Today’s Speakers

**Manfred Moser**  
Developer, author, and trainer at Starburst

Manfred is an open source developer and advocate. He is co-author of the book *Presto: The Definitive Guide*, and a seasoned trainer and conference presenter. He has trained over 20,000 developers for companies such as Walmart Labs, Sonatype, and Telus.

**Martin Traverso**  
Co-creator of Presto and CTO at Starburst

Martin is a co-creator of Presto, co-founder of the Presto Software Foundation, and CTO at Starburst. Prior to Starburst, Martin worked as a Software Engineer at Facebook, and a Software Architect at Proofpoint and Ning.
Agenda

- Presto overview
- Introduction to the Presto query execution
  - Query lifecycle
  - Explain the EXPLAIN
  - Optimizations
- Five minute break
  - Understanding the cost-based optimizer
- Q&A
Questions

- Ask any time
- Use the meeting Questions feature
- Manfred screens, collects and interjects
- Dedicated Q&A in break and at end
Some advice for attendees

- This is a fast-paced overview – don't try to follow along during class
- Instead focus and pay attention
- Use the demo video after class to setup Presto and CLI locally
- Learn at your own pace
- Use video recording and slides from class as reference to learn more
- Play with the TPC-H data sets and different catalogs and connectors
- Apply skills for your own use case
Presto overview

... probably just a recap for you
What is Presto?

High performance ANSI SQL engine
- SQL support for any connected data source - SQL-on-anything
- Cost-based query optimizer
- Proven horizontal scalability

Open source project
- Very active, large community
- User driven development
- Huge variety of users
- Prestosql.io

Separation of compute and storage
- Scale query processing and data sources independently
- Query storage directly
- No ETL or data integration necessary

Presto everywhere
- No cloud vendor lock-in
- No storage engine vendor lock-in
- No Hadoop distro vendor lock-in
- No database lock in
Why use Presto?

Fastest time-to-insight
- High performance query processing
- Low barrier of entry for users
- Massive scalability
- High concurrency
- Direct access to storage

Lower cost
- Reduced need to copy and move data
- Avoid complex data processing
- Scale storage and compute independently
- Only run computes when processing queries
- One data consumption layer

Avoid data lock in
- No more data silos, departmental copies
- Query data with the existing skills and tools - SQL + BI tools
- Query any data source
- Move data
- Create optionality
Let’s look inside Presto with Martin
Query lifecycle

Client → Coordinator → Parser → Analyzer → Planner / Optimizer → Scheduler → Worker
SELECT orderstatus, sum(totalprice)
FROM orders
WHERE orderdate > DATE '2020-08-01'
GROUP BY orderstatus
Analysis

Query

FROM orders

WHERE orderdate > DATE '2020-08-01'

GROUP BY orderstatus

SELECT orderstatus, sum(totalprice) FROM hive.tpch.orders

Aggregate

VARCHAR(1) 

AGG. FUNCTION DOUBLE->DOUBLE

DOUBLE

DATE

DATE

TABLE hive.tpch.orders
orderkey BIGINT, 
orderdate DATE, 
orderstatus VARCHAR(1), 
totalprice DOUBLE, ...
Planning

Syntax tree

```
SELECT orderstatus, sum(totalprice)
FROM orders
WHERE orderdate > DATE '2020-08-01'
GROUP BY orderstatus
```

Intermediate representation (Plan IR)

```
Scan hive.tpch.orders
Filter orderdate > DATE '2020-08-01'
Aggregation [orderstatus] sum(totalprice)
Output
```
Optimization

Intermediate representation (Plan IR)

- **Filter**
  - `orddate > DATE '2020-08-01')`

- **Scan**
  - `hive.tpch.orders`

- **Aggregation**
  - `[orderstatus] sum(totalprice)`

Optimized plan

1. **Output**
2. **N:1 Exchange**
3. **Final Aggregation**
   - `[orderstatus] sum(totalprice)`
4. **N:M Exchange**
5. **Partial Aggregation**
   - `[orderstatus] sum(totalprice)`
6. **Filtered Scan**
   - `Hive.tpch.orders` `orddate > DATE '2020-08-01'`
Scheduling and execution
Explain the EXPLAIN
EXPLAIN

