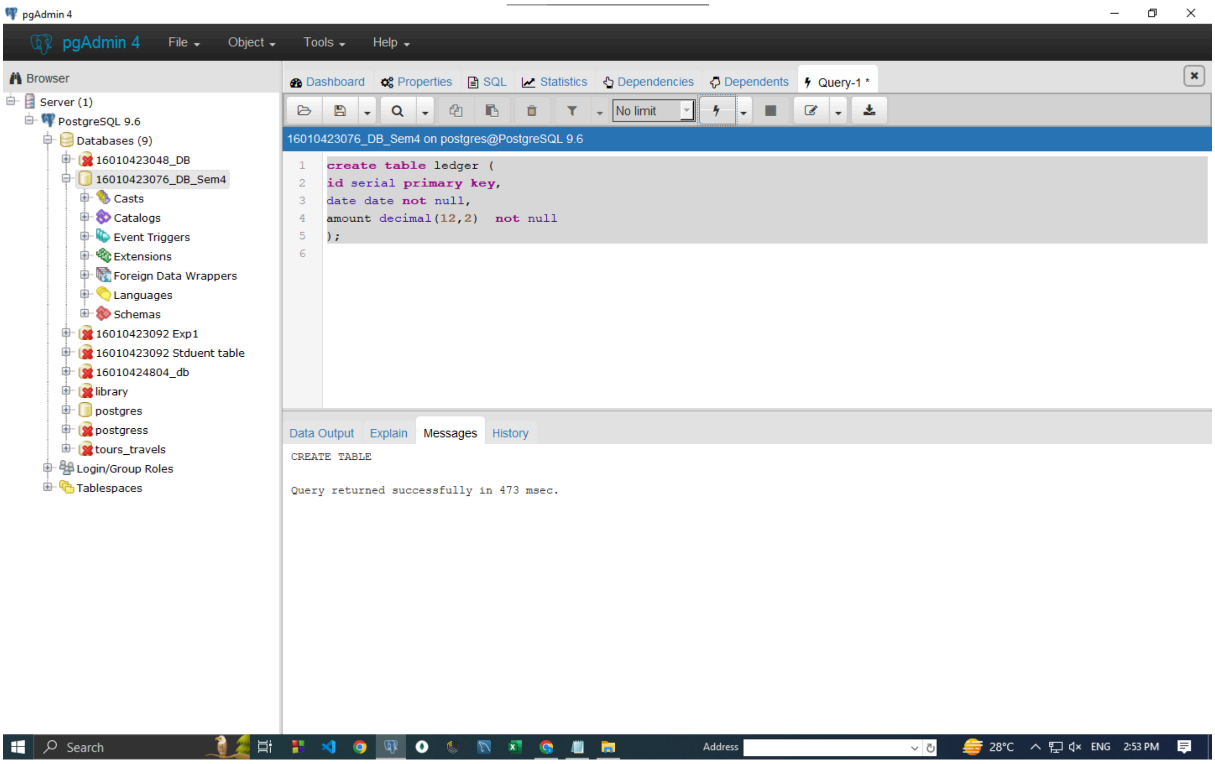
   The results showed that increasing the number of parallel processors reduced the query execution time. For instance:

   When parallelism was disabled, the query took longer because one processor handled the entire task.

   When the degree of parallelism was increased (e.g. 4 processors), the task was divided among them, completing it faster.

