



Fun with **ClickHouse** Window Functions

Robert Hodges and Vitaliy Zakaznikov @ **Altinity**

Presenter Bios and Altinity Introduction

Robert Hodges - CEO

30+ years on DBMS plus
virtualization and security.
ClickHouse is DBMS #20

Vitaliy Zakaznikov - QA Manager

13+ years testing hardware and
software; author of TestFlows
open source testing framework



Altinity

The #1 enterprise ClickHouse provider. Now offering [Altinity.Cloud](#)

Major committer and community sponsor for ClickHouse in US/EU

ClickHouse: a great SQL data warehouse

Understands SQL

Runs on bare metal to cloud

Shared nothing architecture

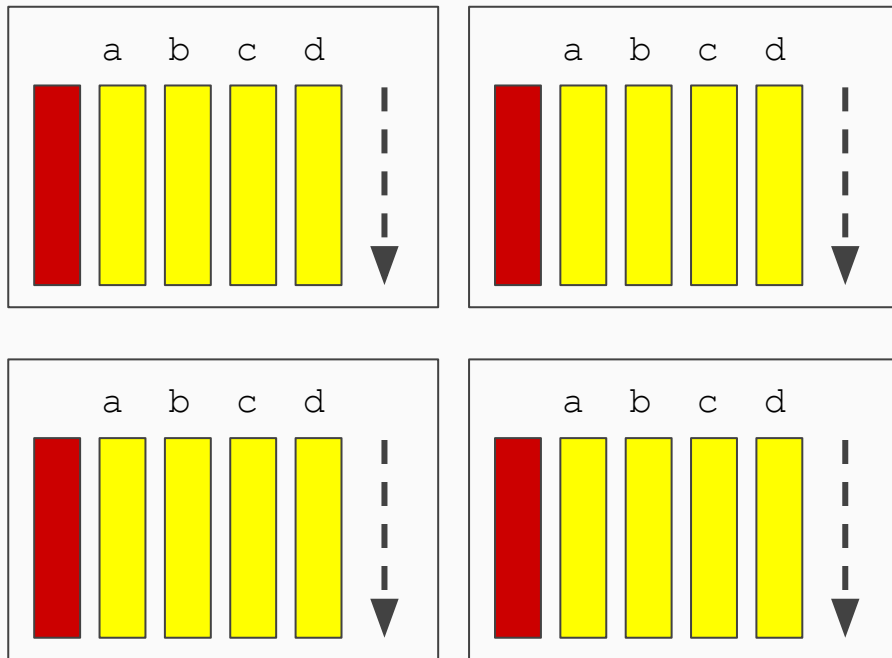
Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

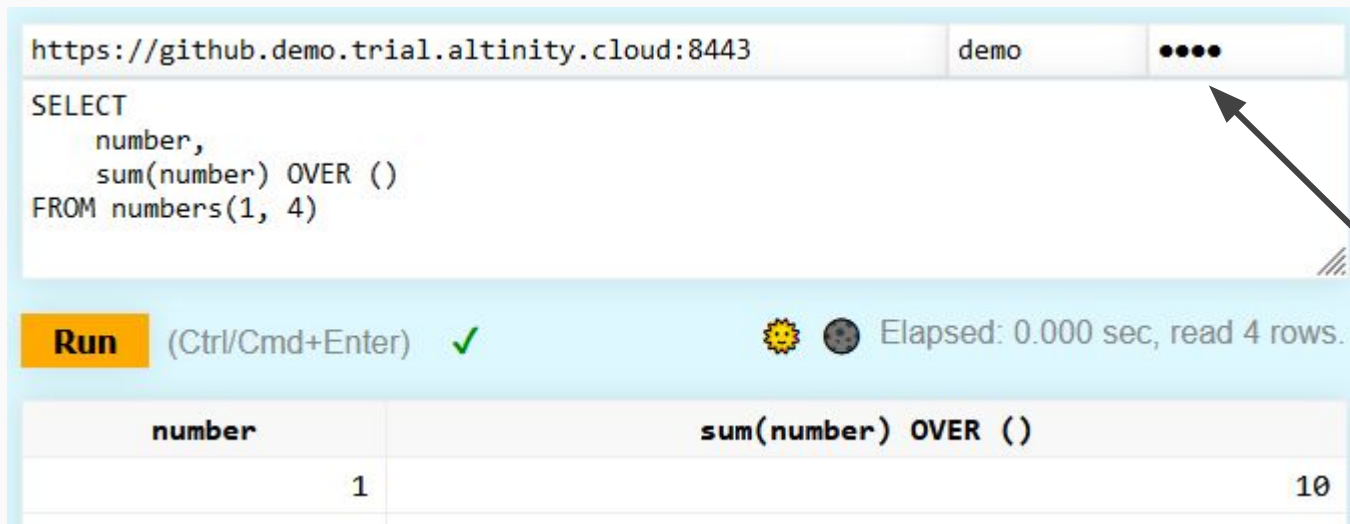
Is Open source (Apache 2.0)