SELECT custkey, sum(totalprice) AS total FROM orders
WHERE orderstatus = 'F' AND orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
GROUP BY custkey
ORDER BY total DESC
EXPLAIN vs EXPLAIN ANALYZE

- **EXPLAIN**: plan structure + cost estimates
- **EXPLAIN ANALYZE**: plan structure + cost estimates + actual execution statistics

Fragment 4 [SOURCE]

CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)

Output layout: [custkey, sum_0]

Output partitioning: HASH [custkey]

Stage Execution Strategy: UNGROUPED_EXECUTION

Aggregate(PARTIAL)[custkey]

| Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)] |
| CPU: 220.00ms (0.64%), Scheduled: 691.00ms (1.65%), Output: 47763 rows (1.64MB) |
| Input avg.: 1335.61 rows, Input std.dev.: 2.44% |
| Collisions avg.: 20.89 (105.76% est.), Collisions std.dev.: 142.37% |

`sum_0 := sum("totalprice")`

└─

ScanFilterProject[table = tpch:orders:sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')] |

| Layout: [custkey:bigint, totalprice:double] |
| CPU: 33.87s (98.57%), Scheduled: 40.62s (97.07%), Output: 48082 rows (845.19kB) |
| Input avg.: 20261.47 rows, Input std.dev.: 0.54% |
| custkey := tpch:custkey |
| totalprice := tpch:totalprice |
| orderdate := tpch:orderdate |
| tpch:orderstatus :=[F] |

Input: 729413 rows (0B), Filtered: 93.41%
Fragment structure

Fragment 4 [SOURCE]
CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)
Output layout: [custkey, sum_0]
Output partitioning: HASH [custkey]
Stage Execution Strategy: UNGROUPED_EXECUTION

Aggregate[PARTIAL][custkey]
Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
CPU: 220.00ms (0.64%), Scheduled: 691.00ms (1.65%), Output: 47763 rows (1.64MB)
Input avg.: 1335.61 rows, Input std.dev.: 2.44%
Collisions avg.: 20.89 (105.76% est.), Collisions std.dev.: 142.37%
sum_0 := sum("totalprice")

ScanFilterProject[table = tpch:orders:sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')]
Layout: [custkey:bigint, totalprice:double]
Estimates: {rows: 729413 (12.52MB), cpu: 16.00M, memory: 0B, network: 0B}/
{rows: 95443 (1.64MB), cpu: 32.00M, memory: 0B, network: 0B}/
{rows: 95443 (1.64MB), cpu: 33.64M, memory: 0B, network: 0B}
CPU: 33.87s (98.57%), Scheduled: 40.62s (97.07%), Output: 48082 rows (845.19kB)
Input avg.: 20261.47 rows, Input std.dev.: 0.54%
custkey := tpch:custkey
totalprice := tpch:totalprice
orderdate := tpch:orderdate
tpch:orderstatus :: [F]]
Input: 729413 rows (0B), Filtered: 93.41%
Fragment 4  

CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)

Output layout: [custkey, sum_0]

Output partitioning: HASH [custkey]

Stage Execution Strategy: UNGROUPED_EXECUTION

Aggregate(PARTIAL)[custkey]

Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]

CPU: 220.00ms (0.64%), Scheduled: 691.00ms (1.65%), Output: 47763 rows (1.64MB)

Input avg.: 1335.61 rows, Input std.dev.: 2.44%

Collisions avg.: 28.89 (105.76% est.), Collisions std.dev.: 142.37%

sum_0 := sum("totalprice")

ScanFilterProject[table = tpch:orders:sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')]

Layout: [custkey:bigint, totalprice:double]

Estimates: {rows: 729413 (12.52MB), cpu: 16.00M, memory: 0B, network: 0B}/
{rows: 95443 (1.64MB), cpu: 32.00M, memory: 0B, network: 0B}/
{rows: 95443 (1.64MB), cpu: 33.64M, memory: 0B, network: 0B}

