Presto 350 Updates and Migration Guide

This document summarizes new features and changes in Presto 350.

Performance Improvements

Here is a list of updates related to performance improvements:

- Performance improvements in querying information _schema tables (#999, #1306, #1329, #1543, #2488)
- Pushdown support for some functions and operators in PostgreSQL (DataTank) connector (#3881, #4111, #4112, #5261)
- Improved query performance by reducing worker to worker communication overhead. (#6126, #5905, #5949)
- Improved performance of `ORDER BY ... LIMIT` queries. (#6072)
- Improved performance of queries with uncorrelated `IN` clauses. (#5582)
- Improved performance of queries that use the `decimal` type. (#4730, #4886, #5181)
- Improved performance and accuracy of `approx_percentile()`. (#5158)
- Improved performance of certain cross join queries. (#5276)
- Reduced latency for queries that perform a broadcast join of a large table. (#5237)
- Improved performance of queries involving comparisons between `DOUBLE` or `REAL` values and integer values. (#3533)
- Improved performance of queries involving `row_number()`. (#3614)
- Improved performance of queries containing `LIKE` predicate. (#3618)
- Improved performance for queries that read fields from nested structures. (#2672)
- Improved performance of queries involving constant scalar subqueries (#3432)
- Improved query performance by removing redundant data reshuffling. (#2853)
- Improved performance of inequality joins involving `BETWEEN`. (#2859)
- Improved join performance for dictionary encoded data. (#2862)
- Improved performance of queries containing redundant scalar subqueries. (#2456)
- Improved performance of `INSERT` and `CREATE TABLE ... AS` queries containing redundant `ORDER BY` clauses. (#2044)
- Improved performance when processing columns of `map` type. (#2015)
- Reduced query memory usage by improving retained size estimation for `VARCHAR` and `CHAR` types. (#4123)
- Improved performance of certain join queries by reducing the amount of data that needs to be scanned. (#1673)
- Improved performance of queries containing complex predicates. (#1515)
- Improved performance of certain window functions when using bounded window frames (e.g., `ROWS BETWEEN ... PRECEDING AND ... FOLLOWING`). (#464)
- Improved performance of certain queries involving coercions and complex expressions in `JOIN` conditions. (#1390)

New UDFs

Many new UDFs are available. See the Trino official documentation to understand the usage of each functions.

- contains_sequence()
- concat_ws()
- murmur3()
- from_unixtime_nanos()
- T-Digest functions: T-Digest functions — Trino 361 Documentation
- from_iso8601_timestamp_nanos()
- human_readable_seconds()
- bitwise_left_shift(), bitwise_right_shift() and bitwise_right_shift_arithmetic()
- luhn_check()
- approx_most_frequent()
- random(m, n)
- starts_with()
- regexp_count(), regexp_position()
- strpos(string, substring, instance)
- Geospatial functions: to_encoded_polyline(), from_encoded_polyline(), line_interpolate_point(), line_interpolate_points(), geometry_from_hadoop_shape(), ST_Length(SphericalGeography)
- at_timezone(), with_timezone()

- last_day_of_month()

New SQL support

- Add IF EXISTS and IF NOT EXISTS syntax to ALTER TABLE. (#4651)
- Add support for INTERSECT ALL and EXCEPT ALL. (#2152)
- Add support for DISTINCT clause in aggregations within correlated subqueries. (#5904)
- Add support for `RANGE BETWEEN <value> PRECEDING AND <value> FOLLOWING` window frames. (#609)
- Add support for window frames based on `GROUPS`. (#5713)
- Add support for `extract()` with `TIMEZONE_HOUR` and `TIMEZONE_MINUTE` for `time with time zone` values. (#5668)
- Add support for correlated subqueries in recursive queries. (#4877)
- Add support for `IN` predicate with subqueries in outer join condition. (#4151)
- Add support for quantified comparisons (e.g., `> ALL (...)`) in aggregation queries. (#4128)
- Add support for variable-precision `TIME` type. (#4381)
- Add support for variable precision `TIME WITH TIME ZONE` type. (#4905)
- Add support for variable-precision `TIMESTAMP` (without time zone) type. (#3783)
- Add support for variable-precision `TIMESTAMP WITH TIME ZONE` type (#3947)
- Allow inserting values of a larger type into as smaller type when the values fit. For example, BIGINT into SMALLINT, or VARCHAR(10) into VARCHAR(3). Values that don't fit will cause an error at runtime. (#2061)
- Allow using `.*` on expressions of type `ROW` in the `SELECT` clause to convert the fields of a row into multiple columns. (#1017)
- Allow references to tables in the enclosing query when using .*. (#1867)
- Add support for IGNORE NULLS for window functions. (#1244)
- Add support for INNER and OUTER joins involving UNNEST. (#1522)

