Prof. Dr. T. Grust, Benjamin Dietrich, Christian Duta SS 2018

Databasesystems 2
Forum: `https://forum-db.informatik.uni-tuebingen.de/c/ss18-db2`

Assignment 8 (19.06.2018)

Submission: Tuesday, 26.06.2018, 10:00 AM

Please note that students currently have the opportunity to **evaluate lectures**. Please help us to improve **your** courses by providing precious feedback. Check your Mailbox now and participate **today**.

1. [8 Points] **B$^+$Tree - Variation**

   Consider the B$^+$Tree of order $o = 2$ in Figure 1 where the `Insert`- and `Delete`-operations do **not** implement *redistribution*.

   (a) Find five *additional* key values $a, \ldots, e$ such that the following holds: If you insert the entries in the order $a, \ldots, e$ and then delete them in the reverse order $e, \ldots, a$, the resulting tree is the **same** as in Figure 1.

   (b) Find five *additional* key values $a, \ldots, e$ such that the following holds: If you insert the entries in the order $a, \ldots, e$ and then delete them in the reverse order $e, \ldots, a$, the resulting tree **differs** from the one in Figure 1.

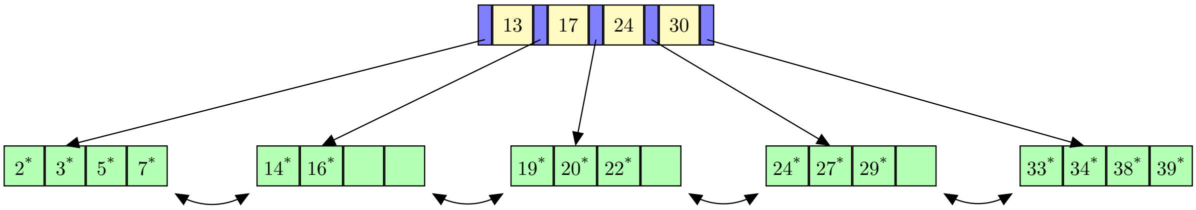


Figure 1: A B$^+$Tree

2. [11 Points] **Cluster 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 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 2a, **briefly**.

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

3. [11 Points] **UB-Trees: B$^+$Tree on Z-order values**

One possibility to index a table on multiple columns are *composite indexes*. However these do not support both dimensions equally. Instead, the first dimension dominates the order of entries in a composite index.

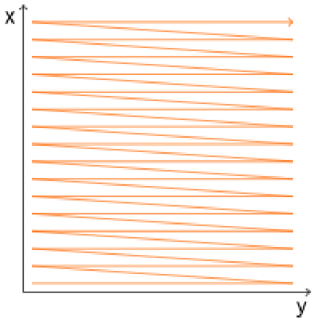
Consider a table `points(x INT, y INT)` representing points in a two dimensional space:

```
-- A two dimensional space
CREATE TABLE points (x INT, y INT);

-- Populate space with points
INSERT INTO points
  (SELECT x, y
   FROM    generate_series(0,99) AS x,  generate_series(0,99) AS y);

-- Create primary key index
CREATE UNIQUE INDEX points_x_y ON points USING btree (x,y);
```

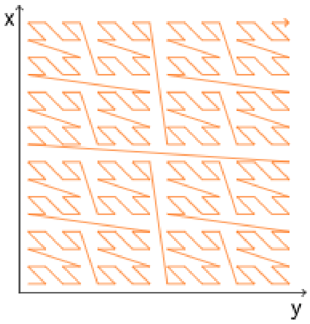
Composite B$^+$Tree indexes reflect the spacial locality of points in a two-dimensional space only in the first dimension `x`. Locality regarding *both* dimensions can be achieved with a B$^+$Tree index over an expression that combines both dimensions in a so called *Z-order value*:

```
CREATE INDEX points_z_value_x_y ON points USING btree (z_order_value(x,y));
ANALYZE;
```

The Z-order value of a point (`x`,`y`) is calculated using the bit-interleaving function provided in `zorder.sql`.[2] This index reflects locality for both input dimensions:



$\langle x{,}y\rangle$ **(concatenated)**



Z-Order

(a) Write a SQL query that, given an index `I` on table `points` and an index leaf page number `n`, returns all points that are located on `I`'s leaf page `n`. Use function `bt_page_items(indexname TEXT, pageno INT)` provided by `CREATE EXTENSION pageinspect` to access the *RID*s of all items on a B$^+$Tree page (e.g. `pageno` = 7). Assume that the given page is a leaf page.

(b) For both indexes, `points_x_y` and `points_z_value_x_y`:

- Choose an arbitrary leaf page.
  (Check that attribute `type` returned by `bt_page_stats(indexname, pageno)` is `'l'`.)
- Collect the points located on this page using the query of 3a.
- Illustrate the `x`, `y` location of these points in the two-dimensional space, plotting them on a grid of size 100 × 100 with *Gnuplot*.

  You can access a web-based version of Gnuplot at `http://gnuplot.respawned.com`.[3]
- Describe and explain your findings **briefly**.

[2] See Wikipedia for more information on Z-order values: `https://en.wikipedia.org/wiki/Z-order_curve`

[3] Examples for *"Data"* and *"Plot script"* can be found in files `data.txt` and `points.plot`.