Cost-based optimizer

Benefits of statistics

Starburst

Lesson objectives

Cost-based optimizer: Benefits of statistics

1. Describe what cost-based optimization (CBO) is and why it matters.
2. Explain the role that table statistics play in CBO.
3. Describe some of the ways in which queries can be optimized using CBO.
4. Demonstrate the nature of query improvements using CBO.

Table Statistics

Table:

- **row count**: the total number of rows in the table

Column:

- **data size**: the size of the data that needs to be read
- **nulls fraction**: the fraction of null values
- **distinct value count**: the number of distinct values
- **low value**: the smallest value in the column
- **high value**: the largest value in the column

```
trino> SHOW STATS FOR datalake.sales.lineitem;
  column_name  |  data_size   | distinct_values_count | nulls_fraction | row_count | low_value  | high_value
---------------+--------------+-----------------------+----------------+-----------+------------+------------
 orderkey      |         NULL |             1547451.0 |            0.0 |      NULL | 1          | 6000000
 partkey       |         NULL |              196274.0 |            0.0 |      NULL | 1          | 200000
 suppkey       |         NULL |               10000.0 |            0.0 |      NULL | 1          | 10000
 linenumber    |         NULL |                   7.0 |            0.0 |      NULL | 1          | 7
 quantity      |         NULL |                  50.0 |            0.0 |      NULL | 1.0        | 50.0
 extendedprice |         NULL |              936107.0 |            0.0 |      NULL | 901.0      | 104949.5
 discount      |         NULL |                  11.0 |            0.0 |      NULL | 0.0        | 0.1
 tax           |         NULL |                   9.0 |            0.0 |      NULL | 0.0        | 0.08
 returnflag    |    6001215.0 |                   3.0 |            0.0 |      NULL | NULL       | NULL
 linestatus    |    6001215.0 |                   2.0 |            0.0 |      NULL | NULL       | NULL
 shipdate      |         NULL |                2526.0 |            0.0 |      NULL | 1992-01-02 | 1998-12-01
 commitdate    |         NULL |                2466.0 |            0.0 |      NULL | 1992-01-31 | 1998-10-31
 receiptdate   |         NULL |                2554.0 |            0.0 |      NULL | 1992-01-04 | 1998-12-31
 shipinstruct  |  7.2006409E7 |                   4.0 |            0.0 |      NULL | NULL       | NULL
 shipmode      |  2.5717034E7 |                   7.0 |            0.0 |      NULL | NULL       | NULL
 comment       | 1.58997209E8 |             4668697.0 |            0.0 |      NULL | NULL       | NULL
 NULL          |         NULL |                  NULL |           NULL | 6001215.0 | NULL       | NULL
(17 rows)
```

Retrieve and calculate statistics

Retrieve statistics – works for all connectors.

```
SHOW STATS FOR table_name;
```

Calculate statistics – only works on the data lake connectors. For others, such as RDBMS-based catalogs, calculate statistics natively at the source.

```
ANALYZE table_name WITH (
   partitions = ARRAY[ARRAY['p2_value1', 'p2_value2']],
   columns = ARRAY['col_1', 'col_2'])
```

Note: Identifying specific columns and/or partitions is optional, but recommended.

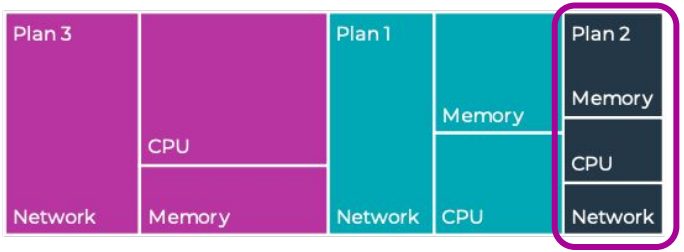
Cost-based optimizer (CBO)

What is it?

- Query Optimizer performs various transformations of a query plan to arrive at the most efficient variant.
- “Cost-based” means alternative plans according to the estimated “cost” to execute :
  - **CPU time, memory requirements, and network bandwidth** usage are the three dimensions that contribute to query execution time, both in single-query and concurrent workloads.

Starburst CBO

- Cost-Based Transformation engine:
  - Join reordering based on selectivity estimates and cost
  - Automatic join type selection (repartitioned vs broadcast)
  - Automatic left/right side selection for joined tables
- Support for statistics coming from connectors



Query Optimization

Pushdown

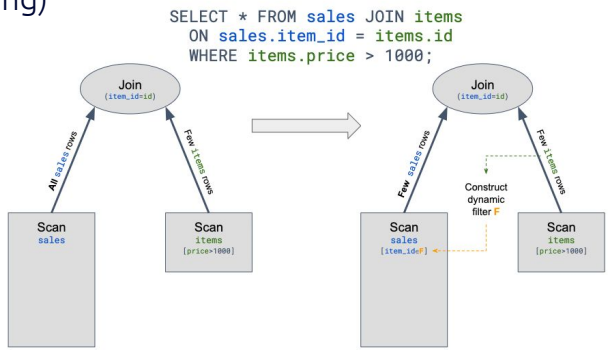
Move the filtering condition as close to the source of the data as possible.

- Some benefits:
  - improved overall query performance
  - reduced network traffic between Trino and the data source
  - reduced load on the remote data source
- Types:
  - Predicate
  - Aggregation
  - Join
  - Limit
  - Top-N

Dynamic Filtering

Significantly improve the performance of queries with selective joins by avoiding reading of data that would be filtered by join condition.

- Improve performance on federated queries
  - Better resource utilization for better data filtering
- Skip loading of partitions (dynamic partitioning pruning)



Lesson summary

Cost-based optimizer: Benefits of statistics

1. The cost-based optimizer (CBO) performs various transformations of a query plan to arrive at the most efficient variant.

