Experiment No.1

Title: Execution of Parallel Database queries.

Batch: A1 Roll No.: 16010422013 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(1,50000000);
```

```
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;
```

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.

```
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:

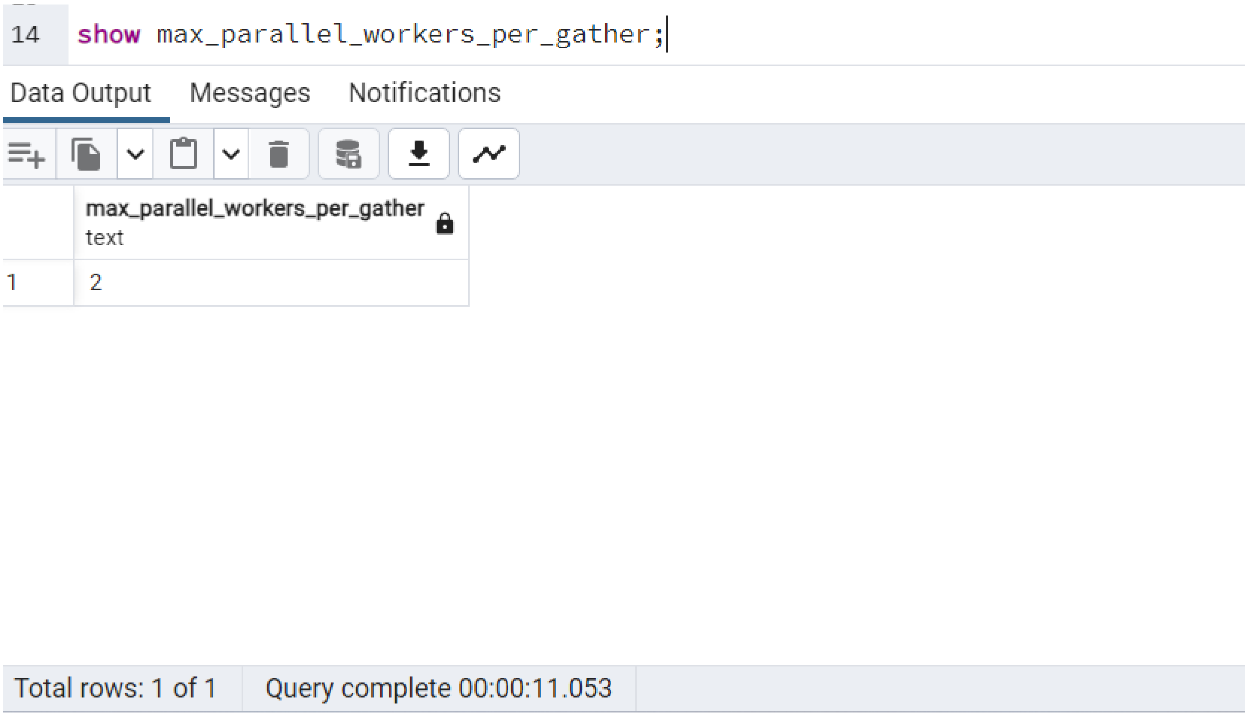
1. Explain the parallelism achieved in the experiment you performed.

   The experiment utilized parallelism through a "Gather Motion" and a "Parallel Hash Join," involving four parallel workers (segments) for increased efficiency. The "Redistribute Motion" operations facilitated data redistribution among workers during the join operation between the "ledger" and "library_transactions" tables. This parallel execution plan aimed to optimize performance by dividing the workload and leveraging the available computing resources. The segmentation allowed for parallel scanning of the tables, enhancing the processing speed, particularly beneficial when dealing with large datasets. The successful parallel execution demonstrated the database system's ability to harness parallelism for improved query performance.

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

The degree of parallelism plays a pivotal role in influencing the efficiency of database operations, as demonstrated in the experiment comparing various degrees of parallelism for the join operation between the "ledger" and "library_transactions" tables. A higher degree of parallelism, characterized by an increased number of parallel processors, generally leads to improved performance. This is particularly notable when handling large datasets, as the workload is effectively distributed across multiple processors, enabling faster query execution. On the other hand, a lower degree of parallelism may result in suboptimal performance, especially for substantial datasets, where parallel processing benefits are underutilized. The optimal degree of parallelism is contingent on factors such as dataset size, system resources, and query complexity.