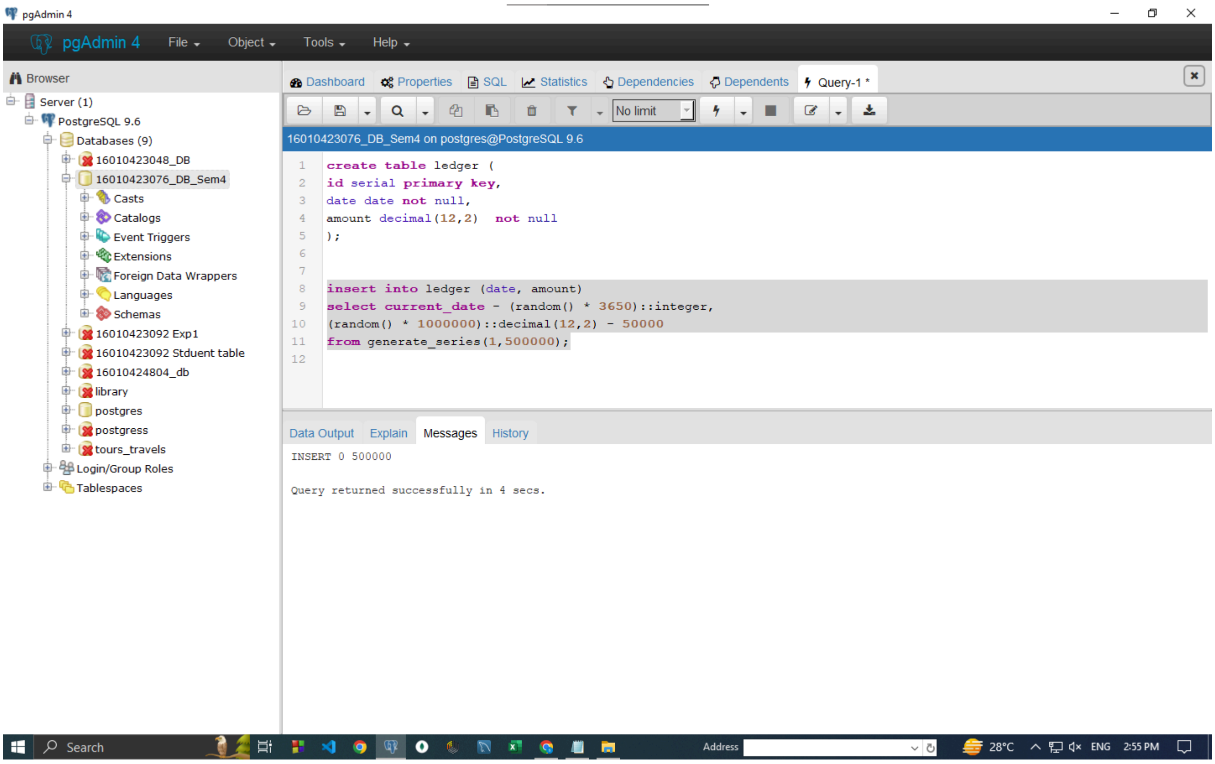
   However, it's important to note that too many parallel workers for a small dataset or non-selective queries might not always improve performance, as communication between processors can add overhead. PostgreSQL optimizes this automatically based on the query and data size.

Results: (Program printout with output)



```
create table ledger (
id serial primary key,
date date not null,
amount decimal(12,2)  not null
);
```

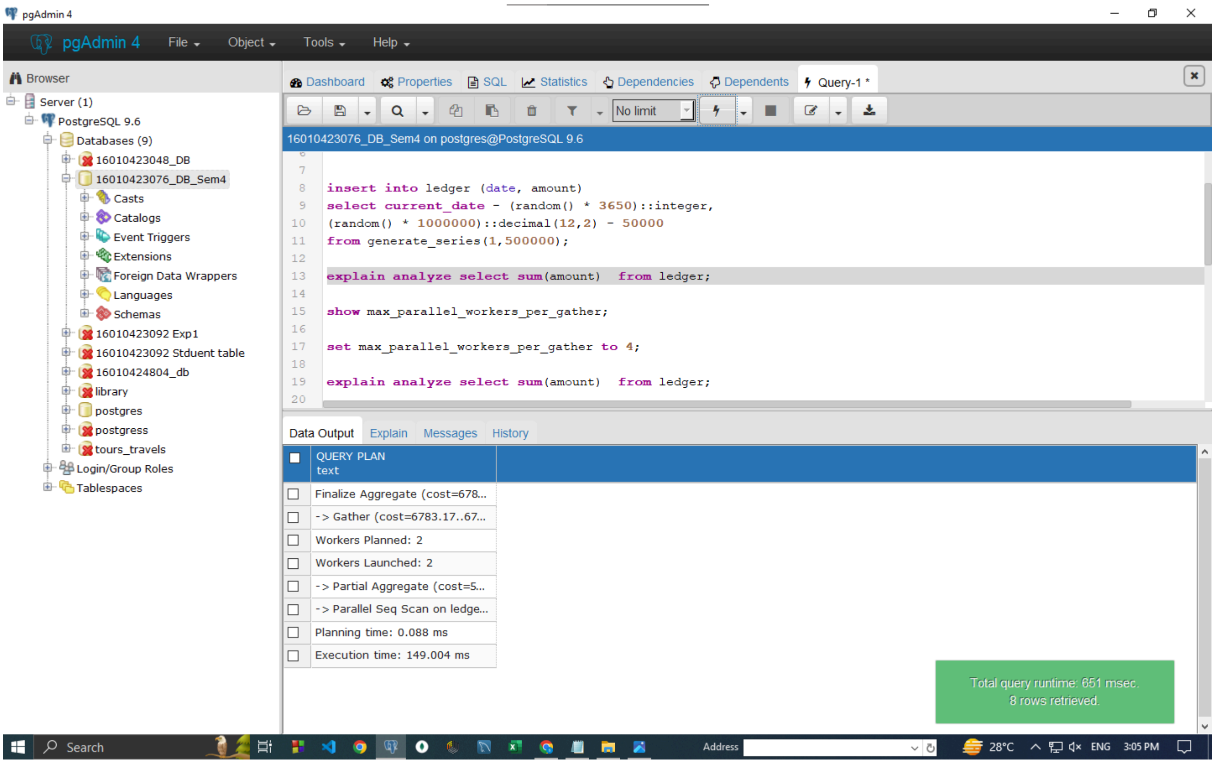
```
CREATE TABLE

Query returned successfully in 473 msec.
```



