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

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

Assignment 9 (26.06.2018)

Submission: Tuesday, 03.07.2018, 10:00 AM

1. [20 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 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 explain your answer briefly:

   (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 has to return the exact same results as the original one, for all *possible* instances of table `unary` and all *possible* vales of `:n`.

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

2. [10 Points] **Performance Impact of Indexes**

   Obviously, indexes have impact on the performance of data modifying operations. But, can an additional index also **decrease** the performance of a `SELECT`-query?

   Consider the following table:

   ```
   CREATE TABLE skewed (
      sort        INTEGER NOT NULL,
      category    INTEGER NOT NULL,
      interesting BOOLEAN NOT NULL
   );

   INSERT INTO skewed
      SELECT i AS sort, i % 1000 AS category, i > 50000 AS interesting
      FROM   generate_series(1, 1000000) i;

   CREATE INDEX skewed_category ON skewed (category);
   ANALYZE skewed;
   ```

   We now want to query the first twenty interesting rows in category 42:

   ```
   SELECT *
   FROM   skewed
   WHERE  interesting AND category = 42
   ORDER  BY sort
   LIMIT  20;
   ```

(a) Print the query plan using `EXPLAIN ANALYZE` and describe in your own words, how query evaluation proceeds and how this is supported by index `skewed_category`.

In general, it is a good idea to support top-N queries (`ORDER BY`/`LIMIT`) by an index on the sort criteria.

(b) Create an additional B$^+$Tree index `skewed_sort` on column `sort` of table `skewed`.

(c) You will notice that index `skewed_sort` will decrease the performance of our query. **Why?**
Again, print the query plan, explain it in your own words and compare it to the plan of 2a. Give a short conclusion of your findings.