SQL features in Trino

Taipei dbt meetup #35
April 23, 2025

David Phillips

Co-creator of Trino and Presto

What is Trino?

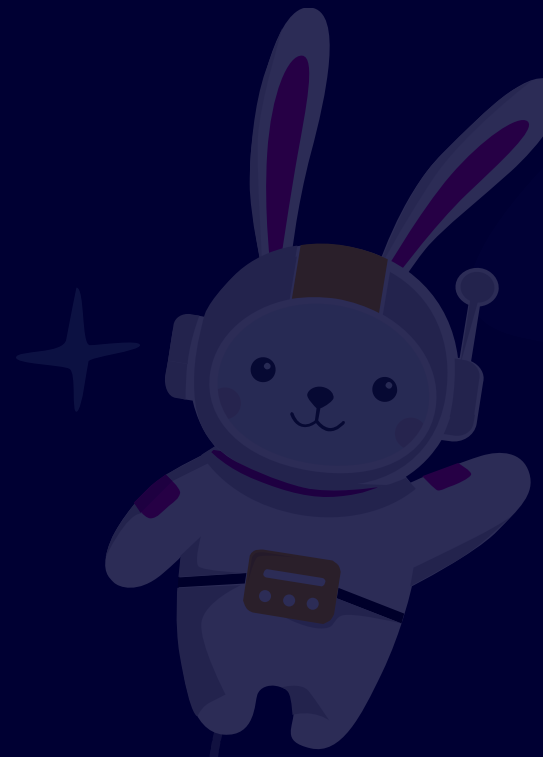
Fast, open source, SQL query engine that can query different data sources.

Apache Iceberg

trino

Why Iceberg?

- Performance and scalability
- Hidden partitioning
- Safe schema evolution
- Time travel and rollback
- ACID transactions



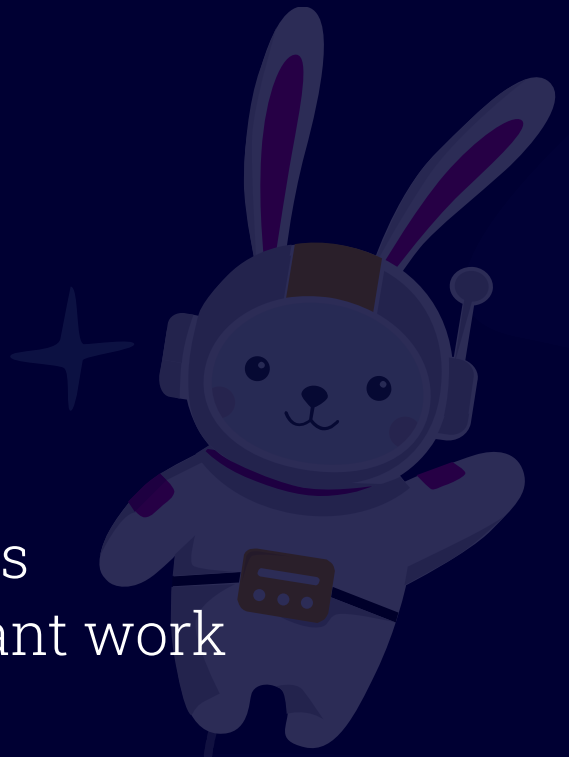
trino

Performance and scalability

- Fast scan planning
- Advanced filtering using metadata
- No file listing
- Supports huge tables

Hive partitioning

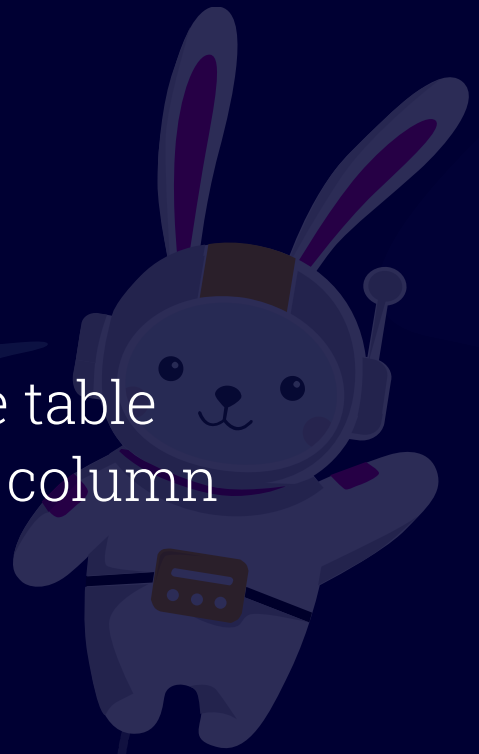
- Partitions are explict columns in the table
- Must use value-based partitioning
- Queries must explicitly filter on partition columns
- Partitioning **cannot be changed** without significant work