16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
create table ledger (
id serial primary key,
date date not null,
amount decimal(12,2)  not null
);


insert into ledger (date, amount)
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,500000);
```

```
INSERT 0 500000

Query returned successfully in 4 secs.
```



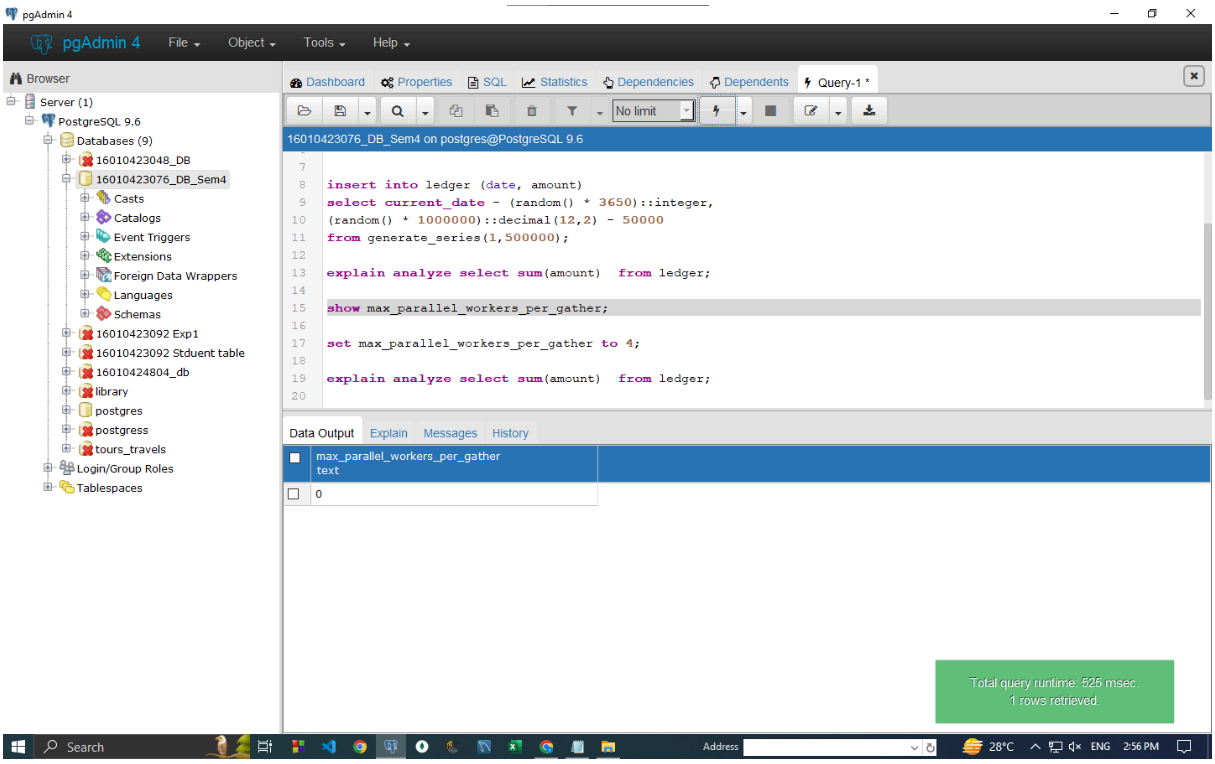
16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```

insert into ledger (date, amount)
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,500000);

explain analyze select sum(amount)  from ledger;

show max_parallel_workers_per_gather;

set max_parallel_workers_per_gather to 4;

explain analyze select sum(amount)  from ledger;

```

□	QUERY PLAN text
□	Finalize Aggregate (cost=678...
□	-> Gather (cost=6783.17..67...
□	Workers Planned: 2
□	Workers Launched: 2
□	-> Partial Aggregate (cost=5...
□	-> Parallel Seq Scan on ledge...
□	Planning time: 0.088 ms
□	Execution time: 149.004 ms

Total query runtime: 651 msec.
8 rows retrieved.

First the Sum operation took **651 msec** time for execution.