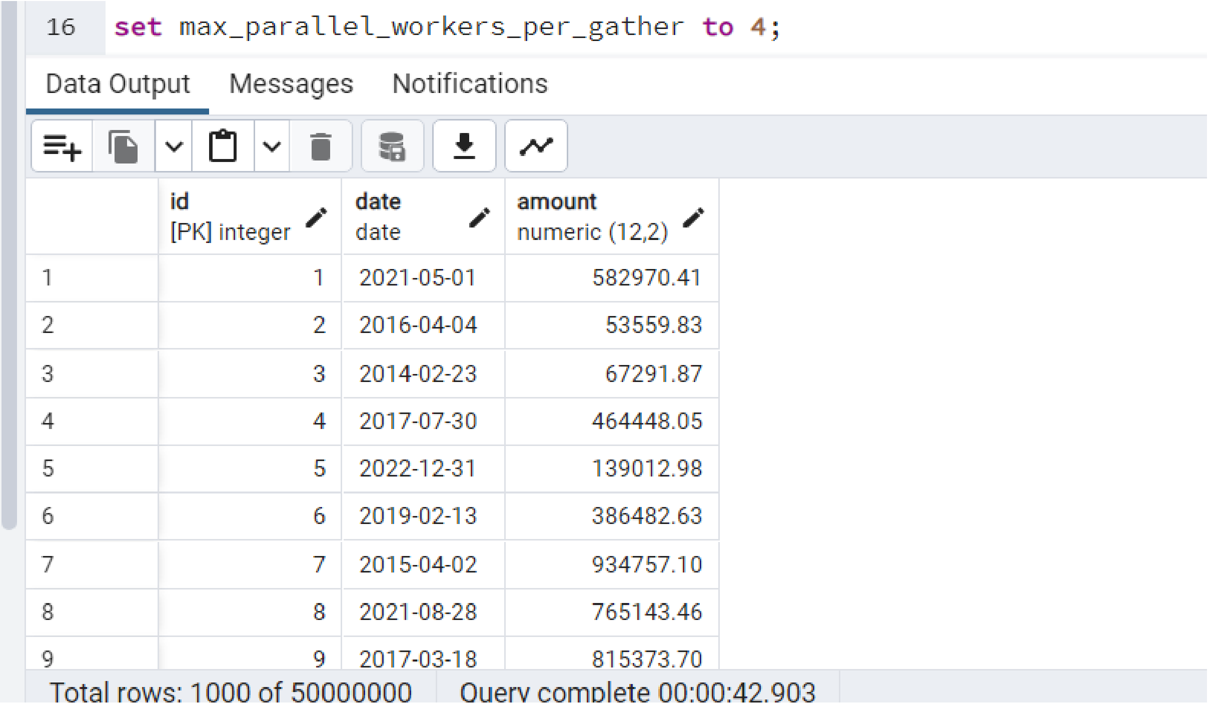
Results: (Program printout with output)



```
show max_parallel_workers_per_gather;
```

	max_parallel_workers_per_gather text
1	2

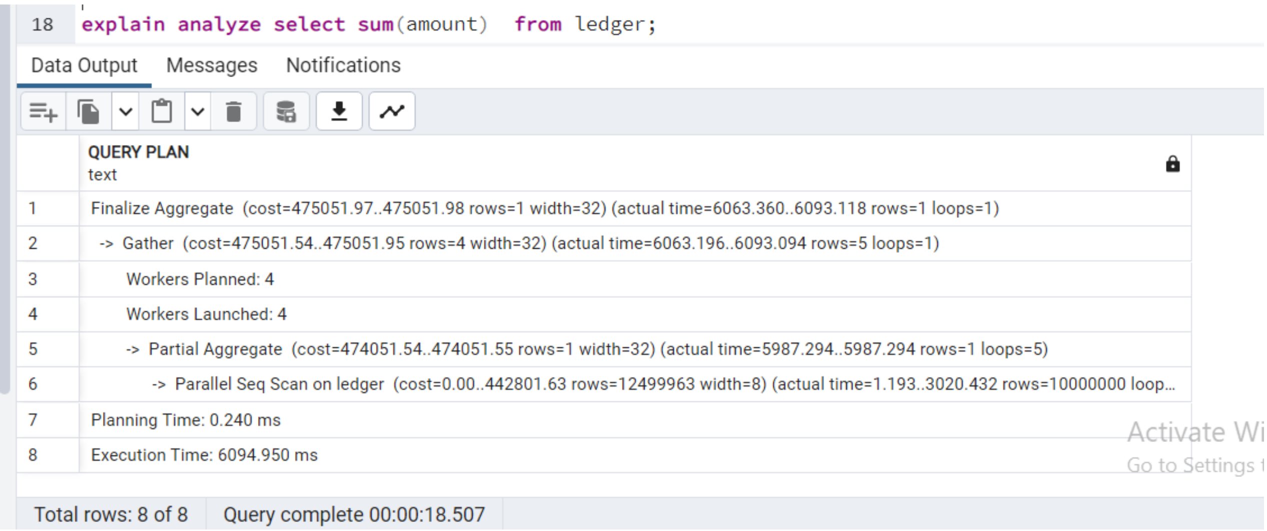
Total rows: 1 of 1 Query complete 00:00:11.053



```
set max_parallel_workers_per_gather to 4;
```

