ClickHouse and The One Billion Row Challenge

Derek Chia, Senior Support Engineer @ ClickHouse

ClickHouse

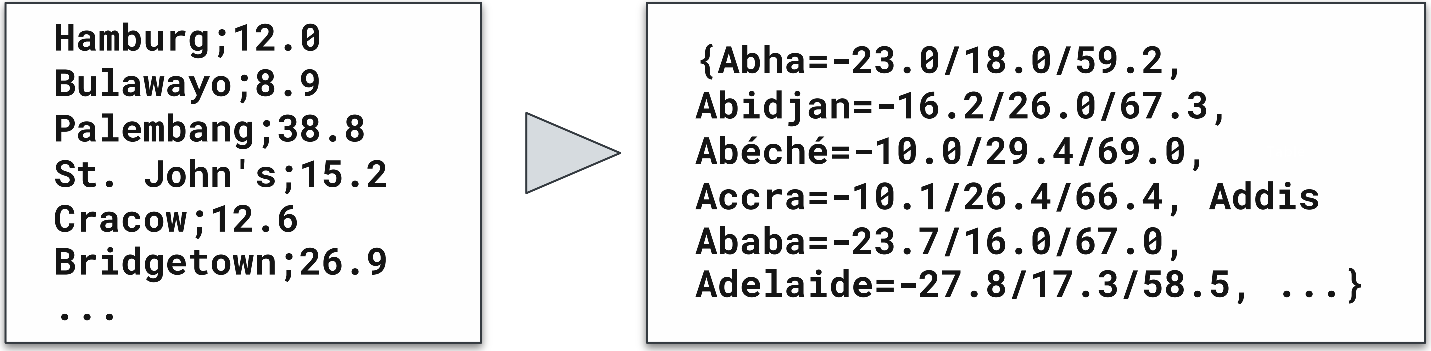
https://clickhouse.com/blog/clickhouse-one-billion-row-challenge

ClickHouse and The One Billion Row Challenge

1️⃣🐝🏎️ - https://github.com/gunnarmorling/1brc

The original task

- Write a *Java program* which reads the file, calculates the **min**, **mean**, and **max** temperature value per weather station, and emits the results on stdout like this:
    - **sorted** alphabetically by **station name**, and the result values per station
    - in the format <min>/<mean>/<max>, **rounded** to one fractional digit



```
Hamburg;12.0
Bulawayo;8.9
Palembang;38.8
St. John's;15.2
Cracow;12.6
Bridgetown;26.9
...
```

<Station name>;<average temperature>

```
{Abha=-23.0/18.0/59.2,
Abidjan=-16.2/26.0/67.3,
Abéché=-10.0/29.4/69.0,
Accra=-10.1/26.4/66.4, Addis
Ababa=-23.7/16.0/67.0,
Adelaide=-27.8/17.3/58.5, ...}
```

{<Station name>;<min>/<mean>/<max>, ...}

ClickHouse and The One Billion Row Challenge

Generating dataset using Java and Python

- The repo has a tool (create_measurements.{sh,py}) to generate **one billion random points** by sampling a **Gaussian distribution with a mean and variance of 10** using a list of (413) distinct stations and their average temperatures

```
Hamburg;12.0
Bulawayo;8.9
Palembang;38.8
St. John's;15.2
Cracow;12.6
Bridgetown;26.9
...
```

<Station name>; <average temp.>

```
# clone and build generation tool. Output omitted.
git clone git@github.com:gunnarmorling/1brc.git
./mvnw clean verify
./create_measurements.sh 1000000000

Created file with 1,000,000,000 measurements in 435900 ms
```

Java - *435900 ms = 435.900 s = 7.27 mins (too slow!)*

```
python3 create_measurements.py 1_000_000_000
Estimated max file size is:  14.8 GiB.
Building test data...
[==================================================] 100%
Test data successfully written to 1brc/data/measurements.txt
Actual file size:  14.8 GiB
Elapsed time: 13 minutes 36 seconds
Test data build complete.
```

Python - *~13 mins (wayyy too slow!)*

```
❯ head -n 3 data/measurements.txt
Monatélé;65.2
Glendora;-13.0
Gundumāl;-67.2

❯ wc -l data/measurements.txt
 1000000000 data/measurements.txt

❯ ls -lh data/measurements.txt |
awk '{print $5, $9}'
15G data/measurements.txt
```

