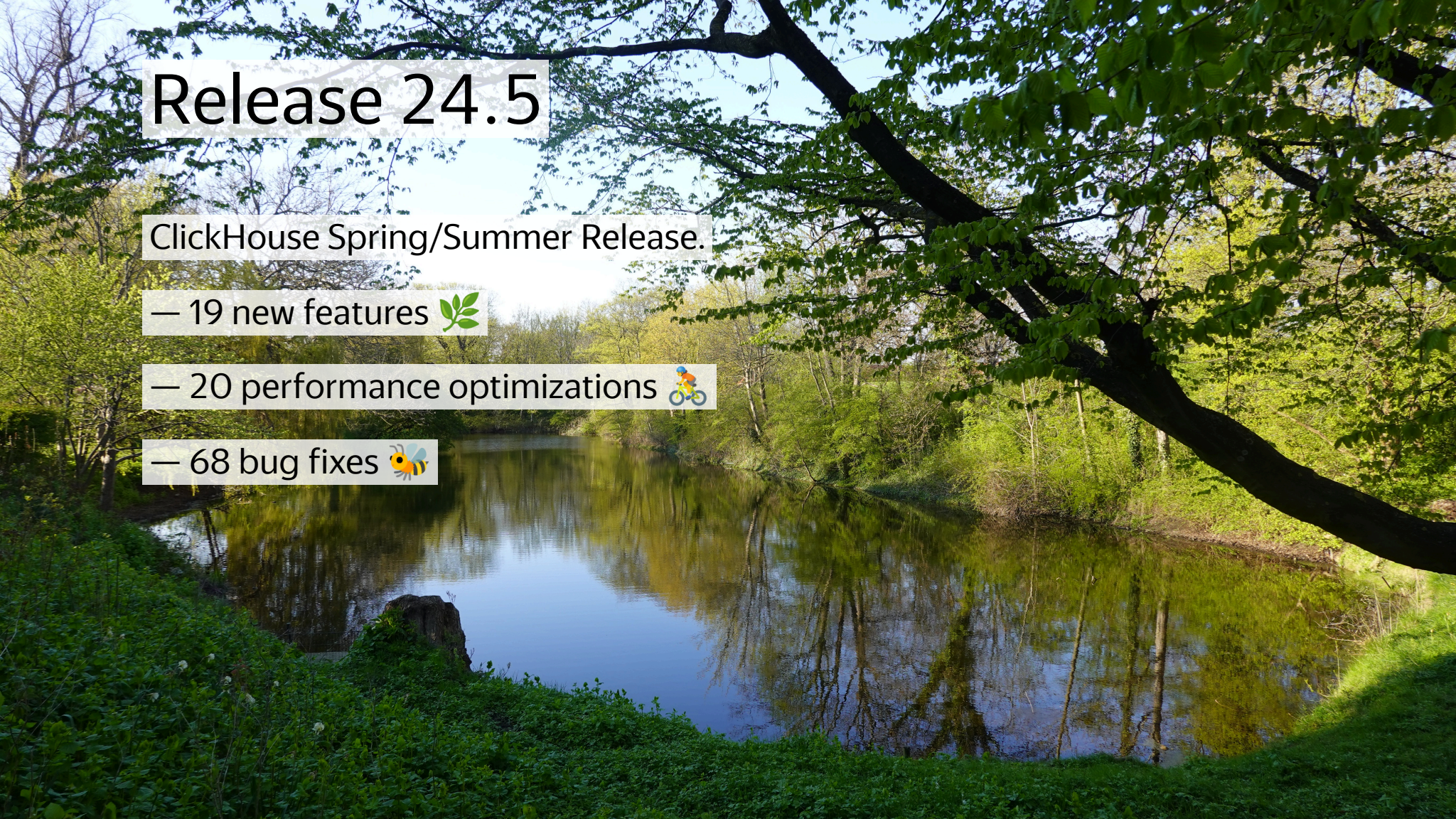

ClickHouse Release
24.5

Release 24.5 Webinar

1. (55 min) What's new in ClickHouse 24.5.