2. CBO uses table statistics to identify the best plan.

3. SQL can be used to returned the optimized plan with, or without runtime details, generated by the CBO.

4. Queries are typically optimized for pushdown, join optimizations, and dynamic filtering.

Cost-based optimizer

Query plan analysis

Lesson objectives

Cost-based optimizer: Query plan analysis

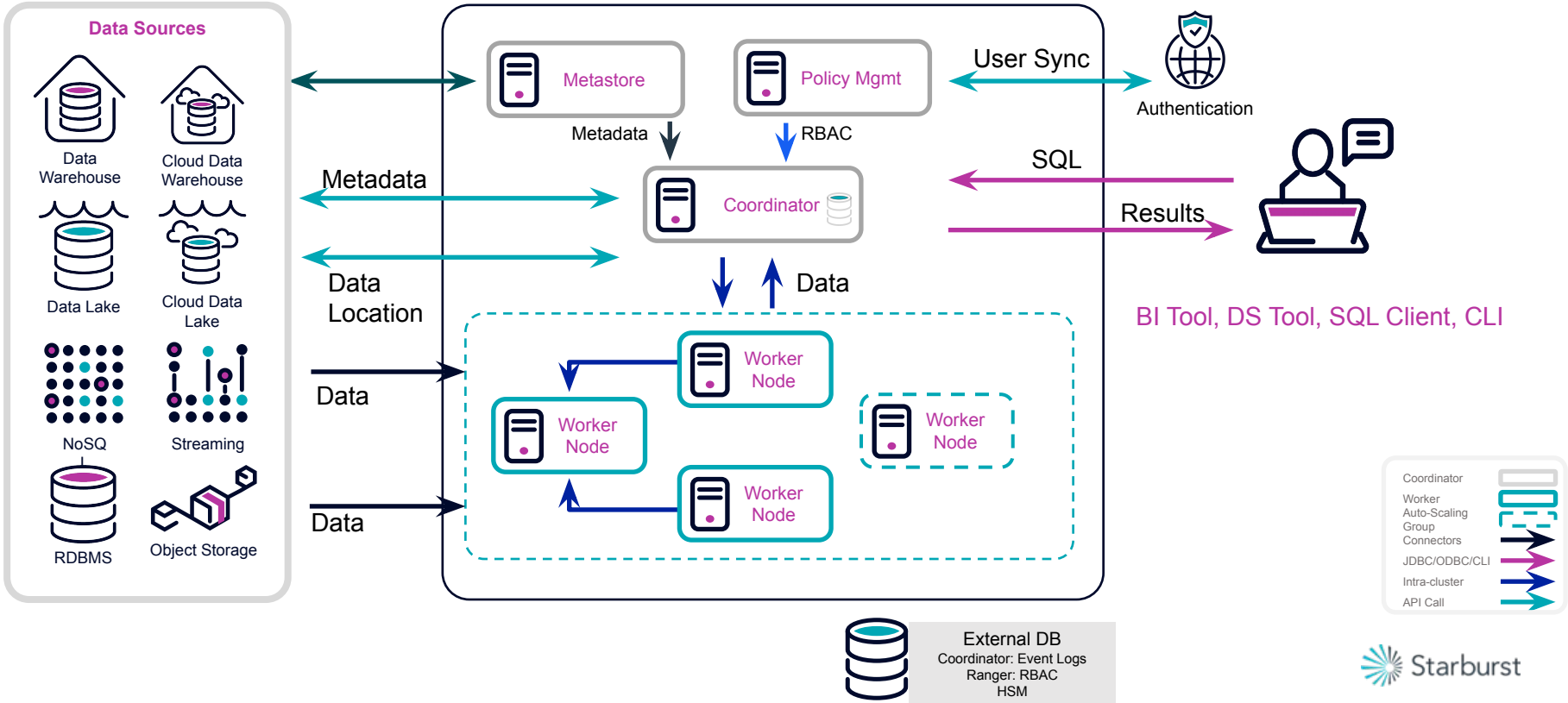
1. Explain what query plans are and how they are achieved using Starburst architecture.
2. Describe the purpose of query plan analysis.
3. Analyze the typical output of an explain plan to determine value.
4. Explain how optimization is achieved and the mechanisms used to achieve it.
5. Demonstrate how to read a visual representation of a query plan.