CPU: 33.87s (98.57%), Scheduled: 40.62s (97.07%), Output: 48082 rows (845.19kB)

Input avg.: 28261.47 rows, Input std.dev.: 0.54%

custkey := tpch:custkey
totalprice := tpch:totalprice
orderdate := tpch:orderdate
tpch:orderstatus
:: [F]]

Input: 729413 rows (0B), Filtered: 93.41%
Row layout

Fragment 4 [SOURCE]
  CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)
  Output layout: [custkey, sum_0]
  Output partitioning: HASH [custkey]
  Stage Execution Strategy: UNGROUPED_EXECUTION
  Aggregate(PARTIAL)[custkey]
    Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
    CPU: 220.00ms (0.64%), Scheduled: 691.00ms (1.65%), Output: 47763 rows (1.64MB)
    Input avg.: 1335.61 rows, Input std.dev.: 2.44%
    Collisions avg.: 20.89 (105.76% est.), Collisions std.dev.: 142.37%
    sum_0 := sum("totalprice")
    └─ ScanFilterProject[table = tpch:orders: sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')]
      Layout: [custkey:bigint, totalprice:double]
      Estimates: {rows: 729413 (12.52MB), cpu: 16.00M, memory: 0B, network: 0B}/
      {rows: 95443 (1.64MB), cpu: 32.00M, memory: 0B, network: 0B}/
      {rows: 95443 (1.64MB), cpu: 33.64M, memory: 0B, network: 0B}
      CPU: 33.87s (98.57%), Scheduled: 40.62s (97.07%), Output: 48082 rows (845.19kB)
      Input avg.: 20261.47 rows, Input std.dev.: 0.54%
      custkey := tpch:custkey
totalprice := tpch:totalprice
orderdate := tpch:orderdate
tpch:orderstatus
:: [[F]]
Input: 729413 rows (0B), Filtered: 93.41%
Estimates

Fragment 4 [SOURCE]

CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)
Output layout: [custkey, sum_0]
Output partitioning: HASH [custkey]
Stage Execution Strategy: UNGROUPED_EXECUTION
Aggregate(PARTIAL)[custkey]
  └─ ScanFilterProject[table = tpch:orders:sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')]
Performance stats

Fragment 4 [SOURCE]

CPU: 34.09s, Scheduled: 41.31s, Input: 729413 rows (0B); per task: avg.: 243137.67 std.dev.: 206.19, Output: 47763 rows (1.64MB)

Output layout: [custkey, sum_0]
Output partitioning: HASH [custkey]
Stage Execution Strategy: UNGROUPED_EXECUTION
Aggregate(PARTIAL)[custkey]

  Layout: [custkey:bigint, sum_0:row(bigint, boolean, double, boolean)]
  CPU: 220.00ms (0.64%), Scheduled: 691.00ms (1.65%), Output: 47763 rows (1.64MB)
  Input avg.: 1335.61 rows, Input std.dev.: 2.44%
  Collisions avg.: 20.89 (105.76% est.), Collisions std.dev.: 142.37%
  sum_0 := sum("totalprice")

└─ ScanFilterProject[table = tpch:orders:sf1.0, grouped = false, filterPredicate = ("orderdate" BETWEEN DATE '1995-01-01' AND DATE '1995-12-31')]
  Layout: [custkey:bigint, totalprice:double]
  Estimates: {rows: 729413 (12.52MB), cpu: 16.00M, memory: 0B, network: 0B}/
              {rows: 95443 (1.64MB), cpu: 32.00M, memory: 0B, network: 0B}/
              {rows: 95443 (1.64MB), cpu: 33.64M, memory: 0B, network: 0B}
  CPU: 33.87s (98.57%), Scheduled: 40.62s (97.07%), Output: 48082 rows (845.19kB)
  Input avg.: 20261.47 rows, Input std.dev.: 0.54%
custkey := tpch:custkey
totalprice := tpch:totalprice
orderdate := tpch:orderdate
tpch:orderstatus
  :: [F]
Input: 729413 rows (0B), Filtered: 93.41%
Exchanges