And it's really fast!



Using the Altinity.Cloud public endpoint

<https://github.demo.trial.altinity.cloud:8443/play>



The screenshot shows a web-based SQL client interface. At the top, the URL is `https://github.demo.trial.altinity.cloud:8443` and the user is `demo`. The password field is masked with four dots. The SQL query entered is:

```
SELECT
  number,
  sum(number) OVER ()
FROM numbers(1, 4)
```

The query was executed successfully, as indicated by the **Run** button and the status `Elapsed: 0.000 sec, read 4 rows.`. The results are displayed in a table:

number	sum(number) OVER ()
1	10

User
"demo"

Password
"demo"

```
clickhouse-client --host=github.demo.trial.altinity.cloud  
-s --user=demo --password
```

What are Window Functions?

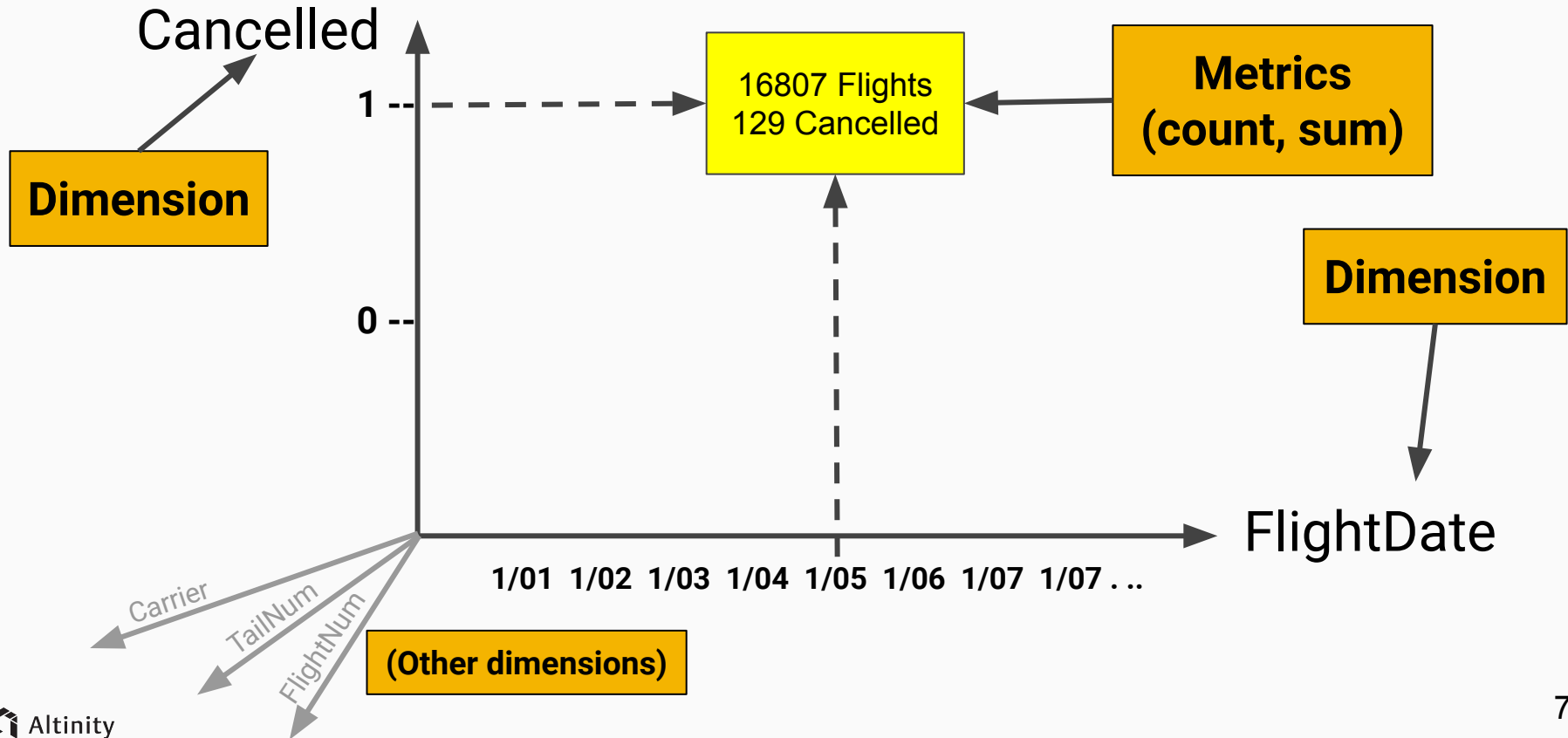
Let's start with a simple query...

```
SELECT FlightDate,  
       count() AS Flights,  
       sum(Cancelled) AS Sum_Cancelled  
FROM ontime  
WHERE toYYYYMM(FlightDate) = 201901  
GROUP BY FlightDate  
ORDER BY FlightDate
```