trino

Hidden partitioning

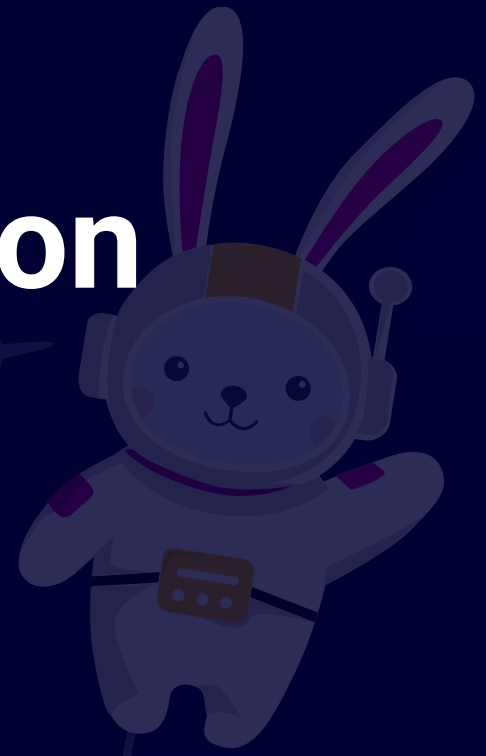
- Partitions are implicit: no partition columns in the table
- Example: partition by day or hour on a timestamp column
- Queries filter on the original columns
- Partition layouts can **evolve** over time



trino

Safe schema evolution

- Add, drop, or rename columns
- Reorder columns in a nested struct
- Widen column types
  - Convert from `int` to `bigint`



Time travel and rollback

- View historical versions of a table
- Rollback to a previous version

ACID transactions

- Data files are tracked in the table metadata
- Readers always see a consistent view of the data
- Concurrent writes are safe from conflicts

What's new in Trino?

Python user-defined functions

```
WITH FUNCTION double_it(x int)
  RETURNS int
  LANGUAGE PYTHON
  WITH (handler = 'twice')
  AS $$
  def twice(a):
      return a * 2
  $$
SELECT double_it(21); -- returns 42
```

```
CREATE FUNCTION double_it(x int)
RETURNS int
LANGUAGE PYTHON
...
```

trino

Spooling client protocol

- Much higher throughput for data transfer
- Faster query completion
- Reduces CPU usage on coordinator
- Direct download of data from spooling storage

Other improvements

- Access control with Open Policy Agent or Apache Ranger
- Lineage tracking with OpenLineage
- Observability with OpenTelemetry and OpenMetrics

trino

SQL features in Trino

trino

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
```

trino

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
```

trino

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.
```

trino

TRY expression

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

TRY avoids these failures:

- Division by zero
- Cannot cast 'abc' to INT
- integer addition overflow: 2000000000 + 2000000000

trino

Conditional and conversion

```
SELECT try_cast('abc' AS integer); -- null

SELECT coalesce(null, null, 'abc', '123'); -- 'abc'

SELECT nullif('abc', 'abc'); -- null
SELECT nullif('abc', '123'); -- 'abc'

SELECT typeof('123'), typeof(123), typeof(123.0);
```

trino

Format function

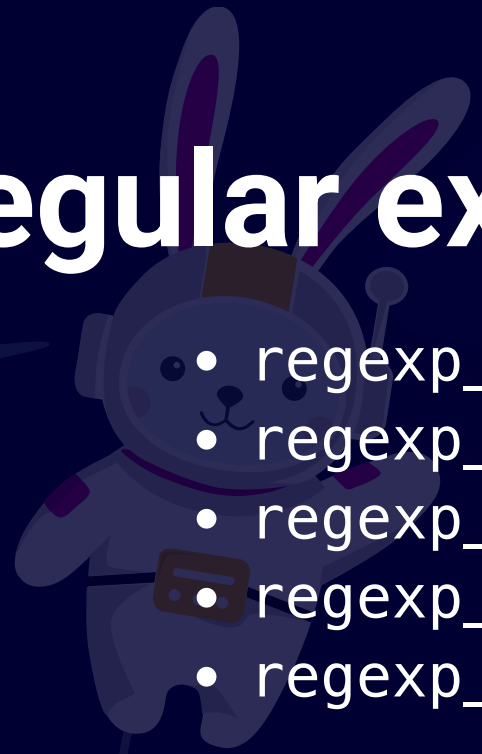
```
SELECT format('pi = %.5f', pi()); -- 3.14159