	id [PK] integer	date date	amount numeric (12,2)
1	1	2021-05-01	582970.41
2	2	2016-04-04	53559.83
3	3	2014-02-23	67291.87
4	4	2017-07-30	464448.05
5	5	2022-12-31	139012.98
6	6	2019-02-13	386482.63
7	7	2015-04-02	934757.10
8	8	2021-08-28	765143.46
9	9	2017-03-18	815373.70

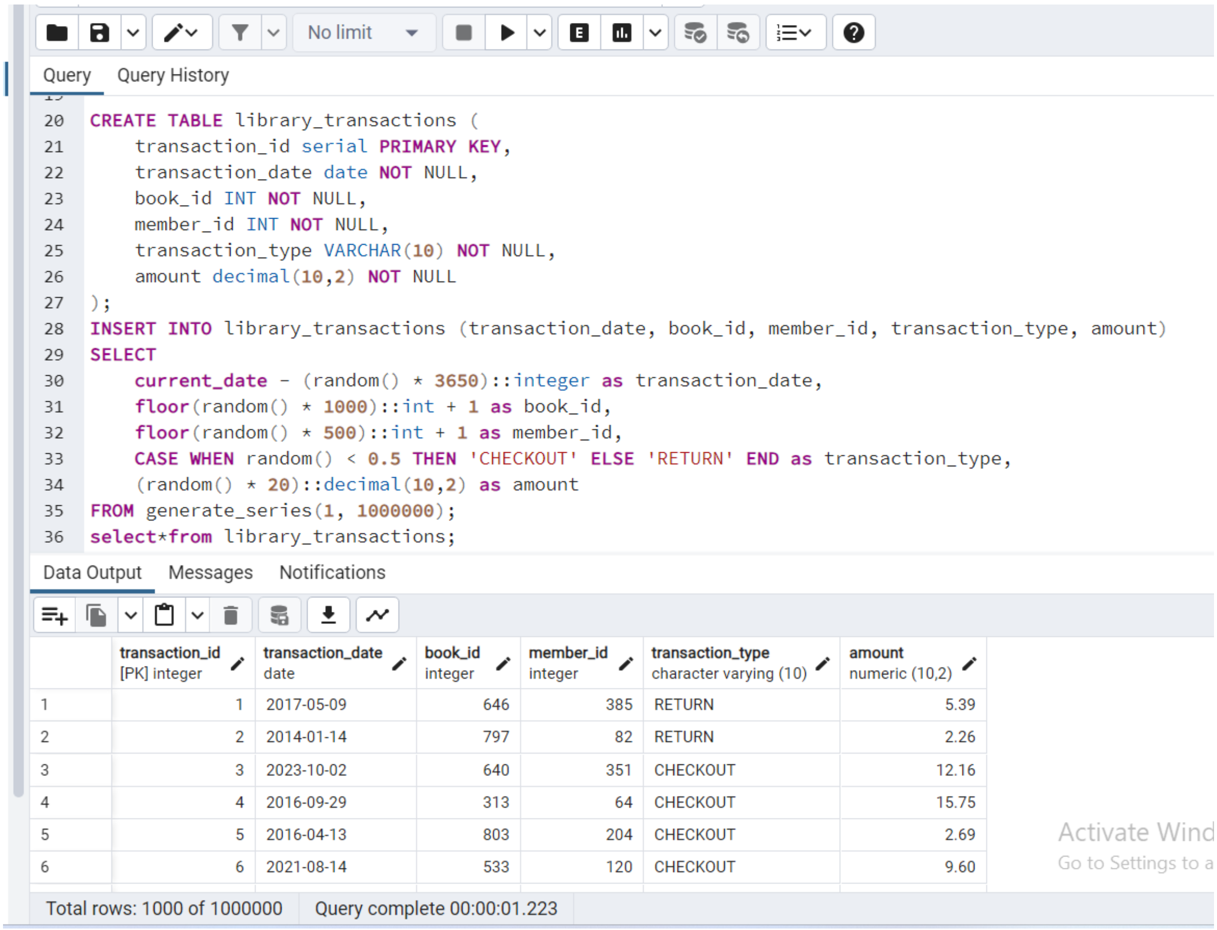
Total rows: 1000 of 50000000 Query complete 00:00:42.903



