Presto Training Series, Session 1: Using Advanced SQL Features In Presto

Try Presto: www.prestosql.io

David Phillips and Manfred Moser

29 July 2020

Today's Speakers

Manfred Moser

Developer, author,
and trainer at Starburst

Manfred is an open source developer and advocate. He is an Apache Maven committer, 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.

David Phillips

Co-creator of Presto and
CTO at Starburst

David is a co-creator of Presto, co-founder of the Presto Software Foundation, and CTO at Starburst. Prior to Starburst, David was a software engineer at Facebook, and held senior engineering positions at Proofpoint, Ning and Adknowledge.

Agenda

- Presto overview / review
- Using advanced SQL features in Presto
  - General SQL features
  - Using JSON
  - Advanced aggregation techniques
- Five minute break
  - Array and map functions
  - Window functions
- Q&A

Questions

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

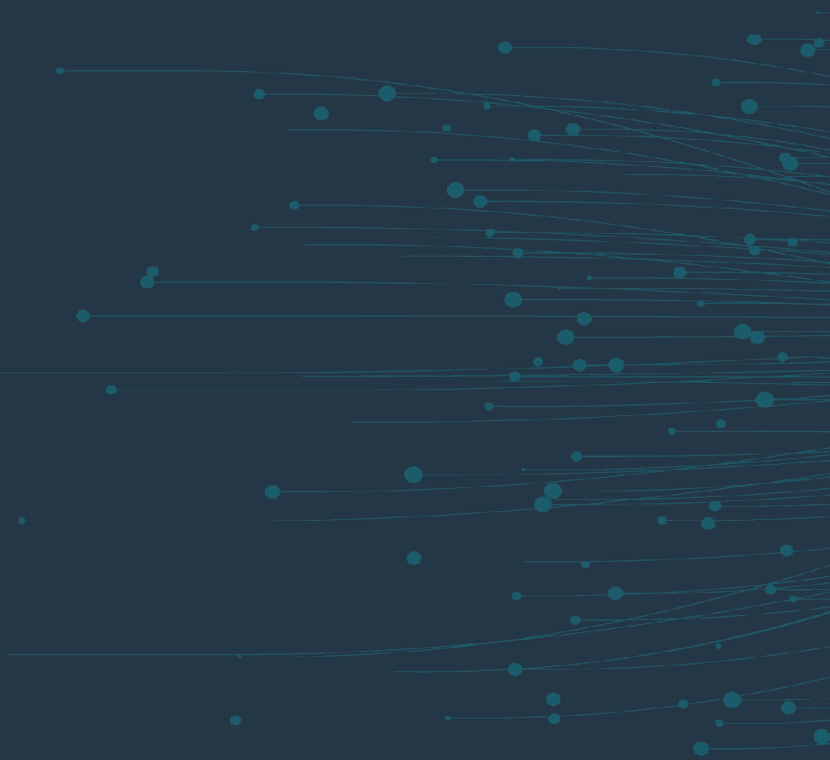
Starburst

Some advice for attendees

- This is a fast-paced overview – don't try to follow along during class
- Instead focus and pay attention – this is advanced SQL
- 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
- 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 scalabilty
- 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

General SQL Features

Format function

```
VALUES format('pi = %.5f', pi()),
       format('agent %03d', 7),
       format('$%,.2f', 1234567.89),
       format('%-7s,%7s', 'hello', 'world'),
       format('%2$s %3$s %1$s', 'a', 'b', 'c'),
       format('%1$tA, %1$tB %1$te, %1$tY', date '2006-07-04');
```

```
pi = 3.14159
agent 007
$1,234,567.89
hello  ,  world
b c a
Tuesday, July 4, 2006
```

Starburst

Simple case expressions

```
SELECT n,
       CASE n
           WHEN 1 THEN 'one'
           WHEN 2 THEN 'two'
           ELSE 'many'
       END AS name
FROM (VALUES 1, 2, 3, 4) AS t (n);
```

```
 n | name
---+------
 1 | one
 2 | two
 3 | many
 4 | many
```

Starburst

Searched case expressions

```
SELECT n,
       CASE
           WHEN n = 1 THEN 'aaa'
           WHEN n IN (2, 3) THEN 'bbb'
           ELSE 'ccc'
       END AS name
FROM (VALUES 1, 2, 3, 4) AS t (n);
```

```
 n | name
---+------
 1 | aaa
 2 | bbb
 3 | bbb
 4 | ccc
```

Starburst

IF expression

```
SELECT format('I have %s cat%s.', n,
              IF(n = 1, '', 's')) AS text
FROM (VALUES 0, 1, 2, 3) AS t (n);
```

```
      text
----------------
 I have 0 cats.
 I have 1 cat.
 I have 2 cats.
 I have 3 cats.
```

Starburst

TRY expression

```
SELECT try(8 / 0) div_zero,
       try(cast('abc' AS integer)) not_integer,
       try(2000000000 + 2000000000) overflow;

 div_zero | not_integer | overflow
----------+-------------+----------
     NULL |        NULL |     NULL
```