SELECT format('agent %03d', 7); -- agent 007

SELECT format('$%,.2f', 1234567.89); -- $1,234,567.89

SELECT format('%-7s,%7s', 'hello', 'world');
-- hello  ,   world

SELECT format('%2$s %3$s %1$s', 'a', 'b', 'c'); -- b c a

SELECT format('%1$tB %1$te, %1$tY', DATE '2001-07-04');
-- July 4, 2001
```

String comparison

`LIKE` operator with `_` and `%` placeholders

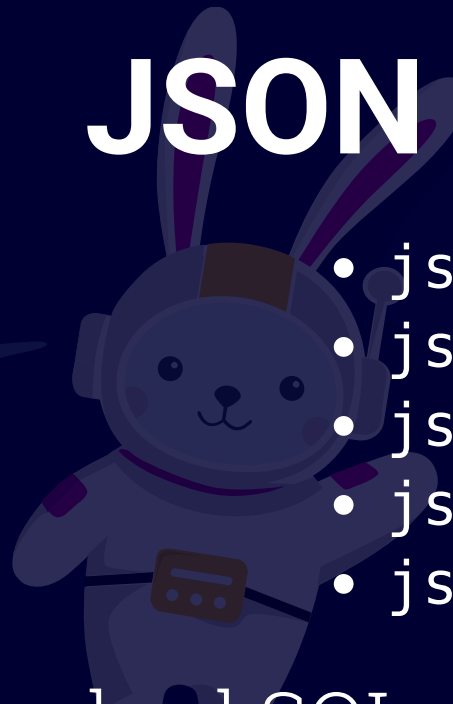
```
SELECT name FROM tpch.tiny.nation
WHERE name LIKE 'A%'
OR name LIKE '____N';
```

Regular expressions



- `regexp_count`
- `regexp_extract`
- `regexp_like`
- `regexp_position`
- `regexp_replace`

JSON functions



- `json_exists`
- `json_query`
- `json_value`
- `json_array`
- `json_object`

Standard SQL JSON path language

Date and timestamp functions

```sql
SELECT date_format(now(), '%d-%M-%Y %H'); -- MySQL

SELECT format_datetime(now(),'y-M-dd'); -- Java

SELECT human_readable_seconds(37462);

SELECT format('%1$tB %1$te, %1$tY', DATE '2023-09-07');

SELECT extract(MONTH FROM TIMESTAMP '2023-11-20 18:10:00');

SELECT month(TIMESTAMP '2023-11-20 18:10:00');
```

trino

Many other functions

```
SELECT parse_data_size('2.3MB');

SELECT contains('10.0.0.0/8', IPADDRESS '11.255.255.255');

SELECT url_extract_host('https://trino.io/docs/');

SELECT uuid(); -- new random UUID with each call
```

trino

Structural data types

Many functions for arrays, maps, and rows

```
SELECT
  ARRAY[1, 2, 3],
  ARRAY['foo', 'bar', 'bazz'],
  MAP(ARRAY['foo', 'bar'], ARRAY[1, 2]),
  ROW(1, 2.0);''
```

trino

Associated max value

Find the clerk with the most expensive order:

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

trino

Pivoting using filtering

Order counts by priority

```
SELECT priority, count(*)
FROM orders
GROUP BY priority;
```

With separate columns per priority

