ClickHouse:
My Favorite Features

Best Features 2022

From the first half of 2022: Jan..Jun

— my favorite features...

— your favorite features!

Best Features 2022

Try to guess — what's my favorite new feature.

And win a T-shirt!

Schema Inference

Now:

```
SELECT * FROM url(
    'https://datasets.clickhouse.com/github_events_v2.native.xz') LIMIT 10
```

Read the structure:

```
DESCRIBE url('https://datasets.clickhouse.com/github_events_v2.native.xz')
```

Developer: Pavel Kruglov.

```
SELECT * FROM url('https://datasets.clickhouse.com/github_events_v2.native.xz', Native,
$$
    file_time DateTime, event_type Enum('CommitCommentEvent' = 1, 'CreateEvent' = 2, 'DeleteEvent'
= 3, 'ForkEvent' = 4, 'GollumEvent' = 5, 'IssueCommentEvent' = 6, 'IssuesEvent' = 7, 'MemberEvent'
= 8, 'PublicEvent' = 9, 'PullRequestEvent' = 10, 'PullRequestReviewCommentEvent' = 11, 'PushEvent'
= 12, 'ReleaseEvent' = 13, 'SponsorshipEvent' = 14, 'WatchEvent' = 15, 'GistEvent' = 16,
'FollowEvent' = 17, 'DownloadEvent' = 18, 'PullRequestReviewEvent' = 19, 'ForkApplyEvent' = 20,
'Event' = 21, 'TeamAddEvent' = 22), actor_login LowCardinality(String), repo_name
LowCardinality(String), created_at DateTime, updated_at DateTime, action Enum('none' = 0,
'created' = 1, 'added' = 2, 'edited' = 3, 'deleted' = 4, 'opened' = 5, 'closed' = 6, 'reopened' =
7, 'assigned' = 8, 'unassigned' = 9, 'labeled' = 10, 'unlabeled' = 11, 'review_requested' = 12,
'review_request_removed' = 13, 'synchronize' = 14, 'started' = 15, 'published' = 16, 'update' =
17, 'create' = 18, 'fork' = 19, 'merged' = 20), comment_id UInt64, body String, path String,
position Int32, line Int32, ref LowCardinality(String), ref_type Enum('none' = 0, 'branch' = 1,
'tag' = 2, 'repository' = 3, 'unknown' = 4), creator_user_login LowCardinality(String), number
UInt32, title String, labels Array(LowCardinality(String)), state Enum('none' = 0, 'open' = 1,
'closed' = 2), locked UInt8, assignee LowCardinality(String), assignees
Array(LowCardinality(String)), comments UInt32, author_association Enum('NONE' = 0, 'CONTRIBUTOR'
= 1, 'OWNER' = 2, 'COLLABORATOR' = 3, 'MEMBER' = 4, 'MANNEQUIN' = 5), closed_at DateTime,
merged_at DateTime, merge_commit_sha String, requested_reviewers Array(LowCardinality(String)),
requested_teams Array(LowCardinality(String)), head_ref LowCardinality(String), head_sha String,
base_ref LowCardinality(String), base_sha String, merged UInt8, mergeable UInt8, rebaseable UInt8,
mergeable_state Enum('unknown' = 0, 'dirty' = 1, 'clean' = 2, 'unstable' = 3, 'draft' = 4),
merged_by LowCardinality(String), review_comments UInt32, maintainer_can_modify UInt8, commits
UInt32, additions UInt32, deletions UInt32, changed_files UInt32, diff_hunk String,
original_position UInt32, commit_id String, original_commit_id String, push_size UInt32,
push_distinct_size UInt32, member_login LowCardinality(String), release_tag_name String,
release_name String, review_state Enum('none' = 0, 'approved' = 1, 'changes_requested' = 2,
'commented' = 3, 'dismissed' = 4, 'pending' = 5)
$$)
LIMIT 10
```

Table Structure Autodetection

For formats that already contain structure:
— Native, Protobuf, Avro, Parquet, ORC, Arrow.
— CSVWithNamesAndTypes, TSVWithNamesAndTypes.

