Introduction to Databases, Spring 2019

Homework #3 (May 13, 2019)

Student ID _____

Name _____

(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)**

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

For Insertion :

```
void _insert(BTreeNode* present, int k)
{
        // Initialize index as index of rightmost element
        int i = present->n;

        // If this is a leaf node
        if (present->leaf == true)
        {
                                while (i > 0 && k < present->keys[i - 1]) {
                        present->keys[i] = present->keys[i - 1];
                        i--;
                }

                present->keys[i] = k;
                present->n += 1;
                _balancing(present);
        }
        else // If this node is not leaf
        {

                while (i > 0 && k < present->keys[i-1])
                        i--;

                _insert(present->C[i], k);
        }
}
```

For Deletion :

```
void _remove(BTreeNode* present, int k)
{
	int i = 0;
	while (i < present->n && k > present->keys[i])
		i++;

	// If the element is founded
	if (present->keys[i] == k)
	{
		BTreeNode* Sib;
		// If it is not leaf, and there is left child
		if (present->C[i] != NULL) {
			Sib = present->C[i];
			int maxind = Sib->n;
			present->keys[i] = Sib->keys[maxind - 1];

			// Mitigate to the deletion of left child
			_remove(present->C[i], Sib->keys[maxind - 1]);
		}
		// If it is not leaf, and there is right child
		else if (present->C[i + 1] != NULL) {
			Sib = present->C[i + 1];
			present->keys[i] = Sib->keys[0];

			// Mitigate to the deletion of right child
			_remove(present->C[i], Sib->keys[0]);
		}

		// The leaf case
		else
		{
			present->n--;
			while (i < present->n) {
				present->keys[i] = present->keys[i + 1];
				i++;
			}

		}
		// Balancing and finish
		_balancingAfterDel(present);
		return;
	}

	// If there are no element are founded
	else {
		// If it is leaf, it means that there are no elements
		if (present->leaf == true) {
			printf("No such element exist");
			return;
		}
		// If it is not a leaf, go to its sibling
		else {
			_remove(present->C[i], k);
			return;
		}
	}
```

(b) Write the snapshot results for your test.

Inserted 1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25 and deleted 13

For Insertion :

1) Max degree = 3

```
max degree : 3, After Insertion
[10 15 ]
[3 ] [13 ] [18 24 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 ] [25 ]
```

2) Max degree = 4

```
max degree : 4, After Insertion
[10 ]
[3 ] [13 15 18 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 24 25 ]
```

For Deletion :

1) Max degree = 3

```
max degree : 3, After Deletion
[10 18 ]
[3 ] [15 ] [24 ]
[1 ] [7 ] [11 14 ] [16 ] [19 ] [25 ]
```

2) Max degree = 4

```
max degree : 4, After Deletion
[10 ]
[3 ] [15 18 ]
[1 ] [7 ] [11 14 ] [16 ] [19 24 25 ]
```

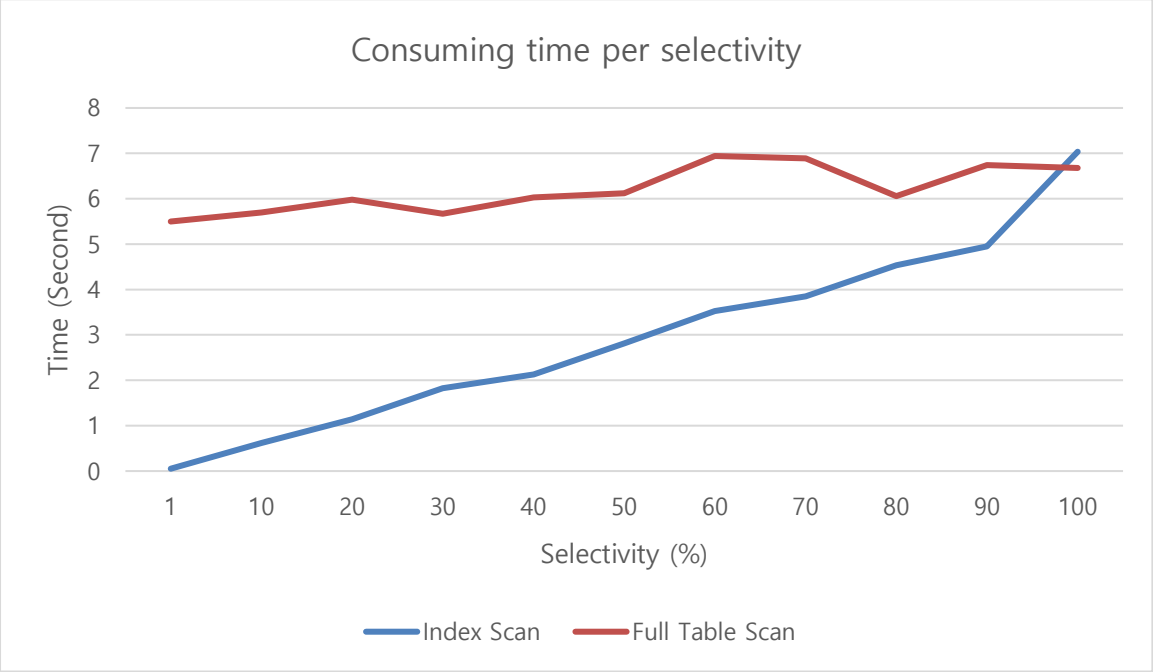
(2) [20 pts] Compare the index scan and full table scan using SQL queries on MySQL. The selectivity of a predicate indicates how many rows from a row set will satisfy the predicate.

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

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 10,000,000). You also should explain the experimental results.

(a) Comparison graph for running time over selectivity.

총 각 selectivity마다 5번의 실험을 했고, 소요 시간의 산술 평균값을 사용했습니다.



(b) Explain comparison results.

Index Scan은 selectivity에 영향을 크게 받는 방면, full table scan은 selectivity에 크게 영향을 받지 않는다. 이는 index를 통해 찾는 경우 몇 개를 찾는 지에 따라 찾아야 하는 행의 수가 정해진다. 찾아야 하는 행의 수가 적으면, 적게 찾으면 되고, 많으면 또 그만큼 많이 찾아야 한다. 반면, Full table scan은 몇 개의 행을 찾는지에 상관없이 모든 테이블을 다 살펴야 하기 때문에, selectivity에 영향을 받지 않는 양상을 보인다.

즉, selectivity가 낮은 경우에는 index를 활용한 scan을 사용하는 것이 효율적이다. Selectivity가 높은 경우에는, index로 인한 오버헤드를 굳이 사용할 필요가 없으므로 full table scan을 활용하는 것이 더 바람직한 것으로 보인다. 이는 selectivity가 100%일 때 index scan이 full table scan보다 더 많은 시간이 걸린다는 사실에서부터 확인할 수 있다.