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

	QUERY PLAN text
1	Finalize Aggregate (cost=475051.97..475051.98 rows=1 width=32) (actual time=6063.360..6093.118 rows=1 loops=1)
2	-> Gather (cost=475051.54..475051.95 rows=4 width=32) (actual time=6063.196..6093.094 rows=5 loops=1)
3	Workers Planned: 4
4	Workers Launched: 4
5	-> Partial Aggregate (cost=474051.54..474051.55 rows=1 width=32) (actual time=5987.294..5987.294 rows=1 loops=5)
6	-> Parallel Seq Scan on ledger (cost=0.00..442801.63 rows=12499963 width=8) (actual time=1.193..3020.432 rows=10000000 loop...
7	Planning Time: 0.240 ms
8	Execution Time: 6094.950 ms

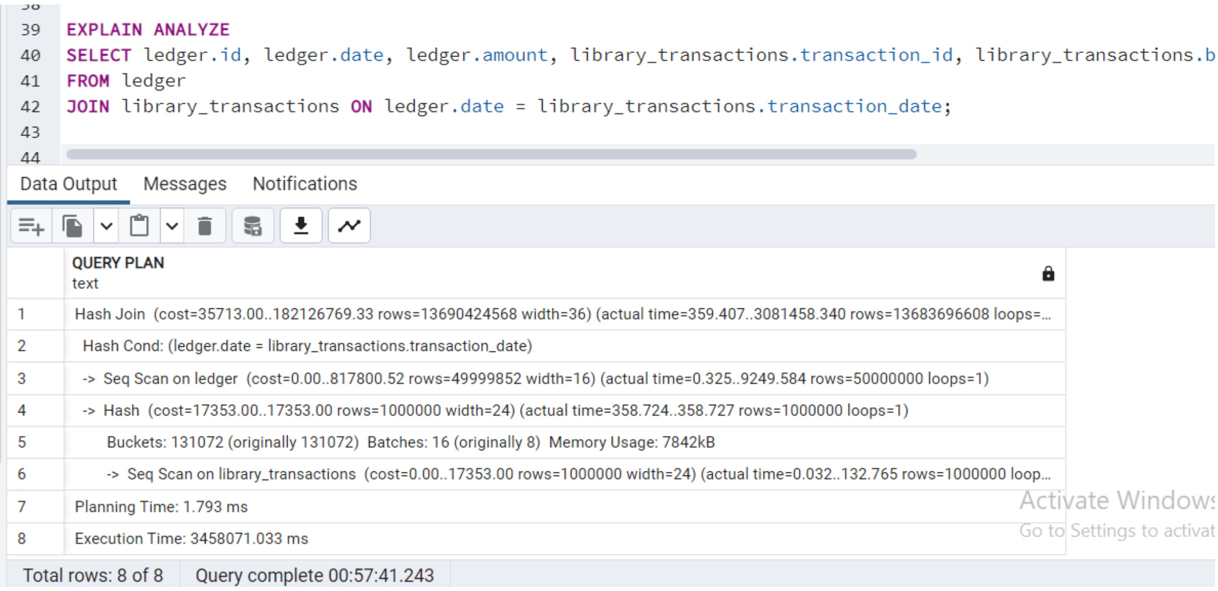
Total rows: 8 of 8   Query complete 00:00:18.507



```sql
CREATE TABLE library_transactions (
    transaction_id serial PRIMARY KEY,
    transaction_date date NOT NULL,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    transaction_type VARCHAR(10) NOT NULL,
    amount decimal(10,2) NOT NULL
);
INSERT INTO library_transactions (transaction_date, book_id, member_id, transaction_type, amount)
SELECT
    current_date - (random() * 3650)::integer as transaction_date,
    floor(random() * 1000)::int + 1 as book_id,
    floor(random() * 500)::int + 1 as member_id,
    CASE WHEN random() < 0.5 THEN 'CHECKOUT' ELSE 'RETURN' END as transaction_type,
    (random() * 20)::decimal(10,2) as amount
FROM generate_series(1, 1000000);
select*from library_transactions;
```