2. (5 min) Q&A.


Release 24.5
ClickHouse Spring/Summer Release.
— 19 new features 🌿
— 20 performance optimizations 🚴
— 68 bug fixes 🐝

Small And Nice Features

Unrestricted Map Data Type

```
CREATE TEMPORARY TABLE test (
   x Map(Array(Map(Float64, String)), String));

INSERT INTO test VALUES
   ({[]: 'Hello'}),
   ({[{1.1: 'test'}]: 'World'});

SELECT * FROM test1;

   ┌─x─────────────────────────────┐
1. │ {[]:'Hello'}                  │
2. │ {[{1.1:'test'}]:'World'} │
   └───────────────────────────────┘
```

Developer: TaiYang Li.

Npy As An Output Format

It was available for data import since 23.10.

Since 24.5 it also supports data export.

```
SELECT [number, number * 2]
FROM numbers(10) INTO OUTFILE 'test.npy' -- or FORMAT Npy

SELECT * FROM 'test.npy'

python3 -c "
    import numpy as np;
    array = np.load('test.npy');
    print(array)"
```

Demo.

Developer: HowePa.

clamp

```
:) SELECT greatest(5, least(10, x))

:) SELECT max2(5, min2(10, x))

:) SELECT clamp(x, 5, 10)
```

Developer: skyoct.

Reading From Archives On S3

```
SELECT * FROM s3(
  's3://umbrella-static/top-1m-2024-01-*.csv.zip :: *.csv')
  LIMIT 10;

SELECT _path, c1 FROM s3(
  's3://umbrella-static/top-1m-2024-01-*.csv.zip :: *.csv')
  WHERE c2 = 'clickhouse.com' ORDER BY _path;
```

The contents of container formats, such as ***.tar.***, ***.zip**, ***.7z** can be read directly from S3.

For the local filesystem — already supported since 23.8.

Developer: Dan Ivanik.

Usability Improvements

Trailing Commas In INSERT

```
INSERT INTO test (c1, c2,) VALUES ('Hello', 'World',);

INSERT INTO test
(
    c1,
    c2,
)
VALUES
(
    'Hello',
    'World',
);
```

Developer: Alexey Milovidov.

Autodetection Of Compression In StdOut

```
ch -q "SELECT * FROM 'test.parquet' INTO OUTFILE 'test.csv.gz'"
```

Since version 24.5 it works even with shell redirects!

```
ch -q "SELECT * FROM 'test.parquet'" > test.csv.gz
```

Developers: v01dXYZ.

Command Line Simplification

```
clickhouse-local --query "SELECT 1"
clickhouse-local -q "SELECT 1"
echo "SELECT 1" | clickhouse local
ch -q "SELECT 1"
echo "SELECT 1" | ch
ch "SELECT 1"  # since 24.5
```

Developers: Alexey Milovidov.

English Quotes

```
SELECT 'Hello, world';

SELECT ‘Hello, world’;
```

Why??? Demo.

Developers: Alexey Milovidov.

Performance Improvements

Fast Parquet Reader

It is enabled by default.

No comments. Try it, and you will see the difference.

Developer: ZhiHong Zhang.

Faster String Functions

UTF8 functions have shortcuts when a string is ASCII-only:

— substringUTF8, substringIndexUTF8, reverseUTF8, lower/upperUTF8, left/rightPadUTF8.

splitByRegexp has a shortcut when a regular expression is a single character.

Developer: TaiYang Li.

Faster Index Analysis

For indices of type **set**.

Index analysis can take a long time, delaying query processing
especially for tables with trillions of rows or with a low index_granularity.

Now up to 1.5 times faster!

Developer: Alexey Milovidov.

Index Analysis For DateTime64

When DateTime is compared with a constant DateTime64.

Developer: Alexey Milovidov.