```
SELECT
  count(*) FILTER (WHERE priority = 'URGENT') AS urgent,
  count(*) FILTER (WHERE priority = 'HIGH') AS high,
  count(*) FILTER (WHERE priority = 'MEDIUM') AS medium,
  count(*) FILTER (WHERE priority = 'LOW') AS low
FROM orders;
```

trino

ROLLUP with single column

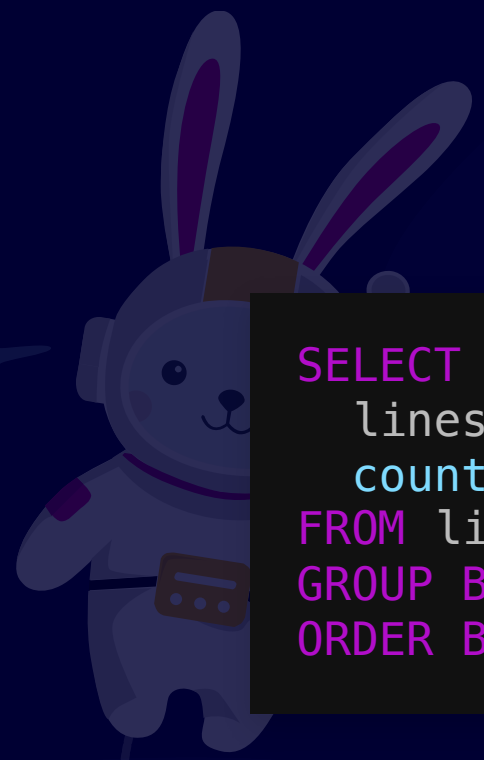
```
SELECT
  orderpriority,
  count(*) AS orders
FROM orders
GROUP BY ROLLUP (orderpriority)
ORDER BY orderpriority;
```

trino

ROLLUP with multiple columns

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

trino



CUBE

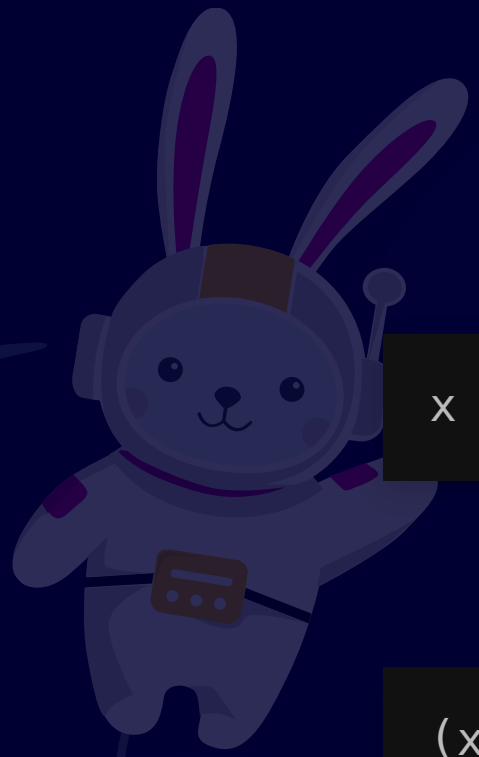
```
SELECT
  linestatus, returnflag,
  count(*) AS items
FROM lineitem
GROUP BY CUBE (linestatus, returnflag)
ORDER BY linestatus, returnflag;
```

trino

GROUPING SETS

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

trino



Lambda expressions

Lambda expression with one input:

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

Lambda expression with two inputs:

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

trino

Aggregating into an array

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

Note the `ORDER BY` clause

trino

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);
```

trino

Sorting arrays

```
SELECT array_sort(ARRAY['a', 'xyz', 'bb', 'abc', 'z', 'b']);
```

Optionally supply a comparator function

```
SELECT array_sort(ARRAY[3, 2, null, 5, null, 1, 2],
                  -- sort null first with descending order
                  (x, y) -> CASE WHEN x IS NULL THEN -1
                                 WHEN y IS NULL THEN 1
                                 WHEN x < y THEN 1
                                 WHEN x = y THEN 0
                                 ELSE -1 END);
-- [null, null, 5, 3, 2, 2, 1]
```

trino

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);
```

trino

Filtering elements

Return data if function results in `true`

```
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);
```

trino

Transforming elements

Convert each element with a function

```
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);
```

trino

Converting arrays to strings

```
SELECT array_join(sequence(1, 5), '/'); -- 1/2/3/4/5
```

trino

Unnest an array

Split it up into multiple rows

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

trino

Unnest 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);
```

trino

Creating maps

From arrays of keys and values

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

From an array of entry rows

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

trino

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);
```

trino

Unnesting a map

Split into multiple rows

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

Thank you! Questions?

More Trino resources:

- Blog: trino.io/blog
- Documentation: trino.io/docs
- Community: trino.io/community
- Join us on Slack: trino.io/slack