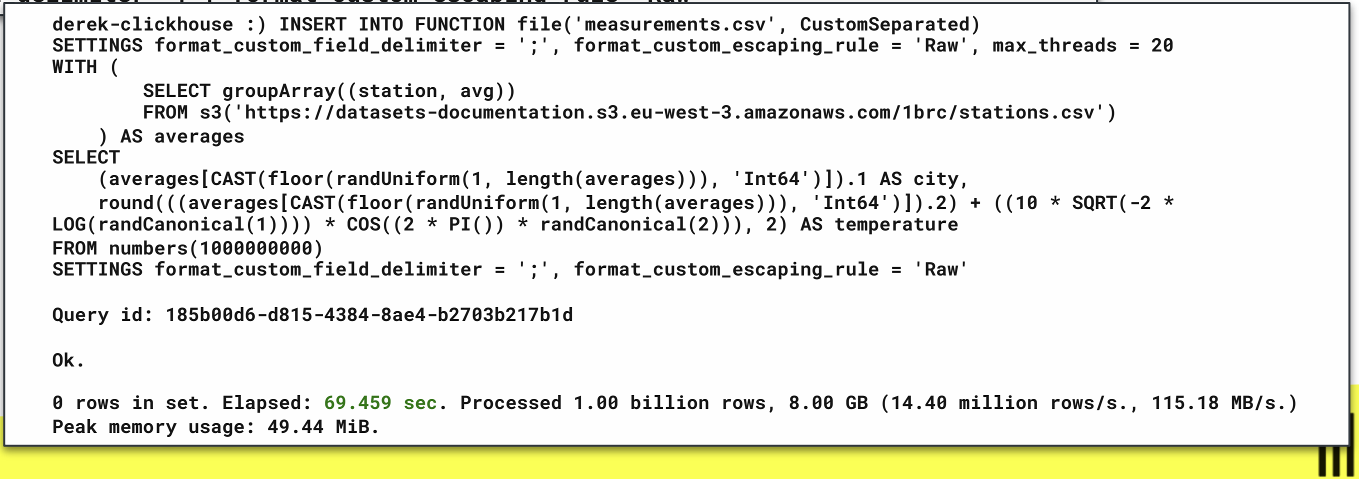
measurements.txt with 1B rows, taking up approx. 15 GB

ClickHouse and The One Billion Row Challenge

Generating dataset using ClickHouse

```sql
INSERT INTO FUNCTION file('measurements.csv', CustomSeparated)
WITH (
      SELECT groupArray((station, avg))
      FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/1brc/stations.csv')
) AS averages
SELECT
             averages[floor(randUniform(1, length(averages)))::Int64].1 as city,
             round(averages[floor(randUniform(1, length(averages)))::Int64].2 +
      (10 * SQRT(-2 * LOG(randCanonical(1))) * COS(2 * PI() * randCanonical(2))), 2) as temperature
FROM numbers(1_000_000_000)
SETTINGS format_custom_field_delimiter=';', format_custom_escaping_rule='Raw'
```

