1 Introduction

The Postgres query planner does not necessarily return the best plan even with interesting orders. As mentioned in the CORDS paper, one reason is the estimated numbers of rows returned by a scan or join can be way off [1]. For instance, imagine the estimate claiming that 1 row is returned from both relations in a join. The optimizer might estimate one row because of the indepedence assumption of AND clauses. The optimizer might do nested loop because everything fits in memory.

As a result, I've created this prototype to explore the possibility of an database operator manually manipulating the plan, or changing the estimates of the number rows returned from a base or joined relation.

2 Features

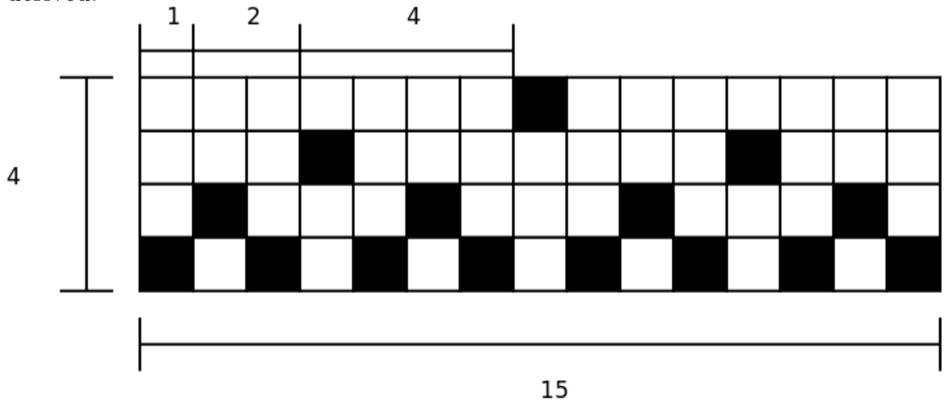
I have implemented a UI that displays the plan that the optimizer initially computes. The UI can display any number of levels of joins. From that UI, you can change which join algorithm should be applied on two relations. Lastly, the UI also allows you to modify the the estimated number of rows for each relation and then the query planner re-searches for a new plan.

3 How it Works

Here I provided a summary of the most complicated components of this project. A lot of challenges I encountered are not included here to keep the report brief.

3.1 UI

The UI was built using GTK+. The most complicated component is the display of tree. This was solved using a table like GUI abstraction, which a GUI widget can be placed at any row and column of the table. The follow diagram illustrates how the formula was derived.



The number of rows is the height of the plan. The child node of a parent, is 1 row below the parent. Additionally, it is $2^{ParentHeight-2}$ units to the left or the right of the parent.

3.2 Changing Joins

We wrap the plan tree datastructure returned by query planner inside our own tree so that each node of the tree has pointers to any important GUI widgets that the user may have manipulated. When the user changes the join on the GUI, we recursively rebuild the path upwards starting from the node that was changed.

3.3 Changing a Relation's Rows Estimates

We add a hashtable to the PlannerInfo struct. The hashtable is a map from relids of the relation to the overridden number of rows. If the hashtable does not contain an entry for a given relids, then the estimate for that relation has not been overridden. Where ever the estimated number of rows was accessed in costsize.c, we prepended a lookup to the hashtable to check if it has been overridden.

4 Future Plans

Firstly, this GUI runs on the database process. In order to keep the scope of the project minimal, I decided to not create somone messaging protocol for communicating back and forth with the database server. Secondly, currently there is no way to maniupate the structure of the tree. Thirdly, there is no way to change which scan should be performed on a based relation. Fourthly, only one pathkey is considered for the merge join. A variety of options should be presented to the user selecting merge join. Lastly, there is absolutely no validation.

References

[1] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 647–658, New York, NY, USA, 2004. ACM.