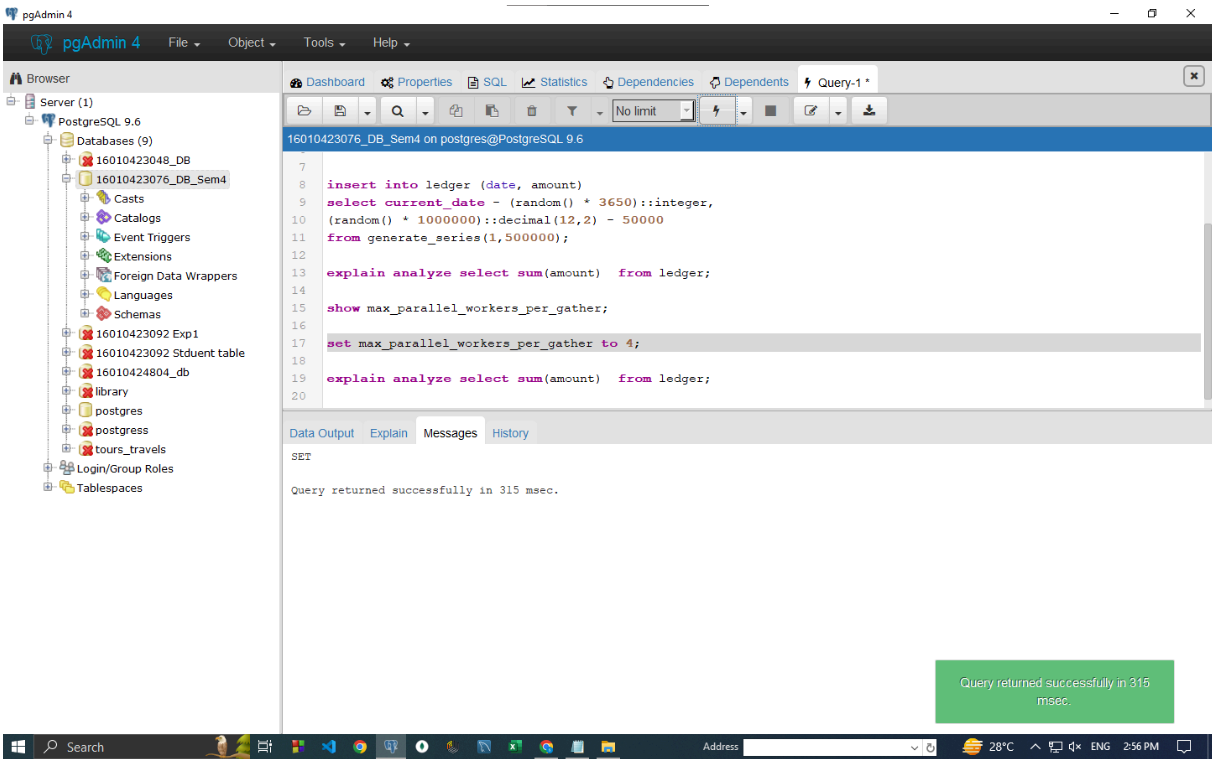


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
insert into ledger (date, amount)
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,500000);

explain analyze select sum(amount)  from ledger;

show max_parallel_workers_per_gather;

set max_parallel_workers_per_gather to 4;

explain analyze select sum(amount)  from ledger;
```

max_parallel_workers_per_gather text
0

Total query runtime: 525 msec.
1 rows retrieved.

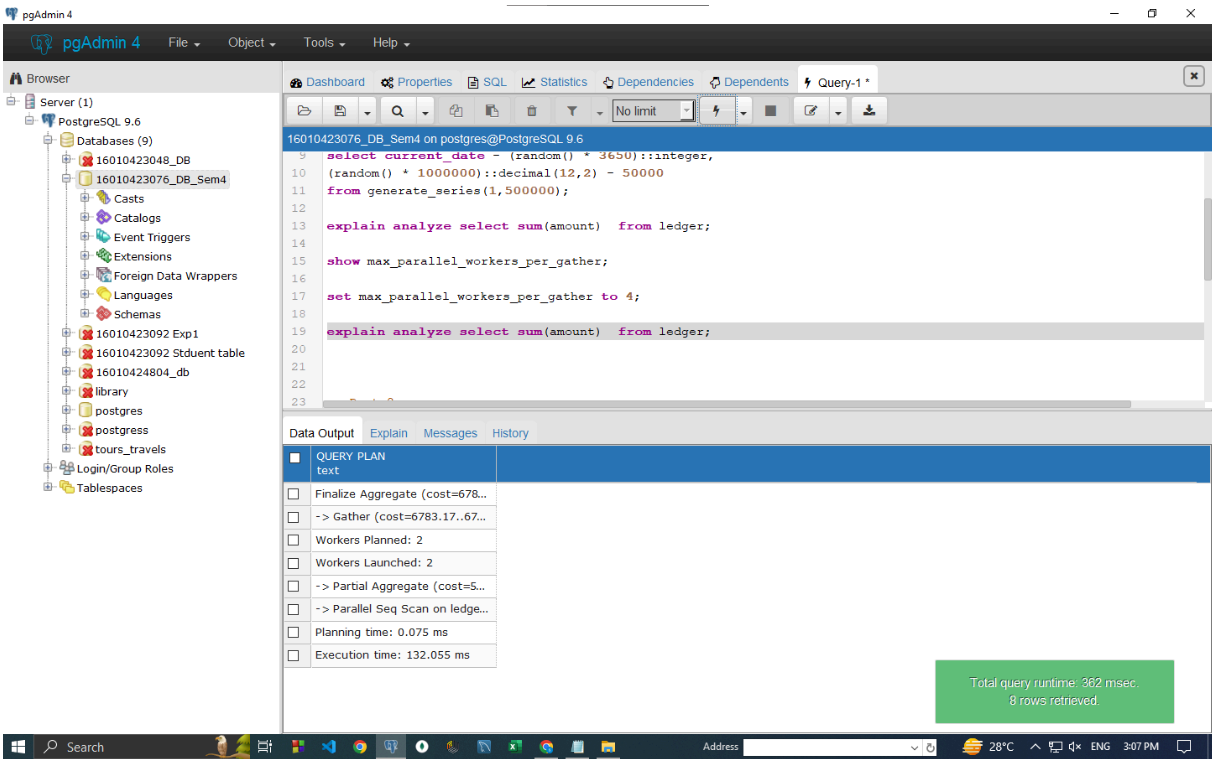


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
insert into ledger (date, amount)
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,500000);

explain analyze select sum(amount)  from ledger;

show max_parallel_workers_per_gather;

set max_parallel_workers_per_gather to 4;

explain analyze select sum(amount)  from ledger;
```

```
SET

Query returned successfully in 315 msec.
```

