Erin Zahner

Dr. F

CSC 351

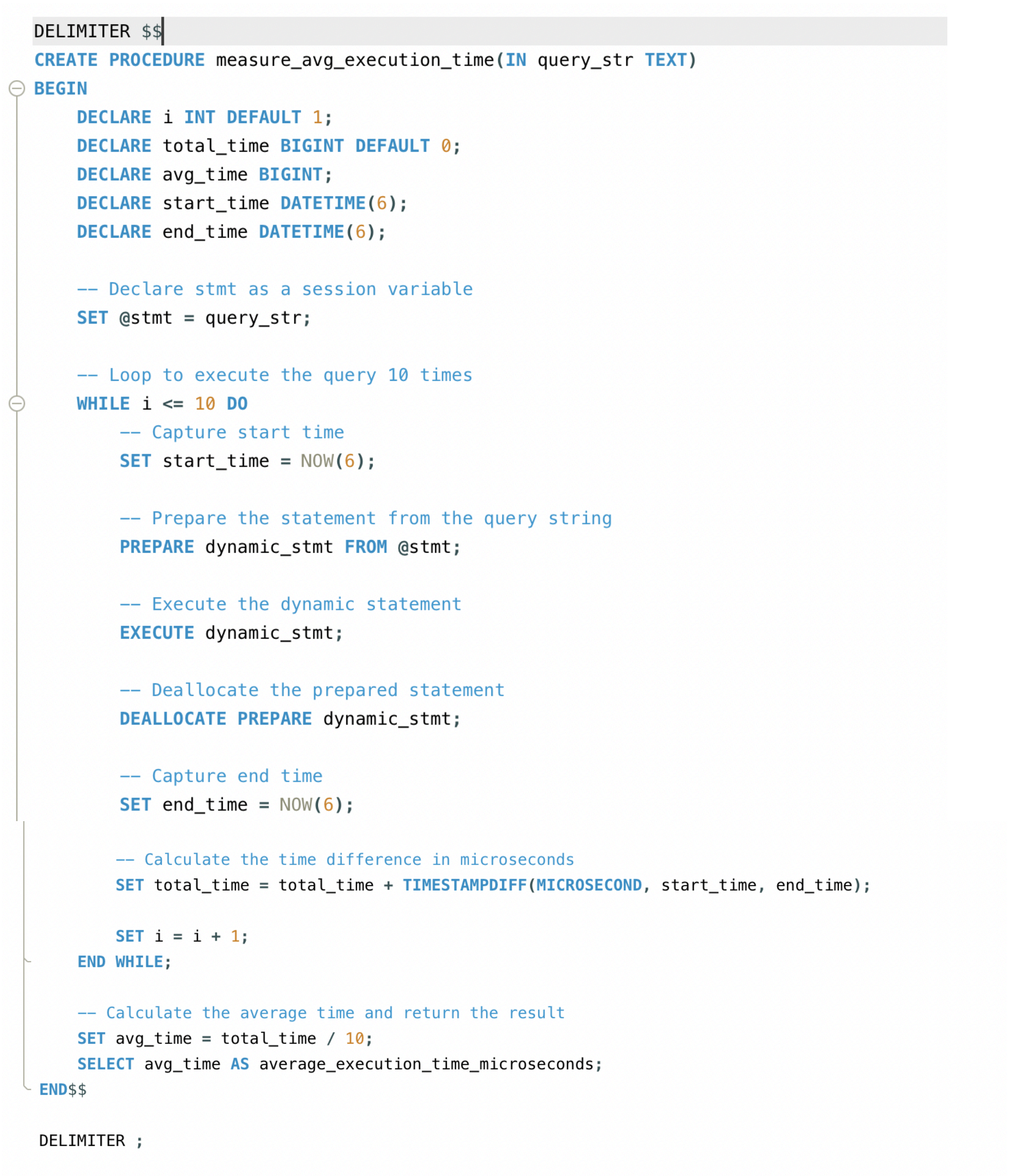
10 December 2024

Assignment 6

Query Type	Description	Dataset Size	Index Type	(Microseconds)
Point Query 1	SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance = 50000	50,000 records	Without Indexes	29121
			With Index (e.g. bname)	332
Point Query 2	SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance = 50000'	100,000 records	Without Indexes	50796
			With Index (e.g. balance)	415
Point Query 3	SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance = 50000'	150,000 records	Without Indexes	73326
			With Index (e.g. bname)	426
Range Query 1	SELECT count(*) FROM accounts WHERE branch_name =	50,000 records	Without Indexes	97481

	"Downtown" AND balance BETWEEN 10000 AND 5000;			
			With Index (e.g. balance)	159
Range Query 2	SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance BETWEEN 10000 AND 5000;	100,000 records	Without Indexes	121781
			With Index (e.g. bname)	163
Range Query 3	SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance BETWEEN 10000 AND 5000;	150,000 records	Without Indexes	147125
			With Index (e.g. balance)	163

