CS-631: ITDBMS
PROJECT REPORT

TITLE: Automatic Creation of Indices in PostgreSQL

PROBLEM STATEMENT:

In this project, you will modify the access layer of PostgreSQL to keep track of relation scans that could have benefited from an index, and if there are many such scans for a particular index, create the index automatically. (The index creation will be done by a separate process, not by the scanning process, since it can be expensive.) A commonly used technique is to check if the benefit of the index * number of queries > cost of creation of the index, and when this gets satisfied, you create the index. This can be extended to consider the cost of index maintenance if time permits.

IMPLEMENTATION:

In PostgreSQL, indices are not created for all tables and columns. In cases where we perform a query that accesses a column very frequently, that does not have an index associated with it, the cost of the query will be high, and over a long period of time, this cost adds up to make query execution very expensive.

In such cases, we could create an index on those accessed frequently and reduce the cost of subsequent executions. However, we must ensure that the index creation cost does not outweigh the rest of the execution cost.

To do so, we should track the execution cost and the index creation cost, and if the index creation cost is less than the execution cost, we will create the index.

However, keeping track and creating an index manually would be time-consuming, so we could automate this process to create indices in the background.

We could change the source code of PostgreSQL such that:

- On every SELECT query execution, if we have an equality predicate in that query, we track the column and the table.

- We do this by making the code output (table_name, column_name) to a file on such select queries.

- We do this by modifying the execMain.c file in src/backend/executor. This file contains the standard_ExecutorRun(), which is the function that executes queries.

- We get the coloumn_name and Relation id from the sourceText, part of queryDecs, a variable passed to the standard_ExecutorRun().

- We get the table_name from the relation id using the get_table_name() function.

- Then, we output the (table_name, column_name) to the file if the conditions are met.

- Using a different process, we can count the number of (table_name, column_name) occurrences and initiate the index creation process only if the number crosses a certain threshold.

- After creating the index, we add the list of created indices to another file so we do not try to generate the index multiple times.

FUNCTIONS ADDED:

1. bool has_equality_predicate(const char* query): Checks whether the query has equality in it or not. This indicates whether we are referring to a particular column.

2. void increment_full_scan_count(Oid, relid, const char* column_name): This is the function that writes (table_name, column_name) to the file whenever called.

3. char* extract_column_name(const char* query): This function extracts the column name from the query.

POTENTIAL IMPROVEMENTS:

1.) In this project, we used another process to create the index. We have written this process in Python. Ideally, this process should be a Postgres background worker that runs on the server and creates the index.

2.) Another potential improvement is the cost estimation. We used the number of times a column is referenced as the cost. If it crosses a certain threshold, we create the index. However, a more complex cost estimation should be used to obtain accurate and optimal results in actual implementation.

3.) Instead of using files to store stats and a list of created indices, we could create separate relations to contain this data so that the entire implementation stays within Postgres and does not need any external dependencies.

Done by:

24M0766 Nitin Chakravarthy Chilukuri

24M0802 Abhishek Kumar