**Cancelled flights for
Jan 2019**

FlightDate	Flights	Sum_Cancelled
2019-01-01	18009	141
2019-01-02	20384	173

SQL queries work like “cubes”



...But what happens if we want to...

Rank particular days by number of cancelled flights?

Print cumulative cancellations for each month?

Print trailing 7-day average cancellations?



How can I do that in SQL??

This is a job for window functions!

But first we need to enable them...

Set session variable



```
clickhouse101 :) SET allow_experimental_window_functions = 1
SET allow_experimental_window_functions = 1
Query id: f8aec38c-7f31-4544-96df-bcdb4034f0ac
Ok.
```

```
<yandex>
  <profiles>
    <default>
      <allow_experimental_window_functions>1</allow_...tions>
      .
      .
    </default></profiles></yandex>
```

Set in user profile



Window functions add a new option to SQL

```
SELECT FlightDate, count() AS Flights,  
sum(Cancelled) AS Daily_Cancelled,  
avg(Daily_Cancelled)  
  OVER (ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)  
  AS Avg_Cancelled_7  
FROM ontime_ref  
WHERE Year = 2019 GROUP BY FlightDate ORDER BY FlightDate
```

Window function!

```
FlightDate|Flights|Daily_Cancelled|Avg_Cancelled_7  |  
-----|-----|-----|-----|  
. . .  
2019-01-30|  19102|          2145|  805.5714285714286|  
2019-01-31|  19962|          1775|                999.0|  
2019-02-01|  20045|           459|1037.2857142857142|
```

How window functions work conceptually

```
SELECT FlightDate,  
count() AS Flights,  
sum(Cancelled) AS Daily_Cancelled,  
avg(Daily_Cancelled)  
  OVER (ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)  
  AS Avg_Cancelled_7  
FROM ontime_ref  
WHERE Year = 2019  
GROUP BY FlightDate  
ORDER BY FlightDate
```