**Fragment 0 [SINGLE]**
- Output[orderdate, _col1]
  - RemoteMerge[1]

**Fragment 1 [ROUND_ROBIN]**
- LocalMerge[orderdate ASC_NULLS_LAST]
  - PartialSort[orderdate ASC_NULLS_LAST]
    - RemoteSource[2]

**Fragment 2 [HASH]**
- Aggregate(FINAL)[orderdate]
  - LocalExchange[HASH] ("orderdate")
    - RemoteSource[3]

**Fragment 3 [HASH]**
- Aggregate(PARTIAL)[orderdate]
  - Project[]
    - InnerJoin["orderkey_0" = "orderkey"]
      - RemoteSource[5]

**Fragment 4 [SOURCE]**
- ScanFilter[table = hive:tpch:lineitem, ...]

**Fragment 5 [SOURCE]**
- TableScan[hive:tpch:orders, ...]
Optimizations
Optimizations

- Constant folding
- Limit pushdown
- Predicate pushdown
- Aggregation pushdown
- Join reordering and type selection
SELECT orderkey
FROM orders
WHERE orderdate >= current_date - INTERVAL '30' DAY

Fragment 1 [SOURCE]
Output layout: [orderkey]
Output partitioning: SINGLE [
Stage Execution Strategy: UNGROUPED_EXECUTION
ScanFilterProject[table = tpch:orders:sf0.01,..., filterPredicate = ("orderdate" >= DATE '2020-07-10'))
  Layout: [orderkey:bigint]
  Estimates: ...
  orderkey := tpch:orderkey
  orderdate := tpch:orderdate
  tpch:orderstatus
  :: [[F], [O], [P]]
Column pruning

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

SELECT orderstatus, sum(totalprice) 
FROM orders 
GROUP BY orderstatus
```

Fragment 2 [SOURCE]
- Output layout: [orderstatus, sum_0]
- Output partitioning: HASH [orderstatus]
- Stage Execution Strategy: UNGROUPED_EXECUTION
- Aggregate(PARTIAL)[orderstatus]
  - Layout: [orderstatus:varchar(1), sum_0:row(bigint, boolean, double, boolean)]
  - CPU: 24.06s (51.10%), Scheduled: 39.11s (37.50%), Output: 180 rows (5.80kB)
  - Input avg.: 1339285.71 rows, Input std.dev.: 93.43%
  - sum_0 := sum("totalprice")
  - TableScan[hive:tpch:orders, grouped = false]
    - Layout: [totalprice:double, orderstatus:varchar(1)]
    - Estimates: {rows: 150000000 (2.10GB), cpu: 2.10G, memory: 0B, network: 0B}
    - CPU: 23.00s (48.86%), Scheduled: 1.09m (62.46%), Output: 150000000 rows (2.10GB)
    - Input avg.: 1339285.71 rows, Input std.dev.: 93.43%
    - orderstatus := orderstatus:varchar(1):REGULAR
    - totalprice := totalprice:double:REGULAR
```
Nested column pruning

```sql
SELECT details.orderstatus, sum(details.totalprice)
FROM orders_nested
GROUP BY details.orderstatus;
```