Query plan analysis

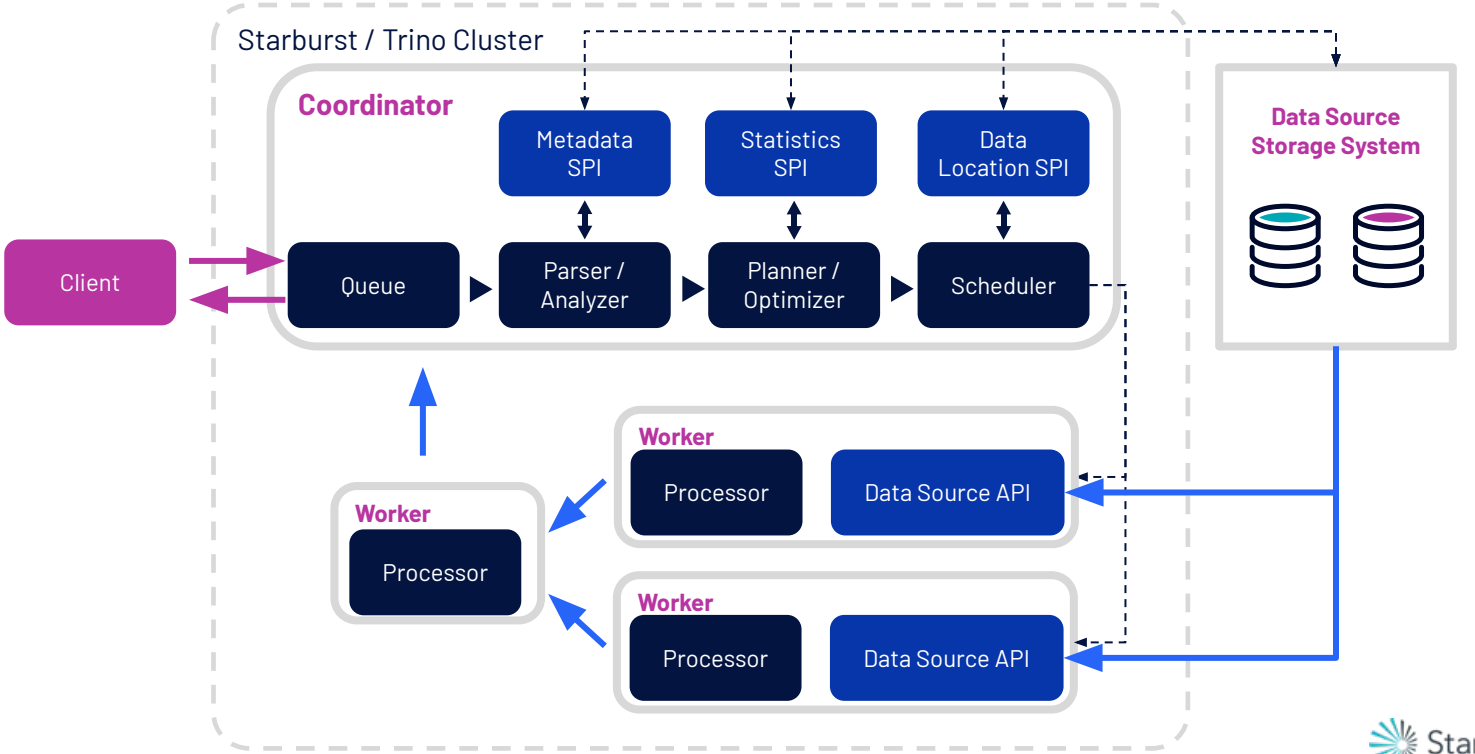
The execution model

Starburst

Starburst Logical Architecture

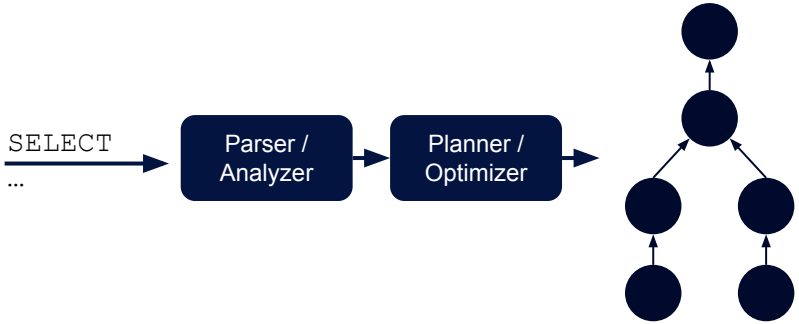


Starburst / Trino Execution Flow

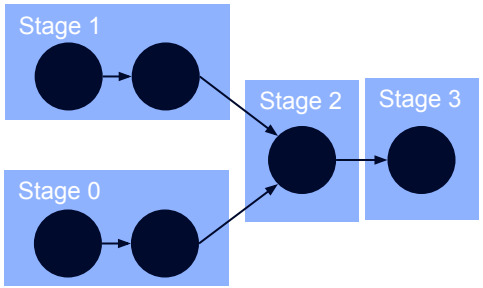


Starburst

Query Execution Model

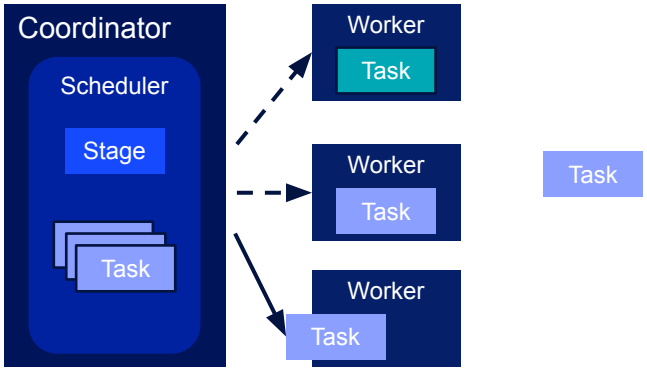


1

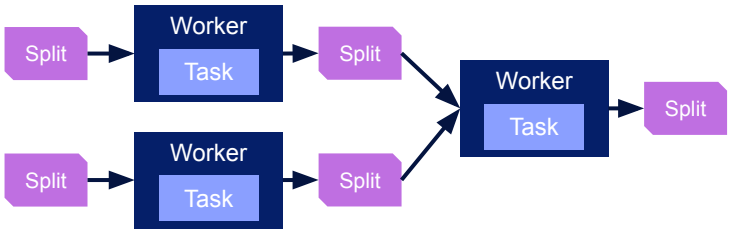


2

3



4



Starburst

Query Execution Model Nomenclature

- **Statement**: Trino executes ANSI-compatible SQL statements, which consists of clauses, expressions, and predicates.
- **Query**: a conversion of a parsed statement into a distributed query plan, which is then realized as a series of interconnected stages.
- **Stage**: hierarchy execution of different sections of a distributed plan
- **Tasks**: a stage is implemented as a series of tasks distributed over a network of workers
- **Split**: sections of a larger data set. Stages retrieve data via splits from connectors as well as other stages higher up in a distributed query plan.
- **Driver**: Tasks contain one or more parallel drivers. Drivers act upon data and combine operators to produce output that is then aggregated by a task and then delivered to another task in another stage.
- **Operator**: An operator consumes, transforms and produces data.
- **Exchange**: transfer data between Trino nodes for different stages of a query

Query plan analysis

Understanding output of `ANALYZE`

Starburst

Introducing the Explain Plan

Problem we're solving:

- What tools are available for us to troubleshoot query issues?
- How can we use the tools to identify bottlenecks?
- How can we understand what is going on in the engine?
- What can we recommend to improve query performance?
- What information does Support need to troubleshoot issues?