Query returned successfully in 315 msec.



16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

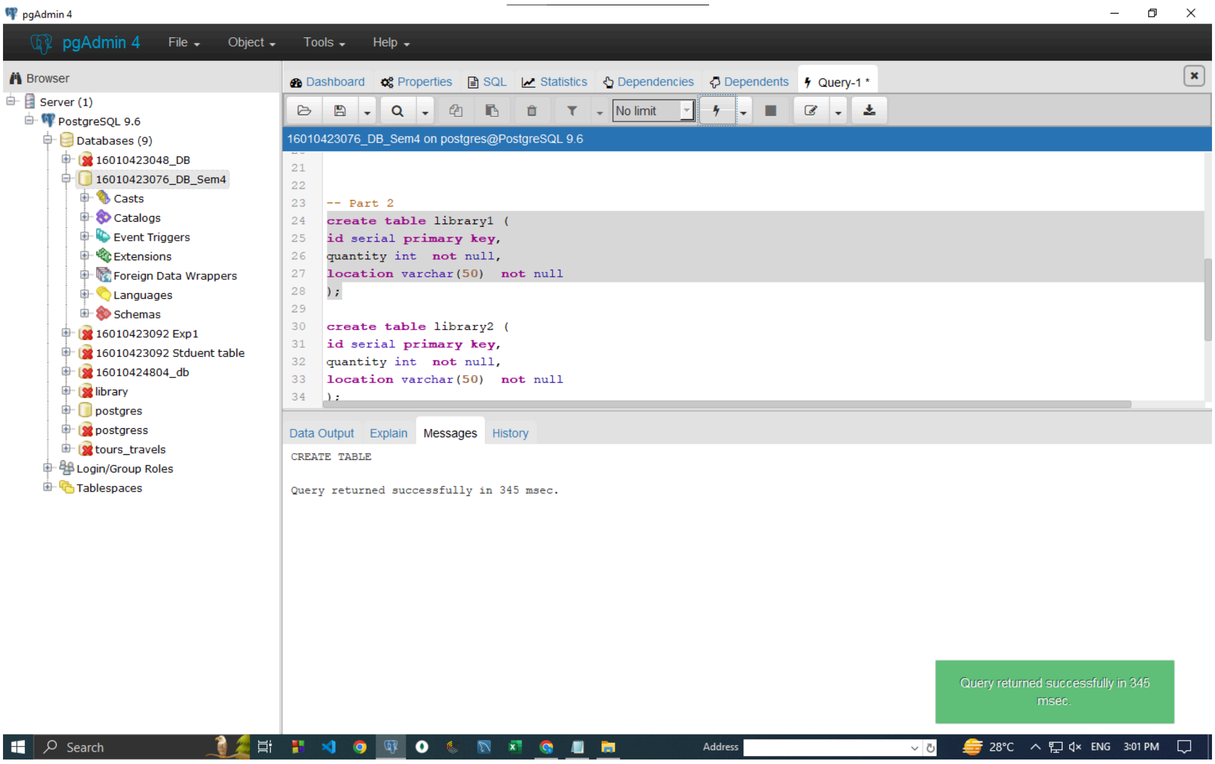
```
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,500000);

explain analyze select sum(amount)  from ledger;

show max_parallel_workers_per_gather;

set max_parallel_workers_per_gather to 4;

explain analyze select sum(amount)  from ledger;
```

```
QUERY PLAN
text
Finalize Aggregate (cost=678...
-> Gather (cost=6783.17..67...
Workers Planned: 2
Workers Launched: 2
-> Partial Aggregate (cost=5...
-> Parallel Seq Scan on ledge...
Planning time: 0.075 ms
Execution time: 132.055 ms
```

Total query runtime: 362 msec.
8 rows retrieved.

After setting parallel workers to 4 the same SUM query took **362 msec** for execution.

Part 2 : Similarly testing parallel execution for two tables.