Faster GROUP BY

When keys fit in 16 bits.

By optimizing merging of lookup tables.

Developer: Jie Binn.

Faster Glob Matching For S3

```
prefix{1,2,3}/file*.csv
```

Was: make a regular expression
and match it against all paths for a prefix.

Now: interpret **{1,2,3}** as lazy generators,
only match **file*.csv** inside **prefix1/**, **prefix2/**, **prefix3/**.

Faster when there is a huge number of files.

Developer: Andrey Zvonov.

Faster Merges

With sparse columns.

Sparse columns — is a special data format used for
columns with a high ratio of default values.

The query engine automatically takes the benefit of the sparse format
by optimizing functions and expressions.

Developer: Anton Popov.

Something Interesting

Dynamic Data Type

A new experimental data type for **semistructured data**.

Similar to the **Variant** data type, but *dynamic*.

Variant(String, UInt64, Array(String)) — anything from the list of types.

Dynamic — anything, automatically extending the list of types.

Developer: Pavel Kruglov

Dynamic Data Type

Using **Dynamic** type in table's definition:

```
CREATE TABLE test (d Dynamic) ENGINE = Memory;
INSERT INTO test VALUES (NULL), (42), ('Hello, World!'), ([1, 2, 3]);
SELECT d, dynamicType(d) FROM test;

┌─d─────────────┬─dynamicType(d)─┐
│ ᴺᵁᴸᴸ          │ None           │
│ 42            │ Int64          │
│ Hello, World! │ String         │
│ [1,2,3]       │ Array(Int64)   │
└───────────────┴────────────────┘
```

Developer: Pavel Kruglov

Dynamic Data Type

CAST from an ordinary column:

```
SELECT 'Hello, World!'::Dynamic as d, dynamicType(d);

┌─d─────────────┬─dynamicType(d)─┐
│ Hello, World! │ String         │
└───────────────┴────────────────┘
```

CAST from a **Variant** column:

```
SET allow_experimental_variant_type = 1, use_variant_as_common_type = 1;
SELECT multiIf(number % 3 = 0, number,
number % 3 = 1, range(number + 1), NULL)::Dynamic AS d, dynamicType(d)
FROM numbers(3)

┌─d─────┬─dynamicType(d)─┐
│ 0     │ UInt64         │
│ [0,1] │ Array(UInt64)  │
│ ᴺᵁᴸᴸ  │ None           │
└───────┴────────────────┘
```

Dynamic Data Type

Reading subcolumns of the Dynamic value:

```
SELECT d, dynamicType(d), d.String, d.`Array(Int64)` FROM test;

┌─d─────────────┬─dynamicType(d)─┬─d.String──────┬─d.Array(Int64)─┐
│ ᴺᵁᴸᴸ          │ None           │ ᴺᵁᴸᴸ          │ []             │
│ 42            │ Int64          │ ᴺᵁᴸᴸ          │ []             │
│ Hello, World! │ String         │ Hello, World! │ []             │
│ [1,2,3]       │ Array(Int64)   │ ᴺᵁᴸᴸ          │ [1,2,3]        │
└───────────────┴────────────────┴───────────────┴────────────────┘

SELECT toTypeName(d.String) FROM test;

┌─toTypeName(d.String)─┐
│ Nullable(String)     │
└──────────────────────┘

SELECT d, dynamicType(d), dynamicElement(d, 'String') FROM test;

┌─d─────────────┬─dynamicType(d)─┬─dynamicElement(d, 'String')─┐
│ ᴺᵁᴸᴸ          │ None           │ ᴺᵁᴸᴸ                        │
│ 42            │ Int64          │ ᴺᵁᴸᴸ                        │
│ Hello, World! │ String         │ Hello, World!               │
│ [1,2,3]       │ Array(Int64)   │ ᴺᵁᴸᴸ                        │
└───────────────┴────────────────┴─────────────────────────────┘
```