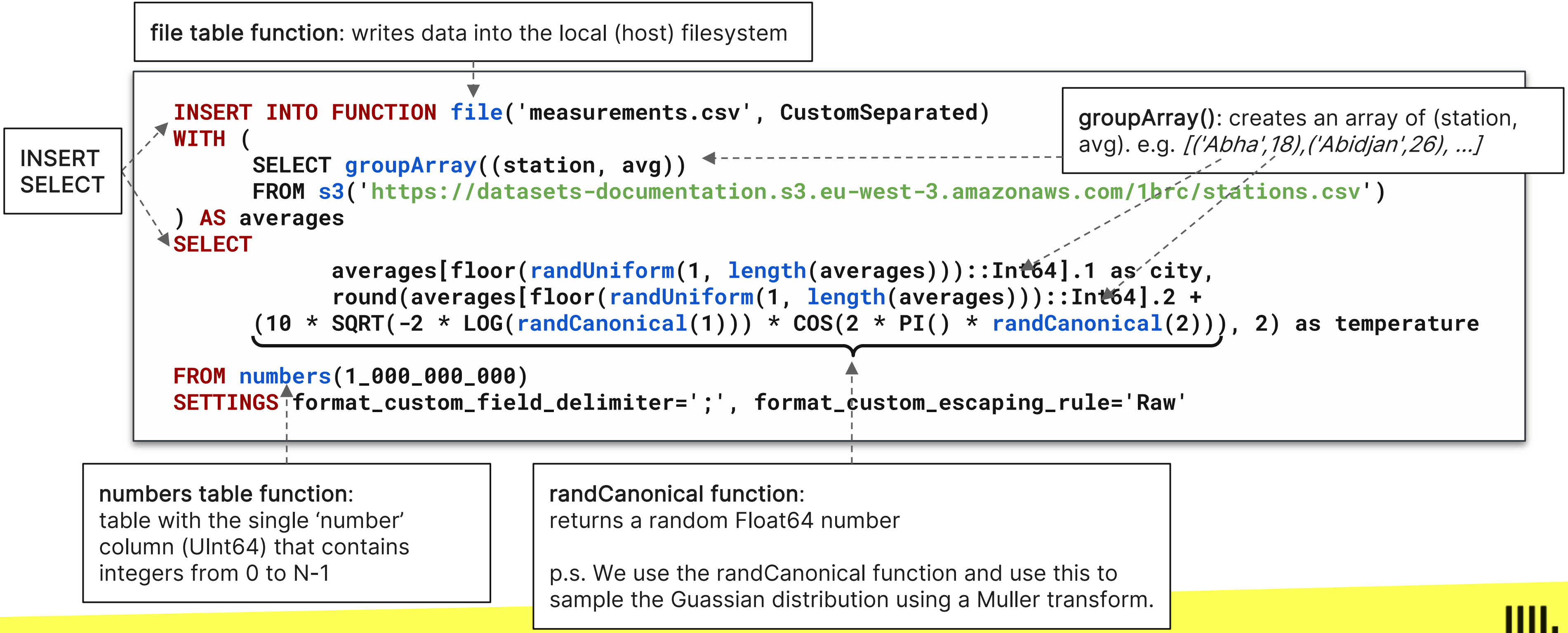
ClickHouse - *~1 min (fast!)*



```
derek-clickhouse :) INSERT INTO FUNCTION file('measurements.csv', CustomSeparated)
SETTINGS format_custom_field_delimiter = ';', format_custom_escaping_rule = 'Raw', max_threads = 20
WITH (
        SELECT groupArray((station, avg))
        FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/1brc/stations.csv')
    ) AS averages
SELECT
    (averages[CAST(floor(randUniform(1, length(averages))), 'Int64')]).1 AS city,
    round(((averages[CAST(floor(randUniform(1, length(averages))), 'Int64')]).2) + ((10 * SQRT(-2 *
LOG(randCanonical(1)))) * COS((2 * PI()) * randCanonical(2))), 2) AS temperature
FROM numbers(1000000000)
SETTINGS format_custom_field_delimiter = ';', format_custom_escaping_rule = 'Raw'

Query id: 185b00d6-d815-4384-8ae4-b2703b217b1d

Ok.

0 rows in set. Elapsed: 69.459 sec. Processed 1.00 billion rows, 8.00 GB (14.40 million rows/s., 115.18 MB/s.)
Peak memory usage: 49.44 MiB.
```

ClickHouse and The One Billion Row Challenge

Generating dataset using ClickHouse, explained



file table function: writes data into the local (host) filesystem

INSERT SELECT

```
INSERT INTO FUNCTION file('measurements.csv', CustomSeparated)
WITH (
      SELECT groupArray((station, avg))
      FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/1brc/stations.csv')
) AS averages
SELECT
            averages[floor(randUniform(1, length(averages)))::Int64].1 as city,
            round(averages[floor(randUniform(1, length(averages)))::Int64].2 +
      (10 * SQRT(-2 * LOG(randCanonical(1))) * COS(2 * PI() * randCanonical(2))), 2) as temperature

FROM numbers(1_000_000_000)
SETTINGS format_custom_field_delimiter=';', format_custom_escaping_rule='Raw'
```

groupArray(): creates an array of (station, avg). e.g. *[('Abha',18),('Abidjan',26), …]*

numbers table function:
table with the single 'number' column (UInt64) that contains integers from 0 to N-1

randCanonical function:
returns a random Float64 number

p.s. We use the randCanonical function and use this to sample the Guassian distribution using a Muller transform.