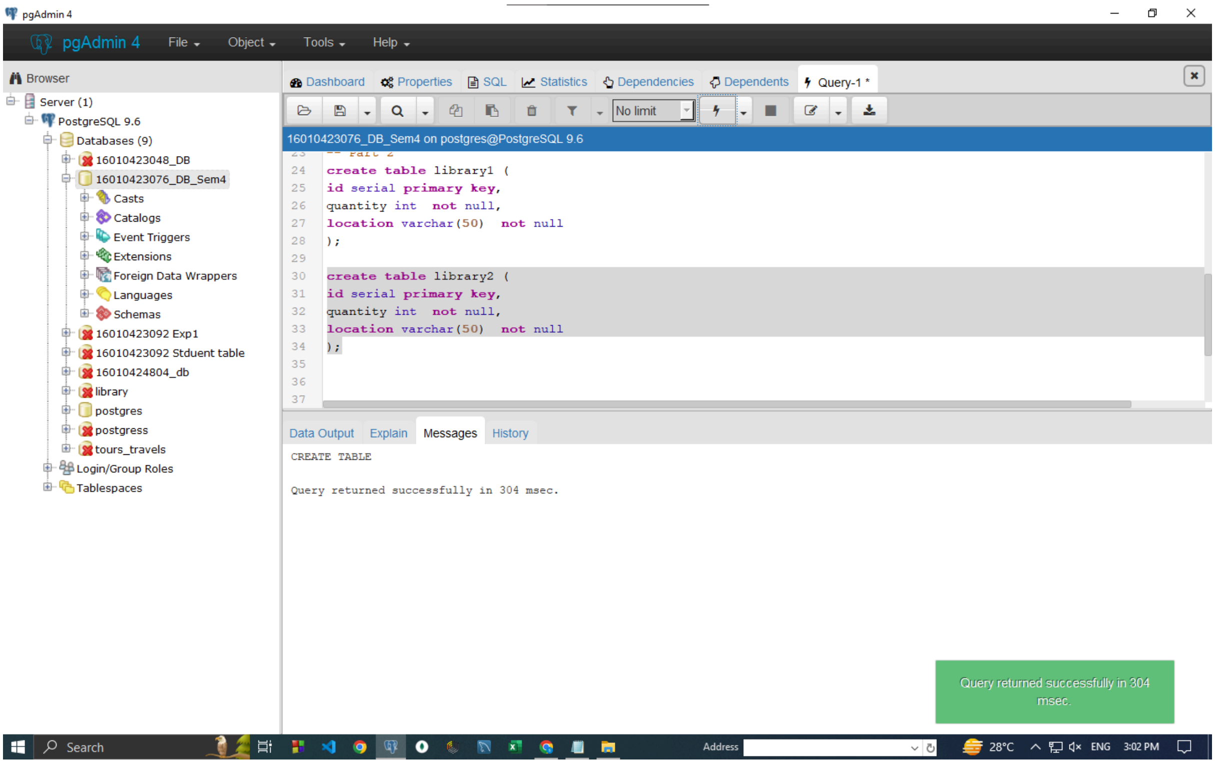


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
-- Part 2
create table library1 (
id serial primary key,
quantity int  not null,
location varchar(50)  not null
);

create table library2 (
id serial primary key,
quantity int  not null,
location varchar(50)  not null
);
```

```
CREATE TABLE

Query returned successfully in 345 msec.
```

Query returned successfully in 345 msec.

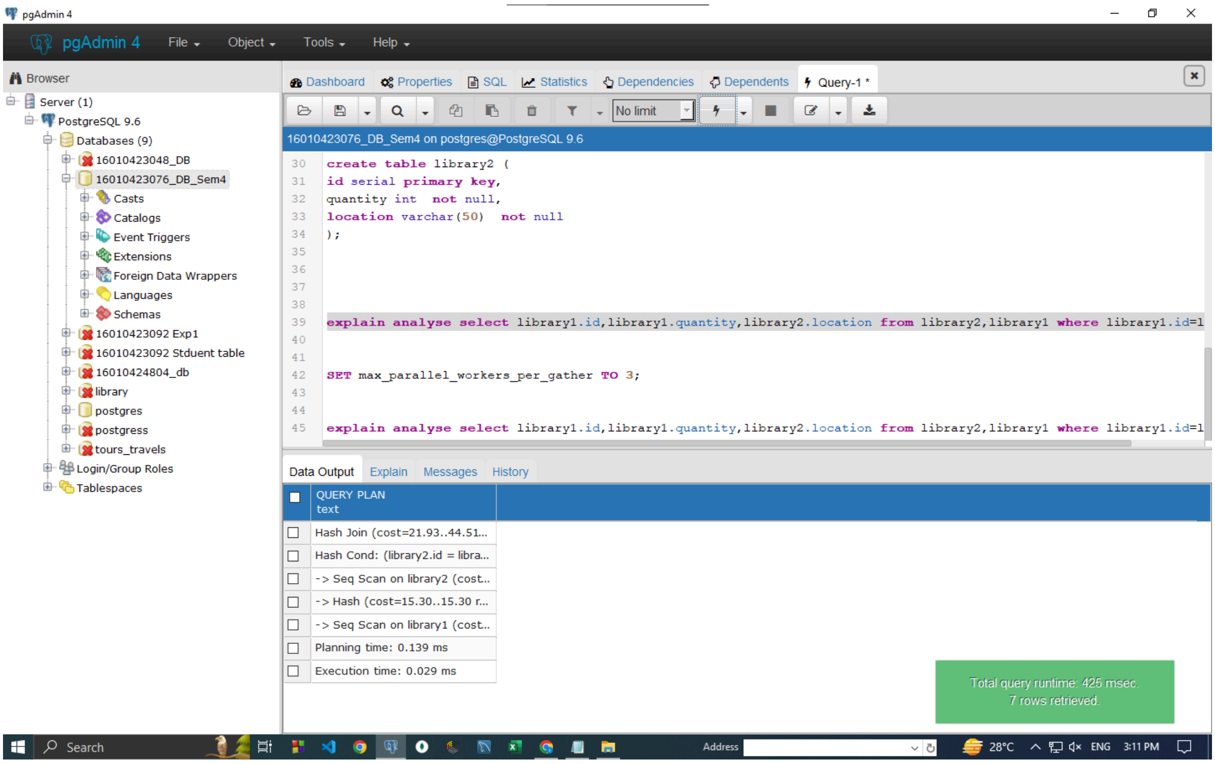


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
create table library1 (
id serial primary key,
quantity int  not null,
location varchar(50)  not null
);

create table library2 (
id serial primary key,
quantity int  not null,
location varchar(50)  not null
);
```

```
CREATE TABLE

Query returned successfully in 304 msec.
```

