Introduction to Databases, Spring 2020

Homework #4 (May 27, 2020)

Student ID 2015313754

Name TaeHyung Gil

Compress 'main.c', 'BTREE.c', 'BTREE.h' and 'your report' (this current document file) and submit with the filename 'HW4_STUDENT ID.zip'

NOTE: You can add/modify functions if you want, but don't use additional libraries.

(1) [**30 pts**] Implement **insertion** and **deletion** operations of B-tree and write the codes. In addition, for given element sequences, show the results together. **(Insertion : 15pts, Deletion 15 pts)**

Definition of B-tree

1. Every node has at most m children (m: B-tree of order).

2. Every non-leaf node (except root) has at least ⌈m/2⌉ children.

3. A non-leaf node with k children contains k−1 keys.

4. All leaves appear in the same level

To implement the B-tree, please refer to the following sites.

- https://en.wikipedia.org/wiki/B-tree#Terminology

- https://www.cs.usfca.edu/~galles/visualization/BTree.html

(a) You should fill the implementation code in the B-tree template.

Answer: Submit your code to i-campus. Don't write your code here.

(b) Show the B-tree for each case.

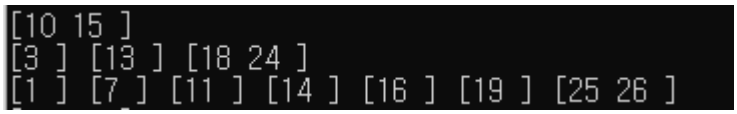
Answer: Show your results. (Drawing or Snapshot)

1) Max degree = 3

Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26)

2) Max degree = 4

Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26)



For Deletion :

1) Max degree = 3

Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26) Remove (13)

2) Max degree = 4

Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26) Remove (13)

(2) [20 pts] Compare the index scan and full table scan using SQL queries on MySQL. The selectivity of a predicate indicates how many rows from a row set will satisfy the predicate.

$$\text{selectivity} = \frac{Numbers\ of\ rows\ satisfying\ a\ predicate}{Total\ number\ of\ rows} \times 100\%$$

Compare the running time between index scan and full table scan according to different data selectivity and draw the graph to compare two scan methods depending on the selectivity. (Fix the total number of rows as 20,000,000). You also should explain the experimental results.

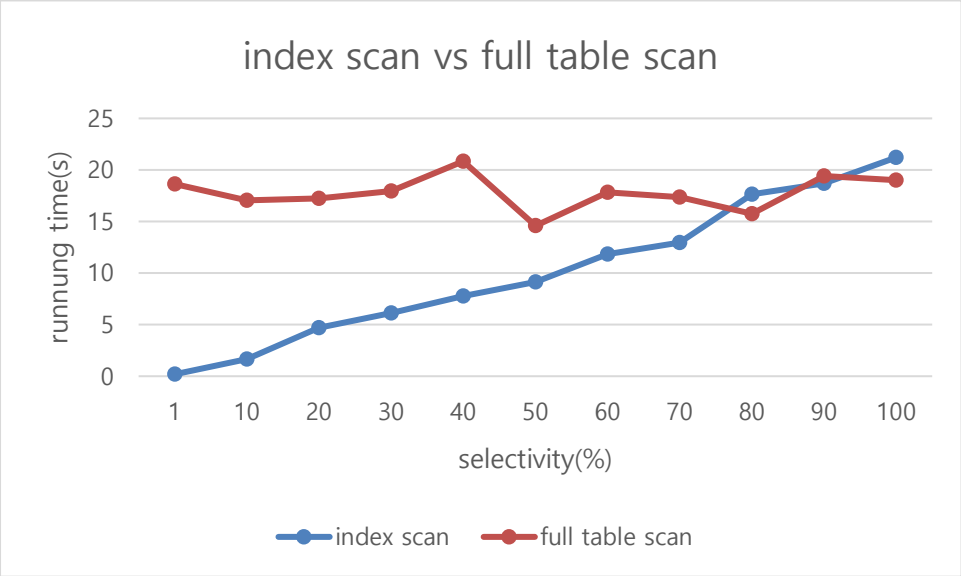
Example code for generating synthetic table.

```
/* Make a table */
DROP TABLE TEST;
CREATE TABLE TEST (a INT, b INT);

DELIMITER $$

DROP PROCEDURE IF EXISTS loopInsert $$

CREATE PROCEDURE loopInsert()
BEGIN
            DECLARE i INT DEFAULT 1;
            WHILE i <= 20000000 DO
                        INSERT INTO TEST (a, b) VALUES (i, i);
                        SET i = i + 1;
            END WHILE;
            COMMIT;
END$$

DELIMITER ;

SET autocommit=0;
CALL loopInsert;
COMMIT;
SET autocommit=1;

/* Make a index */
ALTER TABLE TEST ADD INDEX(a);

/* Compare the running time between index scan and full table scan at selectivity 50% */
SELECT SUM(a)
FROM TEST
WHERE a > 10000000;

SELECT SUM(b)
FROM TEST
WHERE b > 10000000;
```

(a) Comparison graph for running time over selectivity.

Please note that this figure is only for example. It is incorrect.

Answer: Show the comparison graph.



(b) Explain why index scan is faster or slower than full table scan depending on the selectivity in your comparison results.

Answer: Explain your comparison graph(= answer of (2)-(a)).

Full table scan은 table에 있는 모든 Row들을 순차적으로 읽습니다.

예를 들어,

SELECT SUM(b)

FROM TEST

WHERE b > 10000000;

Query를 수행한다면, 모든 row들을 읽으며 해당 row의 b attribute가 10000000보다 큰지 확인하며 query를 수행하게 됩니다. 10000000보다 크다면, 해당 row는 선택되어 sum(b)에 영향을 줄 것입니다.

따라서, full table scan의 running time은 이론적으로 일정해야 합니다. 저의 그래프에서는 변동이 있지만, 이는 실험 환경에 따라 다를 수 있다고 생각합니다.

그에 반해, table의 index인 a를 이용한 query인

SELECT SUM(a)

FROM TEST

WHERE a > 10000000;

Query를 수행할 때에는 index range scan을 수행하게 됩니다. B+ Tree와 같이 Record들이 index에 따라 정렬되어 있다면, 최초의 시작점인 a=10000000 인 record(leaf node)를 먼저 찾고, 그 이후로는 그 다음 leaf node로 이동하며 종료점까지 Scan하게 됩니다. 즉, 한번의 Random access(a=10000000 인 record(leaf node) 찾기)를 수행하고, 그 이후로는 sequential access를 수행합니다.

따라서 index range scan의 running time은 random access에 수행되는 시간이 매우 적다면, scan하게 되는 range에 비례하여 증가합니다.

Scan해야 하는 range가 적은 selectivity에서는 index scan이 full table scan보다 running time이 적게 필요합니다. 이는, index scan의 sequential access가 적게 필요하기 때문입니다.

Scan해야 하는 range가 많은 selectivity에서는 index scan과 full table scan이 어떻게 optimization이 되어 있느냐에 따라 결과가 다를 수 있다고 생각합니다. Selectivity가 90%이상이 된다면 index scan의 sequential access 범위가 전체 table의 크기와 비슷해지므로 index scan에서 한 leaf node에서 다음 leaf node로 이동하는 cost와 full table scan에서 한 data block에서 다음 data block으로 이동하는 cost가 유사하다면, running time, 또한 비슷한 값을 갖게 될 것입니다.

따라서, (2)-(a)의 그래프가, 낮은 selectivity에서는 index scan의 running time이 월등히 낮았고, 80%이상의 높은 selectivity에서는 유사한 running time을 보였습니다.