**Operates on the
computed aggregate**

**Computes average
within a window
“frame” of 7 rows**

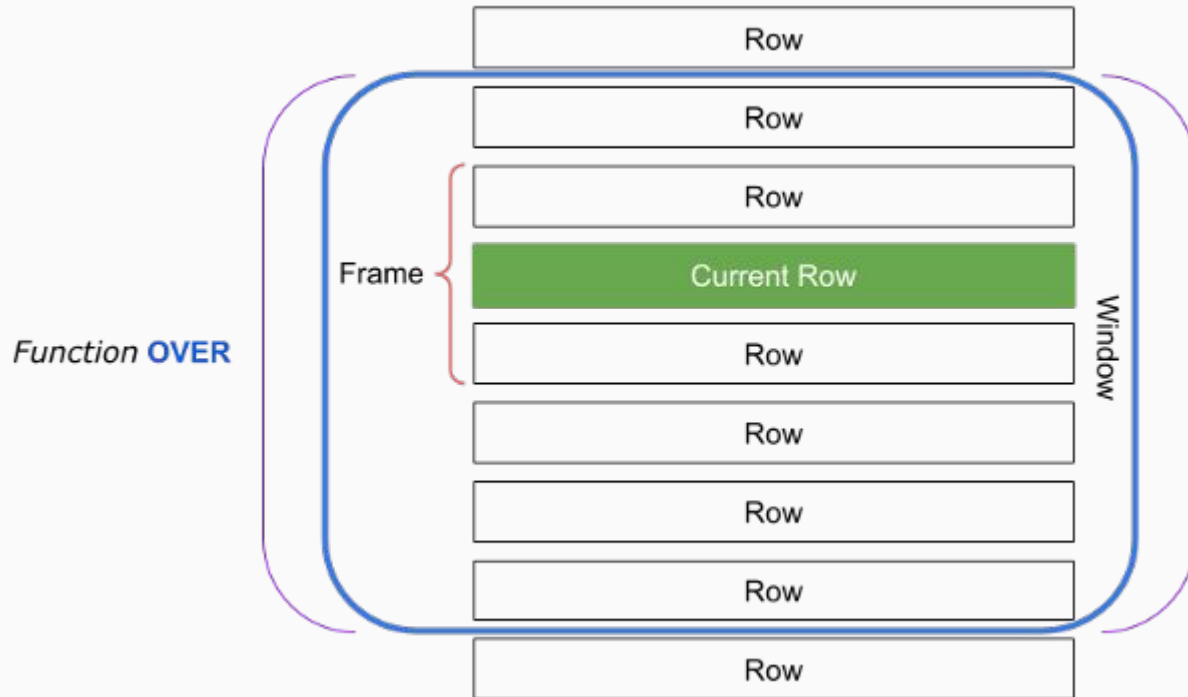
**Result is another
output column**

Window Functions -- The gory details

21.3 (LTS) - First experimental support

21.8 - Pre-release experimental feature (should be enabled by default soon)

How do window functions work for users?



Why do we need “gory details” anyway?

Window function behavior is not obvious!

```
SELECT
  number,
  sum(number) OVER ()
FROM numbers(1, 5)
```

number	sum(number) OVER ()
1	15
2	15
3	15
4	15
5	15

- Empty OVER clause means that there is **only one window** that includes **all the result rows**
- When no ORDER BY clause is specified then all rows are the **peers of the current row**
- The default frame is **RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**

What can be a window function?

Any aggregate function

- min
- max
- sum
- avg
- etc.

```
SELECT number, min(number)  
OVER () FROM numbers(1,5)
```

Window native function

- row_number
- first_value
- last_value
- rank
- dense_rank
- leadInFrame
- lagInFrame

```
SELECT number, rank() OVER (ORDER BY  
number) FROM numbers(1,5)
```

What is an OVER clause?

OVER defines the window specification

- Can be empty

```
SELECT number,  
       sum(number) OVER ()  
FROM numbers(1,5)
```

- Can contain window specification

```
SELECT number,  
       sum(number) OVER (PARTITION BY number)  
FROM numbers(1,5)
```

- Can refer to a named window

```
SELECT number,  
       sum(number) OVER w  
FROM numbers(1,5)  
WINDOW w AS (PARTITION BY number)
```


What do window specifications look like?

Window Specification clause

```
[partition_clause] [order_clause] [frame_clause]
```

- *PARTITION BY* clause
Defines window partition

```
SELECT number,  
       sum(number) OVER (PARTITION BY number % 2)  
FROM numbers(1,5)
```

- *ORDER BY* clause
Orders rows within a frame

```
SELECT number,  
       sum(number) OVER (ORDER BY number)  
FROM numbers(1,5)
```

- *FRAME* clause
Defines frame within a window partition

```
SELECT number,  
       sum(number) OVER (ROWS BETWEEN  
UNBOUNDED PRECEDING AND CURRENT ROW)  
FROM numbers(1,5)
```

What kind of frames are there?