Was:

```
SELECT * FROM url('https://datasets.clickhouse.com/github_events_v2.native.xz',
Native,
$$
    file_time DateTime, event_type Enum('CommitCommentEvent' = 1, 'CreateEvent' = 2,
'DeleteEvent' = 3, 'ForkEvent' = 4, 'GollumEvent' = 5, 'IssueCommentEvent' = 6,
'IssuesEvent' = 7, 'MemberEvent' = 8, 'PublicEvent' = 9, 'PullRequestEvent' = 10,
'PullRequestReviewCommentEvent' = 11, 'PushEvent' = 12, 'ReleaseEvent' = 13,
'SponsorshipEvent' = 14, 'WatchEvent' = 15, 'GistEvent' = 16, 'FollowEvent' = 17,
'DownloadEvent' = 18, 'PullRequestReviewEvent' = 19, 'ForkApplyEvent' = 20, 'Event' =
21, 'TeamAddEvent' = 22), actor_login LowCardinality(String), repo_name
LowCardinality(String), created_at DateTime, updated_at DateTime, action Enum('none'
= 0, 'created' = 1, 'added' = 2, 'edited' = 3, 'deleted' = 4, 'opened' = 5, 'closed'
= 6, 'reopened' = 7, 'assigned' = 8, 'unassigned' = 9, 'labeled' = 10, 'unlabeled' =
```

Schema Inference

It works even for semistructured and unstructured formats!
— CSV, TSV, CSVWithNames, TSVWithNames,
  JSONEachRow, Values, Regexp, MsgPack...

```
DESCRIBE file('hits.ndjson')

┌─name──────────┬─type───────────────────┬─default_type─┬─default_expression─┬─comm
│ UserID        │ Nullable(String)       │              │                    │
│ URLDomain     │ Nullable(String)       │              │                    │
└───────────────┴────────────────────────┴──────────────┴────────────────────┴─────
```

— Nullable(String) if data is inside string in JSON.
— Nullable(Float64) if it's number.
— Arrays are also supported. Multidimensional arrays.
— Arrays of non-uniform types are parsed as Tuples.

Developer: Pavel Kruglov.

Schema Inference

```
$ echo '{"x":[[123,456]]}' > test.ndjson
$ clickhouse-local --query "SELECT x, toTypeName(x) FROM file('test.ndjson')"

[[123,456]]     Array(Array(Nullable(Float64)))

$ echo '{"x":[123,"Hello",["World"]]}' > test.ndjson
$ clickhouse-local --query "SELECT x, toTypeName(x) FROM file('test.ndjson')"

(123,'Hello',['World']) Tuple(Nullable(Float64), Nullable(String),
Array(Nullable(String)))

$ echo '{"x":"Hello"} {"y":"World"}' > test.ndjson
$ clickhouse-local --query "SELECT * FROM file('test.ndjson')"

\N      Hello
World   \N

$ echo '{"x":[[123,"Hello"]]} {"x":[[123,"Hello",456]]}' > test.ndjson
$ clickhouse-local --query "SELECT * FROM file('test.ndjson')"

Code: 636. DB::Exception: Cannot extract table structure from JSONEachRow
format file. Error: Automatically defined type Array(Tuple(Nullable(Float64),
Nullable(String))) for column x in row 1 differs from type defined by
```

Schema Inference

CSV: String and Float64 types are inferred;
TSV: everything as String;
Column names: c1, c2, ...;
For CSVWithNames, TSVWithNames column names are extracted.

```
:) DESC file('hits.csv')

c1      Nullable(String)
c2      Nullable(Float64)

:) DESC file('hits.tsv')

c1      Nullable(String)
c2      Nullable(String)

:) DESC file('hits2.tsv', TSVWithNames)

URLDomain  Nullable(String)
UserID     Nullable(String)
```

Schema Inference

We also support schema on demand!

```
SELECT c1 AS domain, uniq(c2), count() AS cnt
  FROM file('hits.csv')
  GROUP BY domain ORDER BY cnt DESC LIMIT 10

SELECT c1::String AS domain, uniq(c2::UInt64), count() AS cnt
  FROM file('hits.csv')
  GROUP BY domain ORDER BY cnt DESC LIMIT 10

SELECT URLDomain::String AS domain, uniq(UserID::UInt64), count() AS cnt
  FROM file('hits.ndjson')
  GROUP BY domain ORDER BY cnt DESC LIMIT 10
```

Schema Inference

Bonus: usage example of **LineAsString** and **RawBLOB** formats:

```
:) SELECT extractTextFromHTML(*)
    FROM url('https://news.ycombinator.com/', LineAsString);

:) SELECT extractTextFromHTML(*)
    FROM url('https://news.ycombinator.com/', RawBLOB);

:) DESC url('https://news.ycombinator.com/', LineAsString)

line    String
```

Developer: Pavel Kruglov.

