Experiment No.1

Title: Execution of Parallel Database queries.

Batch: B-4 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

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 operation 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, the 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.
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(1,50000000);

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;

set max_parallel_workers_per_gather to 2;

explain analyze select sum(amount)  from ledger;

set max_parallel_workers_per_gather to 6;

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 joins.

```
CREATE TABLE library1 (
    id serial primary key,
    book_title varchar(255),
    quantity integer
);

INSERT INTO library1 (book_title, quantity)
VALUES
    ('Book1', 100),
    ('Book2', 150),
    ('Book3', 200),
    ('Book4', 75),
    ('Book5', 300);
```

```
CREATE TABLE library2 (
   id serial primary key,
   book_id integer,
   location varchar(100)
);

INSERT INTO library2 (book_id, location)
VALUES
   (1, 'Shelf A'),
   (2, 'Shelf B'),
   (3, 'Shelf C'),
   (4, 'Shelf D'),
   (5, 'Shelf E');

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:

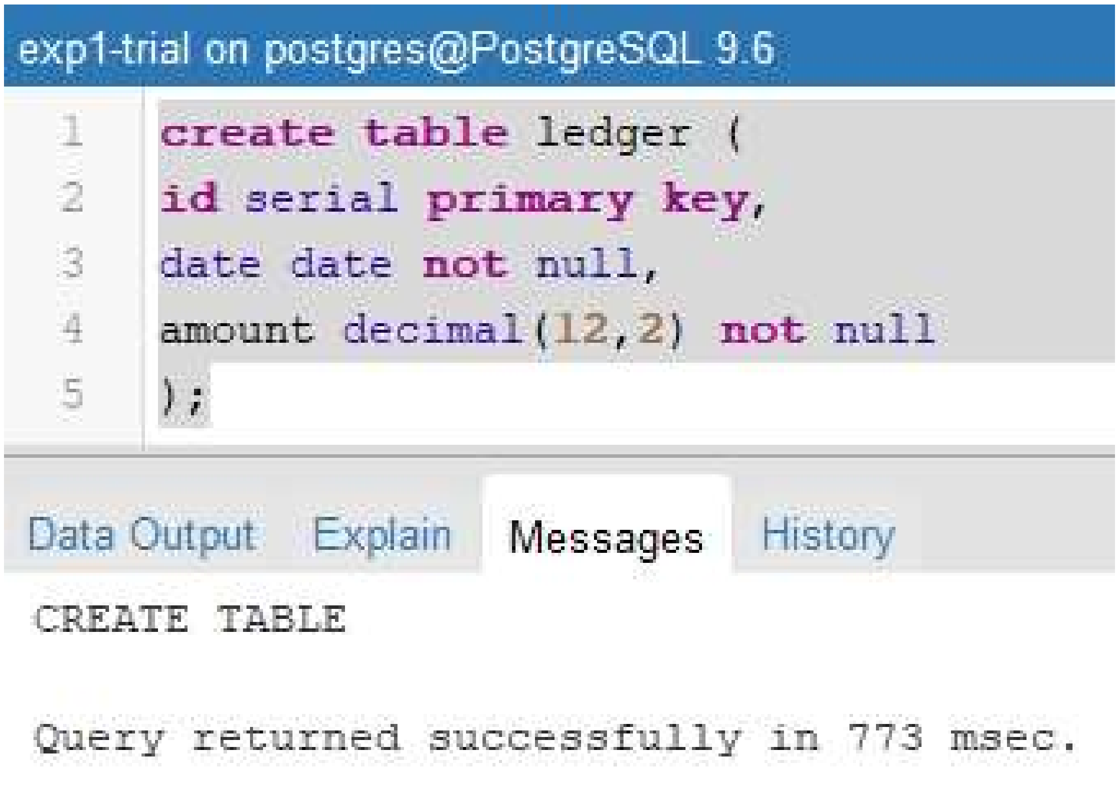
1. Explain the parallelism achieved in the experiment you performed.

Ans: In the experiment we performed, parallelism was achieved by enabling and adjusting the "max_parallel_workers_per_gather" configuration parameter in PostgreSQL. Initially, the query to calculate the sum of the "amount" column in the "ledger" table was executed sequentially. By increasing the "max_parallel_workers_per_gather" to match the number of cores on the workstation, we enabled parallel query execution, resulting in a significant performance improvement.
We then tested different values of "max_parallel_workers_per_gather" and observed the impact on query execution. By setting it to 4, we observed a fourfold increase in query performance compared to the sequential execution. Additionally, we noted that PostgreSQL optimizes the number of workers based on the size of the table and the "min_parallel_relation_size" parameter.
Furthermore, we extended the experiment to include a join operation between the "library1" and "library2" tables and analyzed the impact of parallelism on join queries as well.
Overall, the experiment effectively demonstrated how adjusting the parallelism settings in PostgreSQL can significantly impact query performance, especially for large datasets and complex operations like joins.

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

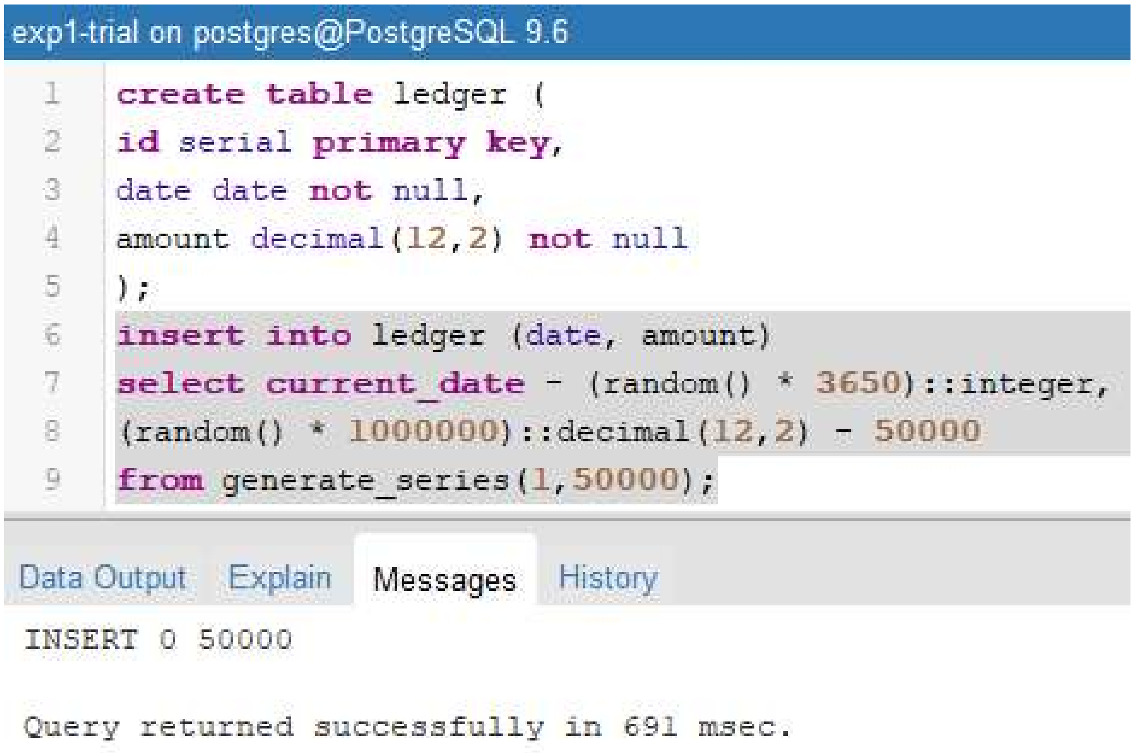
Ans: The degree of parallelism directly affects the efficiency and speed of the operation conducted. A higher degree of parallelism can lead to faster query execution, especially for computationally intensive tasks, while a lower degree of parallelism may result in slower performance. Therefore, it's crucial to carefully adjust the degree of parallelism based on the specific characteristics of the workload and the underlying hardware to achieve optimal performance.

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 773 msec.
```