FRAME clause

- *ROWS* frame
Defines a frame with the range in terms of relationship of rows to the current row number

```
SELECT
  number,
  sum(number) OVER (ORDER BY
number ROWS 1 PRECEDING) AS sum
FROM numbers(1, 3)
```

number	sum
1	1
2	3
3	5

- *RANGE* frame
Defines a frame with the range in terms of row values from the current row value.

```
SELECT
  number,
  sum(number) OVER (ORDER BY number RANGE
1 PRECEDING) AS sum
FROM values('number Int8', 1, 2, 2, 4)
```

number	sum
1	1
2	5
2	5
4	4

What are current rows peers?

CURRENT ROW Peers

Are rows that fall into the same sort bucket and applies only to **RANGE** frame.

- **No** ORDER BY clause

```
SELECT
  number,
  sum(number) OVER () AS sum
FROM values('number Int8', 1, 2, 2, 3,
4, 5)
```

number	sum
1	17
2	17
2	17
3	17
4	17
5	17

- **With** ORDER BY clause

```
SELECT
  number,
  sum(number) OVER (ORDER BY number)
AS sum
FROM values('number Int8', 1, 2, 2, 3,
4, 5)
```

number	sum
1	1
2	5
2	5
3	8
4	12
5	17

How do we define the extent of the frame?

FRAME extent clause

- **frame START**

Defines start of the frame with the end being set implicitly to current row (for both ROWS and RANGE frame)

```
SELECT number, sum(number) OVER
  (ORDER BY number ROWS 1
  PRECEDING) AS sum FROM
  numbers(1, 5)
```

is actually

```
SELECT
  number,
  sum(number) OVER (ORDER BY
  number ASC ROWS BETWEEN 1
  PRECEDING AND CURRENT ROW) AS sum
FROM numbers(1, 5)
```

- **frame BETWEEN**

Defines a frame with start and end specified explicitly

```
SELECT number, sum(number) OVER (ORDER BY
  number ROWS BETWEEN 1 PRECEDING AND 1
  FOLLOWING) AS sum FROM numbers(1, 5)
```

is actually the same

```
SELECT
  number,
  sum(number) OVER (ORDER BY number ASC
  ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
  AS sum
FROM numbers(1, 5)
```

