April 12, 2023

CrateDB

CrateDB Office Hour #3

Welcome 

- Opportunity to talk to CrateDB engineers ☺
- Open Zoom call
- No recording!
- As a starting point, we will show some cool features
- Ask questions, share your use case

Agenda

1. **Window functions in CrateDB**
2. **User defined functions**
3. **Geospatial data and search in CrateDB**
4. **Your questions**

Window functions

- Window functions operate on a subset of rows defined by the window specification
- Help perform calculations that depend on the rows' order or involve comparisons with other rows in the same result set
- To declare window function, use the OVER clause to **define the window**

```
SELECT col1, col2, COUNT(*) OVER(PARTITION BY col1)
FROM table1
ORDER BY col2;
```

Computation window as a partition by col1

Window functions: example

```
SELECT first_name,
       last_name,
       designation,
       COUNT(*) OVER(PARTITION BY designation)
FROM demo.employees
ORDER BY last_name
LIMIT 100;
```

Result from query

OVER(PARTITION BY designation)

first_name	last_name	designation	count(*) OVER (PARTITION BY designation)
AYANNA	AABERG	Team Lead	8601
GERRY	AABY	Software Engineer	60222
CRISTEN	AADLAND	Trainee	8601
LEN	AADLAND	Software Engineer	60222
DREW	AADLAND	Software Engineer	60222
CARMAN	AADLAND	Software Engineer	60222
JERALD	AAGAARD	Software Engineer	60222
LATINA	AAGAARD	Software Engineer	60222
SILVIA	AAGAARD	Software Engineer	60222
LONG	AALBERS	Software Engineer	60222
OSVALDO	AALBERS	Software Engineer	60222
ROB	AALUND	Senior Software Engineer	17203
MARJORY	AAMODT	Software Engineer	60222

Window = Team Leads

Window = Trainees

Window = Software Engineers

Window = Senior Software Engineers

- Count the number of employees for each designation value (e.g., trainee, team lead, software engineer)
- Each designation is a window on which count will be calculated

WINDOW clause

- Defining window using WINDOW clause can simplify the code and allow you to reuse window specification across multiple expressions
- The ORDER and PARTITION define what is referred to as the **window**—the ordered subset of data over which calculations are made

```
SELECT first_name,
       last_name,
       designation,
       salary,
       MAX(salary) OVER w max_salary
FROM demo.employees
WINDOW w AS (PARTITION BY designation ORDER BY salary DESC)
ORDER BY last_name
LIMIT 100;
```

- For each employee, show the name, designation, employee's salary and the *max salary for her/his designation group

* The list of aggreagate and general-purpose functions on the defined window: https://community.crate.io/t/window-functions-in-cratedb/1398

User-Defined Functions (UDFs)

- Allows you to **expand** the capabilities of **CrateDB** with **JavaScript** code, creating reusable functions.
- For example, get_postal_code function can be used to extract postal code from the address
- A dedicated scalar function: regexp_matches

And then, use it as follows:

```
SELECT get_postal_code(address) FROM demo.employees
```

```
CREATE FUNCTION get_postal_code(address STRING)
RETURNS STRING
LANGUAGE JAVASCRIPT
AS '
  function get_postal_code(address) {
    const postalCodeRegex = /\d{5}$/;
    const match = address.match(postalCodeRegex);
    if (match) {
      return match[0];
    } else {
      return null;
    }
  }
';
```

Check other UDF examples here
https://community.crate.io/t/user-defined-function-collection/773
https://community.crate.io/t/advanced-downsampling-with-the-lttb-algorithm/1287

Working with Geospatial data

- GEO_POINT: geographic data type used to store latitude and longitude coordinates.
- GEO_SHAPE: geographic data type used to store 2D shapes defined as GeoJSON geometry objects.
- GEO_SHAPE can store different types of GeoJSON geometry objects:

```
INSERT INTO my_table_geo (id, area) VALUES
    (1, 'POINT (9.7417 47.4108)'),
    (2, 'MULTIPOINT (47.4108 9.7417, 9.7483 47.4106)'),
    (3, 'LINESTRING (47.4108 9.7417, 9.7483 47.4106)'),
    (4, 'MULTILINESTRING ((47.4108 9.7417, 9.7483 47.4106), (52.50463 13.46738, 52.51000 13.47000))'),
    (5, 'POLYGON ((47.4108 9.7417, 9.7483 47.4106, 9.7426 47.4142, 47.4108 9.7417))'),
    (6, 'MULTIPOLYGON (((5 5, 10 5, 10 10, 5 5)), ((6 6, 10 5, 10 10, 6 6)))'),
    (7, 'GEOMETRYCOLLECTION (POINT (9.7417 47.4108), MULTIPOINT (47.4108 9.7417, 9.7483 47.4106))')
```

Geospatial data: Functions

- distance (geo_point1, geo_point2) – calculate the distance between two points on Earth
- intersection (geo_shape1, geo_shape2) – returns true if both shapes share some points or area
- within (shape1, shape2) – returns true if shape1 is within shape 2
- geohash (geo_point) – returns a GeoHash representation based on full precision
- area (geo_shape) – calculates the area of the input shape in square-degrees

For each city, find distance from north pole:

```
SELECT city, distance(location, 'POINT(0.0 90.0)') AS dist_north_pole
FROM "demo"."world_cities"
LIMIT 100;
```

GEO_SHAPE

- GEO_SHAPE is represented as an object containing a valid GeoJSON geometry object:
- Alternatively, a ‘well-known text representation’ string can be used to represent GEO_SHAPE:

```
'POLYGON ((5 5, 10 5, 10 10, 5 10, 5 5))'
```

```
{
    type = 'Polygon',
    coordinates = [
        [
            [100.0, 0.0],
            [101.0, 0.0],
            [101.0, 1.0],
            [100.0, 1.0],
            [100.0, 0.0]
        ]
    ]
}
```

Find if two shapes intersect:

```
SELECT name, intersects(boundary,
       'POLYGON((2.348324 48.849304, 2.360516 48.858374, 2.337510 48.864306, 2.348324 48.849304))')
FROM demo.areas;
```

Your questions

Contact: marija@crate.io

CrateDB

Thank you