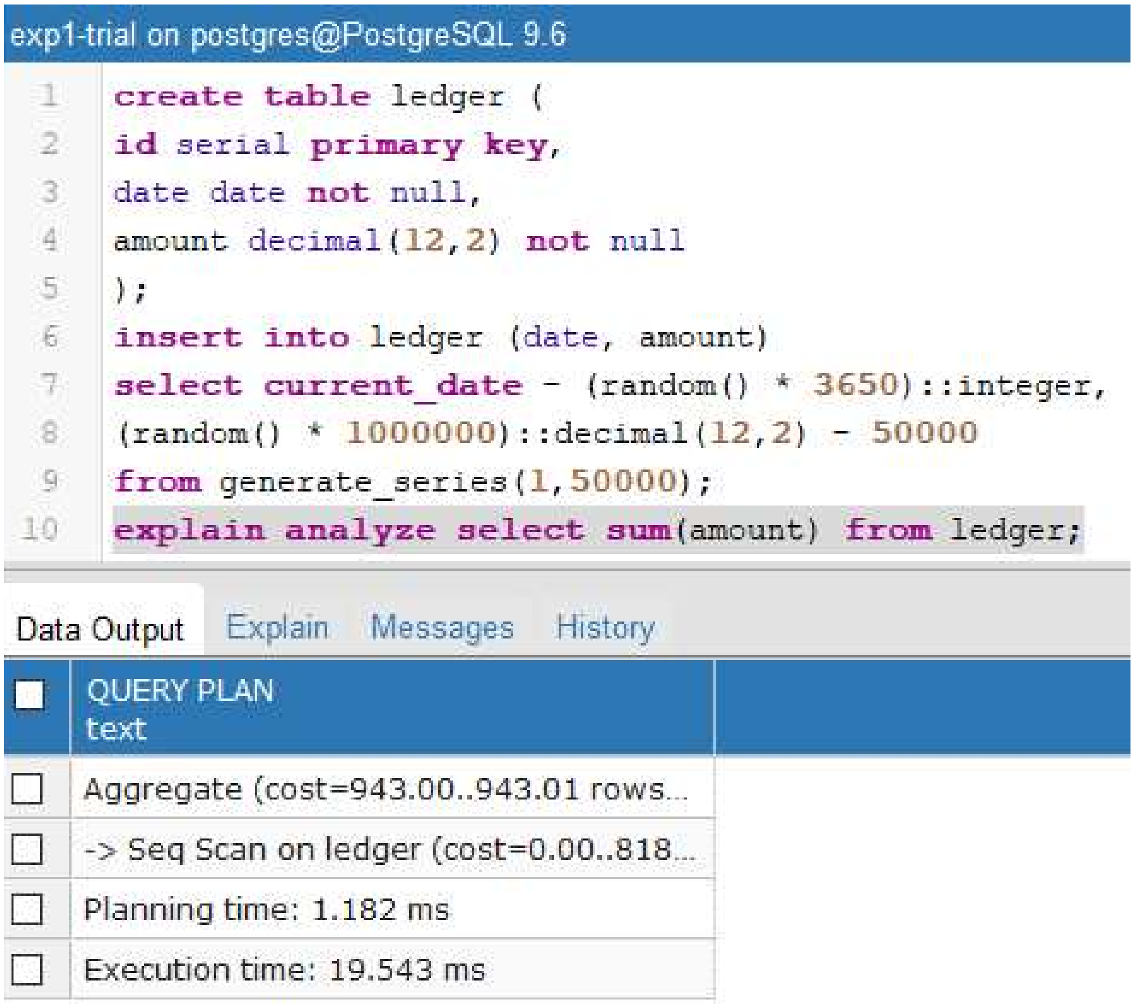


exp1-trial on postgres@PostgreSQL 9.6

```sql
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(1,50000);
```

Data Output | Explain | Messages | History

```
INSERT 0 50000

Query returned successfully in 691 msec.
```

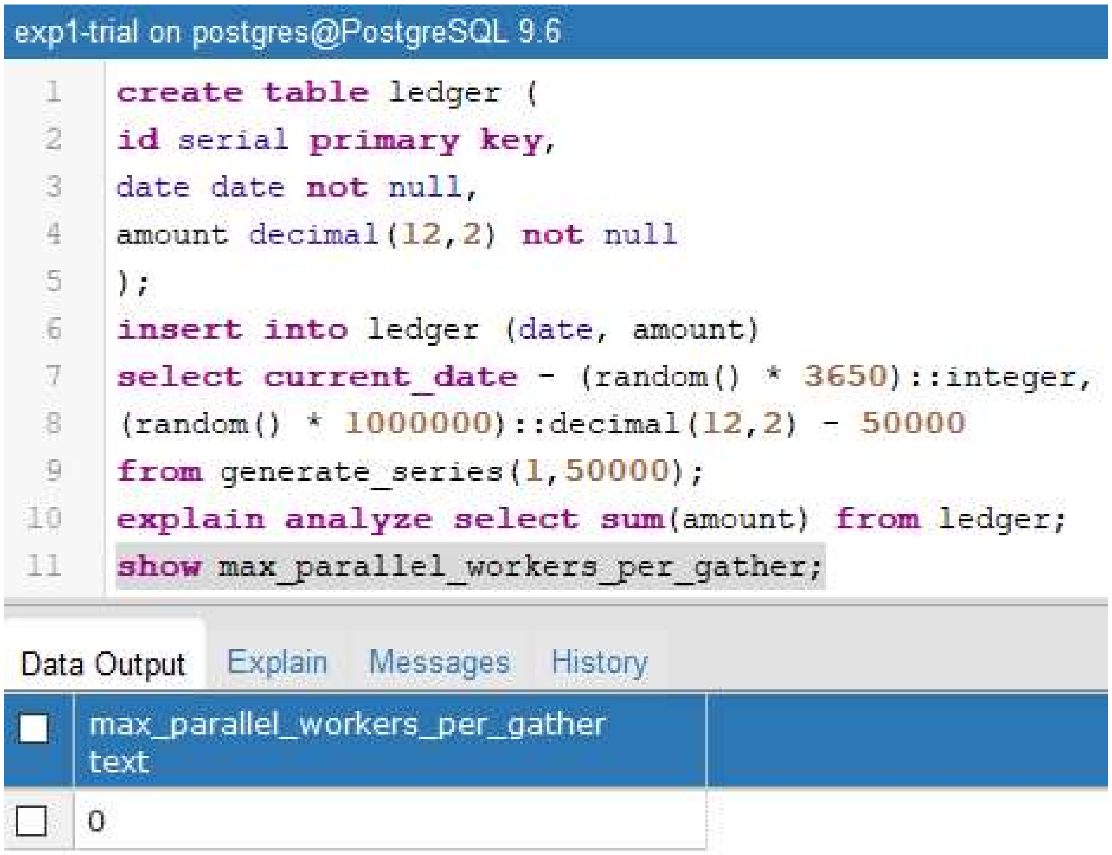


exp1-trial on postgres@PostgreSQL 9.6

```sql
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(1,50000);
explain analyze select sum(amount) from ledger;
```