Goal for today:

- Ability to generate and interpret what's happening at different stages of an Explain Plan

What's an Explain Plan?

- Query Execution Plan
  - What logical tasks, in what order (table reads, joins, sorts, order by, etc)
  - What distribution type (how many workers to use per task)
- Uses Table Statistics
  - Estimates rows per task (influences join type & ordering)
- Output of CBO (cost-based optimizer)
  - Shows execution path / processing order for tasks
  - Shows optimizations (predicate push down, join reordering, etc)

```
SELECT c.custkey, sum(l.extendedprice * l.discount) discount
FROM glue.bootcamp_explain_plan.customer c
        , glue.bootcamp_explain_plan.orders o
        , glue.bootcamp_explain_plan.lineitem l
WHERE c.custkey = o.custkey
        AND l.orderkey = o.orderkey
GROUP BY c.custkey
ORDER BY discount desc
```

```
Fragment 0 [SINGLE]
    Output layout: [custkey, sum]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Output[custkey, total]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   total := sum
    └─ RemoteMerge[1]
           Layout: [custkey:bigint, sum:double]

Fragment 1 [ROUND_ROBIN]
    Output layout: [custkey, sum]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    LocalMerge[sum DESC NULLS LAST]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    └─ PartialSort[sum DESC NULLS LAST]
       │   Layout: [custkey:bigint, sum:double]
       └─ RemoteSource[2]
              Layout: [custkey:bigint, sum:double]

Fragment 2 [HASH]
    Output layout: [custkey, sum]
    Output partitioning: ROUND_ROBIN []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Aggregate(FINAL)[custkey]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   sum := sum("sum_0")
    └─ LocalExchange[HASH][$hashvalue] ("custkey")
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue:bigint]
       │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
       └─ RemoteSource[3]
              Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_1:bigint]

Fragment 3 [SOURCE]
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
((("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')))]
              Layout: [custkey:bigint, totalprice:double]
              Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB),
cpu: 82.97M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
              orderstatus := orderstatus:varchar(1):REGULAR
              custkey := custkey:bigint:REGULAR
              totalprice := totalprice:double:REGULAR
              orderdate := orderdate:date:REGULAR
```

Generating an EXPLAIN Plan

- Prefix any query with the word ‘EXPLAIN’
- Textual output of execution plan of the query in internal engine
- Multiple variants:
  - **Explain:** plan structure + cost estimates
  - **Explain Analyze:** plan structure + cost estimates + actual execution statistics

```
EXPLAIN ANALYZE
select custkey, sum(totalprice) as total
from glue.bootcamp_explain_plan.orders
where
        orderstatus = 'F' AND
        orderdate between DATE '1995-01-01' and DATE
'1995-12-31'
group by custkey
order by total desc;
```

****Note: All Explain Plan output in this deck was generated using Starburst v358.0*

Starburst

Explain Plan Output

- Series of fragments:
  - Representing logical sections of work to be performed in stages in cluster
  - Work executed on single or multiple-nodes

Query: EXPLAIN **select** custkey, **sum**(totalprice) **as** total **from** glue.bootcamp_explain_plan.orders **where** orderstatus = 'F' **AND** orderdate **between DATE** '1995-01-01' **and DATE** '1995-12-31' **group by** custkey **order by** total **desc**;

```
Fragment 0 [SINGLE]
    Output layout: [custkey, sum]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Output[custkey, total]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   total := sum
    └─ RemoteMerge[1]
           Layout: [custkey:bigint, sum:double]

Fragment 1 [ROUND_ROBIN]
    Output layout: [custkey, sum]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    LocalMerge[sum DESC NULLS LAST]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    └─ PartialSort[sum DESC NULLS LAST]
       │   Layout: [custkey:bigint, sum:double]
       └─ RemoteSource[2]
              Layout: [custkey:bigint, sum:double]

Fragment 2 [HASH]
    Output layout: [custkey, sum]
    Output partitioning: ROUND_ROBIN []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Aggregate(FINAL)[custkey]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   sum := sum("sum_0")
    └─ LocalExchange[HASH][$hashvalue] ("custkey")
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue:bigint]
       │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
       └─ RemoteSource[3]
              Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_1:bigint]

Fragment 3 [SOURCE]
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
((("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')))]
              Layout: [custkey:bigint, totalprice:double]
              Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB),
cpu: 82.97M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
              orderstatus := orderstatus:varchar(1):REGULAR
              custkey := custkey:bigint:REGULAR
              totalprice := totalprice:double:REGULAR
              orderdate := orderdate:date:REGULAR
```

Starburst

Exchanges

- Fragment ids indicate how data is transferred between fragments

```
Fragment 0 [SINGLE]
    ...
    Output[custkey, total]
    │   Layout: [custkey:bigint, sum:double]
    │   ...
    │   total := sum
    └─ RemoteMerge[1]
           Layout: [custkey:bigint, sum:double]

 Fragment 1 [ROUND_ROBIN]
    Output layout: [custkey, sum]
    ...
    LocalMerge[sum DESC NULLS LAST]
    │   Layout: [custkey:bigint, sum:double]
    │   ...
    └─ PartialSort[sum DESC NULLS LAST]
       │   Layout: [custkey:bigint, sum:double]
       └─ RemoteSource[2]
              Layout: [custkey:bigint, sum:double]

Fragment 2 [HASH]
    Output layout: [custkey, sum]
    Output partitioning: ROUND_ROBIN []
    ...
    Aggregate(FINAL)[custkey]
    │   Layout: [custkey:bigint, sum:double]
    │   ...
    │   sum := sum("sum_0")
    └─ LocalExchange[HASH][$hashvalue] ("custkey")
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean),
$hashvalue:bigint]
       │   ...
       └─ RemoteSource[3]
              Layout: [custkey:bigint, sum_0:row(bigint, boolean, double,
boolean), $hashvalue_1:bigint]

Fragment 3 [SOURCE]
    Output layout: [custkey, sum_0, $hashvalue_2]
    ...
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean),
$hashvalue_2:bigint]
    │   ...
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped =
false, filterPredicate = (("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE
'1995-01-01' AND DATE '1995-12-31'))]
              Layout: [custkey:bigint, totalprice:double]
```

