Introduction to Databases, Spring 2020

Homework #4 (May 27, 2020)

Student ID : 2016312568

Name : Jung Hee Yoon

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.

(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)

```
[10 15 ]
[3 ] [13 ] [18 24 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 ] [25 26 ]
```

2) Max degree = 4

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

```
[10 15 ]
[3 ] [13 ] [18 24 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 ] [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)

```
[10 18 ]
[3 ] [15 ] [24 ]
[1 ] [7 ] [11 14 ] [16 ] [19 ] [25 26 ]
```

2) Max degree = 4

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

```
[10 18 ]
[3 ] [15 ] [24 ]
[1 ] [7 ] [11 14 ] [16 ] [19 ] [25 26 ]
```

(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.

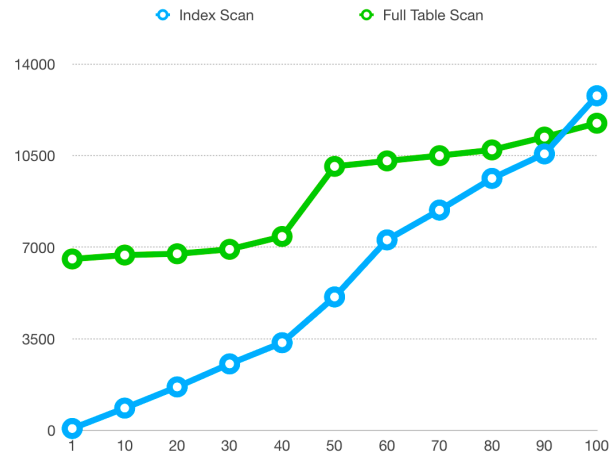
```
/* Make a table */
DROP TABLE TEST;
CREATE TABLE TEST (a INT, b INT);

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.

In graph, Y-axis's unit is milliseconds to express more clearly. X-axis is selectivity and Y-axis is runtime (milliseconds unit).

Comparison graph for running time over selectivity.

	1	10	20	30	40	50	60	70	80	90	100
Index Scan	70	850	1660	2540	3350	5100	7280	8420	9630	10570	12790
Full Table Scan	6550	6700	6750	6920	7410	10090	10300	10500	10720	11210	11740



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

In Graph, Index Scan is more faster than until 90% selectivity, but more than 90% selectivity Full Table Scan is more faster than Index Scan. This result can say Index Scan isn't always faster than Full table scan.

SELECT SUM(a) as selectivity_1_a FROM TEST WHERE a <= 200000;

SELECT SUM(a) as selectivity_1_a FROM TEST WHERE b <= 200000;

Upper query is one of queries to get runtime each selectivity. In Select SUM(a) when a<=200000 case, this query need only index. And in Second query, this query need not index. So in HW case, queries need only index or only not index. There are 3 cases when using index.

1. If column info that query need is only index, query execute by the way named Index scan. This scan is efficient when low selectivity. However, Index scan is inefficient in high selectivity. Because index is role like index of book. So, In High selectivity case, just find sequentially can be more efficient than find by index.

2. If column info that query need is not index, query execute by the way named Full Table Scan. This scan is just scan all rows sequentially. Just sequentially can be faster than find by index. Because when find rows. sequentially finding is lower cost than finding by index.

3. If column info that query need is not 'only' index (need both index and others), query execute by the way Index Scan. but, In this case, find rowID and go rowID to get other column info. This way named random access. It can be inefficient when conditional statement include not index (abandon row when this row is not correct by conditional statement 'after' random access).

In our HW cases are just using index case and just using not index case. In result graph, Index Scan is very efficient in low selectivity, but Index scan is inefficient in high selectivity (in this graph, more than 90%). Because, sequentially finding is lower cost than finding by index. In low selectivity, sequentially finding way is access all row in table but, finding by index way access low row. In High selectivity, sequentially finding way is access all row in table, finding by index was also access high rows. Finding by index ways's cost is more expensive than sequentially finding way's cost. Then In low selectivity case, Index Scan is more efficient than Full Table Scan. But in high selectivity case, Full Table Scan is more efficient than Index Scan.