ClickHouse and The One Billion Row Challenge

The challenge, baseline using Java and ClickHouse

```
time ./calculate_average_baseline.sh
real            4m41.360s
user            4m38.427s
sys             0m4.728s
```

Java: *4m 41.360s (slow!)*

```
SELECT format('{}={}/{}/{}', city, min(temperature), round(avg(temperature), 2), max(temperature))
FROM file('measurements.csv', CSV, 'city String, temperature DECIMAL(8,1)')
GROUP BY city
ORDER BY city ASC
FORMAT CustomSeparated
SETTINGS
  format_custom_result_before_delimiter = '{',
  format_custom_result_after_delimiter = '}',
  format_custom_row_between_delimiter = ', ',
  format_custom_row_after_delimiter = '',
  format_csv_delimiter = ';';

412 rows in set. Elapsed: 23.827 sec. Processed 1.00 billion rows, 14.68 GB (41.97 million rows/s.,
616.08 MB/s.)
Peak memory usage: 183.29 MiB.
```

Reading from CSV performs complete linear scan of the file. This is inefficient

ClickHouse local: 23.827s *(faster, but can be better)*

ClickHouse and The One Billion Row Challenge

The challenge, optimizing ClickHouse

- Using **Materialized View**, which acts as an insert trigger, we can compute the statistics during INSERT time
- This means that we shift the computation from SELECT time to INSERT time

1

```
CREATE TABLE weather (
    `city` String,
    `temperature` Decimal(8, 1)
)
ENGINE = Null;

CREATE TABLE weather_results (
    city String,
    max AggregateFunction(max, Decimal(8, 1)),
    min AggregateFunction(min, Decimal(8, 1)),
    avg AggregateFunction(avg, Decimal(8, 1))
)
ENGINE = AggregatingMergeTree
ORDER BY tuple();
```

2

```
CREATE MATERIALIZED VIEW weather_mv TO weather_results
AS SELECT
        city,
        maxState(temperature) as max,
        minState(temperature) as min,
        avgState(temperature) as avg
FROM weather GROUP BY city;
```

3

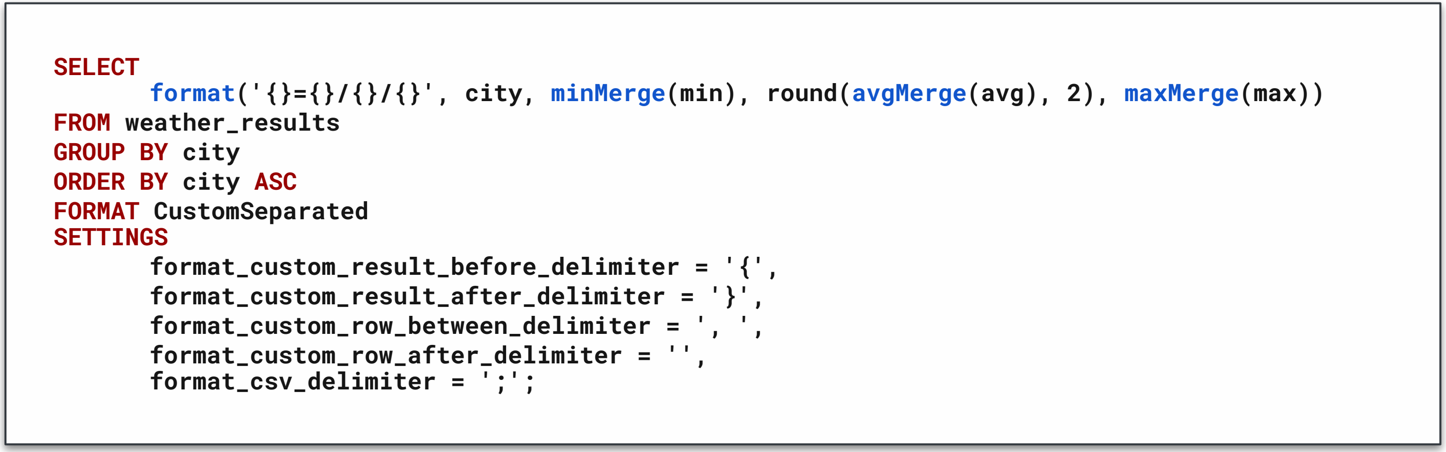
```
INSERT INTO weather
SELECT city, temperature
FROM (
    SELECT
        splitByChar(';', line) AS vals,
        vals[1] AS city,
        CAST(vals[2], 'Decimal(8, 1)') AS temperature
    FROM file('measurements.csv', LineAsString));
```

```
0 rows in set. Elapsed: 24.398 sec. Processed 2.00 billion
rows, 35.64 GB (81.97 million rows/s., 1.46 GB/s.)
Peak memory usage: 242.30 MiB.
```