Fragment 2 [SOURCE]
Output layout: [details#orderstatus, sum_1]
Output partitioning: HASH [details#orderstatus]
Stage Execution Strategy: UNGROUPED_EXECUTION
Aggregate(PARTIAL)[details#orderstatus]
  Layout: [details#orderstatus:varchar(1), sum_1:row(bigint, boolean, double, boolean)]
  CPU: 21.44s (27.23%), Scheduled: 24.52s (21.81%), Output: 48 rows (1.55kB)
  Input avg.: 2830188.68 rows, Input std.dev.: 169.04%
  sum_1 := sum("details#totalprice")
  TableScan[hive:tpch:orders_nested, grouped = false]
    Layout: [details#totalprice:double, details#orderstatus:varchar(1)]
    Estimates: {rows: 150000000 (8.94GB), cpu: 8.94G, memory: 0B, network: 0B}
    CPU: 57.28s (72.76%), Scheduled: 1.46m (78.14%), Output: 150000000 rows (2.10GB)
    Input avg.: 2830188.68 rows, Input std.dev.: 169.04%
    details#totalprice := details#totalprice:double:REGULAR
    details#orderstatus := details#orderstatus:varchar(1):REGULAR
Predicate pushdown

```sql
SELECT sum(extendedprice)
FROM orders JOIN lineitem ON orders.orderkey = lineitem.orderkey
WHERE orders.orderkey < 10
```
Predicate pushdown into connectors

```
SELECT *
FROM orders

Fragment 1 [SOURCE]
CPU: 114.47ms, Scheduled: 225.06ms, Input: 15000 rows (1.86MB); per task: avg.: 15000.00 std.dev.: 0.00, Output: 15000 rows (1.86MB)
...
TableScan[postgresql:tpch.orders tpch.orders, grouped = false]
...
CPU: 115.00ms (100.00%), Scheduled: 225.00ms (100.00%),
Input avg.: 15000.00 rows, Input std.dev.: 0.00%
...

SELECT *
FROM orders
WHERE orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'

Fragment 1 [SOURCE]
CPU: 28.32ms, Scheduled: 56.24ms, Input: 2204 rows (279.58kB); per task: avg.: 2204.00 std.dev.: 0.00, Output: 2204 rows (279.58kB)
...
TableScan[postgresql:tpch.orders tpch.orders, grouped = false]
...
CPU: 28.00ms (100.00%), Scheduled: 56.00ms (100.00%),
Input avg.: 2204.00 rows, Input std.dev.: 0.00%
```
### Predicate pushdown into connectors - Limitations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>((col0 BETWEEN ? AND ?) OR (col0 BETWEEN ? and ?) OR ...) AND ((col1 BETWEEN ? AND ?) OR (col1 BETWEEN ? and ?) OR ...) AND ...</code></td>
<td>✔️</td>
</tr>
<tr>
<td><code>column IN (1, 2, 3)</code></td>
<td>✔️</td>
</tr>
<tr>
<td><code>column LIKE '%hello%world%'</code></td>
<td>❌</td>
</tr>
<tr>
<td><code>log10(column) &gt; 1</code></td>
<td>❌</td>
</tr>
<tr>
<td><code>(col1 = 1 OR col2 = 10) AND (col1 = 2 OR col2 = 20)</code></td>
<td>❌</td>
</tr>
</tbody>
</table>
Predicate pushdown into the Hive connector

- Partition pruning
- Bucket pruning
- Row group skipping for ORC and Parquet
SELECT orderdate, sum(totalprice) total
FROM orders_partitioned
WHERE orderstatus = 'F'
GROUP BY orderdate
ORDER BY total DESC
LIMIT 10

Table contains 150,000,000 rows
Hive partition pruning

```
CREATE TABLE orders_partitioned (...)
WITH (
  format = 'ORC',
  partitioned_by = array['orderstatus', 'orderpriority'])

SELECT orderdate, sum(totalprice) total
FROM orders_partitioned
WHERE cast(substr(orderpriority, 1, 1) as tinyint) = 1
GROUP BY orderdate
ORDER BY total DESC
LIMIT 10

Fragment 3 [SOURCE]
CPU: 7.15s, Scheduled: 13.90s, Input: 29995209 rows (400.85MB); per task: avg.: 7498802.25 std.dev.: 1730167.46, Output: 20682 rows (646.31kB)
Output layout: [orderdate, sum_0]
Output partitioning: HASH [orderdate]
Stage Execution Strategy: UNGROUPED_EXECUTION
Aggregate(PARTIAL)[orderdate]
  Layout: [orderdate:date, sum_0:row(bigint, boolean, double, boolean)]
  CPU: 3.50s (48.52%), Scheduled: 4.98s (34.53%), Output: 20682 rows (646.31kB)
  Input avg.: 1071257.46 rows, Input std.dev.: 94.74%
  Collisions avg.: 18307.11 (142692.06% est.), Collisions std.dev.: 189.30%
  sum_0 := sum("totalprice")
  ScanFilterProject[table = ..., filterPredicate = (CAST("substr"("orderpriority", BIGINT '1', BIGINT '1') AS tinyint) = TINYINT '1')]
  Layout: [totalprice:double, orderdate:date]
  Estimates: ...
  CPU: 3.66s (50.73%), Scheduled: 9.19s (63.72%), Output: 29995209 rows (400.48MB)
  Input avg.: 1071257.46 rows, Input std.dev.: 94.74%
  totalprice := totalprice:double:REGULAR
  orderdate := orderdate:date:REGULAR
  orderpriority := orderpriority:varchar(15):PARTITION_KEY
    :: [[1-URGENT]]
  orderstatus:varchar(1):PARTITION_KEY
    :: [[F], [O], [P]]
Input: 29995209 rows (400.85MB), Filtered: 0.00%
```