TRY avoids these failures:

```
Query failed: Division by zero
Query failed: Cannot cast 'abc' to INT
Query failed: integer addition overflow: 2000000000 + 2000000000
```

Starburst

Lambda expressions overview

Lambda expression with one input:

```
x -> x + 8
```

Lambda expression with two inputs:

```
(x, y) -> x + y
```

Using JSON

JSON data type

```
SELECT
  json_parse('null') j_null,
  json_parse('true') bool,
  json_parse('"hello"') string,
  json_parse('123.45') number,
  json_parse('[1, 3, 5, 9]') array,
  json_parse('["hello", 123, {"xyz": 99, "abc": false}]') mixed;
```

```
 j_null | bool | string  | number |   array   |                mixed
--------+------+---------+--------+-----------+---------------------------------------
 null   | true | "hello" | 123.45 | [1,3,5,9] | ["hello",123,{"abc":false,"xyz":99}]
```

Starburst

Extraction using JSONPath

```
SELECT json_extract(v, '$.languages') AS languages,
       json_extract(v, '$.languages[0].name') AS name_json,
       json_extract_scalar(v, '$.languages[0].name') AS name_scalar
FROM (VALUES JSON '
    {"languages": [{"name": "Java"}, {"name": "Python"}]}
') AS t (v);

              languages               | name_json | name_scalar
--------------------------------------+-----------+-------------
 [{"name":"Java"},{"name":"Python"}] | "Java"    | Java
```

Starburst

Casting from JSON

```
SELECT cast(v AS map(varchar,array(map(varchar,varchar))))
FROM (VALUES JSON '
    {"languages": [{"name": "Java"}, {"name": "Python"}]}
') AS t (v);

                  _col0
-------------------------------------------
 {languages=[{name=Java}, {name=Python}]}
```

Starburst

Partial casting from JSON

```
SELECT cast(v AS map(varchar,array(map(varchar,json))))
FROM (VALUES JSON '
    {"languages":
        [{"id": 123, "name": "Java", "data": [88,99]},
         {"id": 456, "name": "Python"}]}
') AS t (v);

                                    _col0
-------------------------------------------------------------------------------
 {languages=[{data=[88,99], name="Java", id=123}, {name="Python", id=456}]}
```

Starburst

Formatting as JSON

```
SELECT json_format(JSON '[1, 2, 3]') AS array,
       json_format(JSON '{"xyz": 99, "abc": false}') AS map;

  array  |          map
---------+------------------------
 [1,2,3] | {"abc":false,"xyz":99}
```

Advanced Aggregation Techniques

Counting distinct items

How many unique customers do we have?

```
SELECT count(DISTINCT custkey) AS customers
FROM orders;

 customers
-----------
     99996
```

This gives an exact answer, but is slow and memory intensive.

Counting distinct items

Approximately how many unique customers do we have?

```
SELECT approx_distinct(custkey) AS customers
FROM orders;

 customers
-----------
    101655
```

This example has an error of 1.66%.

Counting distinct items

From the approx_distinct() documentation:

> *This function should produce a standard error of 2.3%, which is the standard deviation of the (approximately normal) error distribution over all possible sets. It does not guarantee an upper bound on the error for any specific input set.*

The function uses the HyperLogLog algorithm to approximate the number of distinct items. The error depends on the unique values, not how many times they appear in the input. Both of these produce the same error:

- `approx_distinct(x)`
- `approx_distinct(DISTINCT x)`

Approximate percentiles

What is the order price at the 50th percentile?

```
SELECT round(avg(price)) AS avg,
       approx_percentile(price, 0.5) AS pct
FROM (
  SELECT cast(round(totalprice) AS bigint) AS price
  FROM orders
);
```

```
   avg    |  pct
----------+--------
 151220.0 | 144767
```

Starburst

Approximate percentiles

What are the order prices at the 10th, 20th, 50th, 90th, and 99th percentiles?

```
SELECT approx_percentile(price, array[0.1, 0.2, 0.5, 0.9, 0.99]) AS pct
FROM (
  SELECT cast(round(totalprice) AS bigint) AS price
  FROM orders
);
```

```
                  pct
----------------------------------------
 [39039, 65535, 144767, 274431, 366591]
```

Starburst

Associated max value

Find the clerk who has the most expensive order:

```
SELECT max_by(clerk, totalprice) clerk,
       max(totalprice) price
FROM orders;
```

```
      clerk       |   price
------------------+-----------
 Clerk#000000040 | 555285.16
```

Associated max value using a row type

Find the clerk who has the most expensive order:

```
SELECT max(cast(row(totalprice, clerk) AS
           row(price double, clerk varchar))).*
FROM orders;

   price   |      clerk
-----------+-----------------
 555285.16 | Clerk#000000040
```

Associated max values

Find the clerks who have the most expensive orders:

```
SELECT max_by(clerk, totalprice, 3) clerks
FROM orders;
```

```
                       clerks
-----------------------------------------------------
 [Clerk#000000040, Clerk#000000230, Clerk#000000699]
```