ClickHouse and The One Billion Row Challenge

The challenge, optimizing ClickHouse

- How fast would it be?

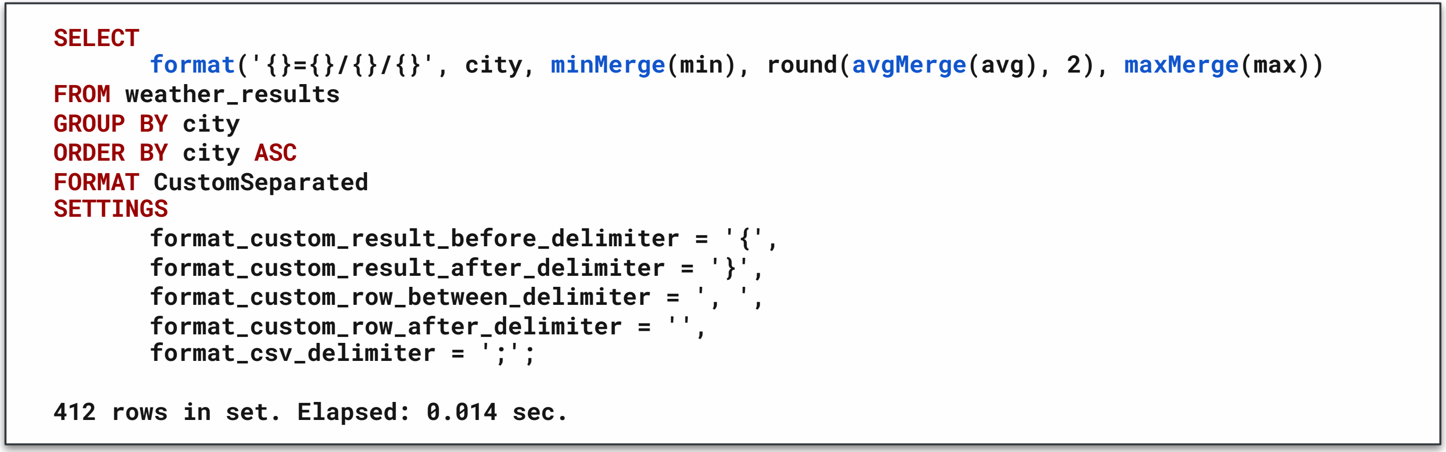


```
SELECT
       format('{}={}/{}/{}', city, minMerge(min), round(avgMerge(avg), 2), maxMerge(max))
FROM weather_results
GROUP BY city
ORDER BY city ASC
FORMAT CustomSeparated
SETTINGS
       format_custom_result_before_delimiter = '{',
       format_custom_result_after_delimiter = '}',
       format_custom_row_between_delimiter = ', ',
       format_custom_row_after_delimiter = '',
       format_csv_delimiter = ';';
```

ClickHouse and The One Billion Row Challenge

The challenge, optimizing ClickHouse

- Using **Materialized View**, query duration lowered from **23s** to **0.014s (~1600 times** speedup)



```
SELECT
        format('{}={}/{}/{}', city, minMerge(min), round(avgMerge(avg), 2), maxMerge(max))
FROM weather_results
GROUP BY city
ORDER BY city ASC
FORMAT CustomSeparated
SETTINGS
        format_custom_result_before_delimiter = '{',
        format_custom_result_after_delimiter = '}',
        format_custom_row_between_delimiter = ', ',
        format_custom_row_after_delimiter = '',
        format_csv_delimiter = ';';

412 rows in set. Elapsed: 0.014 sec.
```

Thank you!

Keep in touch!

clickhouse.com/slack

#clickhouseDB @clickhouseinc

clickhouse