Table contains 150,000,000 rows
CREATE TABLE orders_bucketed (...)
WITH (  
  format = 'ORC',  
  bucketed_by = array['orderkey']
)

SELECT *  
FROM orders_unbucketed  
WHERE orderkey IN (1, 2, 3, 4)

Query 20200810_024110_00017_utfmj, FINISHED, 4 nodes  
Splits: 116 total, 116 done (100.00%)  
0.48 [10K rows, 37.4MB] [21K rows/s, 78.6MB/s]

SELECT *  
FROM orders_bucketed  
WHERE orderkey IN (1, 2, 3, 4)

Query 20200810_024059_00016_utfmj, FINISHED, 4 nodes  
Splits: 8 total, 8 done (100.00%)  
0.43 [40K rows, 59.5MB] [93.5K rows/s, 139MB/s]
Row group skipping

```
SELECT sum(extendedprice)
FROM lineitem
WHERE orderkey = 999
```

Fragment 2
CPU: 17.80s, Scheduled: 1.29m, Input: 9987786 rows (85.78MB); per task: avg.: 2496946.50 std.dev.: 218060.73, Output: 450 rows (11.87kB)

```
...  
Aggregate(PARTIAL)
  
  ScanFilterProject[table = hive:tpch:lineitem, filterPredicate = ("orderkey" = BIGINT '999')]
  
  CPU: 17.68s (99.19%), Scheduled: 1.53m (99.67%), Output: 6 rows (54B)
  Input avg.: 22195.08 rows, Input std.dev.: 84.57%
  Input: 9987786 rows (85.78MB), Filtered: 100.00%
```

Fragment 2
CPU: 1.33s, Scheduled: 3.18s, Input: 310000 rows (2.66MB); per task: avg.: 77500.00 std.dev.: 218060.73, Output: 450 rows (11.87kB)

```
...  
Aggregate(PARTIAL)
  
  ScanFilterProject[table = hive:tpch:lineitem, filterPredicate = ("orderkey" = BIGINT '999')]
  
  CPU: 1.28s (95.88%), Scheduled: 3.37s (97.20%), Output: 6 rows (54B)
  Input avg.: 833.33 rows, Input std.dev.: 331.66%
  Input: 310000 rows (2.66MB), Filtered: 100.00%
```
5 minute break

And if you stick around:

- Browse prestosql.io
- Join us on Slack
- Submit questions
Limit pushdown

```sql
SELECT * 
FROM orders 
WHERE orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31' 
LIMIT 10
```

Fragment 1 [SOURCE]