Data Output | Explain | Messages | History

QUERY PLAN text
Aggregate (cost=943.00..943.01 rows...
-> Seq Scan on ledger (cost=0.00..818...
Planning time: 1.182 ms
Execution time: 19.543 ms

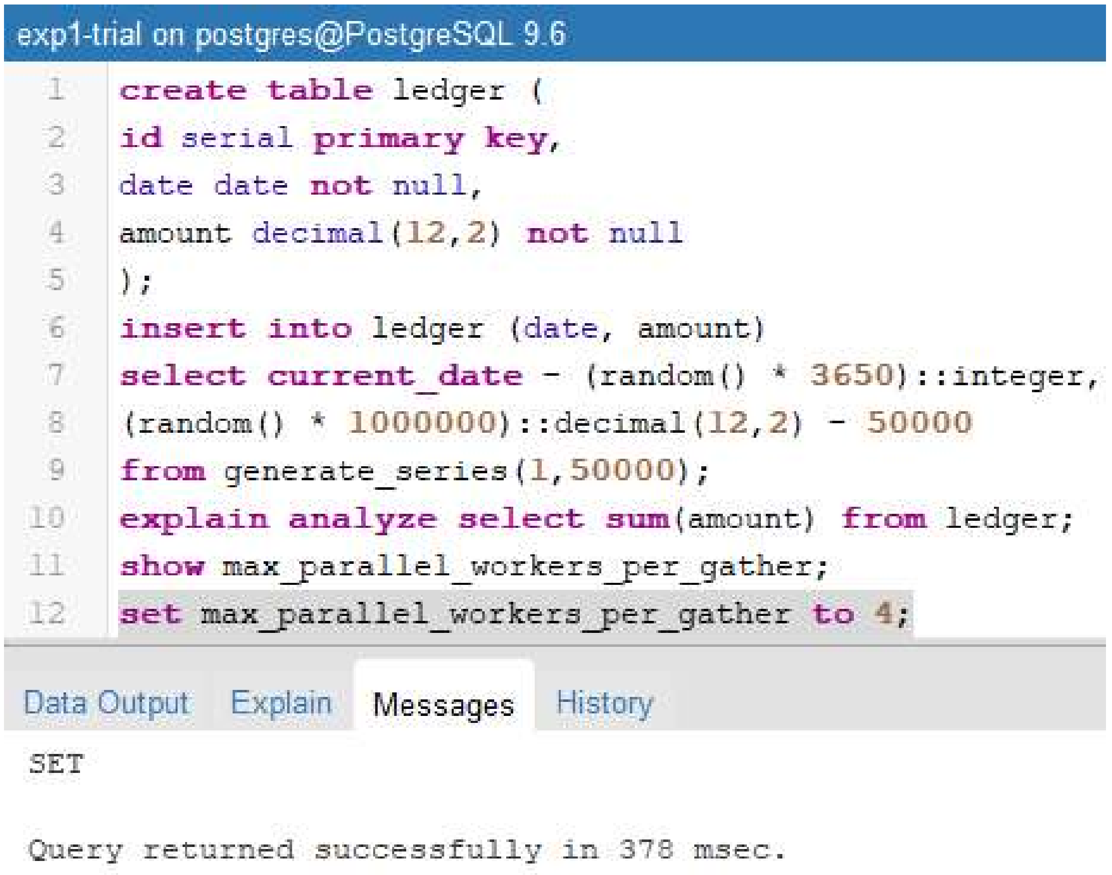


exp1-trial on postgres@PostgreSQL 9.6

```
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(1,50000);
explain analyze select sum(amount) from ledger;
show max_parallel_workers_per_gather;
```

Data Output | Explain | Messages | History

max_parallel_workers_per_gather text
0

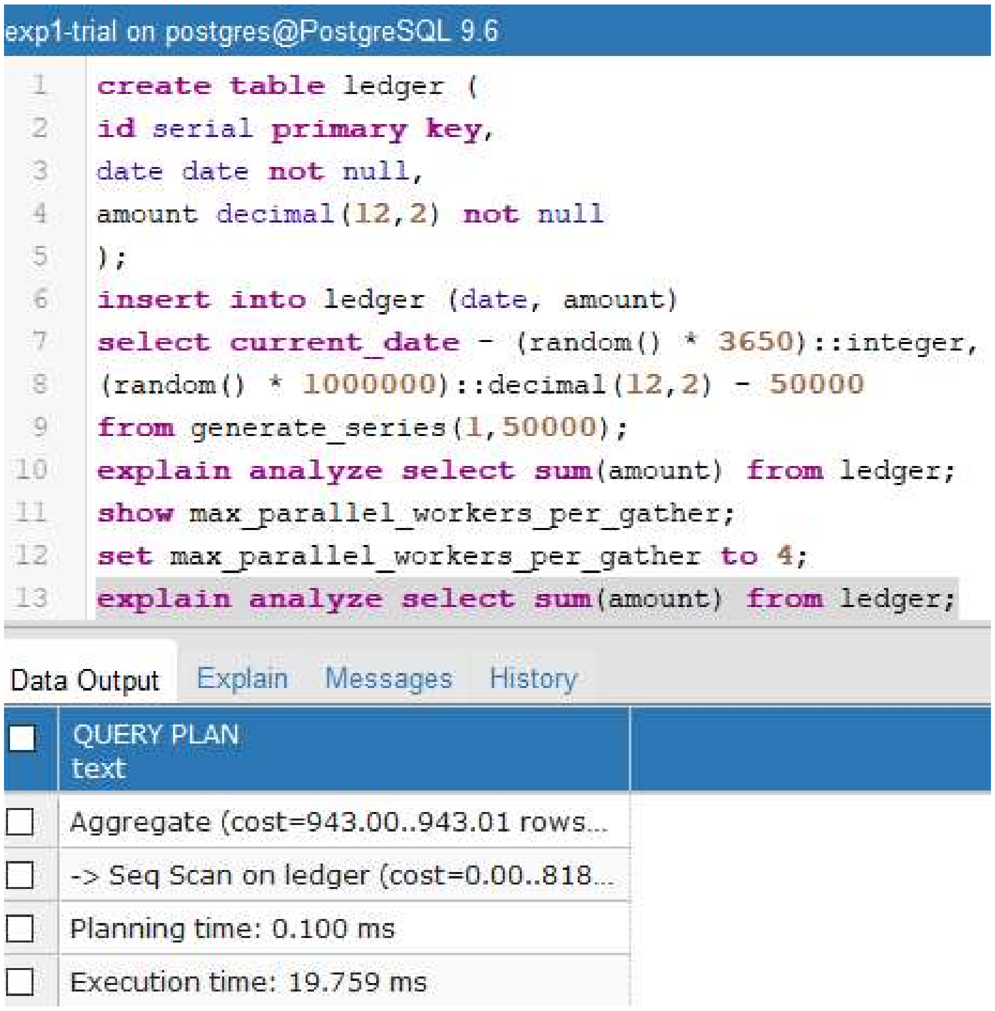


exp1-trial on postgres@PostgreSQL 9.6

```
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(1,50000);
explain analyze select sum(amount) from ledger;
show max_parallel_workers_per_gather;
set max_parallel_workers_per_gather to 4;
```