Query returned successfully in 304 msec.

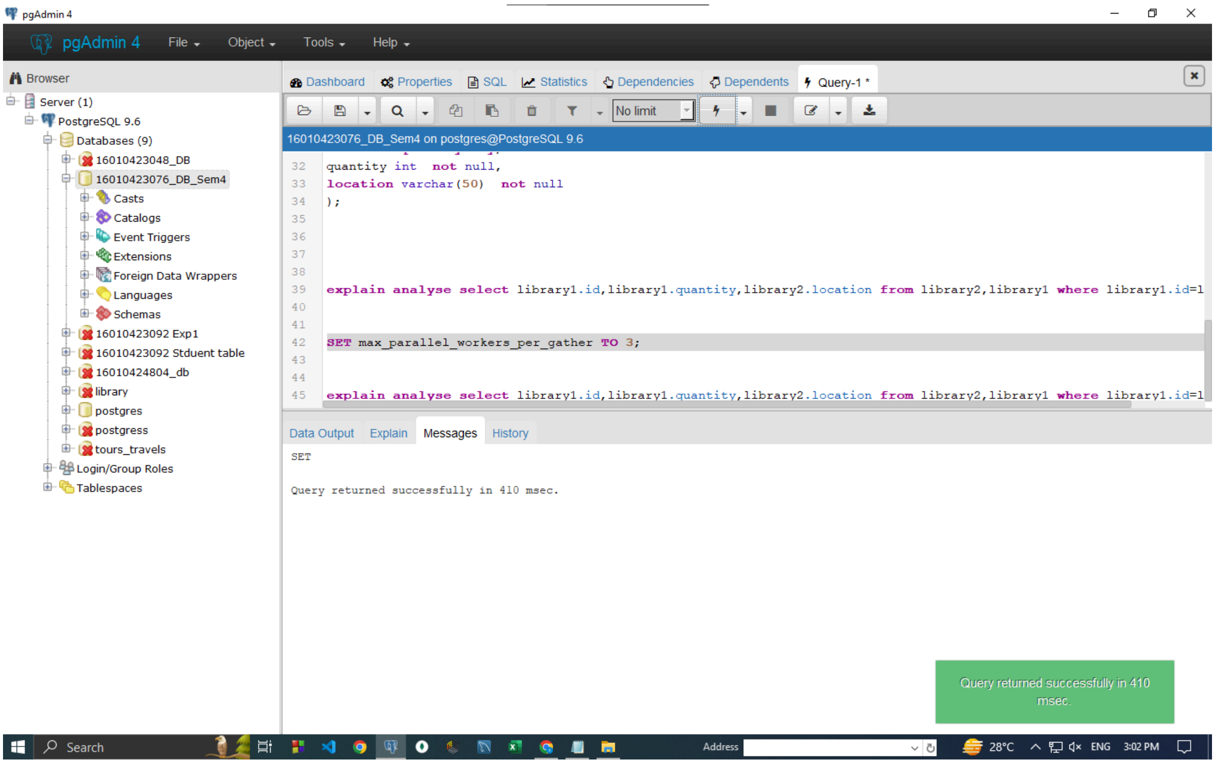


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
create table library2 (
id serial primary key,
quantity int  not null,
location varchar(50)  not null
);




explain analyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=l


SET max_parallel_workers_per_gather TO 3;