CPU: 5.27ms, Scheduled: 24.19ms, Input: 10 rows (1.28kB); per task: avg.: 10.00 std.dev.: 0.00, Output: 10 rows (1.28kB)
Output layout: [orderkey, custkey, orderstatus, totalprice, orderdate, orderpriority, clerk, shippriority, comment]
Output partitioning: SINGLE []
Stage Execution Strategy: UNGROUPED_EXECUTION
TableScan[postgresql:tpch.orders tpch.orders | limit=10, grouped = false]
  Layout: ...
  CPU: 5.00ms (100.00%), Scheduled: 24.00ms (100.00%), Output: 10 rows (1.28kB)
  Input avg.: 10.00 rows, Input std.dev.: 0.00%
  clerk := clerk:varchar(15):varchar
  orderkey := orderkey:bigint:int8
  orderstatus := orderstatus:varchar(1):varchar
  custkey := custkey:bigint:int8
  totalprice := totalprice:double:float8
  comment := comment:varchar(79):varchar
  orderdate := orderdate:date:date
  orderpriority := orderpriority:varchar(15):varchar
  shippriority := shippriority:integer:int4
Partial limit pushdown

```
SELECT *
FROM orders
LIMIT 10
```

Fragment 1 [SINGLE]
- CPU: 1.86ms, Scheduled: 4.00ms, Input: 10 rows (3.88kB); per task: avg.: 10.00 std.dev.: 0.00, Output: 10 rows (3.88kB)
- Output layout: [id, _source, _score, clerk, comment, custkey, orderdate, orderkey, orderpriority, orderstatus, shippriority, totalprice]
- Stage Execution Strategy: UNGROUPED_EXECUTION
- Limit[10]
  - Layout: ...
  - LocalExchange[SINGLE] ()
    - Layout: ...
      - RemoteSource[2]
        - Layout: ...
          - CPU: 0.00ns (0.00%), Scheduled: 0.00ns (0.00%), Output: 10 rows (3.88kB)
          - Input avg.: 2.50 rows, Input std.dev.: 173.21%

Fragment 2 [SOURCE]
- CPU: 3.74ms, Scheduled: 19.56ms, Input: 10 rows (3.88kB); per task: avg.: 10.00 std.dev.: 0.00, Output: 10 rows (3.88kB)
- Output layout: [id, _source, _score, clerk, comment, custkey, orderdate, orderkey, orderpriority, orderstatus, shippriority, totalprice]
- Output partitioning: SINGLE []
- LimitPartial[10]
  - Layout: ...
  - TableScan[elasticsearch:SCAN:orders[limit=10], grouped = false]
    - Layout: ...
      - CPU: 3.00ms (100.00%), Scheduled: 28.00ms (100.00%), Output: 10 rows (3.88kB)
      - Input avg.: 10.00 rows, Input std.dev.: 0.00%
      - ...
SELECT orderstatus, \textit{sum}(totalprice) \\
FROM orders \\
WHERE orderdate \textit{BETWEEN} DATE '1995-01-01' \textit{AND} DATE '1995-12-31' \\
GROUP BY orderstatus
```sql
SELECT count(*)
FROM visits JOIN pages USING (page_id)
WHERE url = 'index.html'
```
Cost-based optimizations
Cost-based optimizations

Data-dependent, based on statistics

Optimize for

- CPU
- Memory requirements
- Network re-shuffles
- Skew avoidance

Optimizations

- Join type selection (Partitioned vs Broadcast)
- Join reordering
Partitioned join
Broadcast join

Join

Scan lineitem

Join

Table

Scan lineitem

Join

Table

Scan orders

Scan orders

broadcast
SELECT orderpriority, SUM(extendedprice * discount) 
FROM lineitem JOIN orders USING (orderkey) 
GROUP BY orderpriority
Join type selection - Broadcast

SELECT orderpriority, SUM(extendedprice * discount)
FROM lineitem JOIN orders USING (orderkey)
WHERE orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-01-31'
GROUP BY orderpriority

Fragment 2 [SOURCE]
...  
| Aggregate(PARTIAL)[orderpriority] |
| Project[] |
| InnerJoin["orderkey_0" = "orderkey"] |
| Distribution: REPLICATED |
| TableScan[hive:tpch:lineitem, grouped = false] |
| LocalExchange[HASH] ("orderkey") |
| ... |
| RemoteSource[3] |