Starburst

Explain vs Explain Analyze output

```
Fragment 2 [HASH]
    Output layout: [custkey, sum]
    Output partitioning: ROUND_ROBIN []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Aggregate(FINAL)[custkey]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   sum := sum("sum_0")
    └─ LocalExchange[HASH][$hashvalue] ("custkey")
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue:bigint]
       │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
       └─ RemoteSource[3]
              Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_1:bigint]

Fragment 3 [SOURCE]
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
(("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'))]
              Layout: [custkey:bigint, totalprice:double]
              Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB),
cpu: 82.97M, memory: 0B, network: 0B}/{rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
              orderstatus := orderstatus:varchar(1):REGULAR
              custkey := custkey:bigint:REGULAR
              totalprice := totalprice:double:REGULAR
              orderdate := orderdate:date:REGULAR
```

Explain Plan

- Explain is useful for:
  - A long running query but you just want the Query Plan
  - Out Of Memory issues for queries

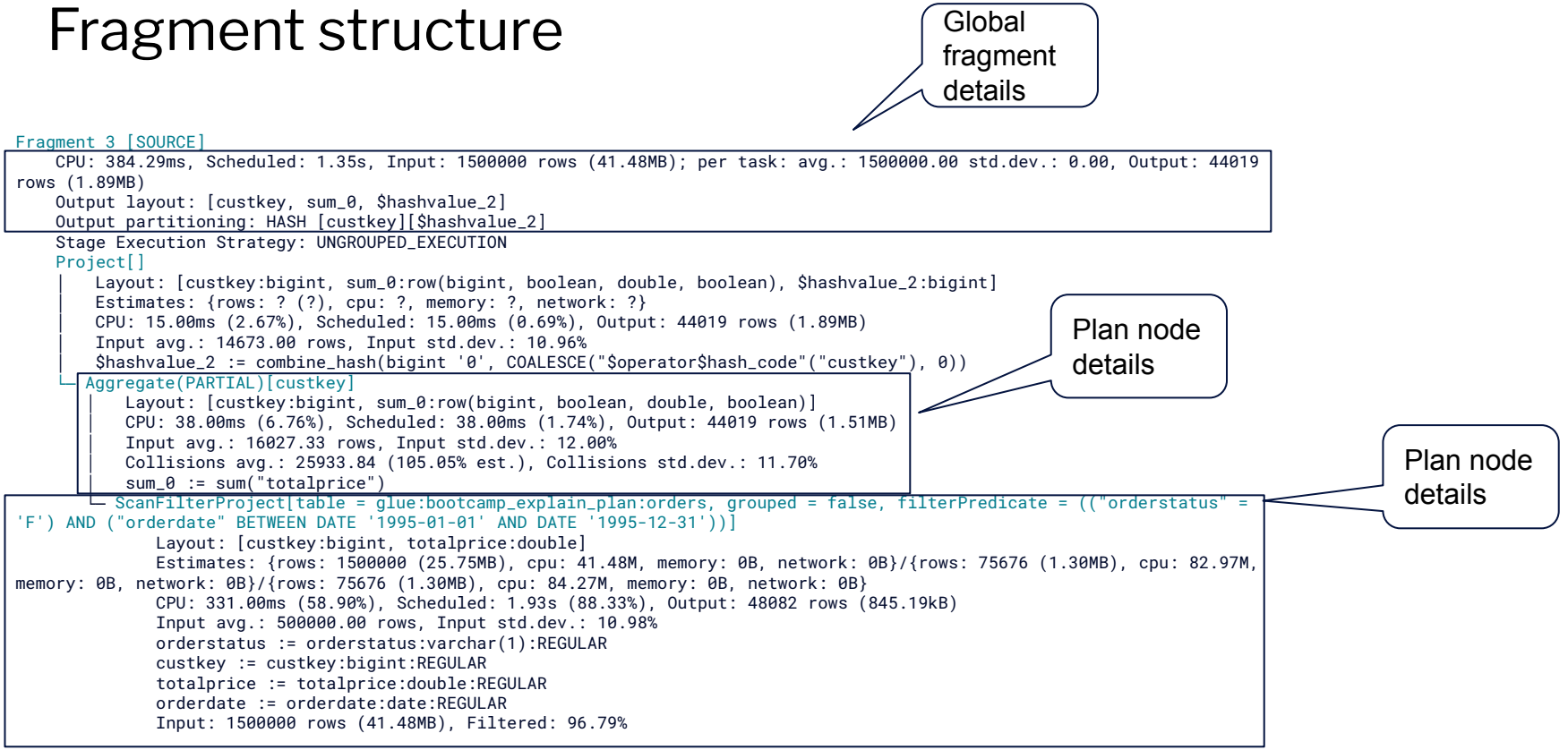
```
Fragment 2 [HASH]
    CPU: 203.43ms, Scheduled: 324.82ms, Input: 44019 rows (1.89MB); per task: avg.: 14673.00 std.dev.: 73.54,
Output: 37377 rows (657.02kB)
    Output layout: [custkey, sum]
    Output partitioning: ROUND_ROBIN []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Aggregate(FINAL)[custkey]
    │   Layout: [custkey:bigint, sum:double]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   CPU: 150.00ms (12.37%), Scheduled: 246.00ms (10.03%), Output: 37377 rows (657.02kB)
    │   Input avg.: 917.06 rows, Input std.dev.: 3.91%
    │   Collisions avg.: 18.87 (98.32% est.), Collisions std.dev.: 18.12%
    │   sum := sum("sum_0")
    └─ LocalExchange[HASH][$hashvalue] ("custkey")
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue:bigint]
       │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
       │   CPU: 29.00ms (2.39%), Scheduled: 38.00ms (1.55%), Output: 44019 rows (1.89MB)
       │   Input avg.: 917.06 rows, Input std.dev.: 233.92%
       └─ RemoteSource[3]
              Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_1:bigint]
              CPU: 3.00ms (0.25%), Scheduled: 4.00ms (0.16%), Output: 44019 rows (1.89MB)
              Input avg.: 917.06 rows, Input std.dev.: 233.92%

Fragment 3 [SOURCE]
    CPU: 811.93ms, Scheduled: 1.25s, Input: 1500000 rows (41.48MB); per task: avg.: 500000.00 std.dev.:
54878.14, Output: 44019 rows (1.89MB)
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   CPU: 29.00ms (2.39%), Scheduled: 29.00ms (1.18%), Output: 44019 rows (1.89MB)
    │   Input avg.: 14673.00 rows, Input std.dev.: 10.96%
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   CPU: 50.00ms (4.12%), Scheduled: 50.00ms (2.04%), Output: 44019 rows (1.51MB)
       │   Input avg.: 16027.33 rows, Input std.dev.: 12.00%
       │   Collisions avg.: 26153.00 (103.29% est.), Collisions std.dev.: 0.00%
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
(("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'))]
              Layout: [custkey:bigint, totalprice:double]
              Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}/{rows: ? (?), cpu: ?, memory: 0B,
network: 0B}/{rows: ? (?), cpu: ?, memory: 0B, network: 0B}
              CPU: 731.00ms (60.26%), Scheduled: 1.81s (73.78%), Output: 48082 rows (845.19kB)
              Input avg.: 500000.00 rows, Input std.dev.: 10.98%
              orderstatus := orderstatus:varchar(1):REGULAR
              custkey := custkey:bigint:REGULAR
              totalprice := totalprice:double:REGULAR
              orderdate := orderdate:date:REGULAR
              Input: 1500000 rows (41.48MB), Filtered: 96.79%
```

