PROJECT REPORT II

NAME: ROHIT NAIR

UFID: 4698-3602

The main components in the project are as follows:

Tables:

- Queries
  - This has the queries
- Advertisers
  - This has the advertiser data
- Keywords
  - This has the bid keywords data
- Query Tokens
  - This table is used to store the tokens from a query
  - This is computed only once per query at the first task running
  - So subsequent tasks need not find the tokens
  - So its precomputation in a way
- Bidders
  - This stores the bidders.
  - Cannot be precomputed since bidders change for each task
- Qualityscores
  - This can be precomputed since its static for each query and set of advertisers who bid it
  - With the initial code the precomputation is not done
  - Hence precomputation can be done now for optimization
- Budget
  - This stores the remaining budget for each task
  - Cannot be precomputed
- Task1ads
  - Stores the output for task1
- Task12ads
  - Stores the output for task2
- Task2ads
  - Stores the output for task3
- Task22ads
  - Stores the output for task4
- Task3ads
  - Stores the output for task5
- Task32ads

    - Stores the output for task6
- Adcount
    - Keeps track of the count of ads displayed
    - Cannot be precomputed

Functions

- Task1 – Takes 7-9 mins for 5000 queries
- Task2 – Takes 5-6 mins for 5000 queries
- Task3 – Takes 5–6 mins for 5000 queries
- Task4 – Takes 5-6 mins for 5000 queries
- Task5 – Takes 5-6 mins for 5000 queries
- Task6 – Takes 5-6 mins for 5000 queries
- Qualityscore – Precomputed along with Task1
- MAIN – The main function which just calls the tasks one after the other

Current Query Plan:

- Tokens are found from query and saved into querytokens
- Keywords and querytokens joined to get bidders and saved into bidder table
- Range query done on bidders to get topk advertisers
- Topk advertisers are saved into the corresponding output table

Optimizations Made

1. Presently the tokeniser works on a complex join query using connect by as well as multiple regular expressions. This can be eliminated and a separate tokeniser function can be called. This function will just retrieve the query using a simple sql query and then get the tokens one by one manually using space as a delimiter and then storing it as a oracle table variable and persist it in db.
2. Presently bidders are found and saved in bidders table from where the topk bidders are found. Here presently the bidders table doesn't have an index. The index can be added on the query id since each time we find the topk bidders for one single query id.
3. After selecting the ads, we update the budget table and the ads table. Here also the indexing is missing on my first implementation. Now we can add the index based on the advertiserid. Since budget is allocated to advertiserids.
4. Qualityscore table was initially made with the intention of precomputing the qualityscore. But I had missed out that on my first implementation. Now currently the qualityscore can be precomputed and the cursor opening can be eliminated from tasks 2 – 6. And also an index can be added to the qualityscore table.

5. From the second task onwards the precomputed querytokens table and the keywords table can be joined to eliminate the queries where it is impossible to have the advertisers.
6. SQLLDR was showing much improved speed when the running parallel for every table. SQLLDR processes were created for each of the dat files. It showed huge improvement in speeds.

Optimized query plan

It is almost the same apart from the way querytokens are found and qualityscore is precomputed. No further changes to queries were needed because all queries except the tokensing query were simple queries.

The difference was going to be made from the newly indexes

No new components were added apart from the indexes.

Performance Improvements

Running time has been saved by a minute for the first task and by about 1 and a half minutes for the rest of the tasks each. Please note that since only the functions and cursors and not queries were not optimized, query times in the optimized code were assumed to be the same. Therefore only the times for the tasks were measured as a whole.

Suspected Reasons

The things that were done has certainly speeded up the execution. Removing cursors I believe was the most important point. Each time the function was called the oracle system had to reserve extra memory for the cursor and init the vars needed for the cursor creation. This has now been avoided since the qualityscore has been precomputed as well as the querytoken cursor has been avoided from the second task onwards.

Things Learnt

Having a cursor in sql is very bad as far as time efficiency is concerned. Cursors should only be used as a last resort. Indexes if added intelligently can speed up large databases but might require additional storage. There are a number of ways where SQLLDR can be made faster. By running it parallel and by sorting the dat file according to the index. But in our case, that was already done in part I.