DB2

Forum: https://forum-db.informatik.uni-tuebingen.de/c/ss20-db2

Assignment 8 (23.06.2020)

Submission: Tuesday, 30.06.2020, 10:00 AM

Relevant videos: up to `DB2 - Chapter 10 - Video #49`.

https://tinyurl.com/DB2-2020

1. [10 Points] **Clustering Factor**

   Even if a table is clustered according to an index, subsequent `INSERT` and `UPDATE` operations may destroy the table's perfect sort order. Instead, tuples may be inserted on any page of the table, possibly reducing the *clustering factor* of the table.

   Clustered Tuple: A tuple t of table T is *clustered* w.r.t an index I, if the tuple t', immediately preceding t in the sequence set of I, is located on the same page of T as t or on a page of T physically located directly before the page of t. The first tuple in the sequence set of I is always clustered.

   Clustering Factor: The *clustering factor* of table T is $\frac{|\textit{clustered} \text{ tuples in } T|}{|\text{tuples in } T|}$

   Load file `cluster-factor.sql` to create table `indexed(a INT, b TEXT, c NUMERIC(3,2))` as introduced in the lectures. Table `indexed` is equipped with two indexes, `indexed_a` on column `a` and `indexed_c_a` on composite key ⟨`c`,`a`⟩. The table is clustered according to index `indexed_c_a`.

   (a) Write a query to compute the *clustering factor* of table `indexed` with respect to index `indexed_c_a`. **Hint:** Use window function `LAG()`[1] to access the tuple immediately preceding the current tuple with respect to the given sort order.

   (b) Write a query to compute the *clustering factor* of table `indexed` with respect to index `indexed_a`. Explain the difference to the result of 1a, **briefly**.

   (c) Perform the following update and remeasure the *clustering factor* w.r.t. `indexed_c_a`. What happened?

   ```
   UPDATE indexed SET b=lower(b) WHERE a%2=0;
   ```

   (d) Table `indexed` was created with a default *fillfactor* of 100% (complete packing). If a smaller fillfactor had been specified (between 10 and 99), the table pages would have been packed only to the indicated percentage; the remaining space on each page would have been reserved for updating rows on that page.[2] Set the *fillfactor* to 60% and restore clustering using the following commands. Repeat the experiment of 1c and explain your observations **briefly**.

   ```
   ALTER TABLE indexed SET ( fillfactor = 60 ); CLUSTER indexed;
   ```

[1] https://www.postgresql.org/docs/current/static/functions-window.html#FUNCTIONS-WINDOW-TABLE

[2] https://www.postgresql.org/docs/current/static/sql-createtable.html#SQL-CREATETABLE-STORAGE-PARAMETERS

2. [5 Points] **B$^+$Tree - Variation**

In this assignment, let us assume that all key values inserted into the B$^+$Tree are unique and the index' `Insert`- and `Delete`-operations do **not implement *redistribution***.

Find five *additional* key values $a, \ldots, e$ such that when you insert the entries in order $a, \ldots, e$ and then delete them in reverse order $e, \ldots, a$, the following holds:

(a) The resulting tree is the **same** as in Figure 1.

(b) The resulting tree **differs** from the one in Figure 1.

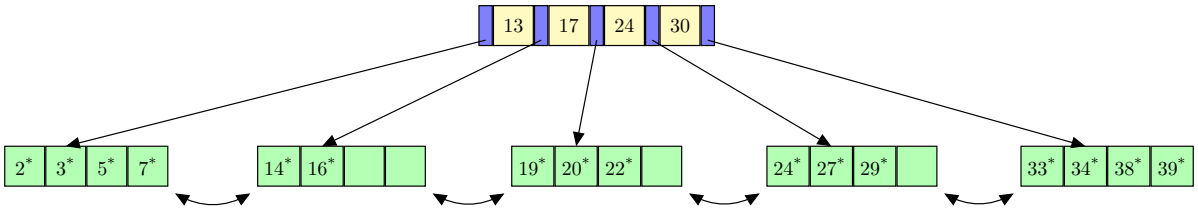

Figure 1: A B$^+$Tree

3. [15 Points] **Solving Predicate Equations**

Create and populate table `unary(a INTEGER PRIMARY KEY)` as provided in `unary.sql`.

Examine the following four queries. Assume that variable `:n` is not `NULL` and set to an *arbitrary* integer value (for example: `\set n 64`).

Q_1:

```
SELECT u.a
FROM unary AS u
WHERE u.a - 1 = :n;
```

Q_2:

```
SELECT u.a
FROM unary AS u
WHERE (u.a::double precision)^2 = :n;
```

Q_3:

```
SELECT u.a
FROM unary AS u
WHERE u.a % :n = 1;
```

Q_4:

```
SELECT u.a
FROM unary AS u
WHERE :n / u.a = 1;
```

For each of the queries, answer the following questions and briefly explain your answer:

(a) Is the query evaluation supported by index `unary_a`?

(b) If not, try to find an **equivalent** query that is supported by index `unary_a`. If there is no such query, explain why. Be careful to not alter the query semantics. This means that each rewritten query must return exactly the same results as the original one, for all *possible* instances of table `unary`, and all *possible* values of `:n`.

(c) Instead of rewriting the query, an *expression index* can be used to support the query. Define this index and check whether it is actually used. With regards to all four queries, what is the disadvantage of defining individual expression indexes?