Pivoting with conditional counting

Order counts by order priority, as separate columns:

```
SELECT
  count_if(orderpriority = '1-URGENT') AS urgent,
  count_if(orderpriority = '2-HIGH') AS high,
  count_if(orderpriority = '3-MEDIUM') AS medium,
  count_if(orderpriority = '4-NOT SPECIFIED') AS not_specified,
  count_if(orderpriority = '5-LOW') AS low
FROM orders;
```

```
 urgent |  high  | medium | not_specified |  low
--------+--------+--------+---------------+--------
 300343 | 300091 | 298723 |        300254 | 300589
```

Starburst

Pivoting with filtering

Order counts by order priority, as separate columns:

```
SELECT
  count(*) FILTER (WHERE orderpriority = '1-URGENT') AS urgent,
  count(*) FILTER (WHERE orderpriority = '2-HIGH') AS high,
  count(*) FILTER (WHERE orderpriority = '3-MEDIUM') AS medium,
  count(*) FILTER (WHERE orderpriority = '4-NOT SPECIFIED') AS not_specified,
  count(*) FILTER (WHERE orderpriority = '5-LOW') AS low
FROM orders;
```

```
 urgent |  high  | medium | not_specified |  low
--------+--------+--------+---------------+--------
 300343 | 300091 | 298723 |        300254 | 300589
```

Starburst

Pivoting averages

Total order price by order priority, as separate columns:

```
SELECT
  avg(totalprice) FILTER (WHERE orderpriority = '1-URGENT') AS urgent,
  avg(totalprice) FILTER (WHERE orderpriority = '2-HIGH') AS high,
  avg(totalprice) FILTER (WHERE orderpriority = '3-MEDIUM') AS medium,
  avg(totalprice) FILTER (WHERE orderpriority = '4-NOT SPECIFIED') AS not_specified,
  avg(totalprice) FILTER (WHERE orderpriority = '5-LOW') AS low
FROM orders;

  urgent   |   high    |  medium   | not_specified |    low
-----------+-----------+-----------+---------------+-----------
 151222.87 | 151553.28 | 151155.45 |     150792.44 | 151373.33
```

Starburst

Aggregating a complex expression

What if we charge a premium based on order priority?

```
SELECT avg(totalprice *
           CASE
             WHEN orderpriority = '1-URGENT' THEN 1.10
             WHEN orderpriority = '2-HIGH' THEN 1.05
             ELSE 1.0
           END) / avg(totalprice) AS premium
FROM orders;

 premium
---------
 1.03005
```

Starburst

Aggregating into an array

Build an array from region names, in descending order:

```
SELECT array_agg(name ORDER BY name DESC) names
FROM region;
```

```
                    names
-----------------------------------------------
 [MIDDLE EAST, EUROPE, ASIA, AMERICA, AFRICA]
```

Starburst

Aggregating using a lambda

Compute the product of the values in the group:

```
SELECT name,
       reduce_agg(value, 1,
                  (a, b) -> a * b,
                  (a, b) -> a * b) AS product
FROM (VALUES ('x', 1), ('x', 3), ('x', 5),
             ('y', 2), ('y', 4), ('y', 6)) AS t (name, value)
GROUP BY name;
```

```
 name | product
------+---------
 x    |      15
 y    |      48
```

Starburst

Order-insensitive checksums

Compare data between tables by computing a checksum:

```
SELECT checksum(orderkey) AS check_orderkey,
       checksum(row(custkey, orderstatus, totalprice)) AS check_multiple
FROM orders;
```

```
     check_orderkey      |     check_multiple
-------------------------+-------------------------
 e8 9a ce bd 9a 26 30 54 | b3 e1 57 6b 07 28 a0 6f
```

ROLLUP with single

```
SELECT orderpriority,
       count(*) AS orders
FROM orders
GROUP BY ROLLUP(orderpriority)
ORDER BY orderpriority;
```

```
  orderpriority   | orders
------------------+---------
 1-URGENT         |  300343
 2-HIGH           |  300091
 3-MEDIUM         |  298723
 4-NOT SPECIFIED  |  300254
 5-LOW            |  300589
 NULL             | 1500000
```

ROLLUP with multiple

```
SELECT linestatus, returnflag,
       count(*) AS items
FROM lineitem
GROUP BY ROLLUP(linestatus, returnflag)
ORDER BY linestatus, returnflag;
```

linestatus	returnflag	items
F	A	1478493
F	N	38854
F	R	1478870
F	NULL	2996217
O	N	3004998
O	NULL	3004998
NULL	NULL	6001215

CUBE

```
SELECT linestatus, returnflag,
       count(*) AS items
FROM lineitem
GROUP BY CUBE(linestatus, returnflag)
ORDER BY linestatus, returnflag;
```

linestatus	returnflag	items
F	A	1478493
F	N	38854
F	R	1478870
F	NULL	2996217
O	N	3004998
O	NULL	3004998
NULL	A	1478493
NULL	N	3043852
NULL	R	1478870
NULL	NULL	6001215