Data Output | Explain | Messages | History

```
SET

Query returned successfully in 378 msec.
```

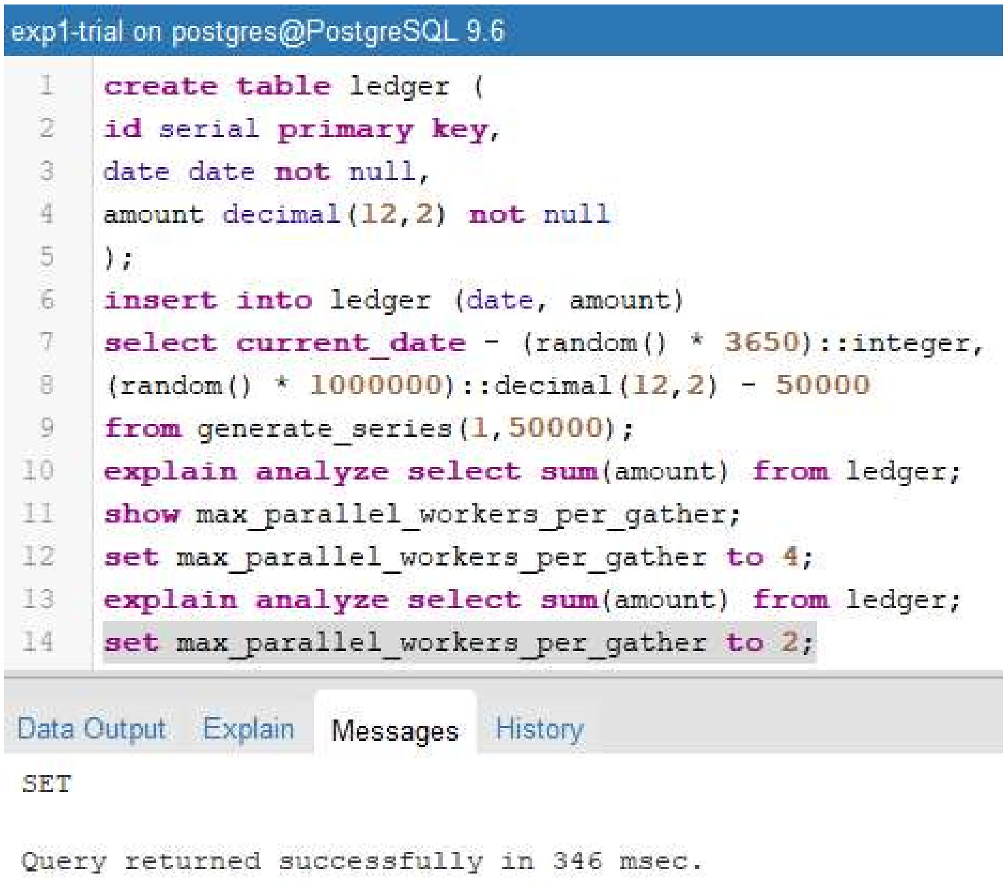


exp1-trial on postgres@PostgreSQL 9.6

```
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(1,50000);
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;
```

Data Output | Explain | Messages | History

QUERY PLAN text
Aggregate (cost=943.00..943.01 rows...
-> Seq Scan on ledger (cost=0.00..818...
Planning time: 0.100 ms
Execution time: 19.759 ms

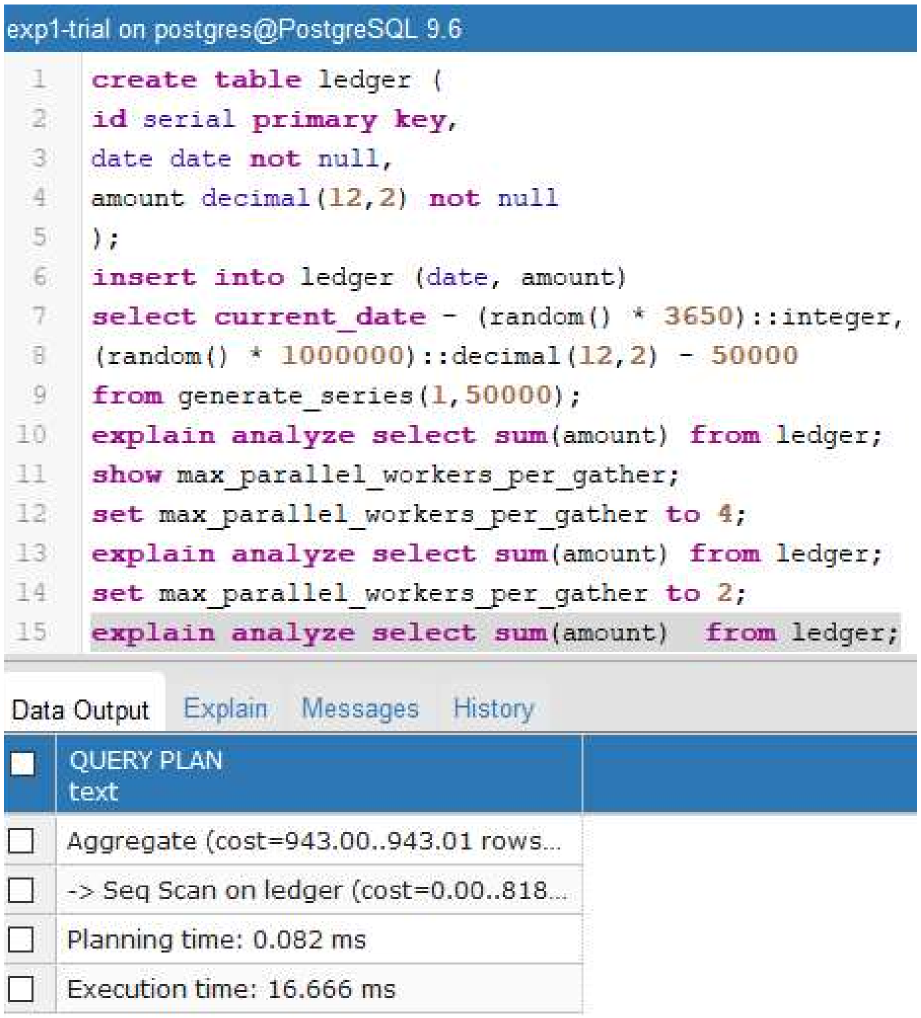


exp1-trial on postgres@PostgreSQL 9.6

```
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(1,50000);
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 max_parallel_workers_per_gather to 2;
```