	transaction_id [PK] integer	transaction_date date	book_id integer	member_id integer	transaction_type character varying (10)	amount numeric (10,2)
1	1	2017-05-09	646	385	RETURN	5.39
2	2	2014-01-14	797	82	RETURN	2.26
3	3	2023-10-02	640	351	CHECKOUT	12.16
4	4	2016-09-29	313	64	CHECKOUT	15.75
5	5	2016-04-13	803	204	CHECKOUT	2.69
6	6	2021-08-14	533	120	CHECKOUT	9.60

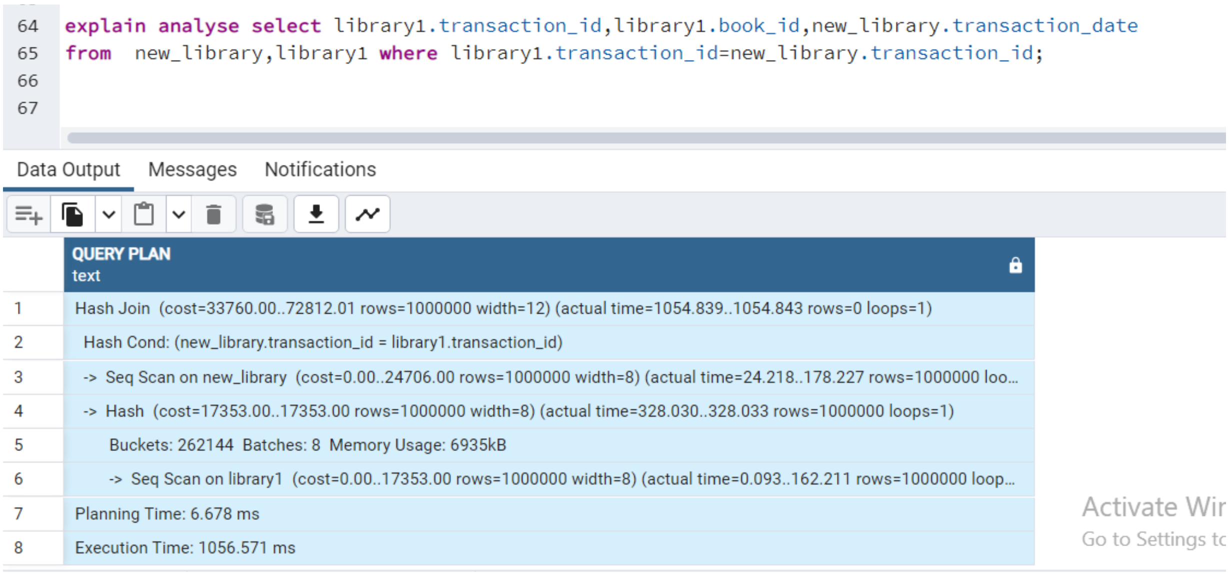
Total rows: 1000 of 1000000   Query complete 00:00:01.223



```sql
EXPLAIN ANALYZE
SELECT ledger.id, ledger.date, ledger.amount, library_transactions.transaction_id, library_transactions.b
FROM ledger
JOIN library_transactions ON ledger.date = library_transactions.transaction_date;
```