Schema Autodetection

Formats with external schema: **Protobuf** and **CapNProto**:

```
SELECT * FROM file('data.protobuf')
    SETTINGS format_schema = 'path_to_schema:message_name'
```

Developer: Pavel Kruglov.

Schema Autodetection

Also works for **Merge**, **Distributed** and **ReplicatedMegreTree**!

Was:

```
CREATE TABLE hits (WatchID UInt64, JavaEnable UInt8, Title String,
GoodEvent Int16, EventTime DateTime, EventDate Date, CounterID UInt32,
ClientIP UInt32, ClientIP6 FixedString(16), RegionID UInt32, UserID
UInt64, CounterClass Int8, OS UInt8, UserAgent UInt8, URL String,
Referer String, URLDomain String, RefererDomain String, Refresh UInt8,
IsRobot UInt8, ... ParsedParams.Key4 Array(String), ParsedParams.Key5
Array(String), ParsedParams.ValueDouble Array(Float64), IslandID
FixedString(16), RequestNum UInt32, RequestTry UInt8)
ENGINE = ReplicatedMegreTree('/clickhouse/tables/{uuid}', '{replica}');
```

Now:

```
CREATE TABLE hits
ENGINE = ReplicatedMegreTree('/clickhouse/tables/{uuid}', '{replica}');
```

Format Autodetection

For INTO OUTFILE, FROM INFILE:

Was:

```
SELECT URLDomain, UserID FROM test.hits
    INTO OUTFILE 'hits.csv' FORMAT CSV
```

Now:

```
SELECT URLDomain, UserID FROM test.hits INTO OUTFILE 'hits.parquet'
SELECT URLDomain, UserID FROM test.hits INTO OUTFILE 'hits.csv'
SELECT URLDomain, UserID FROM test.hits INTO OUTFILE 'hits.ndjson'
SELECT URLDomain, UserID FROM test.hits INTO OUTFILE 'hits.native'
SELECT URLDomain, UserID FROM test.hits INTO OUTFILE 'hits.csv.gz'
```

For table functions and engines:
— file, url, s3, hdfs, s3Cluster, hdfsCluster...

Developers: Pavel Kruglov, ZhongYuanKai.

Best Features 2022

Try to guess — what's my favorite new feature.

And win a T-shirt!

1. Schema Inference.

2. ???

ClickHouse Keeper

Full compatibility with ZooKeeper 3.5 by protocol and data model.

Can run embedded in clickhouse-server. Or separately.

Or replace ZooKeeper in other software stacks.

Passing Jepsen tests, ClickHouse functional and integration tests.
Deployed in production.

Faster than ZooKeeper, consuming less memory.

Compact logs and snapshots.

Developer: Alexander Sapin.

Best Features 2022

Try to guess — what's my favorite new feature.

And win a T-shirt!

1. Schema Inference.

2. ClickHouse Keeper.

3. ???

Flexible Memory Limits

a.k.a. Memory Overcommit

ClickHouse in 2021:

— Memory limit (for query) exceeded: would use 9.39 GiB;
— SET max_memory_usage = ...

ClickHouse in 2022:

— everything works automatically;
— no need to tune any settings.

Developer: Dmitriy Novik.

Best Features 2022

Try to guess — what's my favorite new feature.

And win a T-shirt!

1. Schema Inference.

2. ClickHouse Keeper.

3. Flexible Memory Limits.

4. ???

Projections

Multiple data representations inside a single table.

— different data order;
— subset of columns;
— subset of rows;
— aggregation.

Difference to materialized views:

— projections data is always consistent;
— updated atomically with the table;
— replicated in the same way as the table;
— projection can be automatically used for SELECT query.

Developer — Amos Bird. Experimental since 21.6. Production since 22.3.

```
CREATE TABLE wikistat
(
    time DateTime,
    project LowCardinality(String),
    subproject LowCardinality(String),
    path String,
    hits UInt64,

    PROJECTION total
    (
        SELECT
            project,
            subproject,
            path,
            sum(hits),
            count()
        GROUP BY
            project,
            subproject,
            path
    )
)
ENGINE = ReplicatedMergeTree
ORDER BY (path, time)
```

Best Features 2022

Try to guess — what's my favorite new feature.