Average Time Stored Procedure



```sql
DELIMITER $$
CREATE PROCEDURE measure_avg_execution_time(IN query_str TEXT)
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE total_time BIGINT DEFAULT 0;
    DECLARE avg_time BIGINT;
    DECLARE start_time DATETIME(6);
    DECLARE end_time DATETIME(6);

    -- Declare stmt as a session variable
    SET @stmt = query_str;

    -- Loop to execute the query 10 times
    WHILE i <= 10 DO
        -- Capture start time
        SET start_time = NOW(6);

        -- Prepare the statement from the query string
        PREPARE dynamic_stmt FROM @stmt;

        -- Execute the dynamic statement
        EXECUTE dynamic_stmt;

        -- Deallocate the prepared statement
        DEALLOCATE PREPARE dynamic_stmt;

        -- Capture end time
        SET end_time = NOW(6);

        -- Calculate the time difference in microseconds
        SET total_time = total_time + TIMESTAMPDIFF(MICROSECOND, start_time, end_time);

        SET i = i + 1;
    END WHILE;

    -- Calculate the average time and return the result
    SET avg_time = total_time / 10;
    SELECT avg_time AS average_execution_time_microseconds;
END$$

DELIMITER ;
```

In this stored procedure, I created a method to measure the average execution time of a query by running it dynamically 10 times. I started by accepting the query as input and assigning it to a session variable (@stmt). Inside the loop, I used NOW(6) to capture the start time, prepared the query using PREPARE dynamic_stmt FROM @stmt, and executed it with EXECUTE dynamic_stmt. After execution, I deallocated the prepared statement with

DEALLOCATE PREPARE dynamic_stmt and captured the end time using NOW(6). I calculated the time difference for each execution in microseconds and added it to a total time counter. Finally, after 10 iterations, I divided the total time by 10 to get the average execution time and returned it as a result.

Results

Point Queries with Indexes:

The execution times were very low (332, 415, 426 microseconds) because the indexes allowed the database to efficiently locate specific values directly, avoiding full table scans. The slight increase in execution time with larger dataset sizes is expected, as even with indexes, the database must navigate slightly deeper into the indexed structure for larger tables. However, the logarithmic efficiency of indexes keeps this increase minimal.

Point Queries without Indexes:

These queries took much longer (29,121 to 73,326 microseconds) because, without indexes, the database had to perform a full table scan, checking each record for the specified condition. As the dataset size increased, execution times rose significantly due to the linear scaling of a full table scan. Larger datasets mean more records to check, leading to much slower performance.

Range Queries with Indexes:

The execution times (159, 163, 163 microseconds) were even faster than indexed point queries. This happened because the database used the indexes to efficiently scan only the relevant range of values, benefiting from sequential access in the indexed structure. Notably, the execution times remained nearly constant across dataset sizes, showcasing how indexes maintain performance even as the data grows.

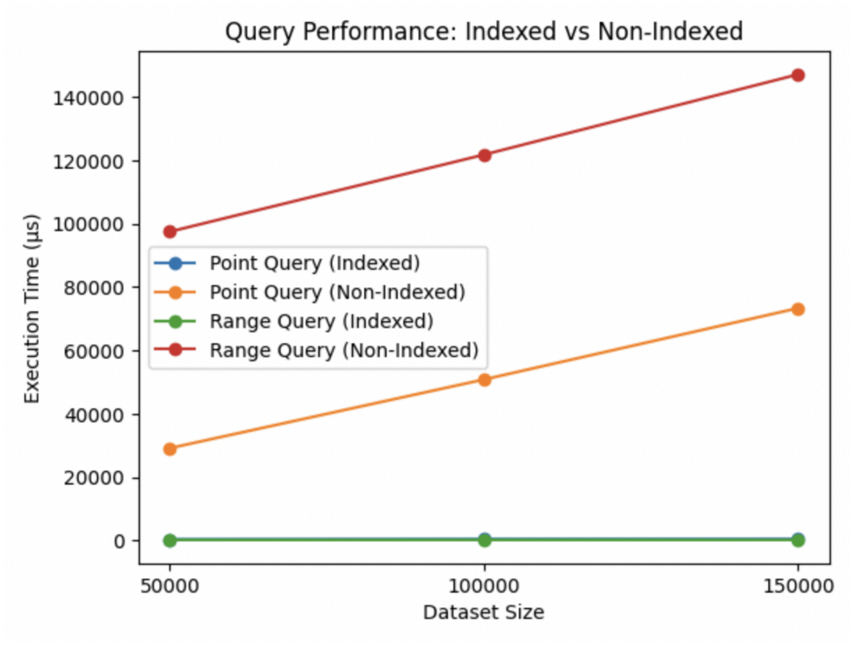
Range Queries without Indexes:

These were the slowest (97,481 to 147,125 microseconds). Without indexes, the database had to perform a full table scan to evaluate whether each record fell within the specified range. The execution time scaled poorly with dataset size, as the database had to process more records, further illustrating the inefficiency of range queries without indexing on larger datasets.

Conclusion:

These results highlight the importance of indexing in query optimization. While both point and range queries benefit significantly from indexes, range queries showed greater efficiency with indexes and much worse performance without them, as they rely heavily on scanning. This demonstrates the value of indexed structures in improving query performance, especially for larger datasets.

Graph of Average Times


Query Performance: Indexed vs Non-Indexed
Execution Time (µs)
Dataset Size
Point Query (Indexed)
Point Query (Non-Indexed)
Range Query (Indexed)
Range Query (Non-Indexed)