Project 2 – Advanced Databases

Problem statement

We want to execute the following query:

(?a)---follows--->(?b)---friendOf--->(?c)---likes--->(?d)---hasReview--->(?e)

Follows, friendOf, likes and hasReview are all triple tables.

We want to join them always on the object of the left triple table and the subject of the right triple table.

In “Dataset Description” the datasets in which the triple tables are stored are further described.

In order to solve this query we need to be able to join two tables. Triple Tables as well as a “result table” and a triple table. A “result table” is the result of a previous join.

The required join algorithms are further described in the section Algorithm description

The code for the project is on GitHub:

https://github.com/Schelle7/Project2-Advanced-Databases

Algorithm description

Hash-join

Input: two tables

The tables are joined on the rightmost attribute of the left and the leftmost attribute of the right table.

Table1 (the left table) is used to build a dict with the join attribute as key and a list of the rows where it is the right most attribute in table1.

The next step is to iterate over the rows of table2 (the right table) and use the join attribute (the left most attribute) to get all rows of table1 with this join value and then join them together.

The result can then be saved.

Sort-merge-join

Input: two tables

The tables are joined on the rightmost attribute of the left and the leftmost attribute of the right table.

Then the algorithm sorts both tables ascending on the join attribute.

Because the two tables are now sorted it is possible to start at the beginning of the two tables.

The left table is denoted as table1 and the right table as table2.

We have one number for each table, that indicates the current position in each table, pos1 and pos2.

Pos1 and pos2 are at the beginning of table1 and table2 respectively.

To start off we compare the entries at pos1 and pos2 in the two tables.

If they have the same join attribute we add them to the result of the join and continue looking at the next entries in table2 and add all entries where the join attribute is still the same together with the row at pos1 of table1.

Then we increase pos1 by 1 and leave pos2 the same.

If the two join attributes from pos1 and pos2 are not the same we increase the position with the smaller join attribute (alphabetically) by 1 and start from the beginning.

Skip sort merge join

Is the same as sort merge join except that we try to increase by more than 1 if the two join attributes are not equivalent.

If the two join attributes are not the same we try to increase the position with the smaller join attribute (alphabetically) by 100. If the join attribute from the same table is still alphabetically smaller we continue from the beginning of the iteration.
Otherwise we just increase the position by 1.

This allows us to go faster over parts of the table where the join attributes are consistently smaller in one of the tables.

Join multiple tables

This function is used to call the three joins after one another with the result of the previous join.

Dataset description

The dataset consists of two files:

Each of them contains a triple table with subject, property, object in each line.
The size of the file and the number of relevant triples is as follows

File1 “100k.txt” with 109’0310 triples.

Num friendOf triples: 45712

Num follows triples: 31887

Num likes triples: 1032

Num hasReview triples: 1453

File2 “watdiv.10M.nt” with 10’916’458 triples.

Num friendOf triples: 4’491’142

Num follows triples: 3’289’307

Num likes triples: 112’401

Num hasReview triples: 149’634

In the data preprocessing step the triples get converted into a dictionary that has the properties as key with a table of the corresponding subject object table.

Experiment and analysis

In order to join the 4 triple-tables we have to perform three joins.
Below I listed the time required for each join as well as the required total time for each join algorithm.

100k.txt

These are the times for the queries on the triple-table from 100k.txt

Hash join

1. Join needed 0.73
2. Join needed 6.63
3. Join needed 76.08

Hash join query needed: 83.91 seconds to run the query

Sort merge join

1. Join needed 1.96
2. Join needed 5.06
3. Join needed 17.51

Sort merge join query needed: 24.53 seconds to run the query

Sort merge join skip

1. Join needed 1.96
2. Join needed 4.09
3. Join needed 16.39

Sort merge join skip query needed: 22.80 seconds to run the query

In my experiment skip sort-merge join was the fastest

sort merge join is second

hash join is last

It is interesting to note that hash join was the fastest on the first join but much worse later on. This is most likely due to saving and retrieving ever growing lists when saving a new value in the dictionary.

Sort merge join and sort merge join skip are unsurprisingly similar in runtime and skipping values (increasing the position by 100) only really gives a benefit once the tables get bigger.

It would be interesting to look into a sort hash algorithm that gets rid of often retrieving and storing the lists in the dictionaries while not having to compare many values to get to the right one.

watdiv.10M.nt

For the second part of the experiment on the "watdiv.10M.nt" file I had to split up the data since my Computer had a MemoryError after using about 16GB RAM and 40-50GB in the committed storage.

This might affect the results but I don't see any other way of doing it.

The reason why I think this might affect the results is, that a) the sorting with runtime n*log(n) is worse on many smaller datasets.

Furthermore the hashing is probably better because the size of the lists of rows we retrieve and store are not as big.

This being said below the results follow.

Sort merge join

Sort merge join query needed: 3594.74 seconds to run the query

Skip sort merge join

Skip sort merge join query needed: 3251.00 seconds to run the query

Hash join

Hash join query needed: 8789.33 seconds to run the query

The results are the same on "watdiv.10M.nt as on "100k.txt".

Skip sort merge join is the fastest

Sort merge join is second

Hash join is last

Conclusion

Skip sort merge join is the fastest of the three selected algorithms.

The biggest problem of hash join is, that we store and retrieve ever growing lists of rows in the dictionaries quite often.

The biggest problem for sort merge join is, that it iterates over join attributes that aren't relevant.

Skip sort merge tries to mitigate that problem.

It would be interesting to combine the algorithms by first sorting table1 to build the dictionary without repeatedly retrieving and storing the rows for the same join attributes.

This could be done by having a list that appends new rows until the join attribute changes and only then storing the list of rows in the dict once.

It might also be better to sort table2 in order to only retrieve each list of rows once.