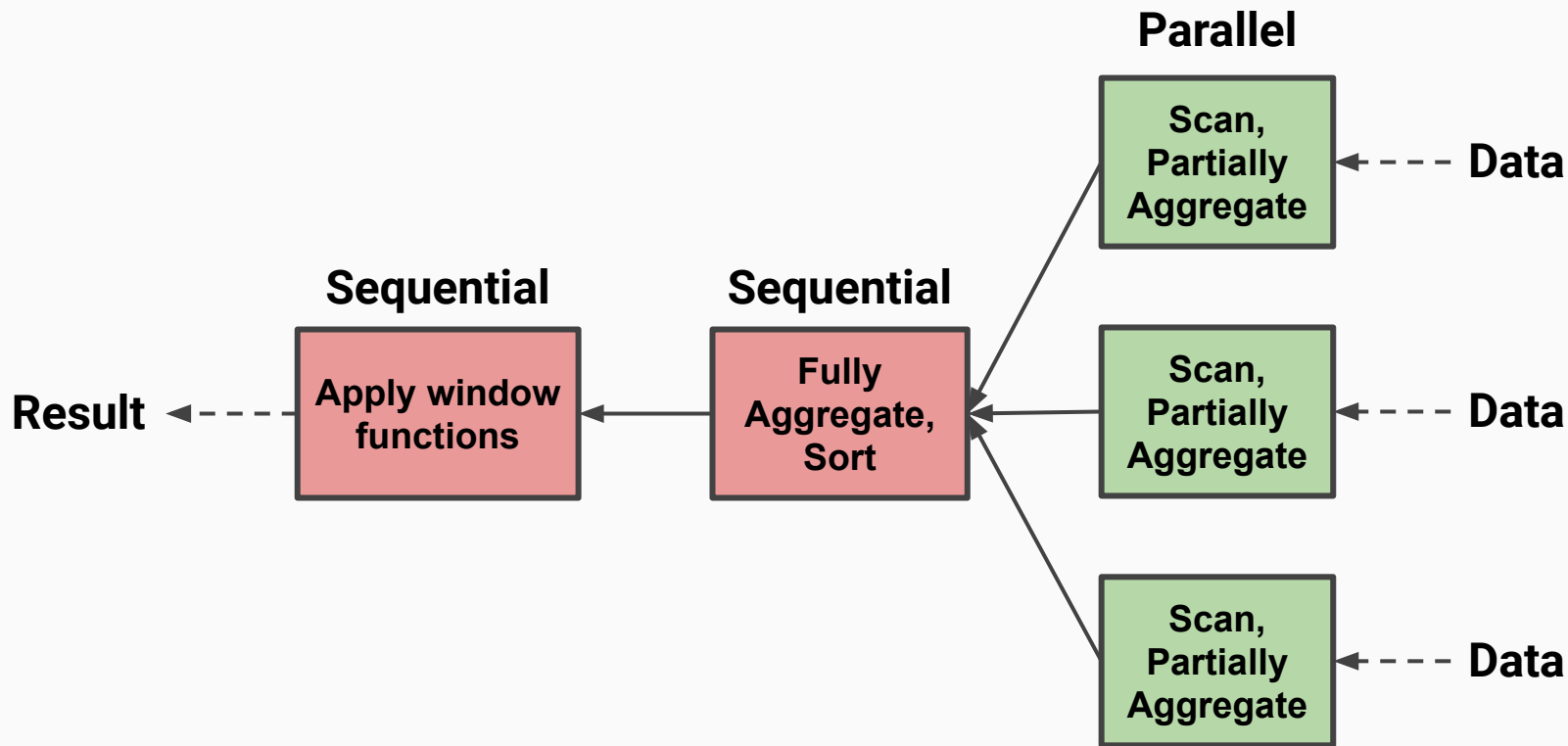
More on frame extents!

FRAME extent clause

Frame **START** and frame **END** offsets can be specified as

- **CURRENT ROW**
Current row as the frame slides through the window
- **UNBOUNDED PRECEDING**
All rows before current row, if **ROWS** frame,
or first row's value in window partition, if **RANGE**
frame
- **UNBOUNDED FOLLOWING**
All rows after current row, if **ROWS** frame,
or last row's value in window partition if **RANGE** frame
- **expr PRECEDING**
Offset in rows before current row, if **ROWS** frame,
or current row value minus **expr**, if **RANGE** frame
- **expr FOLLOWING**
Offset in rows before current row, if **ROWS** frame,
or current row value plus **expr**, if **RANGE** frame

How do window functions work internally?



Using Window functions in practice

21.3 (LTS) - First experimental
support

21.8 - Pre-release experimental
feature

Computing cumulative monthly cancellations

```
SELECT FlightDate, count() AS Flights,  
sum(Cancelled) AS Daily_Cancelled,  
sum(Daily_Cancelled)  
  OVER (PARTITION BY toStartOfMonth(FlightDate)  
        ORDER BY FlightDate)  
  AS Cumul_Cancelled  
FROM ontime  
WHERE Year = 2019 GROUP BY FlightDate ORDER BY FlightDate
```

Group by flight month

sum(Daily_Cancelled)
 OVER (PARTITION BY toStartOfMonth(FlightDate)
 ORDER BY FlightDate)
 AS Cumul_Cancelled

Order by date of light

FlightDate	Flights	Daily_Cancelled	Cumul_Cancelled
2019-01-01	18009	141	141
2019-01-02	20384	173	314

Rank cancellations by week

```
SELECT FlightDate, count() AS Flights,  
sum(Cancelled) AS Daily_Cancelled,  
rank() OVER  
  (PARTITION BY toStartOfWeek(FlightDate)  
   ORDER BY Daily_Cancelled DESC) as Weekly_Rank  
FROM ontime  
WHERE Year = 2019 GROUP BY FlightDate ORDER BY FlightDate
```