Explain Analyze Plan

Starburst

Fragment structure



Starburst

Source Fragment: Column pruning

Explain Analyze Query:

```
EXPLAIN ANALYZE
select orderstatus, sum(totalprice)
from glue.bootcamp_explain_plan.orders
group by orderstatus
```

Table DDL:

```
TABLE orders (
  orderkey bigint,
  custkey bigint,
  orderstatus varchar(1),
  totalprice double,
  orderdate date,
  orderpriority varchar(15),
  clerk varchar(15),
  shippriority integer,
  comment varchar(79))
```

```
Fragment 2 [SOURCE]
    CPU: 307.02ms, Scheduled: 921.85ms, Input: 1500000 rows (21.46MB); per task: avg.: 1500000.00 std.dev.: 0.00, Output: 9 rows
(378B)
    Output layout: [orderstatus, sum_0, $hashvalue_2]
    Output partitioning: HASH [orderstatus][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Aggregate(PARTIAL)[orderstatus][$hashvalue_2]
    |   Layout: [orderstatus:varchar(1), $hashvalue_2:bigint, sum_0:row(bigint, boolean, double, boolean)]
    |   CPU: 92.00ms (29.97%), Scheduled: 92.00ms (7.13%), Output: 9 rows (378B)
    |   Input avg.: 500000.00 rows, Input std.dev.: 10.98%
    |   Collisions avg.: 0.00 (0.00% est.), Collisions std.dev.: ?%
    |   sum_0 := sum("totalprice")
    └─ ScanProject[table = glue:bootcamp_explain_plan:orders, grouped = false]
           Layout: [orderstatus:varchar(1), totalprice:double, $hashvalue_2:bigint]
           Estimates: {rows: 1500000 (34.33MB), cpu: 21.46M, memory: 0B, network: 0B}/{rows: 1500000 (34.33MB), cpu: 55.79M, memory:
0B, network: 0B}
           CPU: 214.00ms (69.71%), Scheduled: 1.20s (92.71%), Output: 1500000 rows (34.33MB)
           Input avg.: 500000.00 rows, Input std.dev.: 10.98%
           $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("orderstatus"), 0))
           orderstatus := orderstatus:varchar(1):REGULAR
           totalprice := totalprice:double:REGULAR
           Input: 1500000 rows (21.46MB), Filtered: 0.00%
```

Starburst

Estimates

- Cost-based optimizer relies on statistics being available from columns in queries
  - Only need stats for:
    - join keys
    - Columns filtered on
- Estimates section show stats that are available
  - If no stats are available, numbers will be question marks
- Collected stats include:
  - Total size in bytes of columns
  - Number of distinct values
  - Fraction of nulls
  - Number of rows
  - Min/Max value
- Stats collected with Hive connector by running ANALYZE on tables
- Stats for databases collected by database

```
Fragment 3 [SOURCE]
    CPU: 384.29ms, Scheduled: 1.35s, Input: 1500000 rows (41.48MB); per task: avg.: 1500000.00
std.dev.: 0.00, Output: 44019 rows (1.89MB)
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    |   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean),
$hashvalue_2:bigint]
    |   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    |   CPU: 15.00ms (2.67%), Scheduled: 15.00ms (0.69%), Output: 44019 rows (1.89MB)
    |   Input avg.: 14673.00 rows, Input std.dev.: 10.96%
    |   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       |   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       |   CPU: 38.00ms (6.76%), Scheduled: 38.00ms (1.74%), Output: 44019 rows (1.51MB)
       |   Input avg.: 16027.33 rows, Input std.dev.: 12.00%
       |   Collisions avg.: 25933.84 (105.05% est.), Collisions std.dev.: 11.70%
       |   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false,
filterPredicate = (("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE
'1995-12-31'))]
             Layout: [custkey:bigint, totalprice:double]
             Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/
                        {rows: 75676 (1.30MB), cpu: 82.97M, memory: 0B, network: 0B}/
                        {rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
             CPU: 331.00ms (58.90%), Scheduled: 1.93s (88.33%), Output: 48082 rows (845.19kB)
             Input avg.: 500000.00 rows, Input std.dev.: 10.98%
             orderstatus := orderstatus:varchar(1):REGULAR
             custkey := custkey:bigint:REGULAR
             totalprice := totalprice:double:REGULAR
             orderdate := orderdate:date:REGULAR
             Input: 1500000 rows (41.48MB), Filtered: 96.79%
```