GROUPING SETS

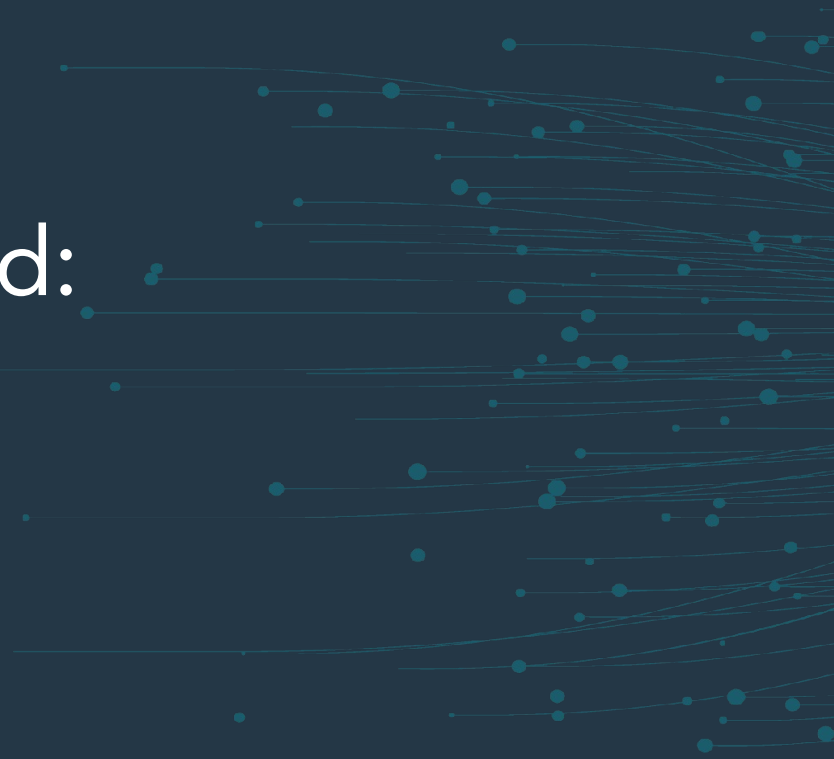
```
SELECT linestatus, returnflag,
       count(*) AS items
FROM lineitem
GROUP BY GROUPING SETS (
    (linestatus),
    (returnflag),
    (linestatus, returnflag),
    ()
)
ORDER BY linestatus, returnflag;
```

linestatus	returnflag	items
F	A	1478493
F	N	38854
F	R	1478870
F	NULL	2996217
O	N	3004998
O	NULL	3004998
NULL	A	1478493
NULL	N	3043852
NULL	R	1478870
NULL	NULL	6001215

5 minute break

And if you stick around:

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



Array and Map Functions

Creating arrays

```
SELECT ARRAY[4, 5, 6] AS integers,
       ARRAY['hello', 'world'] AS varchars;

 integers  |    varchars
-----------+----------------
 [4, 5, 6] | [hello, world]
```

Starburst

Accessing array elements

```
SELECT a[2] AS second1,
       element_at(a, 2) AS second2,
       element_at(a, -2) AS second_from_last,
       element_at(a, 99) AS bad
FROM (VALUES ARRAY[4, 5, 6, 7, 8]) AS t (a);
```

```
 second1 | second2 | second_from_last | bad
---------+---------+------------------+------
       5 |       5 |                7 | NULL
```

Accessing an invalid subscript with [] fails:

```
Query failed: Array subscript must be less than or equal to array length: 8 > 5
Query failed: Array subscript is negative: -2
```

Starburst

Sorting arrays

```
SELECT array_sort(ARRAY['a', 'xyz', 'bb', 'abc', 'z', 'b'],
    (x, y) -> CASE
                  WHEN length(x) < length(y) THEN -1
                  WHEN length(x) > length(y) THEN 1
                  ELSE 0
              END) AS sorted;

         sorted
-------------------------
 [a, z, b, bb, xyz, abc]
```

Starburst

Matching elements

Do any, all, or none of the elements equal 8?

```
SELECT a,
       any_match(a, e -> e = 8) AS any,
       all_match(a, e -> e = 8) AS all,
       none_match(a, e -> e = 8) AS none
FROM (VALUES ARRAY[4, 5, 6, 7, 8]) AS t (a);
```

```
        a        | any  |  all  | none
-----------------+------+-------+-------
 [4, 5, 6, 7, 8] | true | false | false
```

Starburst

Filtering elements

```
SELECT a,
       filter(a, x -> x > 0) AS positive,
       filter(a, x -> x IS NOT NULL) AS non_null
FROM (VALUES ARRAY[5, -6, NULL, 7]) AS t (a);
```

```
        a         | positive |  non_null
------------------+----------+------------
 [5, -6, NULL, 7] | [5, 7]   | [5, -6, 7]
```

Starburst

Transforming elements

```
SELECT a,
       transform(a, x -> abs(x)) AS positive,
       transform(a, x -> x * x) AS squared
FROM (VALUES ARRAY[5, -6, NULL, 7]) AS t (a);
```

```
        a         |    positive     |      squared
------------------+-----------------+--------------------
 [5, -6, NULL, 7] | [5, 6, NULL, 7] | [25, 36, NULL, 49]
```