Fragment 3 [SOURCE]
Output partitioning: BROADCAST []
{rows: 187110 (2.50MB), cpu: 543.84M, memory: 0B, network: 0B}/
{rows: 187110 (2.50MB), cpu: 546.33M, memory: 0B, network: 0B}
Disabling cost-based optimizations

\texttt{SET SESSION join\_distribution\_type = 'BROADCAST'}

\texttt{SET SESSION join\_distribution\_type = 'PARTITIONED'}

\texttt{SET SESSION join\_reordering\_strategy = 'NONE'}
Join reordering

```
SELECT c.custkey, sum(l.extendedprice * l.discount) AS discount
FROM customer c, orders o, lineitem l
WHERE c.custkey = o.custkey AND l.orderkey = o.orderkey
GROUP BY c.custkey
ORDER BY discount DESC
```
Table statistics

SHOW STATS FOR orders

<table>
<thead>
<tr>
<th>column_name</th>
<th>data_size</th>
<th>distinct_values_count</th>
<th>nulls_fraction</th>
<th>row_count</th>
<th>low_value</th>
<th>high_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderkey</td>
<td>NULL</td>
<td>1.5E7</td>
<td>0.0</td>
<td>NULL</td>
<td>1</td>
<td>60000000</td>
</tr>
<tr>
<td>custkey</td>
<td>NULL</td>
<td>1014186.0</td>
<td>0.0</td>
<td>NULL</td>
<td>1</td>
<td>1499999</td>
</tr>
<tr>
<td>orderstatus</td>
<td>1.5E7</td>
<td>3.0</td>
<td>0.0</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>totalprice</td>
<td>NULL</td>
<td>1.2476914E7</td>
<td>0.0</td>
<td>NULL</td>
<td>838.05</td>
<td>558822.56</td>
</tr>
<tr>
<td>orderdate</td>
<td>NULL</td>
<td>2449.0</td>
<td>0.0</td>
<td>NULL</td>
<td>1992-01-01</td>
<td>1998-08-02</td>
</tr>
<tr>
<td>orderpriority</td>
<td>1.2600876E8</td>
<td>5.0</td>
<td>0.0</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>clerk</td>
<td>2.25E8</td>
<td>9806.0</td>
<td>0.0</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>shippriority</td>
<td>NULL</td>
<td>1.0</td>
<td>0.0</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>comment</td>
<td>7.27385523E8</td>
<td>1.3839831E7</td>
<td>0.0</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>1.5E7</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Computing statistics

Automatically when inserting data into tables
Via ANALYZE command
Resources

Blog Posts:

Intro to Cost-based Optimizer: https://prestosql.io/blog/2019/07/04/cbo-introduction.html
Dynamic partition pruning: https://prestosql.io/blog/2020/06/14/dynamic-partition-pruning.html
Dynamic filtering: https://prestosql.io/blog/2019/06/30/dynamic-filtering.html
Cast Optimization: https://prestosql.io/blog/2019/05/21/optimizing-the-casts-away.html
Removing redundant ORDER BY: https://prestosql.io/blog/2019/06/03/redundant-order-by.html

Documentation

Optimizer: https://prestosql.io/docs/current/optimizer.html
ANALYZE: https://prestosql.io/docs/current/sql/analyze.html
SHOW STATS: https://prestosql.io/docs/current/sql/show-stats.html
Wrapping up
Presto Training Series

Join the Presto creators again for more:

• Advanced SQL in Presto with David - recording available

• Securing Presto with Dain (26 Aug)

• Configuring and Tuning Presto Performance with Dain (9 Sept)
Presto Summit series

Diverse information about Presto and real world usage

• State of Presto - recording available

• Presto as Query Layer at Zuora - recording available

• Presto Migration at Arm Treasure Data - recording available

• Presto for Analytics at Pinterest - 19 Aug
And finally …

- Learn more from our website and documentation at prestosql.io
- Join us on slack at prestosql.io/slack
- Get a free digital copy of Presto: The Definitive Guide
- Thank you for hanging out with us
- See you next time
Your question
Our answers ...