Dynamic Data Type

Type inference from data formats:

```
SELECT d, dynamicType(d)
FROM format(JSONEachRow, 'd Dynamic',
'
    {"d" : "Hello, World!"},
    {"d" : 42},
    {"d" : 42.42},
    {"d" : "2020-01-01"},
    {"d" : [1, 2, 3]}
')

┌─d─────────────┬─dynamicType(d)─┐
│ Hello, World! │ String         │
│ 42            │ Int64          │
│ 42.42         │ Float64        │
│ 2020-01-01    │ Date           │
│ [1,2,3]       │ Array(Int64)   │
└───────────────┴────────────────┘
```

Azure

Support for **"workload identity"** — Vinay Suryadevara.

Optimizing **backups** with the server-side copy — Alexander Sapin.

Support for **plain_rewritable** metadata for **azure** blob storage — Julia Kartseva.

Azure is **production ready** since 24.5!

What for?

Azure

What for?

We launched ClickHouse Cloud on Azure (beta)!

https://clickhouse.com/blog/clickhouse-cloud-is-now-on-azure-in-public-beta

ClickHouse Cloud: now on AWS, GCP, **and Azure**,
... and a partner service on AliCloud.

Form Data Format

```
INSERT INTO test FORMAT Form
name=Vasya&email=vasya@pupkin.com&text=I%20like%20ClickHouse%20so%20much!
```

Inserting directly from HTML forms (**x-www-form-urlencoded** MIME type)

Direct integration with JavaScript libraries, such as Boomerang.js.

Supports inserting a single record
(asynchronous inserts are recommended).

Developer: Shaun Struwig.

JOIN Improvements

From now on, you will see JOIN improvements in every ClickHouse release.

In-memory compression for CROSS (comma) JOIN:
— fast processing of a large table's product.

On-disk processing for CROSS (comma) JOIN:
— generate a large table's product even when
the right hand side is too large.

Developer: Maksim Alekseev.

Non-Equal JOIN

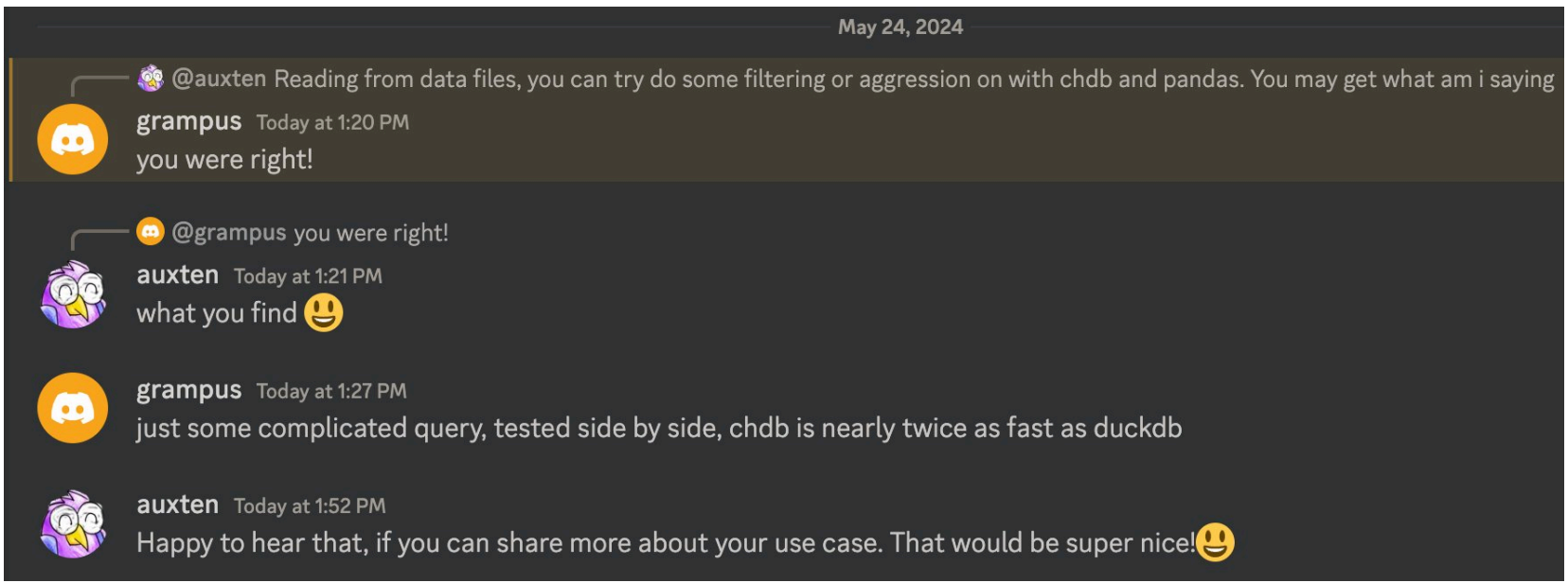
```
SET allow_experimental_join_condition = 1;
```