Starburst

Converting arrays to strings

```
SELECT array_join(sequence(3, 7), '/') AS joined;

  joined
-----------
 3/4/5/6/7

SELECT a,
       array_join(transform(a, e -> format('%,d', e)), ' / ') AS value
FROM (VALUES ARRAY[12345678, 987654321]) AS t (a);

           a           |          value
-----------------------+--------------------------
 [12345678, 987654321] | 12,345,678 / 987,654,321
```

Computing array product

```
SELECT a,
       reduce(a, 1,
              (a, b) -> a * b,
              x -> x) AS product
FROM (VALUES ARRAY[1, 2, 3, 4, 5]) AS t (a);
```

```
        a         | product
------------------+---------
 [1, 2, 3, 4, 5] |     120
```

Starburst

Unnesting an array

```
SELECT name
FROM (
    VALUES ARRAY['cat', 'dog', 'mouse']
) AS t (a)
CROSS JOIN UNNEST(a) AS x (name);
```

```
 name
-------
 cat
 dog
 mouse
```

Unnesting an array with ordinality

```
SELECT id, name
FROM (
    VALUES ARRAY['cat', 'dog', 'mouse']
) AS t (a)
CROSS JOIN UNNEST(a) WITH ORDINALITY AS x (name, id);
```

```
 id | name
----+-------
  1 | cat
  2 | dog
  3 | mouse
```

Starburst

Creating maps

Create a map from arrays of keys and values:

```
SELECT map(ARRAY['x', 'y'], ARRAY[123, 456]);
```

Create a map from an array of entry rows:

```
SELECT map_from_entries(ARRAY[('x', 123), ('y', 456)]);
```

```
     _col0
----------------
 {x=123, y=456}
```

Starburst

Accessing map elements

```
SELECT m,
       m['xyz'] AS xyz,
       element_at(m, 'abc') AS abc,
       element_at(m, 'bad') AS missing
FROM (VALUES map_from_entries(ARRAY[('abc', 123), ('xyz', 456)])) AS t (m);
```

```
         m          | xyz | abc | missing
--------------------+-----+-----+---------
 {abc=123, xyz=456} | 456 | 123 |    NULL
```

Accessing an invalid key with [] fails:

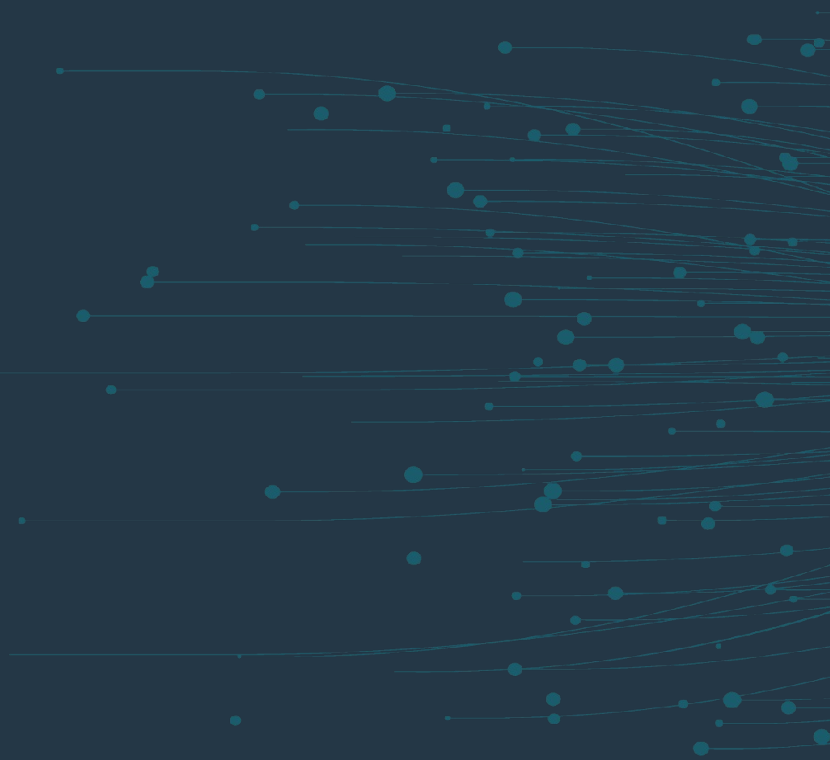
```
Query failed: Key not present in map: bad
```

Unnesting a map

```
SELECT key, value
FROM (
    VALUES map_from_entries(ARRAY[('abc', 123), ('xyz', 456)])
) AS t (m)
CROSS JOIN UNNEST(m) AS x (key, value);
```

```
 key | value
-----+-------
 abc |   123
 xyz |   456
```

Window Functions



Window function overview

Window functions run across rows of the result. Processing order:

1. FROM and JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. Window functions 
6. SELECT
7. DISTINCT
8. ORDER BY
9. LIMIT

Row numbering

Assign each region a unique number, in name order:

```
SELECT name,
       row_number() OVER (ORDER BY name) AS id
FROM region
ORDER BY name;
```

```
    name     | id
-------------+----
 AFRICA      |  1
 AMERICA     |  2
 ASIA        |  3
 EUROPE      |  4
 MIDDLE EAST |  5
```