JDBC Improvements

Query parameters are supported in LIMIT, OFFSET and FETCH FIRST clauses.

```
Connection conn = ...
PreparedStatement stmt = conn.prepareStatement(
  "SELECT * FROM sample_datasets.www_access OFFSET ? LIMIT ?");
stmt.setInt(1, 10); // OFFSET = 10
stmt.setInt(2, 20); // LIMIT = 20
ResultSet rs = stmt.executeQuery();
...
```

time_partitioning_shuffle magic comment for INSERT/CTAS

Treasure Data supports a TD-specific setting to customize a partition size for INSERT/CTAS queries for better query performance. The following is an example of a time_partitioning_range_session property that is supported as a magic comment.

```
-- set session time_partitioning_range = '12h'
```

This comment can also be used to disable shuffling by specifying `no-shuffle` as a value. In this case, however, the partition size cannot be customized. In Presto 350, we introduced a new magic comment `time_partitioning_shuffle = 'true'|'false'` so that you can specify partition shuffling and time partition ranges independently.

Migration Guide

Use the following information to understand the changes required in your queries to use with Presto 350.

SELECT DISTINCT, ORDER BY

When SELECT DISTINCT is used with ORDER BY statement, expressions must appear in the select list.

The following query works in the current version of Presto.

```
SELECT DISTINCT from_unixtime(time) , COUNT(*), time
FROM sample_datasets.www_access
GROUP BY from_unixtime(time), time
ORDER BY from_unixtime(time)
```

This query fails on Presto 350 with an error message like `For SELECT DISTINCT, ORDER BY expressions must appear in select list`. To solve this issue on Presto 350, rewrite this query as follows.

```
SELECT DISTINCT from_unixtime(time) , COUNT(*), time
FROM sample_datasets.www_access
GROUP BY from_unixtime(time) , time
ORDER BY from_unixtime(sample_datasets.www_access.time)
```

lag() and lead()

ORDER BY

lag() and lead() require ORDER BY in Presto 350. For example, the following query works on the current version, but it doesn't on Presto 350.

```
SELECT
  time,
  LAG (path, 1) OVER () AS lag_data,
  path,
  LEAD (path, 1) OVER () AS lead_data
FROM
  sample_datasets.www_access
```

To support this query in Presto 350, you must explicitly use the ORDER BY clause as follows.

```
SELECT
  time,
  LAG (path, 1) OVER (ORDER BY time) AS lag_data,
  path,
  LEAD (path, 1) OVER (ORDER BY time) AS lead_data
FROM
  sample_datasets.www_access
```

Window Frame

Presto 350 has a new semantic check for lag() and lead(); the frame cannot be specified. For example, the following query works on the current version but doesn't produce expected results. However, this query fails on Presto 350.

```
SELECT
  time,
  user,
  LAG (path, 1) OVER (ORDER BY time ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS lag_data,
  path,
  LEAD (path, 1) OVER (ORDER BY time ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS lead_data
FROM
  sample_datasets.www_access
```

To fix this query in Presto 350, remove ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING.

```
SELECT
  time,
  user,
  LAG (path, 1) OVER (ORDER BY time) AS lag_data,
  path,
  LEAD (path, 1) OVER (ORDER BY time) AS lead_data
FROM
  sample_datasets.www_access
```

TIME and TIMESTAMP Behavior Changes

In Presto 350, TIME and TIMESTAMP behaviors follow SQL standards; it has some incompatibilities with the current version.

