Date 4/5/2022

Expt. No. 11 Page No. 10

Experiment - 11

Aim: Indexing and query processing

SQL queries:

1) analyze;

2) select * from comments where ID = 99999;

→ Time to execute: 102 ms, 83 ms, 104 ms, 97 ms, 90 ms

2) select * from comments where rating = 5 and item_id = 99982;

→ Time to execute: 98 ms, 108 ms, 126 ms, 112 ms, 111 ms

* The difference in time for same query can be explained as below:

All operating systems and all applications use a concept called 'caching'.

It means - when the data is first read from a slow memory device (like, a hard disk), it is saved in a fast memory device (like, RAM) for some time, to facilitate faster lookups.

The same applies to RDBMs.
First time the data blocks that build up your results are read from disk, second time they are read from memory.

Teacher's Signature : _____

The difference in time for both query can be explained as follows:

When planning a query, Postgres will look at each potentially relevant index. It will then either pick what appears to be the most suitable index for the query or will decide to use no index at all if none of the relevant indexes appear suitable at all.

If the query is already using an index and you create a less suitable index, or if the query is not using an index and you create an unsuitable index; Postgres will take longer to plan the query as it now has another index file it has to look at and compare to other potential query plans and hence the as no index is present at start the overhead time is required to decide one for index scan.

*/

3) explain select * from comments where ID = 99999

c1: 0.29, c2: 8.31, row: 1, width: 78

/* An index scan is when SQL server must scan the data or index pages to find the appropriate records. The index scan performs a B-tree traversal, walks through the leaf nodes to find all matching entries, and fetches the corresponding table data. */

explain select * from comments where rating = 5 and item_id = 99982;
c1: 0.00, c2: 2857, row: 1, width: 78

Teacher's Signature : _____

Date

Expt. No.

Page No. 12

/* Analyze command give time execution of 392 ms suggesting that indexing the data make the analysis more easier which in turn takes less time to gathers statistics for the query planner to create the most efficient query execution paths. */

select * from comments where ID = 99999;

→ Time to execute: 82 ms, 81 ms, 95 ms, 103 ms, 84 ms

/* The time of execution have comparative slightly lower down after creating indexes because index on id is present for query plan. */

select * from comments where rating = 5 and item_id = 99982;

→ Time to execute: 95 ms, 108 ms, 93 ms, 90 ms, 79 ms

/* The time of execution have lowered down after creating indexes as id is index which leads the filtered data from rating to go through index scan */

5) explain select * from comments where rating = 5;

explain select * from Comments where rating = 5 and comment = '0';

create index id ON comments (item_id);

Time for q1 execution before indexing: 82 ms, 81 ms, 95 ms, 103 ms, 84 ms
Time for q2 execution after indexing: 65 ms, 61 ms, 68 ms, 70 ms, 73 ms
Time for q2 execution before indexing: 95 ms, 108 ms, 93 ms, 90 ms, 79 ms

Teacher's Signature: _____

Date

Expt. No. Page No. 13

Time for q2 execution after indexing : 184, 200, 188, 176, 189 ms.

Query q1 plan after indexing :
"Index Scan using ID on comments (cost=0.29..8.31 rows=1 width=98)"
"Index Cond: (item_id = 99999)"

Query q2 plan after indexing :
"Index Scan using ID on comments (cost=0.29..8.31 rows=1 width=78)"
"Index Cond: (item_id = 99982)"
"Filter: (rating = 5)"

Teacher's Signature :

Date 11/5/2022

Expt. No. 12 Page No. 14

Experiment-12

Aim: Transactions and Concurrency

SQL queries:

Transactions:

1)

1) begin;

2) select * from student where name = 'Tanaka';

3) delete ~~student~~ from student where name = 'Tanaka';

4) select * from student where name = 'Tanaka';

5) roll back;

6) select * from student where name = 'Tanaka';

Observation:

In the database there is one student having name = 'Tanaka' so when it is deleted table and if select operation is executed to find student with name = 'Tanaka' the query will return empty table. But if a rollback is executed and then if select operation is executed the original entry can be seen again.

Teacher's Signature : _____

1) Query returned successfully in 66 msec.

2)

id	name	dept_name	tot_cred
98988	Tanaka	Biology	120

3) Query returned successfully in 92 msec.

4)

id	name	dept_name	tot_cred

5) Query returned successfully in 80 msec

6)

id	name	dept_name	tot_cred
98988	Tanaka	Biology	120

Date

Expt. No. Page No. 15

Conclusion:

The above scenario is observed due to execution of begin and roll back command. As we know by executing begin command indicate beginning of a transaction so till it is indicated that transaction is ended the changes can be reflected in database but are not yet committed. Rollback command takes the database to last committed state which was before execution of begin.

Concurrency:

2)
```
begin;
update student set tot_cred = 55 where name = 'Tanaka';

begin;
select * from student where name = 'Tanaka';

show transaction isolation level;
```

/* The transaction isolation level used is read committed i.e. default. So when both the transaction are started concurrency is maintained before starting of transaction but as transaction is ongoing and not yet committed the changes of tot_cred done in first window is not updated in window 2. */

/* The first commit ensures the update in database to be seen in all database copies hence tot_cred value in 2nd window gets update to 55 as commit has been executed and database has been made concurrent */

Teacher's Signature :

Id	name	dept_name	tot_cred
98988	Tanaka	Biology	120

Transaction-isolation
read committed

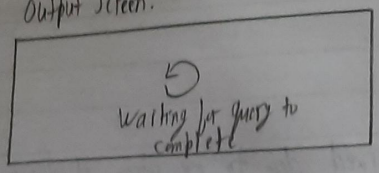
Date

Expt. No. Page No. 16

```
begin;
update student set tot_cred = 44 where name = 'Tanaka';

begin;
select min (tot_cred) from student where name = 'Tanaka'.
Update student set tot_cred = (select min (tot_cred) from student where
 name = 'Tanaka') + 20 where name = 'Tanaka';

/* The second window updates the tot_cred value from 55 as till
 then the tot_cred value update to 44 from 1st window is not yet
 committed and hence when the second window is committed the tot_cred
 value is 75 and not 64. */

begin;
 set transaction isolation level serializable;

select * from ~~tanaka~~ student where name = 'Tanaka';

select min (tot_cred) from student where name = 'Tanaka';
update student set tot_cred = (select min (tot_cred) from student
                               where name = 'Tanaka') + 20
                               where name = 'Tanaka';

select id, salary from instructor where id in ('22222', '15151');

begin;
Set transaction isolation level serializable;
```

Teacher's Signature :

Output screen:



Id	name	dept_name	tot_cred
98988	Tanaka	Biology	55

Id	Salary
15151	40000
22222	95000

Date

Expt. No. Page No. 17

update instructor set salary = (select salary from instructor where id = '22222') where id = '15151';

update instructor set salary = (select salary from instructor where id = '15151') where id = '22222';

commit;

/* Error occurs since serializability is used and both are trying to work on the same snapshot and postgres detects these operations cannot be committed together nor in serial manner and therefore postgres throws an error saying serialize access not possible and when we select and see the output, both the salaries are same and they do not get interchanged as desired */

Teacher's Signature :