And win a T-shirt!

1. Schema Inference.

2. ClickHouse Keeper.

3. Flexible Memory Limits.

4. Projections.

5. ???

Backup & Restore

— full and incremental backups;

— tables, partitions, databases, all databases;

— full or partial restore;

— a bunch of files or a single archive;

— atomic snapshot for MergeTree,
best effort snapshot for multiple tables;

Developer — Vitaliy Baranov.

Backup & Restore

```
BACKUP TABLE t TO File('backup_20220629');

BACKUP TABLE t TO File('backup_20220629.zip');

BACKUP TABLE t TO File('backup_20220630')
    SETTINGS base_backup = File('backup_20220629');

BACKUP TABLE system.users, system.grants TO ...
BACKUP TABLE system.functions TO ...
```

Developer — Vitaliy Baranov.

Backup & Restore

```
BACKUP|RESTORE
  TABLE [db.]table_name [AS [db.]table_name_in_backup]
    [PARTITION[S] partition_expr [,...]] |
  DICTIONARY [db.]dictionary_name [AS [db.]name_in_backup] |
  DATABASE database_name [AS database_name_in_backup]
    [EXCEPT TABLES ...] |
  TEMPORARY TABLE table_name [AS table_name_in_backup] |
  ALL TEMPORARY TABLES [EXCEPT ...] |
  ALL DATABASES [EXCEPT ...] } [,...]
  [ON CLUSTER 'cluster_name']
  TO|FROM File('path/') | Disk('disk_name', 'path/')
  [SETTINGS base_backup = File(...) | Disk(...)]
```

Developer — Vitaliy Baranov.

Best Features 2022

Try to guess — what's my favorite new feature.

1. Schema Inference.

2. ClickHouse Keeper.

3. Flexible Memory Limits.

4. Projections.

5. Backup & Restore.

6. I just want to give more T-shirts...

Support for Aarch64

First ClickHouse builds for Aarch64 (ARM64) — in **Feb 2016**.

ClickHouse builds have been tested on:

— APM X-Gene;
— Cavium ThunderX 1, 2;
— Raspberry Pi;
— Pinebook;
— Google Pixel;
— Apple M1;
— Huawei Taishan;
— AWS Graviton 1, 2 and 3;
— Ampere Altra;
— and on secret Google CPUs;

Making Aarch64 Production Ready

— automated builds, packages, Docker;

— full functional test runs;

— automated performance tests;

— JIT compilation for queries;

— optimizations with ARM NEON;

— query profiling and introspection;

— Hyperscan, GRPC;

Best Features 2022

Some features are still experimental
but already available for testing:

— Semistructured Data
a.k.a. dynamic subcolumns and JSON data type.

— Transactions.

Do you use these features?

Support For Semistructured Data

JSON data type:

```
CREATE TABLE games (data JSON) ENGINE = MergeTree;
```

You can insert arbitrary nested JSONs.

Types are automatically inferred on INSERT and merge.

Data is stored in columnar format: columns and subcolumns.

Query nested data naturally.

Support For Semistructured Data

Example: NBA games dataset

```
CREATE TABLE games (data String)
ENGINE = MergeTree ORDER BY tuple();

SELECT JSONExtractString(data, 'teams', 1, 'name')
FROM games;
```

— 0.520 sec.

```
CREATE TABLE games (data JSON)
ENGINE = MergeTree;

SELECT data.teams.name[1] FROM games;
```

— 0.015 sec.

Support For Semistructured Data

```
DESCRIBE TABLE games
SETTINGS describe_extend_object_types = 1

name: data
type: Tuple(                                                     <-- inferred type
    `_id.$oid` String,
    `date.$date` String,
    `teams.abbreviation` Array(String),
    `teams.city` Array(String),
    `teams.home` Array(UInt8),
    `teams.name` Array(String),
    `teams.players.ast` Array(Array(Int8)),
    `teams.players.blk` Array(Array(Int8)),
    `teams.players.drb` Array(Array(Int8)),
    `teams.players.fg` Array(Array(Int8)),
    `teams.players.fg3` Array(Array(Int8)),
    `teams.players.fg3_pct` Array(Array(String)),
    `teams.players.fg3a` Array(Array(Int8)),
    `teams.players.fg_pct` Array(Array(String)),
    `teams.players.fga` Array(Array(Int8)),
    `teams.players.ft` Array(Array(Int8)),
    `teams.players.ft_pct` Array(Array(String)),
    `teams.players.fta` Array(Array(Int8)),
    `teams.players.mp` Array(Array(String)),
    `teams.players.orb` Array(Array(Int8)),
    `teams.players.pf` Array(Array(Int8)),
    `teams.players.player` Array(Array(String)),
    `teams.players.plus_minus` Array(Array(String)),
```

Support For Semistructured Data

Flexible schema.