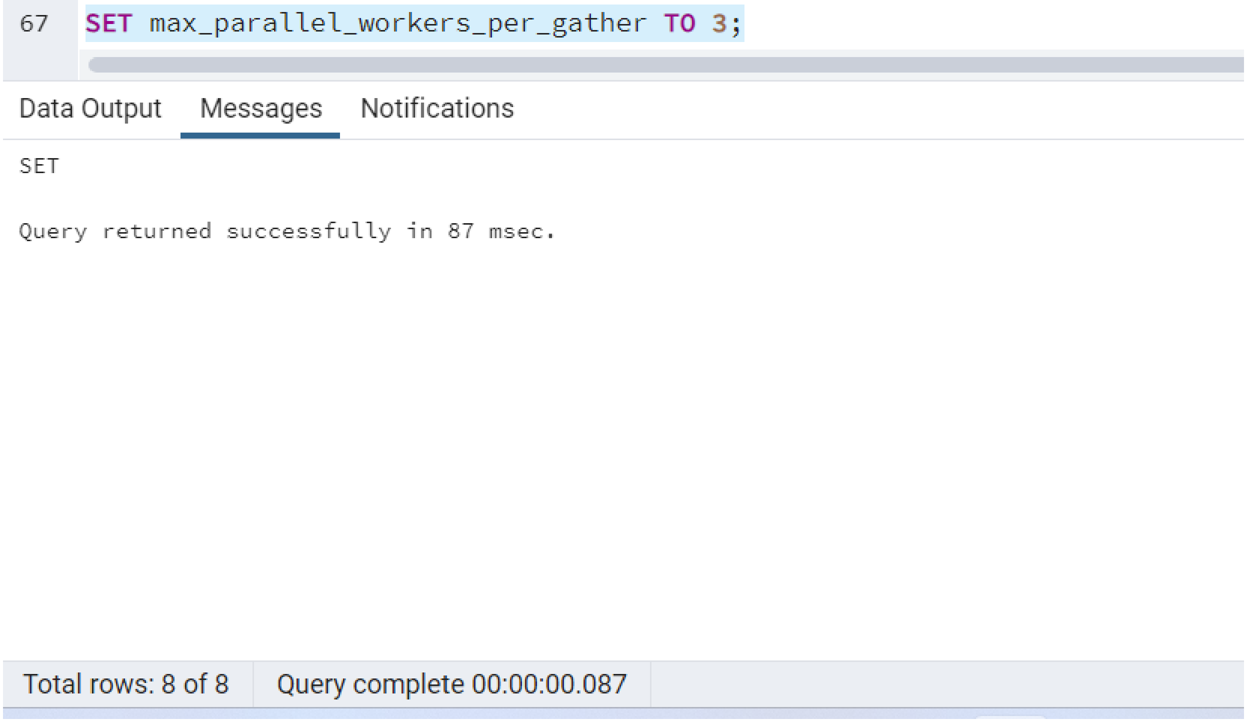
	QUERY PLAN text
1	Hash Join (cost=35713.00..182126769.33 rows=13690424568 width=36) (actual time=359.407..3081458.340 rows=13683696608 loops=...
2	Hash Cond: (ledger.date = library_transactions.transaction_date)
3	-> Seq Scan on ledger (cost=0.00..817800.52 rows=49999852 width=16) (actual time=0.325..9249.584 rows=50000000 loops=1)
4	-> Hash (cost=17353.00..17353.00 rows=1000000 width=24) (actual time=358.724..358.727 rows=1000000 loops=1)
5	Buckets: 131072 (originally 131072) Batches: 16 (originally 8) Memory Usage: 7842kB
6	-> Seq Scan on library_transactions (cost=0.00..17353.00 rows=1000000 width=24) (actual time=0.032..132.765 rows=1000000 loop...
7	Planning Time: 1.793 ms
8	Execution Time: 3458071.033 ms

Total rows: 8 of 8 Query complete 00:57:41.243



```sql
explain analyse select library1.transaction_id,library1.book_id,new_library.transaction_date
from  new_library,library1 where library1.transaction_id=new_library.transaction_id;
```

	QUERY PLAN text
1	Hash Join (cost=33760.00..72812.01 rows=1000000 width=12) (actual time=1054.839..1054.843 rows=0 loops=1)
2	Hash Cond: (new_library.transaction_id = library1.transaction_id)
3	-> Seq Scan on new_library (cost=0.00..24706.00 rows=1000000 width=8) (actual time=24.218..178.227 rows=1000000 loo...
4	-> Hash (cost=17353.00..17353.00 rows=1000000 width=8) (actual time=328.030..328.033 rows=1000000 loops=1)
5	Buckets: 262144 Batches: 8 Memory Usage: 6935kB
6	-> Seq Scan on library1 (cost=0.00..17353.00 rows=1000000 width=8) (actual time=0.093..162.211 rows=1000000 loop...
7	Planning Time: 6.678 ms
8	Execution Time: 1056.571 ms