Row numbering order

Assign each region a unique number, in descending name order:

```
SELECT name,
       row_number() OVER (ORDER BY name DESC) AS id
FROM region
ORDER BY name;
```

```
    name     | id
-------------+----
 AFRICA      |  5
 AMERICA     |  4
 ASIA        |  3
 EUROPE      |  2
 MIDDLE EAST |  1
```

Starburst

Row numbering with limit

Assign each region a number, in descending name order, limited to three rows:

```
SELECT name,
       row_number() OVER (ORDER BY name DESC) AS row_number
FROM region
ORDER BY name
LIMIT 3;
```

```
    name     | id
-------------+----
 AFRICA      |  5
 AMERICA     |  4
 ASIA        |  3
```

Starburst

Rank

Assign a rank to each region, in descending name order:

```
SELECT name,
       rank() OVER (ORDER BY name DESC) AS rank
FROM region
ORDER BY name;
```

```
    name     | rank
-------------+------
 AFRICA      |    5
 AMERICA     |    4
 ASIA        |    3
 EUROPE      |    2
 MIDDLE EAST |    1
```

Rank with ties

Assign a rank to each region, based on first letter of name:

```
SELECT name,
       rank() OVER (ORDER BY substr(name, 1, 1)) AS rank
FROM region
ORDER BY name;
```

```
    name     | rank
-------------+------
 AFRICA      |    1
 AMERICA     |    1
 ASIA        |    1
 EUROPE      |    4
 MIDDLE EAST |    5
```

Dense rank with ties

Assign a rank to each region, based on first letter of name:

```
SELECT name,
       dense_rank() OVER (ORDER BY substr(name, 1, 1)) AS rank
FROM region
ORDER BY name;
```

```
    name     | rank
-------------+------
 AFRICA      |    1
 AMERICA     |    1
 ASIA        |    1
 EUROPE      |    2
 MIDDLE EAST |    3
```

Ranking without ordering

Assign a rank to each region:

```
SELECT name,
       rank() OVER (ORDER BY null) AS x,
       rank() OVER () AS y
FROM region
ORDER BY name;
```

```
    name     | x | y
-------------+---+---
 AFRICA      | 1 | 1
 AMERICA     | 1 | 1
 ASIA        | 1 | 1
 EUROPE      | 1 | 1
 MIDDLE EAST | 1 | 1
```

Row numbering without ordering

Assign a rank to each region:

```
SELECT name,
       row_number() OVER (ORDER BY null) AS x,
       row_number() OVER () AS y
FROM region
ORDER BY name;
```

```
    name     | x | y
-------------+---+---
 AFRICA      | 1 | 1
 AMERICA     | 2 | 2
 ASIA        | 3 | 3
 EUROPE      | 4 | 4
 MIDDLE EAST | 5 | 5
```

Assigning rows to buckets

Assign rows into three buckets, in name order:

```
SELECT name,
       ntile(3) OVER (ORDER BY name) AS bucket
FROM region
ORDER BY name;
```

```
    name     | bucket
-------------+--------
 AFRICA      |      1
 AMERICA     |      1
 ASIA        |      2
 EUROPE      |      2
 MIDDLE EAST |      3
```

Percentage ranking

Percentage rank of rows, in name order:

```
SELECT name,
       percent_rank() OVER (ORDER BY name) AS percent
FROM region
ORDER BY name;
```

```
    name     | percent
-------------+---------
 AFRICA      |     0.0
 AMERICA     |    0.25
 ASIA        |     0.5
 EUROPE      |    0.75
 MIDDLE EAST |     1.0
```

Starburst

Partitioning

Divide regions by first letter of name, then assign ranks:

```
SELECT name,
       rank() OVER (PARTITION BY substr(name, 1, 1) ORDER BY name) AS rank
FROM region
ORDER BY name;
```

```
    name     | rank
-------------+------
 AFRICA      |    1
 AMERICA     |    2
 ASIA        |    3
 EUROPE      |    1
 MIDDLE EAST |    1
```

Partitioning on the same value

Assign a rank to each region:

```
SELECT name,
       rank() OVER (PARTITION BY null ORDER BY name) AS x,
       rank() OVER (ORDER BY name) AS y
FROM region
ORDER BY name;
```

```
    name     | x | y
-------------+---+---
 AFRICA      | 1 | 1
 AMERICA     | 2 | 2
 ASIA        | 3 | 3
 EUROPE      | 4 | 4
 MIDDLE EAST | 5 | 5
```

Accessing leading and trailing rows

Access a value in the row behind and ahead of the current row:

```
SELECT name,
       lag(name) OVER (ORDER BY name) AS lag,
       lead(name) OVER (ORDER BY name) AS lead
FROM region
ORDER BY name;
```

```
    name     |   lag   |    lead
-------------+---------+-------------
 AFRICA      | NULL    | AMERICA
 AMERICA     | AFRICA  | ASIA
 ASIA        | AMERICA | EUROPE
 EUROPE      | ASIA    | MIDDLE EAST
 MIDDLE EAST | EUROPE  | NULL
```

