Experiment No.1

Title: Execution of Parallel Database queries.

Batch:B1 Roll No.:16010420133 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;

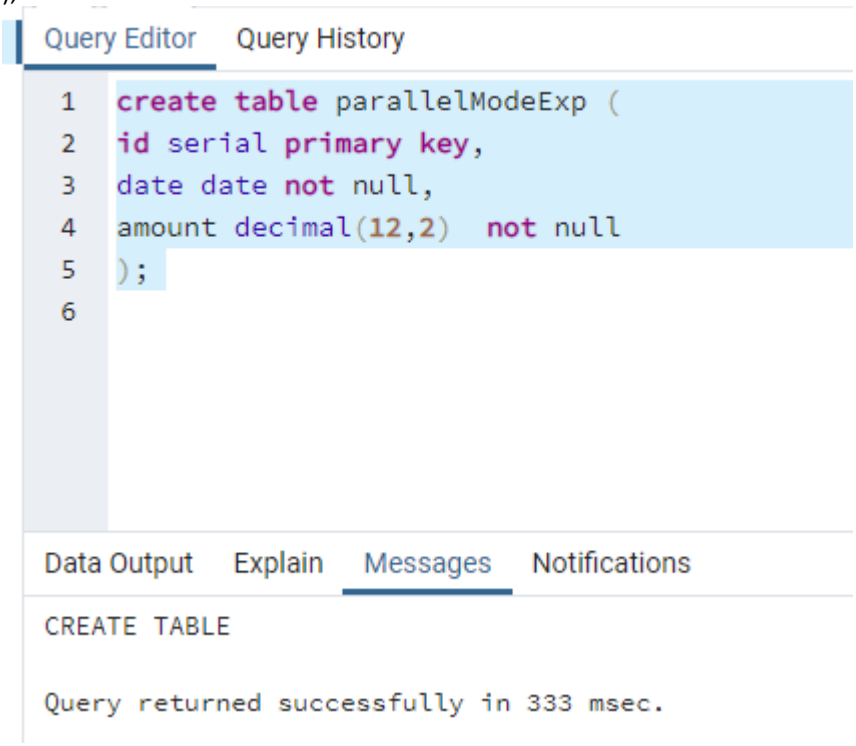
Questions:

1. Explain the parallelism achieved in the experiment you performed.
2. With comparison of the results explain how degree of parallelism (no of parallel processors) affect the operation conducted.

Results: (Program printout with output)

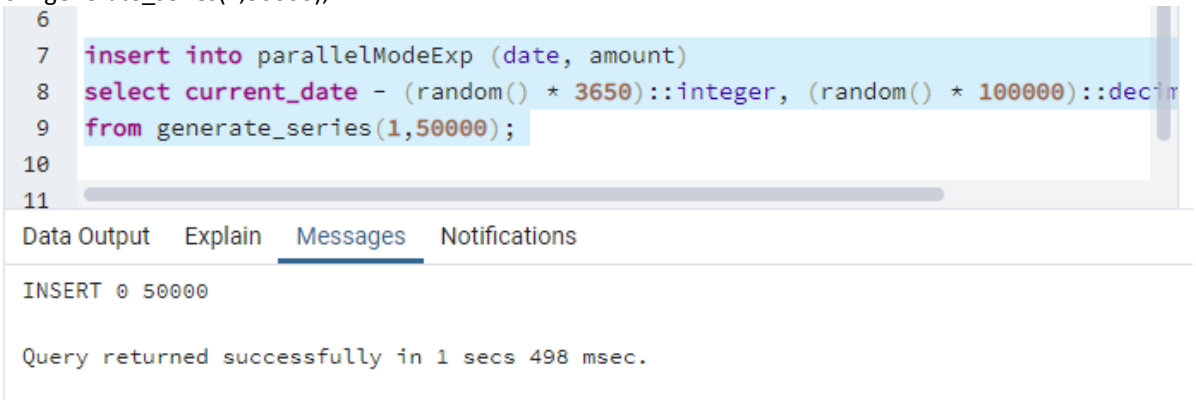
Create Table:

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



Insert Query:

```
insert into parallelModeExp (date, amount)
select current_date - (random() * 3650)::integer, (random() * 100000)::decimal(12,2) - 50000
from generate_series(1,50000);
```



Explain Analyze Query

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

```
11  explain analyze select sum(amount)  from parallelModeExp;
12
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=934.00..934.01 rows=1 width=32) (actual time=22.640..22.641 rows=1 loops=1)
2	-> Seq Scan on parallelmodeexp (cost=0.00..809.00 rows=50000 width=8) (actual time=0.032..6.836 rows=50000 loops=1)
3	Planning Time: 2.633 ms
4	Execution Time: 22.754 ms

Show Query:

show max_parallel_workers_per_gather;

```
13  show max_parallel_workers_per_gather;
14
```

Data Output | Explain | Messages | Notifications

	max_parallel_workers_per_gather text
1	2

Set Query:

set max_parallel_workers_per_gather to 4;

```
15  set max_parallel_workers_per_gather to 4;
16
```

Data Output | Explain | Messages | Notifications

```
SET

Query returned successfully in 191 msec.
```

Explain Analyze Query:

explain analyze select sum(amount) from parallelModeExp;

```
17  explain analyze select sum(amount)  from parallelModeExp;
18
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=934.00..934.01 rows=1 width=32) (actual time=25.378..25.379 rows=1 loops=1)
2	-> Seq Scan on parallelmodeexp (cost=0.00..809.00 rows=50000 width=8) (actual time=0.031..7.323 rows=50000 loops=1)
3	Planning Time: 0.158 ms
4	Execution Time: 25.435 ms

Sort Query:

```
create table tableToSort(
        id serial primary key,
        name varchar(20),
        amount decimal(12,2) not null
);
insert into tableToSort(name,amount) values('a',250);
insert into tableToSort(name,amount) values('b',200);
insert into tableToSort(name,amount) values('c',150);
insert into tableToSort(name,amount) values('d',50);
```

select * from tableToSort;

explain analyze select * from tableToSort order by name;

set max_parallel_workers_per_gather to 4;

explain analyze select * from tableToSort order by name;

```
30  explain analyze select * from tableToSort order by name;
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=54.62..56.54 rows=770 width=78) (actual time=0.113..0.116 rows=4 loops=1)
2	Sort Key: name
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on tabletosort (cost=0.00..17.70 rows=770 width=78) (actual time=0.025..0.027 rows=4 loops=1)
5	Planning Time: 0.183 ms
6	Execution Time: 0.141 ms

```
33  set max_parallel_workers_per_gather to 4;
```

Data Output | Explain | Messages | Notifications

```
SET

Query returned successfully in 112 msec.
```

```
35  explain analyze select * from tableToSort order by name;
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=54.62..56.54 rows=770 width=78) (actual time=0.166..0.171 rows=4 loops=1)
2	Sort Key: name
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on tabletosort (cost=0.00..17.70 rows=770 width=78) (actual time=0.075..0.080 rows=4 loops=1)
5	Planning Time: 0.217 ms
6	Execution Time: 0.215 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)

_____ We executed Parallel Database queries.

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/