Presto has the following data types to represent time and timestamp:

- TIME
- TIME WITH TIME ZONE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE

In the current version, TIME and TIMESTAMP are similar to WITH TIME ZONE types, but in Presto 350, TIME and TIMESTAMP don't have an associated timezone. This might affect query results.

Examples

The result of the following query will be different depending on the Presto version:

```
SELECT CAST('2000-01-01 00:00:00 US/Eastern' AS TIMESTAMP)
--> 2000-01-01 05:00:00.000 on Presto 317
--> 2000-01-01 00:00:00.000 on Presto 350
```

When you convert WITH TIME ZONE types to TIME or TIMESTAMP, timezone is considered in the current version, but not in Presto 350. If you have queries that contain such conversion, you might need to revise them for Presto 350. In the previous example, the query can be rewritten as follows:

```
SELECT CAST( -- Convert to UTC before CAST as TIMESTAMP TIMESTAMP '2000-01-01 00:00:00 US/Eastern' AT TIME ZONE
'UTC' AS TIMESTAMP)
```

In addition, political timezones (e.g. `America/Los_Angeles`) are no longer allowed in TIME WITH TIME ZONE, to avoid issues around DST and possible future policy changes.

Therefore, the following query works on the current version, but not on Presto 350.

```
SELECT TIME '01:02:03.456 America/Los_Angeles'
```

Rewrite this query as follows:

```
SELECT TIME '01:02:03.456 -07:00'
```

Query results can be also different. `SELECT CURRENT_TIME` generates `09:29:52.540 UTC` on the current version; in Presto 350, use `09:29:52.540+00:00`.

Query Length Limitation

Presto 350 might generate a longer byte-code internally, so long queries might hit the query length limitation. If you get error messages like the following, you need to shorten the query.

- `java.lang.IllegalArgumentException: bad parameter count 256`
- `io.prestosql.spi.PrestoException: Query exceeded maximum columns`

checksum()

checksum function implementation has changed in Presto 350. The checksum() function generates a different result from previous Presto versions.

approx_percentile()

approx_percentile() now uses T-digest as an internal data structure, which is more accurate and faster than the previous version. The function produces slightly different results.

However, if you specify the `accuracy` parameter, approx_percentile() doesn't use T-digest because T-digest doesn't allow the `accuracy` parameter. If you want to benefit from T-digest-based approx_percentile(), consider dropping the `accuracy` parameter.

approx_percentile() doesn't allow infinite values (values divide by zero). If such a value is given, the query fails with `java.lang.IllegalArgumentException: value must be finite`. In this case, you have to exclude infinite values before approx_percentile(), or you can use the old version of approx_percentile() by specifying `accuracy` parameter intentionally as a workaround as follows:

```
-- Before
SELECT approx_percentile(
  column1 / column2, -- can be inifinite value
  0.5                -- percentile
) FROM ...

-- After
SELECT approx_percentile(
  column1 / column2, -- can be infinite value
  1,                 -- weight
  0.5,               -- percentile
  0.01                -- accuracy
) FROM ...
```

double NaN to integer

Presto 350 doesn't allow convert NaN value to INTEGER, however, it can be converted to 0 by CAST in the current version. The following query works on the current version but fails on Presto 350 with `Cannot cast double NaN to integer` error message.

```
SELECT CAST(nan() AS INTEGER)
```

You can restore the original behavior by using TRY_CAST and COALESCE as follows:

```
SELECT COALESCE(TRY_CAST(nan() AS INTEGER), 0)
```

information_schema.columns

`comment` and `extra_info` columns have been removed from `information_schema.columns`. If you have queries that refer to these columns, you have to revise them.

Here is a query result of `information_schema.columns` on Presto 350.

```
presto> select * from information_schema.columns limit 10;
 table_catalog |    table_schema    |    table_name    |   column_name   | ordinal_position | column_default |
is_nullable | data_type
---------------+--------------------+------------------+-----------------+------------------+----------------
+-------------+-----------
 td-presto     | information_schema | tables           | table_catalog   |                1 | NULL           |
YES         | varchar
 td-presto     | information_schema | tables           | table_schema    |                2 | NULL           |
YES         | varchar
...
```