Starburst

‘Layout’ rows

- Columns passed as output from one section to the next
- Provides tracking of columns from beginning to final stages of query execution
- Internal names can be assigned to columns during execution

```
Fragment 3 [SOURCE]
    CPU: 384.29ms, Scheduled: 1.35s, Input: 1500000 rows (41.48MB); per task: avg.: 1500000.00 std.dev.: 0.00,
Output: 44019 rows (1.89MB)
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   CPU: 15.00ms (2.67%), Scheduled: 15.00ms (0.69%), Output: 44019 rows (1.89MB)
    │   Input avg.: 14673.00 rows, Input std.dev.: 10.96%
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   CPU: 38.00ms (6.76%), Scheduled: 38.00ms (1.74%), Output: 44019 rows (1.51MB)
       │   Input avg.: 16027.33 rows, Input std.dev.: 12.00%
       │   Collisions avg.: 25933.84 (105.05% est.), Collisions std.dev.: 11.70%
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
(("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'))]
              Layout: [custkey:bigint, totalprice:double]
              Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/
                    {rows: 75676 (1.30MB), cpu: 82.97M, memory: 0B, network: 0B}/
                    {rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
              CPU: 331.00ms (58.90%), Scheduled: 1.93s (88.33%), Output: 48082 rows (845.19kB)
              Input avg.: 500000.00 rows, Input std.dev.: 10.98%
              orderstatus := orderstatus:varchar(1):REGULAR
              custkey := custkey:bigint:REGULAR
              totalprice := totalprice:double:REGULAR
              orderdate := orderdate:date:REGULAR
              Input: 1500000 rows (41.48MB), Filtered: 96.79%
```

Starburst

Performance stats

- Scheduled time = total amount of time work was scheduled. Accounts for:
  - CPU time
  - Reading data from source (ex: tables, files)
  - Transferring data across network
- CPU time= portion of Scheduled time where CPU was active
- Difference in metrics show time waiting for data to come into cluster
- (%) = percentage of total query time

```
Fragment 3 [SOURCE]
    CPU: 384.29ms, Scheduled: 1.35s, Input: 1500000 rows (41.48MB); per task: avg.: 1500000.00 std.dev.: 0.00,
Output: 44019 rows (1.89MB)
    Output layout: [custkey, sum_0, $hashvalue_2]
    Output partitioning: HASH [custkey][$hashvalue_2]
    Stage Execution Strategy: UNGROUPED_EXECUTION
    Project[]
    │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean), $hashvalue_2:bigint]
    │   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}
    │   CPU: 15.00ms (2.67%), Scheduled: 15.00ms (0.69%), Output: 44019 rows (1.89MB)
    │   Input avg.: 14673.00 rows, Input std.dev.: 10.96%
    │   $hashvalue_2 := combine_hash(bigint '0', COALESCE("$operator$hash_code"("custkey"), 0))
    └─ Aggregate(PARTIAL)[custkey]
       │   Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
       │   CPU: 38.00ms (6.76%), Scheduled: 38.00ms (1.74%), Output: 44019 rows (1.51MB)
       │   Input avg.: 16027.33 rows, Input std.dev.: 12.00%
       │   Collisions avg.: 25933.84 (105.05% est.), Collisions std.dev.: 11.70%
       │   sum_0 := sum("totalprice")
       └─ ScanFilterProject[table = glue:bootcamp_explain_plan:orders, grouped = false, filterPredicate =
(("orderstatus" = 'F') AND ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'))]
             Layout: [custkey:bigint, totalprice:double]
             Estimates: {rows: 1500000 (25.75MB), cpu: 41.48M, memory: 0B, network: 0B}/
                   {rows: 75676 (1.30MB), cpu: 82.97M, memory: 0B, network: 0B}/
                   {rows: 75676 (1.30MB), cpu: 84.27M, memory: 0B, network: 0B}
             CPU: 331.00ms (58.90%), Scheduled: 1.93s (88.33%), Output: 48082 rows (845.19kB)
             Input avg.: 500000.00 rows, Input std.dev.: 10.98%
             orderstatus := orderstatus:varchar(1):REGULAR
             custkey := custkey:bigint:REGULAR
             totalprice := totalprice:double:REGULAR
             orderdate := orderdate:date:REGULAR
             Input: 1500000 rows (41.48MB), Filtered: 96.79%
```