Data Output | Explain | Messages | History

```
SET

Query returned successfully in 346 msec.
```

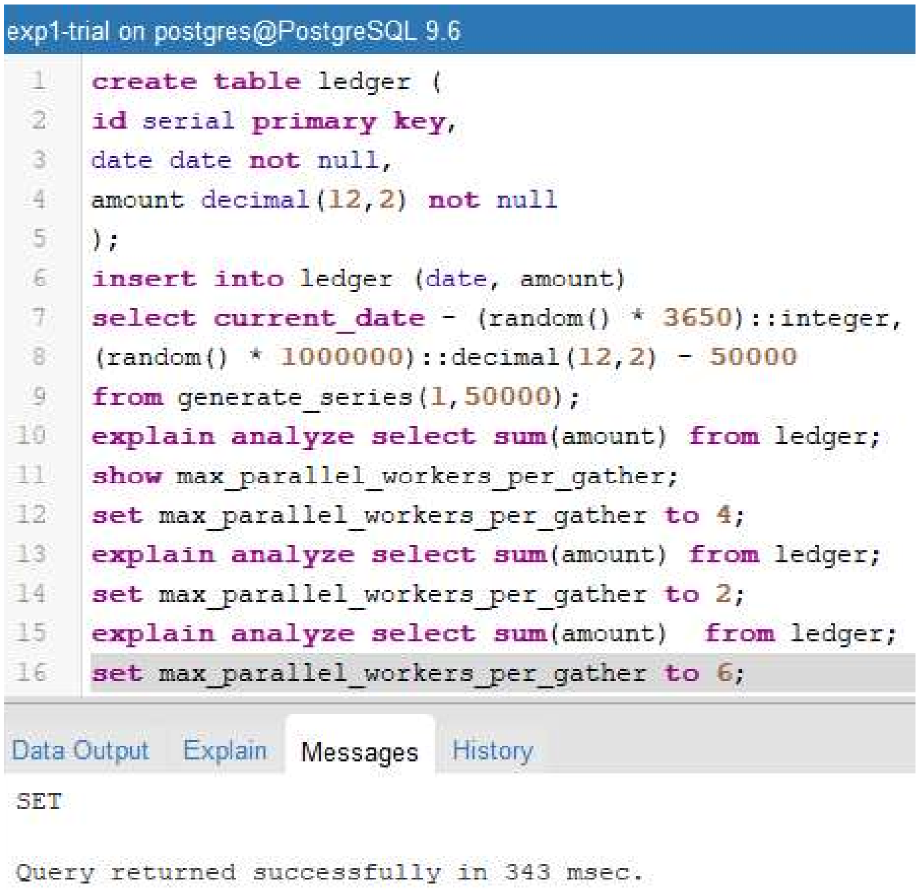


exp1-trial on postgres@PostgreSQL 9.6

```sql
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(1,50000);
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 max_parallel_workers_per_gather to 2;
explain analyze select sum(amount)  from ledger;
```

Data Output | Explain | Messages | History

QUERY PLAN text
Aggregate (cost=943.00..943.01 rows...
-> Seq Scan on ledger (cost=0.00..818...
Planning time: 0.082 ms
Execution time: 16.666 ms

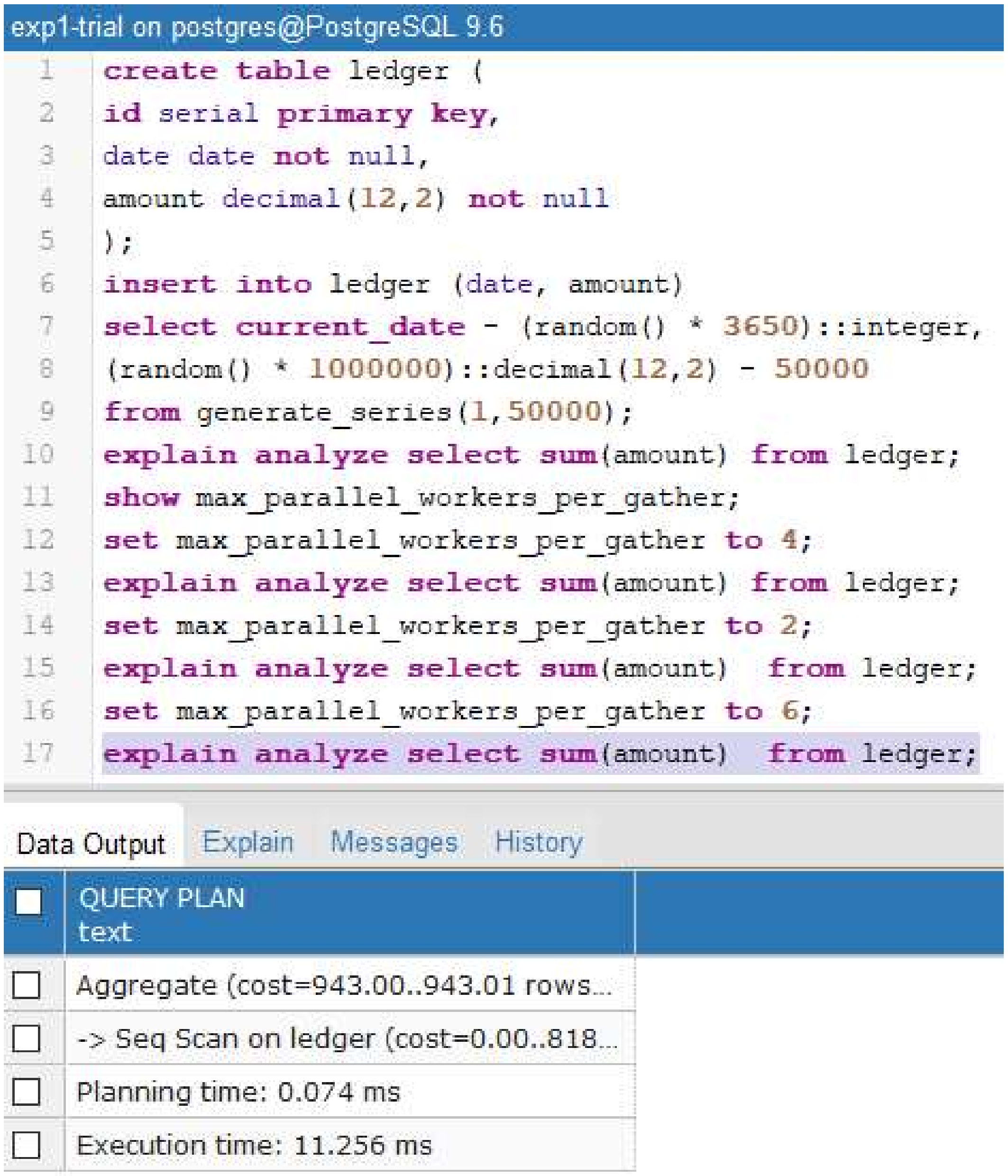


exp1-trial on postgres@PostgreSQL 9.6

```sql
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(1,50000);
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 max_parallel_workers_per_gather to 2;
explain analyze select sum(amount)  from ledger;
set max_parallel_workers_per_gather to 6;
```