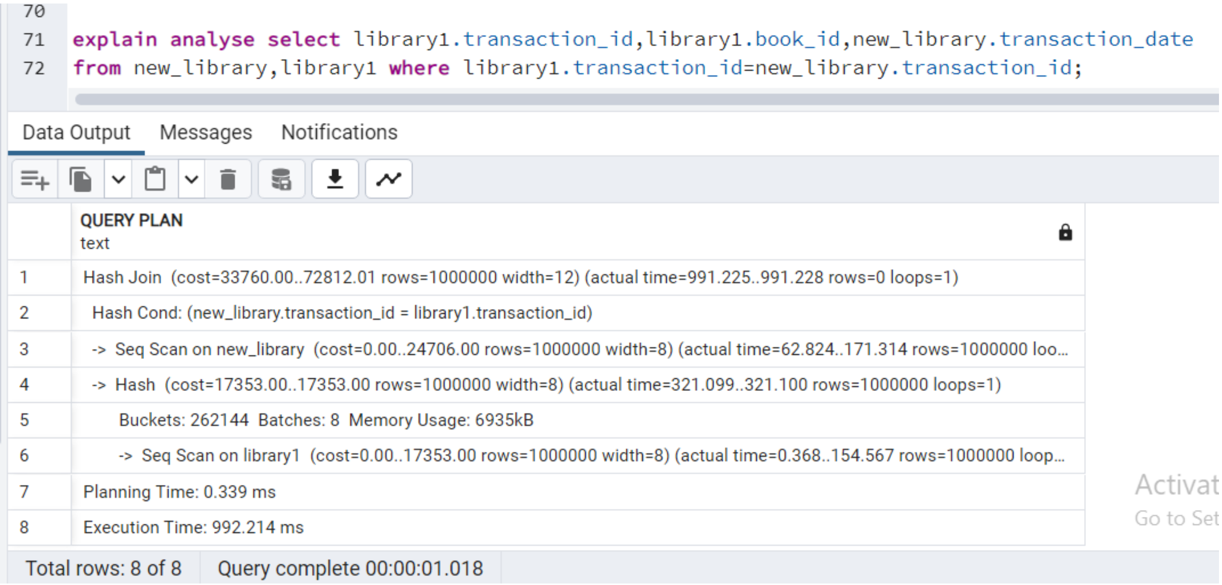


```sql
SET max_parallel_workers_per_gather TO 3;
```

```
SET

Query returned successfully in 87 msec.
```

Total rows: 8 of 8 Query complete 00:00:00.087



```
explain analyse select library1.transaction_id,library1.book_id,new_library.transaction_date
from new_library,library1 where library1.transaction_id=new_library.transaction_id;
```

	QUERY PLAN text
1	Hash Join (cost=33760.00..72812.01 rows=1000000 width=12) (actual time=991.225..991.228 rows=0 loops=1)
2	Hash Cond: (new_library.transaction_id = library1.transaction_id)
3	-> Seq Scan on new_library (cost=0.00..24706.00 rows=1000000 width=8) (actual time=62.824..171.314 rows=1000000 loo...
4	-> Hash (cost=17353.00..17353.00 rows=1000000 width=8) (actual time=321.099..321.100 rows=1000000 loops=1)
5	Buckets: 262144 Batches: 8 Memory Usage: 6935kB
6	-> Seq Scan on library1 (cost=0.00..17353.00 rows=1000000 width=8) (actual time=0.368..154.567 rows=1000000 loop...
7	Planning Time: 0.339 ms
8	Execution Time: 992.214 ms

Total rows: 8 of 8   Query complete 00:00:01.018

Outcomes:

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

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

In conclusion, the experiment highlighted the pivotal role of the degree of parallelism in shaping the efficiency of the join operation between the "ledger" and "library_transactions" tables. A higher degree of parallelism demonstrated improved performance, particularly beneficial for large datasets, where the workload was effectively distributed among multiple processors. Conversely, a lower degree of parallelism resulted in suboptimal performance, emphasizing the importance of finding an optimal balance based on specific workload characteristics and system resources.

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/