And now ClickHouse supports not only equality conditions in ON.

Developer: Lgbo-USTC.

chdb

Fast processing of Pandas Dataframes.



Developer: Auxten

chdb

Custom data types: any user-provided data type could be processed:

```
class myReader(chdb.PyReader):
    def __init__(self, data):
        self.data = data
        self.cursor = 0
        super().__init__(data)

    def read(self, col_names, count):
        block = [self.data[col] for col in col_names]
        block = [col[self.cursor : self.cursor + count] for col in block]
        self.cursor += block[0].shape[0]
        return block

reader = myReader(df_old)

ret = chdb.query("""
   SELECT RegionID, SUM(AdvEngineID), COUNT(*) AS c
   FROM Python(reader) GROUP BY RegionID ORDER BY c DESC LIMIT 10""",
    "DataFrame")
```

Developer: Auxten


Bonus

Bonus: loongarch64

Now we have:

— **x86_64** (production);
— **aarch64** (production);
— **risc-v** (experimental);
— **powerpc64-le** (experimental);
— **s390x** (experimental);
— **loongarch64** (experimental);

Developer: QiangXuhui


Integrations

Integrations

ClickPipes:
— **Kinesis** support;
— allow to set **offsets** for Kafka;
— meta-columns **key**, **timestamp**, **headers**, **topic**, **offset** for Kafka;

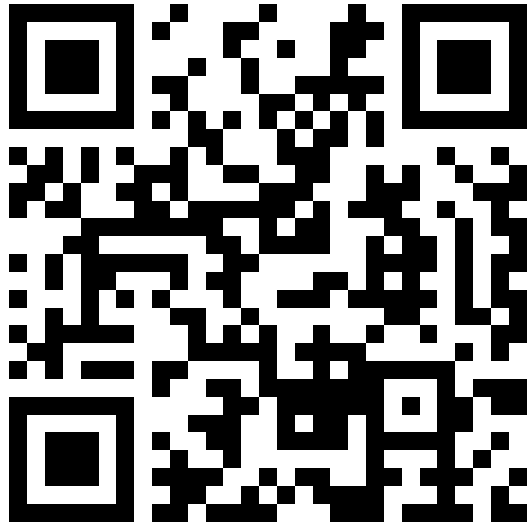
Metabase — directly import CSV into ClickHouse.

Updates for **dbt** and **Tableau** integrations,
as well as **Java**, **Go**, and **Python** drivers.

+ Thanks for many fixes to our contributors:
R. Joel Norgren, Guo Qiang Wang, Jiří Kozlovský, Abhishek Gahlot, Frederik Eychenié, Cal Herries.

Reading Corner

https://clickhouse.com/blog/



— Real-world AWS Graviton 4 performance;
— The new SQL console;
— API endpoints;
— Flexible backups;
— How to become a certified ClickHouse developer;
— Why ClickHouse is faster than Elasticsearch;


Q&A