Starburst

Accessing leading and trailing rows

Access a value in the row behind and ahead of the current row, with default:

```
SELECT name,
       lag(name, 1, 'none') OVER (ORDER BY name) AS lag,
       lead(name, 1, 'none') OVER (ORDER BY name) AS lead
FROM region
ORDER BY name;
```

```
    name     |   lag   |    lead
-------------+---------+-------------
 AFRICA      | none    | AMERICA
 AMERICA     | AFRICA  | ASIA
 ASIA        | AMERICA | EUROPE
 EUROPE      | ASIA    | MIDDLE EAST
 MIDDLE EAST | EUROPE  | none
```

Starburst

Accessing leading and trailing rows

Access a value two rows back and two rows ahead, with default:

```
SELECT name,
       lag(name, 2, 'none') OVER (ORDER BY name) AS lag2,
       lead(name, 2, 'none') OVER (ORDER BY name) AS lead2
FROM region
ORDER BY name;
```

```
    name     |  lag2   |    lead2
-------------+---------+-------------
 AFRICA      | none    | ASIA
 AMERICA     | none    | EUROPE
 ASIA        | AFRICA  | MIDDLE EAST
 EUROPE      | AMERICA | none
 MIDDLE EAST | ASIA    | none
```

Starburst

Accessing leading and trailing rows with nulls

Access a value in the row behind and ahead of the current row, respecting nulls:

```
SELECT id, v,
       lag(v) OVER (ORDER BY id) AS lag,
       lead(v) OVER (ORDER BY id) AS lead
FROM (VALUES (1, 'a'), (2, 'b'), (3, null), (4, 'd'), (5, null)) AS t (id, v)
ORDER BY id;
```

```
 id |  v   | lag  | lead
----+------+------+------
  1 | a    | NULL | b
  2 | b    | a    | NULL
  3 | NULL | b    | d
  4 | d    | NULL | NULL
  5 | NULL | d    | NULL
```

Starburst

Accessing leading and trailing rows without nulls

Access a value in the row behind and ahead of the current row, ignoring nulls:

```
SELECT id, x,
       lag(x) IGNORE NULLS OVER (ORDER BY id) AS lag,
       lead(x) IGNORE NULLS OVER (ORDER BY id) AS lead
FROM (VALUES (1, 'a'), (2, 'b'), (3, null), (4, 'd'), (5, null)) AS t (id, x)
ORDER BY id;
```

```
 id |  x   | lag  | lead
----+------+------+------
  1 | a    | NULL | b
  2 | b    | a    | d
  3 | NULL | b    | d
  4 | d    | b    | NULL
  5 | NULL | d    | NULL
```

Starburst

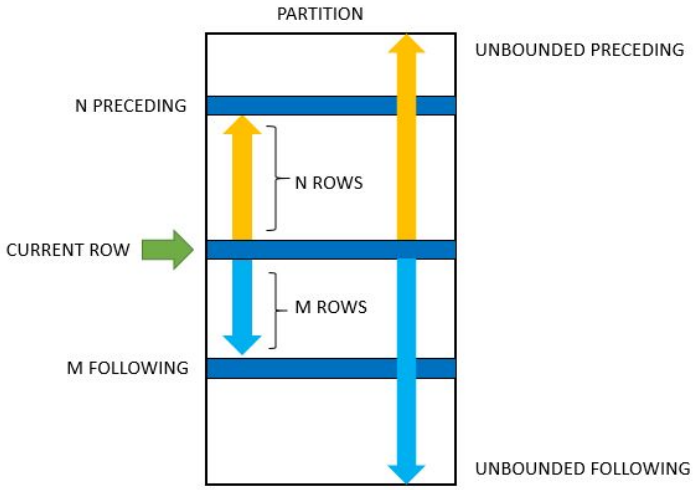
Window frames

Each row in a partition has a frame:

- ROWS: physical frame based on an exact number of rows
- RANGE: logical frame that includes all rows that are *peers* within the ordering

Examples:

- `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
- `RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`
- `ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING`
- `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
- `ROWS BETWEEN UNBOUNDED PRECEDING AND 5 FOLLOWING`
- `ROWS BETWEEN 3 PRECEDING AND UNBOUNDED FOLLOWING`



Source: https://www.sqlitetutorial.net/sqlite-window-functions/sqlite-window-frame/

Starburst

Accessing the first value

```
SELECT name,
       first_value(name) OVER (
           ORDER BY name
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
       ) AS value
FROM region;
```

```
    name     | value
-------------+--------
 AFRICA      | AFRICA
 AMERICA     | AFRICA
 ASIA        | AFRICA
 EUROPE      | AFRICA
 MIDDLE EAST | AFRICA
```

Starburst

Accessing the last value

```
SELECT name,
       last_value(name) OVER (
           ORDER BY name
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
       ) AS value
FROM region;
```

```
    name     |    value
-------------+-------------
 AFRICA      | MIDDLE EAST
 AMERICA     | MIDDLE EAST
 ASIA        | MIDDLE EAST
 EUROPE      | MIDDLE EAST
 MIDDLE EAST | MIDDLE EAST
```

