Cost Model & Statistics

Vivian Huang, Masha Oreshko, Kevin Geng

Goals Completed So Far

- Implement pg_statistic table for stats storage
  - Ability to store statistics in the table directly as a replacement for the current StatsStorage API
  - Ability to persist binary blobs e.g. TopKAggregate as VARBINARY
- Cost model that successfully chooses hash joins over nested loop joins
  - Implemented approximations of the nested loop join & hash join cost formulas from Postgres
  - Wrote tests that use various mock statistics, and having the cost model calculate costs based on those stats which verify that formulas are correctly identifying which operator to use

Goals To Still Complete

- Optimize cost model with quick lower bound cost to potentially avoid doing heavy costing work
- Write benchmarks to get real cpu costs for operations and emitting tuples
- Write more sophisticated costing formulas for other operators and test them
- Integration with bytecode, i.e. test ability to update stats storage including TopKAggregate from TPL

Testing

How we tested the correctness of our implementation

- Cost model tests:
  - Test to ensure nested loop join order is correct (table with smaller rows is the outer table)
  - Test to make sure that given a hash join vs. nested loop join, the hash join has the smaller cost
- pg_statistic functionality test:
  - Test that pg_statistic is updated properly on table creation/deletion
  - TODO: tests to ensure stats can be persisted / restored properly

Code quality

Strengths:

- Code is organized relatively well

Improvements to make:

- More documentation in cost model formulas to improve understanding & readability
- More comments in test code to make it easier to follow and maintain

Concrete tasks for future work

- Complete the rest of our goals
- Expand tests to use real selectivity & cardinality calculations from the statistics files (improving testing for those files & consequently working on improving those estimates)