15721 FINAL PRESENTATION
Stats & Cost Model

Allison Wang, Xiaoyou Wang, Xian Zhang
Motivation

1. Enable the collection of table statistics and cardinality estimation
2. Store stats in Catalog
3. Build cost model for optimizer to predict the optimal query plan
Goals

- 75%: Collect basic table statistics ✓
- 100%: Estimate cardinality and build cost model using statistics ✓
  - Handle a single table query
- 125%: Estimate join query cardinality

We’ve completed the 100% - 5% GOAL !!!
ANALYZE table1

Clients

Stats Collectors

Stats Managers

Stats Tables

- TableStatsCollector
- ColumnStatsCollector
- TupleSampler
- StatsStorage
- TupleSamplesStorage
- pg_catalog
  - pg_column_stats
- samples_db
  - 1_1
  - 1_2
  - 2_3
  - 3_4
Stats Collectors

- **cardinality**: hyperloglog
- **most common** `<values, frequency>`: count-min sketch + topk (priority queue)
- **histogram bounds**: “A Streaming Parallel Decision Tree Algorithm”
- More stats: `num_rows`, `frac_null`
Testing

- 2 types of tests: correctness and performance
- Full coverage: comprehensive unit tests suite for each class
  - 12 test files under optimizer folder
- Testing on **TPC-H** benchmark (1GB dataset)
  - Extra work: fixed some TPC-H bugs in Peloton
- Customer Table: 24MB (150000 tuples)
- Part Table: 24MB (200000 tuples)
- Lineitem Table: 725MB (6001215 tuples)
Result: Cardinality Estimation

**Table: Part**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Real</th>
<th>Est</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_brand</td>
<td>1e+03</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_comment</td>
<td>1e+03</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_container</td>
<td>1e+01</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_mfgr</td>
<td>1e+01</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_name</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>p_partkey</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>p_retailprice</td>
<td>1e+03</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_size</td>
<td>1e+03</td>
<td>1e+01</td>
</tr>
<tr>
<td>p_type</td>
<td>1e+03</td>
<td>1e+01</td>
</tr>
</tbody>
</table>

**Table: Customer**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Real</th>
<th>Est</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_acctbal</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_address</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_comment</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_custkey</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_mktsegment</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_name</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_nationkey</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
<tr>
<td>c_phone</td>
<td>1e+05</td>
<td>1e+05</td>
</tr>
</tbody>
</table>
Result: Most Frequent Value & Freq

Most Common Value & Frequency (brand)

<table>
<thead>
<tr>
<th>Brand#35</th>
<th>Brand#12</th>
<th>Brand#52</th>
<th>Brand#33</th>
<th>Brand#53</th>
</tr>
</thead>
<tbody>
<tr>
<td>8233</td>
<td>8167</td>
<td>8158</td>
<td>8141</td>
<td>8088</td>
</tr>
</tbody>
</table>

Most Common Value & Frequency (nationkey)

<table>
<thead>
<tr>
<th>nationkey</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>6161</td>
<td>real</td>
</tr>
<tr>
<td>6100</td>
<td>est</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nationkey</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>6012</td>
<td>real</td>
</tr>
<tr>
<td>6012</td>
<td>est</td>
</tr>
</tbody>
</table>
Result: Histogram Bounds

Real Histogram Bounds vs Equal Height Histogram Approximation

Frequency

p_size

0
10
20
30
40
50
ANALYZE

- Two interfaces for collecting table stats
  - AnalyzeStatsForTable, AnalyzeStatsForAllTables
- Implement command ANALYZE to collect stats for a specific table
- Demo:
Cost Model
Cost Model (Single Table)

SELECT id, name, COUNT(project_id)
FROM table
WHERE class_id = 15721 AND year < 2017
GROUP BY id, name
LIMIT 10

sel (P1 \land P2) = sel(P1) \times sel(P2)

sel (P1 \lor P2) = sel(P1) + sel(P2) - sel (P1 \land P2)
Standalone Cost Calculator Lib

- SeqScanCost
- IndexScanCost
- CombineConjunctionStats
- SortGroupByCost
- HashGroupByCost
- AggregateCost
- DistinctCost
- ProjectCost
- LimitCost
Standalone Selectivity Lib

- LessThan
- GreaterThan
- Equal
- NotEqual
- LessThanOrEqualTo
- GreaterThanOrEqualTo
- Like
- NotLike
Estimation Performance

SELECT * FROM part WHERE p_partkey < 10000;
- Actual: 9999 | Postgres: 9906 | Ours: 8080

SELECT * FROM part WHERE p_size = 49;
- Actual: 3945 | Postgres: 4213 | Ours: 3695

SELECT * FROM part WHERE p_partkey < 10000 AND p_size = 49;
- Actual: 174 | Postgres: 209 | Ours: 149

SELECT * FROM part WHERE p_partkey < 10000 OR p_size = 49;
- Actual: 13770 | Postgres: 13911 | Ours: 11625
Parting Thoughts

- Accurate cardinality estimation is hard
- Good base table statistics lead to better estimation for complicated predicates and joins
Next Steps

1. Integrate cost model into optimizer cost and stats calculator
2. Support string equality and cardinality using ARRAY type
3. Support LIKE operator selectivity using sampling
4. Cost model for join operators
5. EXPLAIN + new optimizer to fully test cost model performance
Code Quality and Stats

- **8000+** new lines of code
- **43** new files (excluding third-party)
- Group internal PR and code review: **36** PRs
- Well tested
- Highly modularized
Q: What makes you feel heartbroken?

A: make clean