Data Output | Explain | Messages | History

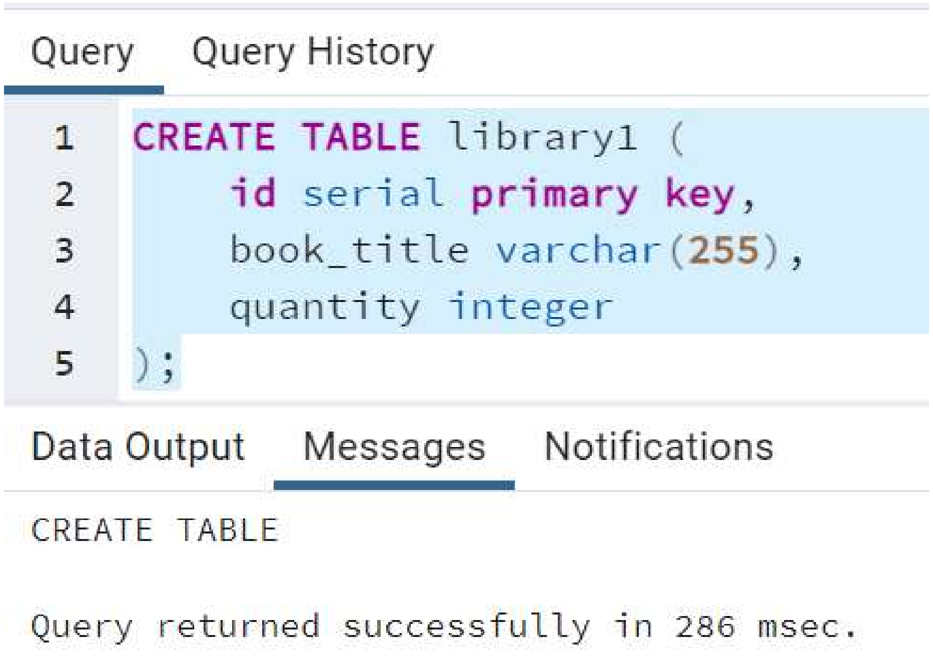
```
SET

Query returned successfully in 343 msec.
```



exp1-trial on postgres@PostgreSQL 9.6

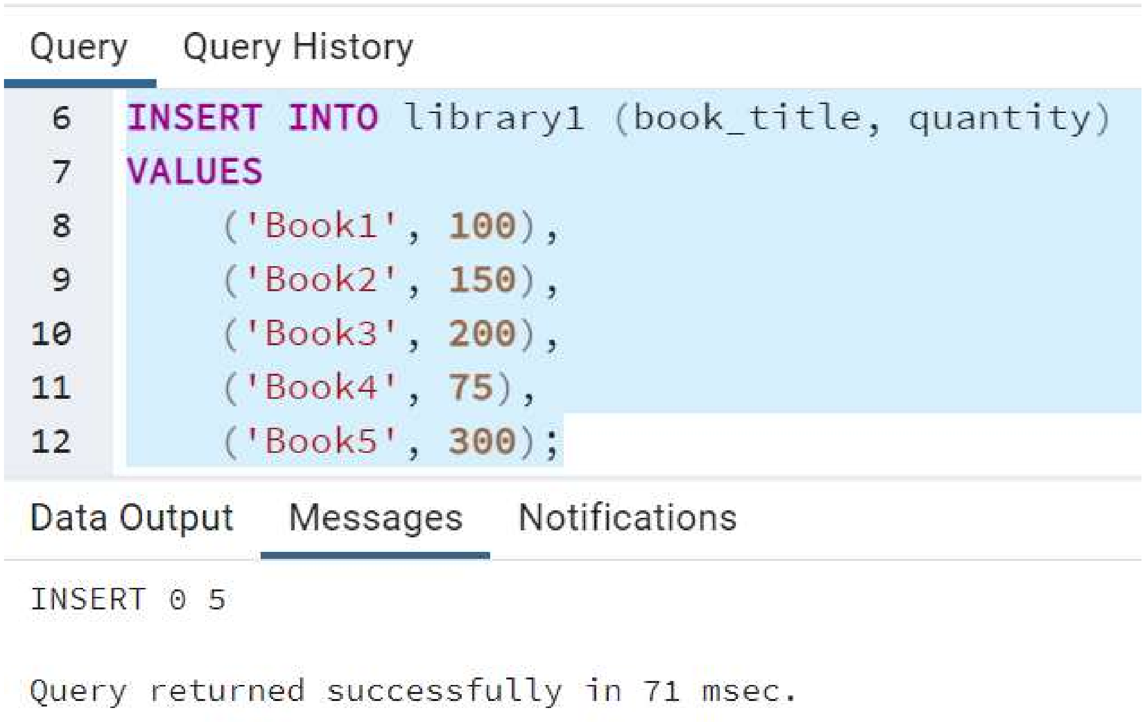
```
create table ledger (
id serial primary key,
date date not null,
amount decimal(12,2) not null
);
insert into ledger (date, amount)
select current_date - (random() * 3650)::integer,
(random() * 1000000)::decimal(12,2) - 50000
from generate_series(1,50000);
explain analyze select sum(amount) from ledger;
show max_parallel_workers_per_gather;
set max_parallel_workers_per_gather to 4;
explain analyze select sum(amount) from ledger;
set max_parallel_workers_per_gather to 2;
explain analyze select sum(amount)  from ledger;
set max_parallel_workers_per_gather to 6;
explain analyze select sum(amount)  from ledger;
```

QUERY PLAN text
Aggregate (cost=943.00..943.01 rows...
-> Seq Scan on ledger (cost=0.00..818...
Planning time: 0.074 ms
Execution time: 11.256 ms