Starburst

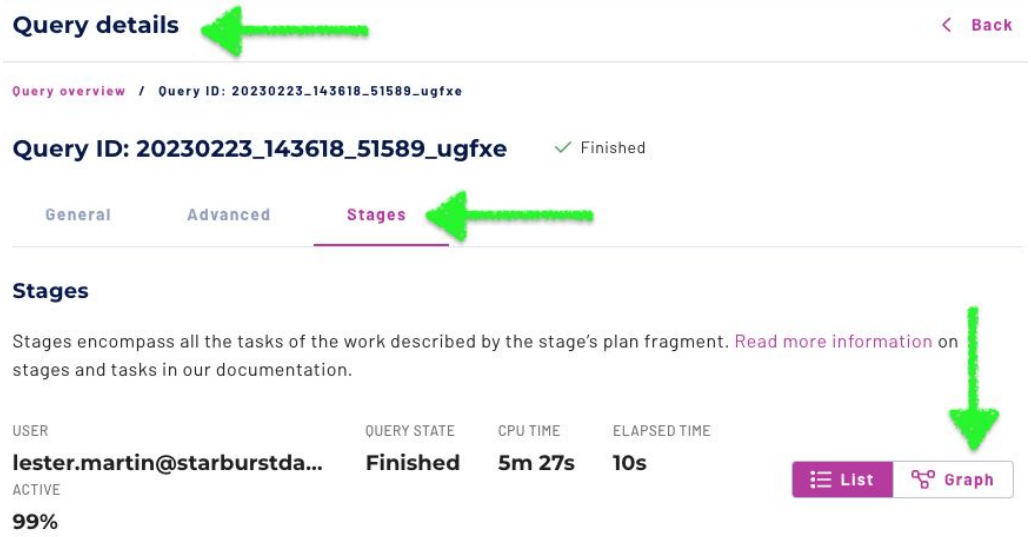
Query plan analysis

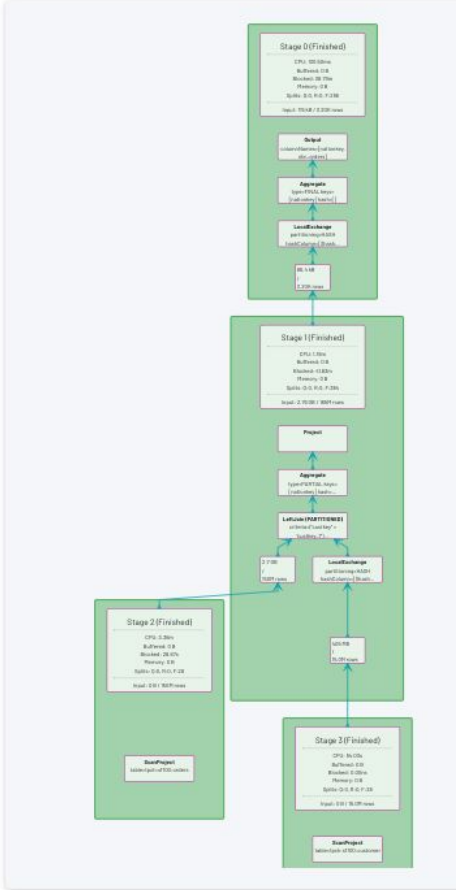
Graphical view

Starburst

Query plan visualized

The web UI features a visual representation by drilling into the **Query details** > **Stages** > **Graph** navigation elements.

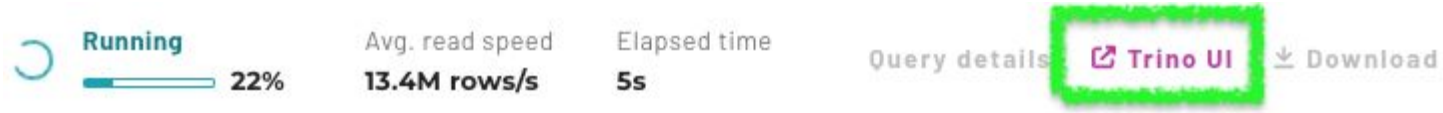




Starburst

Live query plan visualization

For queries currently executing in the **Query editor** you can find a live version of the visual query plan via **Trino UI**.

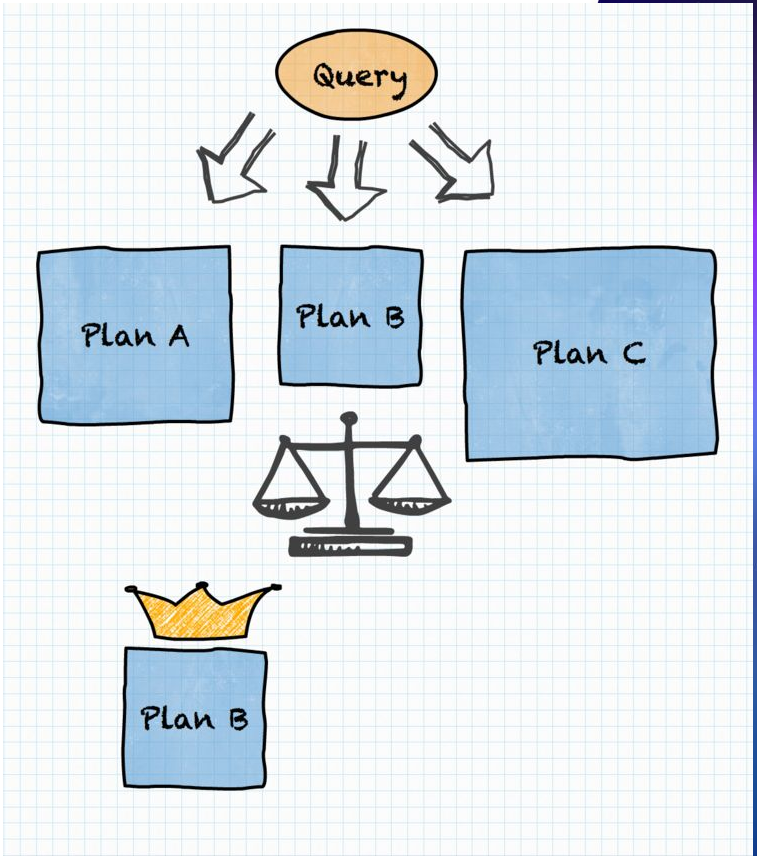


In the surfaced UI, select **Live Plan**.

Real-time updates on which stages are still running and as well as stage-level statistics on both running and completed stages.

Instructor demonstration

Query plan analysis (60 mins)



Starburst

Hands-on exercises

Lab 1: The `EXPLAIN` command (20 mins)

Lab 2: The `EXPLAIN ANALYZE` command (20 mins)

Lab 3: Explore the impact of statistics on query plans (45 mins)

Starburst

Lesson summary

Cost-based optimizer: Query plan analysis

1. Query plans are generated by the system as a processing roadmap. They show each of the steps that Starburst will uses to achieve its results. They can be created with, or without, runtime characteristics.

2. Explain plans list stages as fragments which include the pipeline of operations that are parallelized across the workers.

3. Explain plans can be represented visually; similar to flow charts. This is often helpful to understand the stages involved and assess the value of the plan.