You can have columns with strict and flexible schema in one table.

Queries work as fast as with predefined types!

Production readiness — Q4 2022.

Sparse Encoding For Columns

If a column contains mostly zeros, we can encode it in sparse format
and **automatically optimize** calculations!

```
CREATE TABLE test.hits ...
ENGINE = MergeTree ORDER BY ...
SETTINGS ratio_of_defaults_for_sparse_serialization = 0.9
```

Special column encoding like **LowCardinality**
but it's completely transparent and automatic.

Developer: Anton Popov.

Transactions

ACID. MVCC. Snapshot Isolation.

Available for preview with non-replicated MergeTree tables.

— standard **BEGIN TRANSACTION**, **COMMIT** and **ROLLBACK** statements;

— atomic INSERTs of huge amounts of data;

— atomic INSERTs into multiple tables and materialized views;

— multiple consistent and isolated SELECTs from single snapshot;

— atomicity and isolation for ALTER DELETE / UPDATE;

Transactions

Available for preview with non-replicated MergeTree tables.

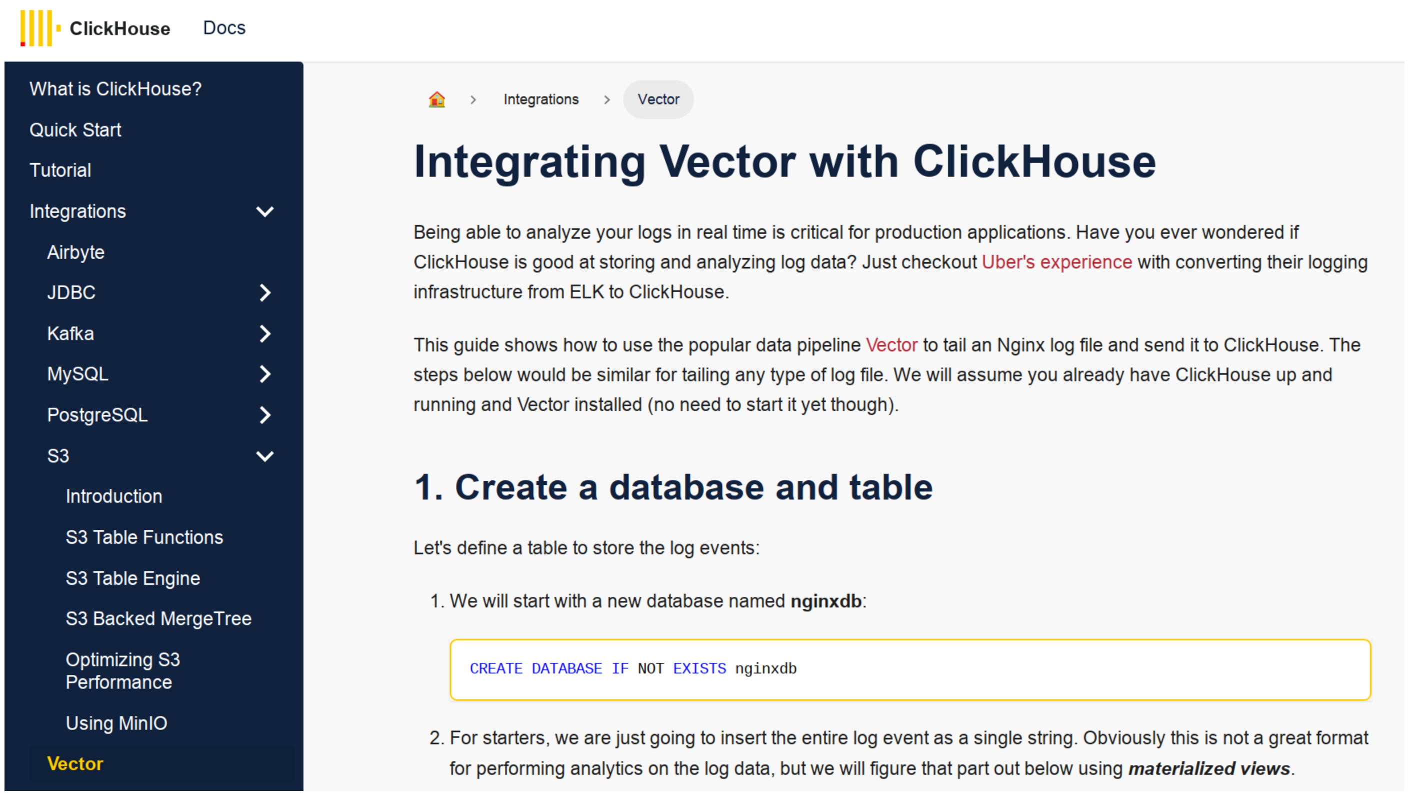
Next steps: Replicated tables and Distributed queries.