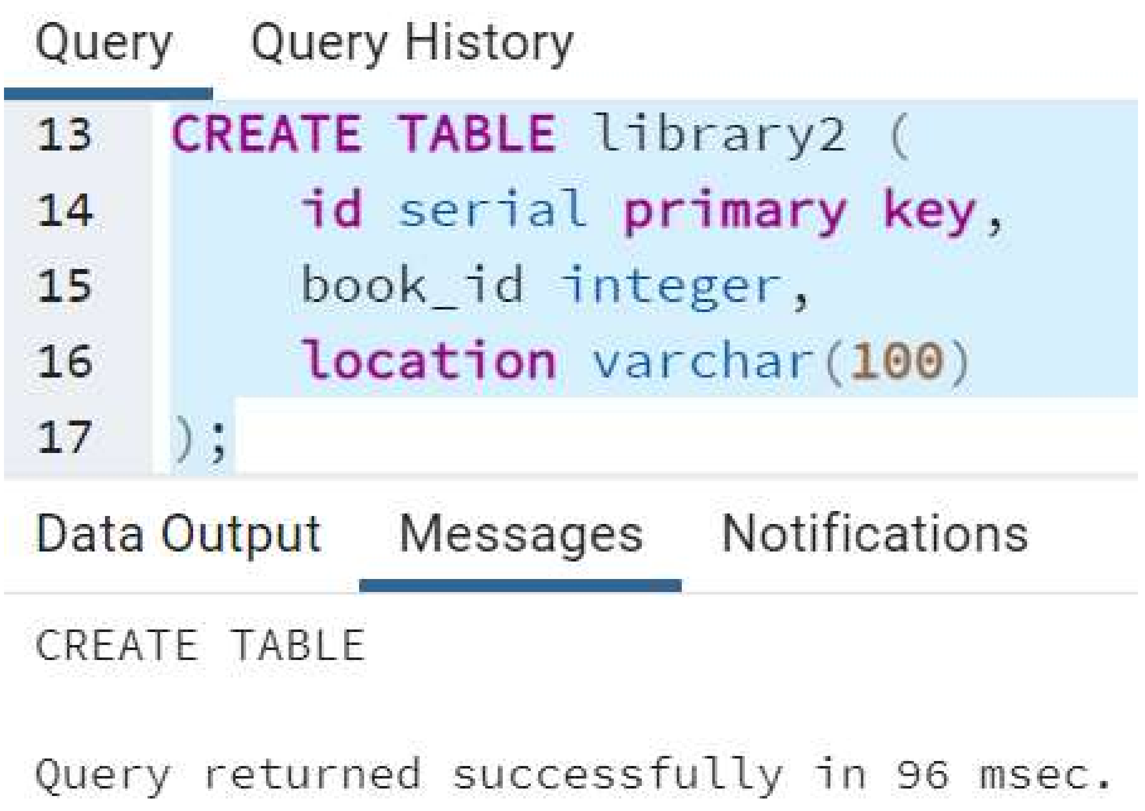
```
CREATE TABLE library1 (
    id serial primary key,
    book_title varchar(255),
    quantity integer
);
```

```
CREATE TABLE

Query returned successfully in 286 msec.
```



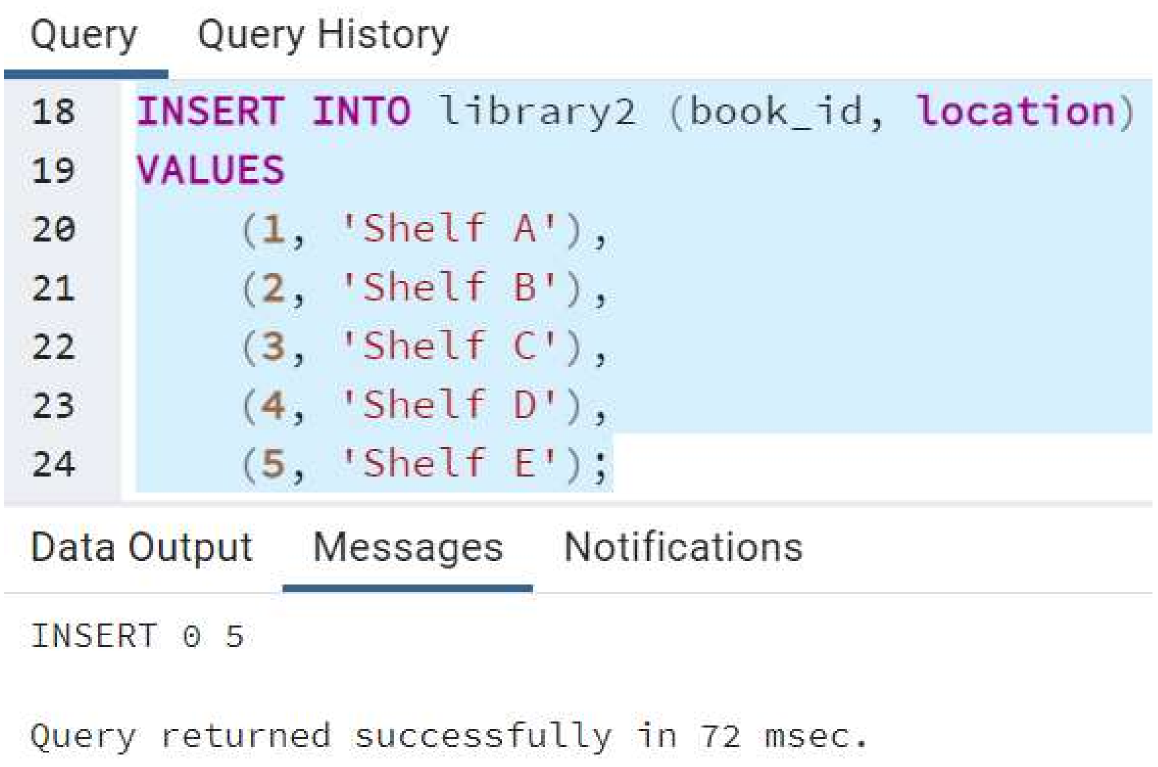
```sql
INSERT INTO library1 (book_title, quantity)
VALUES
    ('Book1', 100),
    ('Book2', 150),
    ('Book3', 200),
    ('Book4', 75),
    ('Book5', 300);
```

```
INSERT 0 5

Query returned successfully in 71 msec.
```



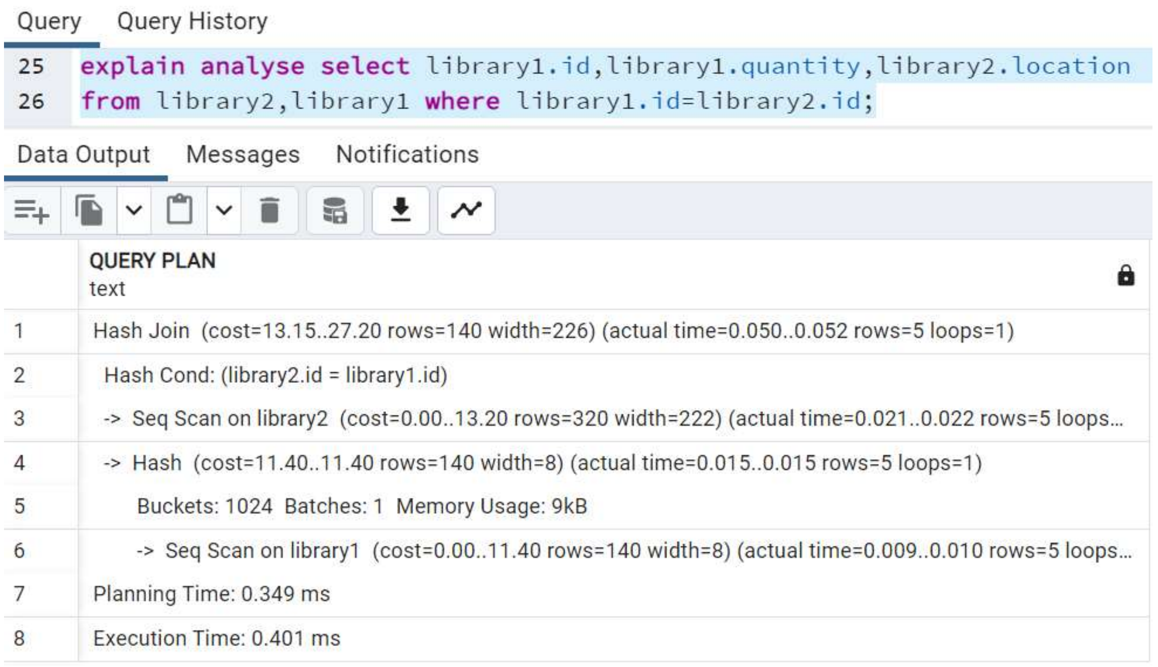
```sql
CREATE TABLE library2 (
    id serial primary key,
    book_id integer,
    location varchar(100)
);
```

```
CREATE TABLE

Query returned successfully in 96 msec.
```