Group by week

FlightDate	Flights	Daily_Cancelled	Weekly_Rank
2019-01-01	18009	141	2
2019-01-02	20384	173	1
2019-01-03	19522	134	3

Multiple ranks for aircraft flights

```
SELECT TailNum, any(Carrier) AS Carrier, count() Flights,  
rank() OVER (ORDER BY Flights DESC) as Overall_Rank,  
rank() OVER (PARTITION BY Carrier ORDER BY Flights DESC) as  
Carrier Rank  
FROM ontime  
WHERE toYYYYMM(FlightDate) = 201901  
GROUP BY TailNum ORDER BY Flights DESC
```

TailNum	Carrier	Flights	Overall_Rank	Carrier_Rank
	OH	2543	1	1
N488HA	HA	361	2	1
N481HA	HA	348	3	2

Reuse window definitions

```
SELECT FlightDate, count() AS Flights,  
sum(Cancelled) AS Daily_Cancelled,  
min(Daily_Cancelled) OVER 7_day as Min_Cancelled_7,  
avg(Daily_Cancelled) OVER 7_day as Avg_Cancelled_7,  
max(Daily_Cancelled) OVER 7_day as Max_Cancelled_7  
FROM ontime WHERE Year = 2019  
GROUP BY FlightDate WINDOW 7_day AS (ROWS BETWEEN 6 PRECEDING  
AND CURRENT ROW) ORDER BY FlightDate
```

```
FlightDate|Flights|Daily_Cancelled|Min_Cancelled_7|...  
-----|-----|-----|-----|...  
2019-01-01| 18009|          141|          141|...  
2019-01-02| 20384|          173|          141|...
```

Are window functions the only way?



“Definitely not!”

Rank cancellations by week using arrays

```
SELECT FlightDate, Flights, Daily_Cancelled, Weekly_Rank FROM
(
  SELECT
    groupArray(FlightDate) AS FlightDate_Arr,
    groupArray(Flights) AS Flights_Arr,
    groupArray(Daily_Cancelled) AS Daily_Cancelled_Arr,
    arrayEnumerate(Daily_Cancelled_Arr) AS Daily_Cancelled_Indexes,
    arraySort((x, y) -> -y, Daily_Cancelled_Indexes, Daily_Cancelled_Arr) as Rank_Array
  FROM
  (
    SELECT FlightDate, count() AS Flights,
    sum(Cancelled) AS Daily_Cancelled
    FROM ontime
    WHERE Year = 2019 GROUP BY FlightDate ORDER BY FlightDate
  )
  GROUP BY toStartOfWeek(FlightDate)
  ORDER BY toStartOfWeek(FlightDate)
)
ARRAY JOIN FlightDate_Arr AS FlightDate, Flights_Arr AS Flights,
Daily_Cancelled_Arr AS Daily_Cancelled, Rank_Array AS Weekly_Rank
ORDER BY FlightDate
```

Roll up values by week

Sort indexes by descending sum of cancelled flights

Unroll arrays again

Roadmap and more information

Not supported or doesn't work

Some features of window functions that are not supported now or don't work

- RANGE frame only works for UIntX/IntX, Date and DateTime types and is not supported for other data types including Nullable
- No INTERVAL support for Date and DateTime types
- No EXCLUDE clause
- No GROUPS frame
- No lag(value, offset) and lag(value, offset) functions but workaround is documented
- Expressions can't use window functions
- Can't use RANGE frame with a named window

More information on window functions

- [ClickHouse window function docs](#)
- [Altinity Blog: ClickHouse Window Functions – Current State of the Art](#)
- [Altinity Software Requirements Spec: SRS019 ClickHouse Window Functions](#)
- [Altinity Knowledge Base, e.g., cumulative sums](#)
- [Blog article on Window Functions by TinyBird.co](#)

And special thanks to:

Alexander Kuzmenkov @ Yandex -- Implemented window functions

Alexey Milovidov @ Yandex -- ClickHouse lead committer

Altinity QA team -- Testing!

Questions?

Thank you!

Altinity

<https://altinity.com>

ClickHouse

<https://github.com/ClickHouse/ClickHouse>

Altinity.Cloud

<https://altinity.com/cloud-database/>

We are hiring!