Javier Palomares

Professor Miranker

EE382 Data Engineering

11/9/2018

Query Plans in SQLite and Postgres

I chose to investigate the query plans SQLite generated for the 20 queries from the assignment. I observed that for several of the join queries, SQLite chooses to scan tables using an automatic index. For the remaining join queries, the behavior was to use full table scans over all tuples of the relations. I investigated the automatic index in SQLite's documentation and found that SQLite might create an automatic index that lasts only for the duration of a single SQL statement. The cost of constructing the automatic index is O(T(R)*logT(R)) and the cost of doing a full table is O(T(R)), so an automatic index will be created if SQLite expects that the lookup will be run more than logT(R) times during the course of the SQL statement. Moreover, the current implementation on SQLite only includes nested joins (no merge-join or hash-join). This behavior explains why I observed no difference when joining relations over join keys where secondary indexes exists compared to joining with secondary indexes. Therefore, the best set of indexes in SQLite for the workload is to have no indexes. Because of this, I do not recommend choosing SQLite when join queries are frequent.

After the difficulties I observed with SQLite, I migrated the experiment to PostgreSQL, which does implement merge-join and hash-join, and was able to observe interesting results that allowed me to conclude the best set of indices is secondary indices on the `ot`,`hund`, and `ten` attributes. The following are the observations I made that formed this conclusion:

1. Query 3 `A’.ht = B’.ht` does not make use of the secondary index on `ht`. The query plan is to use a hash join just like Query 2 where no secondary index is available.

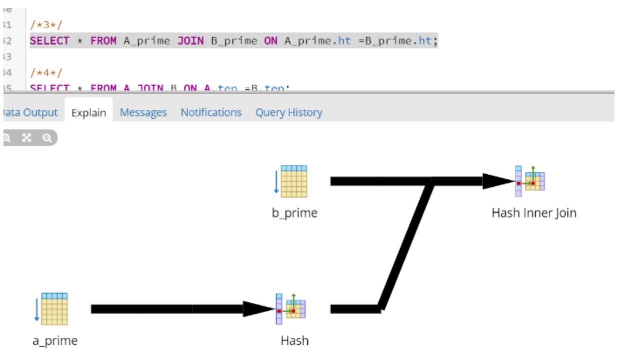

Figure 1. Adding a secondary index on ht does not change the query plan

2. In contrast, Query 5 `A’.ten = B’.ten` produces in a merge sort using the index to avoid a sort. The secondary index is beneficial for Query 5. Query 4, with no available secondary index, must do a sort before the merge join.

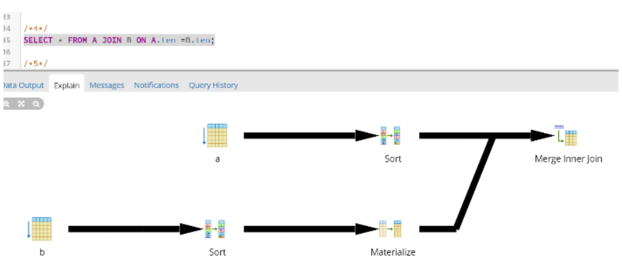

Figure 2. Query 4 A.ht=B.ht. With no secondary index on ten, a sort must be done to use merge join.

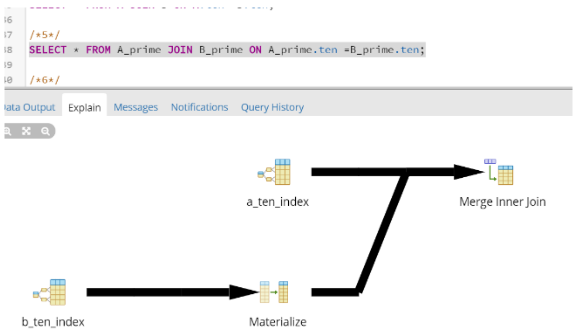

Figure 3. Query 5. A'.ht=B'.ht. With a secondary index, no sorting is needed prior to merge join.

3. Three way joins on attributes with `ht` and `tt` use a hash-join regardless of whether a secondary index exists. In contrast, the query plan is to use merge join when joining on `ot`, `hund`, or `ten` attributes.

Taking these 3 observations in place, I concluded that having a secondary index on `ot`, `hund`, and `ten` is beneficial in reducing the costs of joins on these attributes, and secondary indices on `ht` and `tt` are not beneficial.

The rest of the paper serves as an appendix. I include screenshots and observations for all of the queries from the assignment.

Report_Appendix.pdf