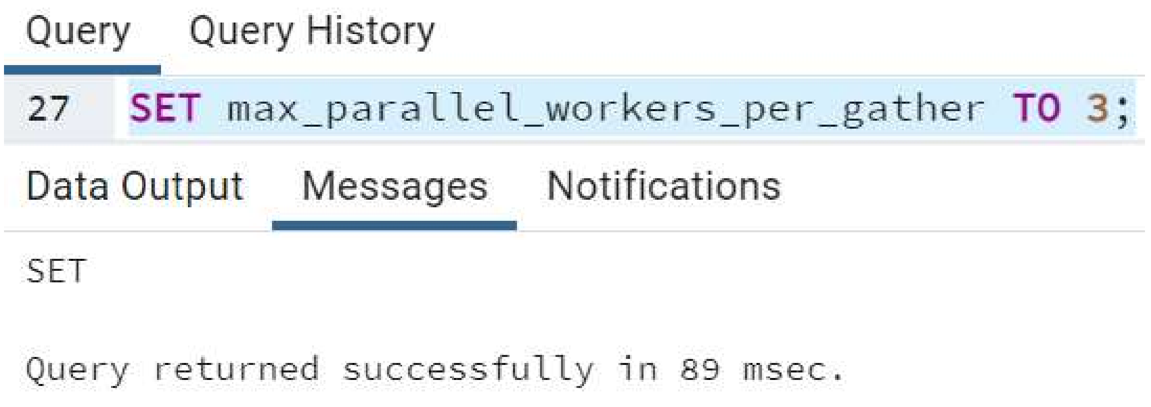
```sql
INSERT INTO library2 (book_id, location)
VALUES
      (1, 'Shelf A'),
      (2, 'Shelf B'),
      (3, 'Shelf C'),
      (4, 'Shelf D'),
      (5, 'Shelf E');
```

```
INSERT 0 5

Query returned successfully in 72 msec.
```



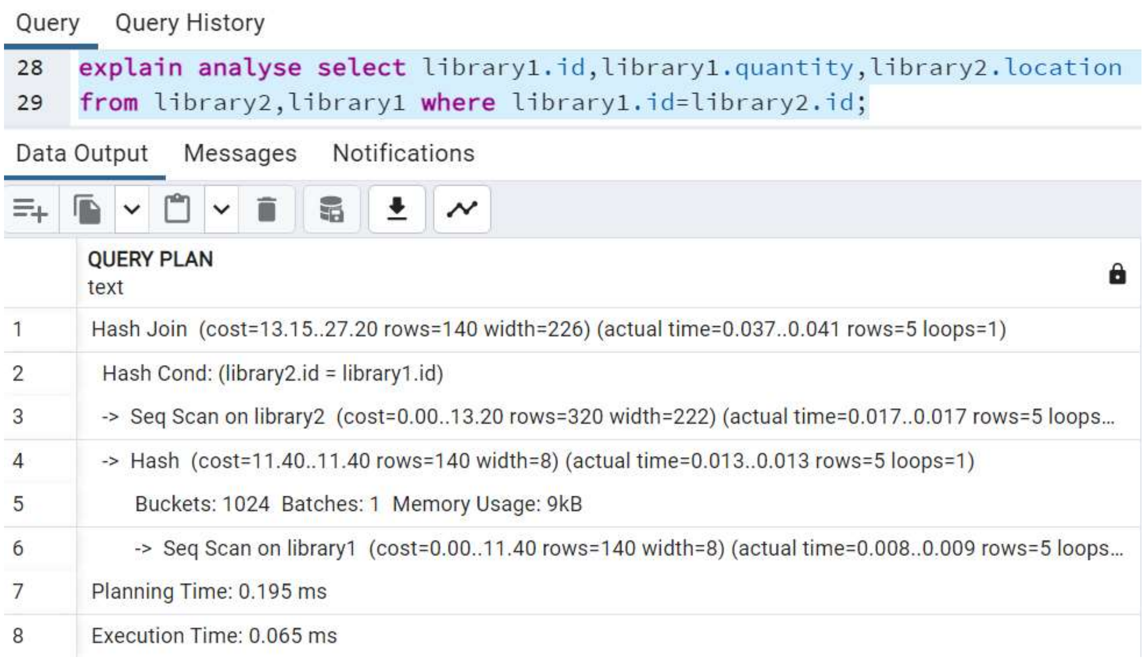
```sql
explain analyse select library1.id,library1.quantity,library2.location
from library2,library1 where library1.id=library2.id;
```

	QUERY PLAN text
1	Hash Join (cost=13.15..27.20 rows=140 width=226) (actual time=0.050..0.052 rows=5 loops=1)
2	Hash Cond: (library2.id = library1.id)
3	-> Seq Scan on library2 (cost=0.00..13.20 rows=320 width=222) (actual time=0.021..0.022 rows=5 loops...
4	-> Hash (cost=11.40..11.40 rows=140 width=8) (actual time=0.015..0.015 rows=5 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on library1 (cost=0.00..11.40 rows=140 width=8) (actual time=0.009..0.010 rows=5 loops...
7	Planning Time: 0.349 ms
8	Execution Time: 0.401 ms



```sql
SET max_parallel_workers_per_gather TO 3;
```

```
SET

Query returned successfully in 89 msec.
```



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

```
QUERY PLAN
text
1  Hash Join  (cost=13.15..27.20 rows=140 width=226) (actual time=0.037..0.041 rows=5 loops=1)
2    Hash Cond: (library2.id = library1.id)
3    ->  Seq Scan on library2  (cost=0.00..13.20 rows=320 width=222) (actual time=0.017..0.017 rows=5 loops...
4    ->  Hash  (cost=11.40..11.40 rows=140 width=8) (actual time=0.013..0.013 rows=5 loops=1)
5          Buckets: 1024  Batches: 1  Memory Usage: 9kB
6          ->  Seq Scan on library1  (cost=0.00..11.40 rows=140 width=8) (actual time=0.008..0.009 rows=5 loops...
7  Planning Time: 0.195 ms
8  Execution Time: 0.065 ms
```

Outcomes: Design advanced database systems using Parallel, Distributed and In - memory databases and its implementation.

Conclusion: (Conclusion to be based on the outcomes achieved)
The experiment underscored the critical role of parallelism in enhancing the efficiency of database operations, particularly for tasks involving aggregation and join operations on substantial datasets. It emphasized the need to carefully consider and adjust the degree of parallelism to match the available computational resources and the nature of the workload, ultimately leading to improved query performance.

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/