explain analyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=l
```

	QUERY PLAN text
□	Hash Join (cost=21.93..44.51...
□	Hash Cond: (library2.id = libra...
□	-> Seq Scan on library2 (cost...
□	-> Hash (cost=15.30..15.30 r...
□	-> Seq Scan on library1 (cost...
□	Planning time: 0.139 ms
□	Execution time: 0.029 ms

Total query runtime: 425 msec.
7 rows retrieved.

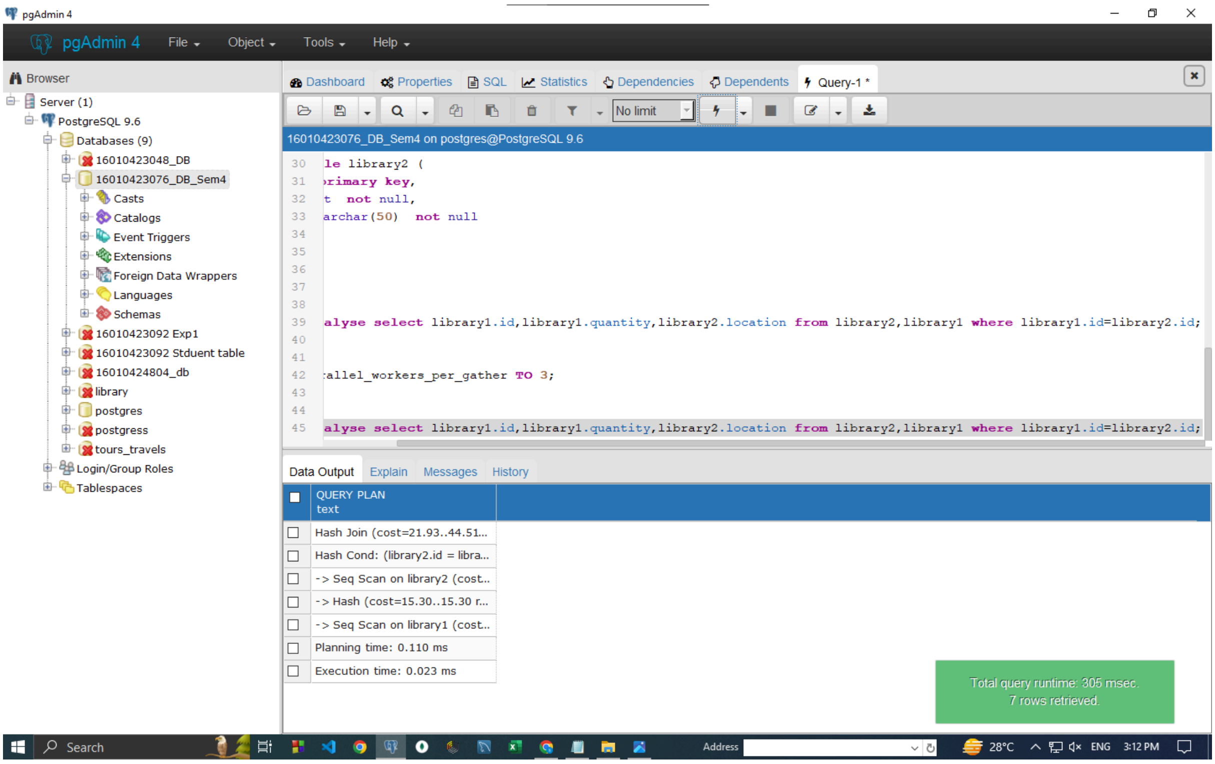


16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
quantity int  not null,
location varchar(50)  not null
);




explain analyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=l


SET max_parallel_workers_per_gather TO 3;


explain analyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=l
```

```
SET

Query returned successfully in 410 msec.
```

Query returned successfully in 410 msec.



16010423076_DB_Sem4 on postgres@PostgreSQL 9.6

```
le library2 (
primary key,
t  not null,
archar(50)  not null




alyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=library2.id;


allel_workers_per_gather TO 3;


alyse select library1.id,library1.quantity,library2.location from library2,library1 where library1.id=library2.id;
```

□	QUERY PLAN text
□	Hash Join (cost=21.93..44.51...
□	Hash Cond: (library2.id = libra...
□	-> Seq Scan on library2 (cost...
□	-> Hash (cost=15.30..15.30 r...
□	-> Seq Scan on library1 (cost...
□	Planning time: 0.110 ms
□	Execution time: 0.023 ms

Total query runtime: 305 msec.
7 rows retrieved.

Outcomes:

CO1: Design advanced database systems using Parallel, Distributed, Object Relational Databases and its implementation.

Conclusion: (Conclusion to be based on the outcomes achieved)

From this experiment, I learned how to configure and execute parallel queries in a database environment, improving query performance through parallel processing. By analyzing the effects of different levels of parallelism, I observed how database systems allocate resources efficiently to enhance computational speed.

This experiment increased my understanding of parallel database systems and their application in handling large-scale data operations effectively.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education
2. https://www.postgresql.org/docs/