Accessing the Nth value

```
SELECT name,
       nth_value(name, 2) OVER (
           ORDER BY name
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
       ) AS value
FROM region;
```

```
    name     |  value
-------------+---------
 AFRICA      | AMERICA
 AMERICA     | AMERICA
 ASIA        | AMERICA
 EUROPE      | AMERICA
 MIDDLE EAST | AMERICA
```

Starburst

Window frame ROWS vs RANGE

```
SELECT id, v,
       array_agg(v) OVER (ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS rows,
       array_agg(v) OVER (ORDER BY id RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS range,
       array_agg(v) OVER (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS rows_tie,
       array_agg(v) OVER (RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS range_tie
FROM (VALUES (1, 'a'), (2, 'b'), (3, 'c'), (3, 'd'), (5, 'e')) AS t (id, v);
```

```
 id | v |      rows       |      range      |    rows_tie     |    range_tie
----+---+-----------------+-----------------+-----------------+-----------------
  1 | a | [a]             | [a]             | [a]             | [a, b, c, d, e]
  2 | b | [a, b]          | [a, b]          | [a, b]          | [a, b, c, d, e]
  3 | c | [a, b, c]       | [a, b, c, d]    | [a, b, c]       | [a, b, c, d, e]
  3 | d | [a, b, c, d]    | [a, b, c, d]    | [a, b, c, d]    | [a, b, c, d, e]
  5 | e | [a, b, c, d, e] | [a, b, c, d, e] | [a, b, c, d, e] | [a, b, c, d, e]
```

Starburst

Rolling and total sum

```
SELECT v,
       sum(v) OVER (ORDER BY v ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS rolling,
       sum(v) OVER () total
FROM (VALUES 1, 2, 3, 4, 5) AS t (v);
```

```
 v | rolling | total
---+---------+-------
 1 |       1 |    15
 2 |       3 |    15
 3 |       6 |    15
 4 |      10 |    15
 5 |      15 |    15
```

Starburst

Partition sum

```
SELECT p, v,
       sum(v) OVER (
           PARTITION BY p ORDER BY v
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS sum
FROM (VALUES ('a', 1), ('a', 2), ('a', 3), ('b', 4), ('b', 5), ('b', 6)) AS t (p, v);
```

```
 p | v | sum
---+---+-----
 a | 1 |   1
 a | 2 |   3
 a | 3 |   6
 b | 4 |   4
 b | 5 |   9
 b | 6 |  15
```

Starburst

Wrapping up

Presto Training Series

Join the Presto creators again for more:

- Understanding and Tuning Query Processing with Martin (12 Aug)
- Securing Presto with Dain (26 Aug)
- Configuring and Tuning Presto Performance with Dain (9 Sept)

Starburst

Presto Summit series

Diverse information about Presto and real world usage

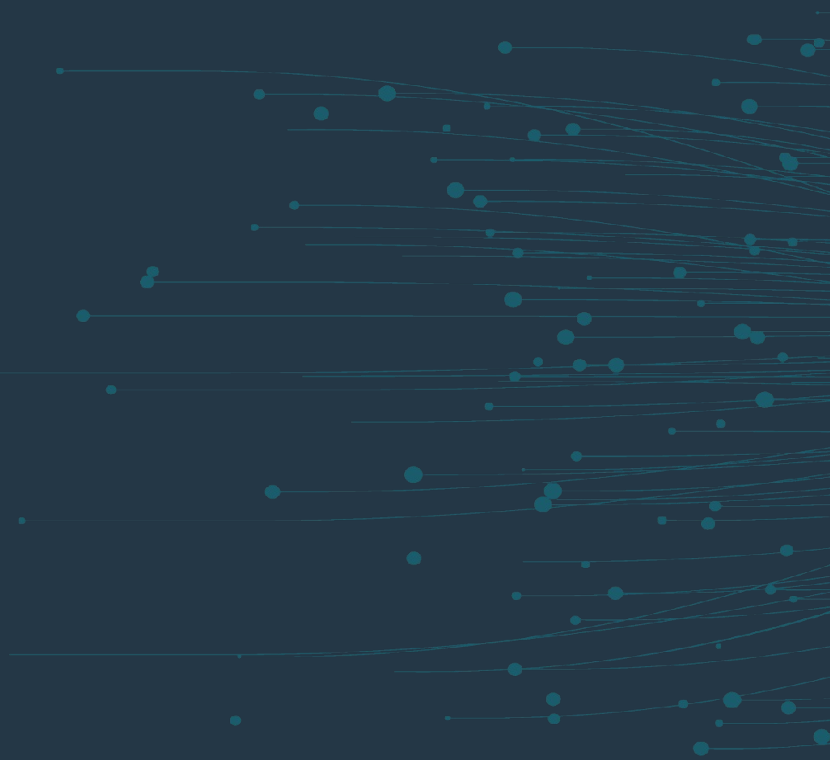
- State of Presto - recording available
- Presto as Query Layer at Zuora - recording available
- Presto Migration at Arm Treasure Data - recording available
- Presto for Analytics at Pinterest - 19 Aug

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

Starburst

Your question
Our answers …



Starburst