We need your feedback.

Production readiness — Q4 2022.

Developer: Alexander Tokmakov.

Bonus: New Documentation



Integrating Vector with ClickHouse

Being able to analyze your logs in real time is critical for production applications. Have you ever wondered if ClickHouse is good at storing and analyzing log data? Just checkout Uber's experience with converting their logging infrastructure from ELK to ClickHouse.

This guide shows how to use the popular data pipeline Vector to tail an Nginx log file and send it to ClickHouse. The steps below would be similar for tailing any type of log file. We will assume you already have ClickHouse up and running and Vector installed (no need to start it yet though).

1. Create a database and table

Let's define a table to store the log events:

1. We will start with a new database named **nginxdb**:

```
CREATE DATABASE IF NOT EXISTS nginxdb
```

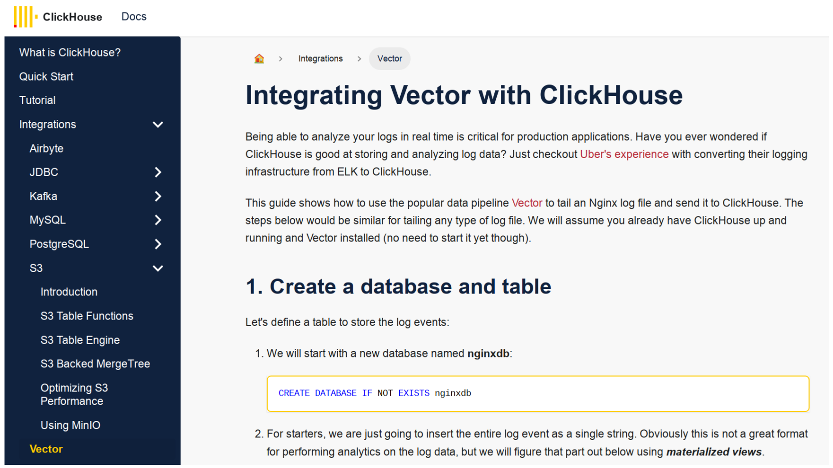
2. For starters, we are just going to insert the entire log event as a single string. Obviously this is not a great format for performing analytics on the log data, but we will figure that part out below using ***materialized views***.

Bonus: New Documentation

Advantages:
— many tutorials about integrations;
— it is fast;

Disadvantages:
— just released; still to do: dark theme;
— no pdf version;

Developer: Rich Raposa.



Integrating Vector with ClickHouse

Being able to analyze your logs in real time is critical for production applications. Have you ever wondered if ClickHouse is good at storing and analyzing log data? Just checkout Uber's experience with converting their logging infrastructure from ELK to ClickHouse.

This guide shows how to use the popular data pipeline Vector to tail an Nginx log file and send it to ClickHouse. The steps below would be similar for tailing any type of log file. We will assume you already have ClickHouse up and running and Vector installed (no need to start it yet though).

1. Create a database and table

Let's define a table to store the log events:

1. We will start with a new database named **nginxdb**:

```
CREATE DATABASE IF NOT EXISTS nginxdb
```

2. For starters, we are just going to insert the entire log event as a single string. Obviously this is not a great format for performing analytics on the log data, but we will figure that part out below using ***materialized views***.

What's Next?

What Else?

Vote for the most weird feature to implement:

— Indices for Vector Search;

— Secondary Indices;

— Streaming Queries;

— Batch Jobs Support;

— GPU Offloading;

— Key-Value Data Marts;

— Embedded ClickHouse Engine;

— Text Processing.

Q&A