ClickHouse

Inverted database indexes: The why, the what, and the how.

Elmi Ahmadov, Software Engineer @ ClickHouse

31.01.2026

What is ClickHouse?

Your (soon-to-be) favorite database!

Open source

- Development started 2009
- Production 2012
- OSS 2016

Column-oriented

- Best for aggregations
- Files per column
- Sorting and indexing
- Background merges

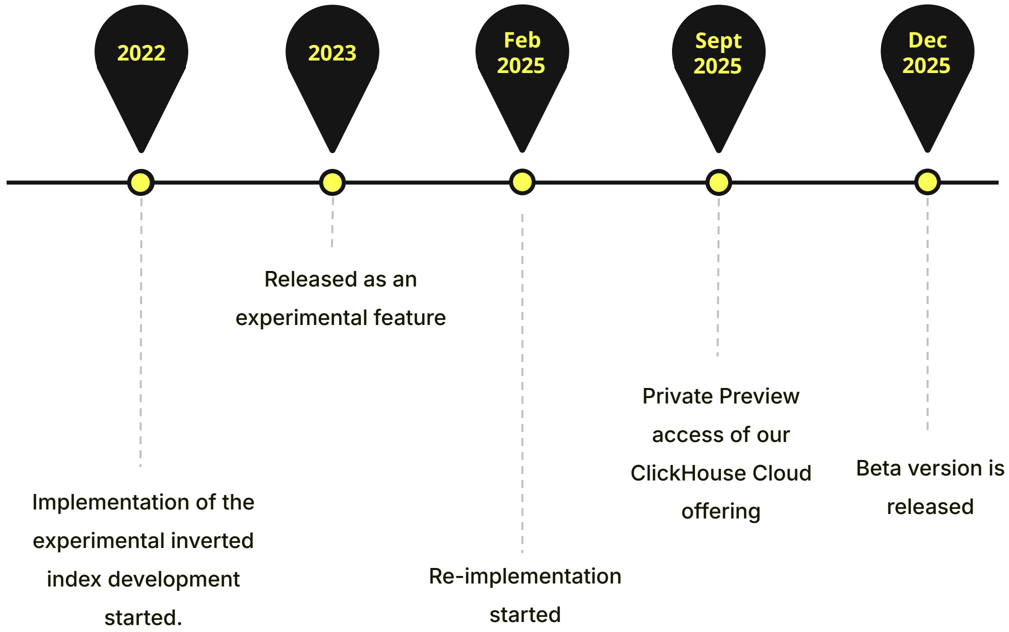
Distributed

- Replication
- Sharding
- Multi-master
- Cross-region

OLAP database

- Analytics use cases
- Aggregations
- Visualizations
- Mostly immutable data

ClickHouse full-text search journey



Implementation of the experimental inverted index development started.

Released as an experimental feature

Re-implementation started

Private Preview access of our ClickHouse Cloud offering

Beta version is released

ClickHouse

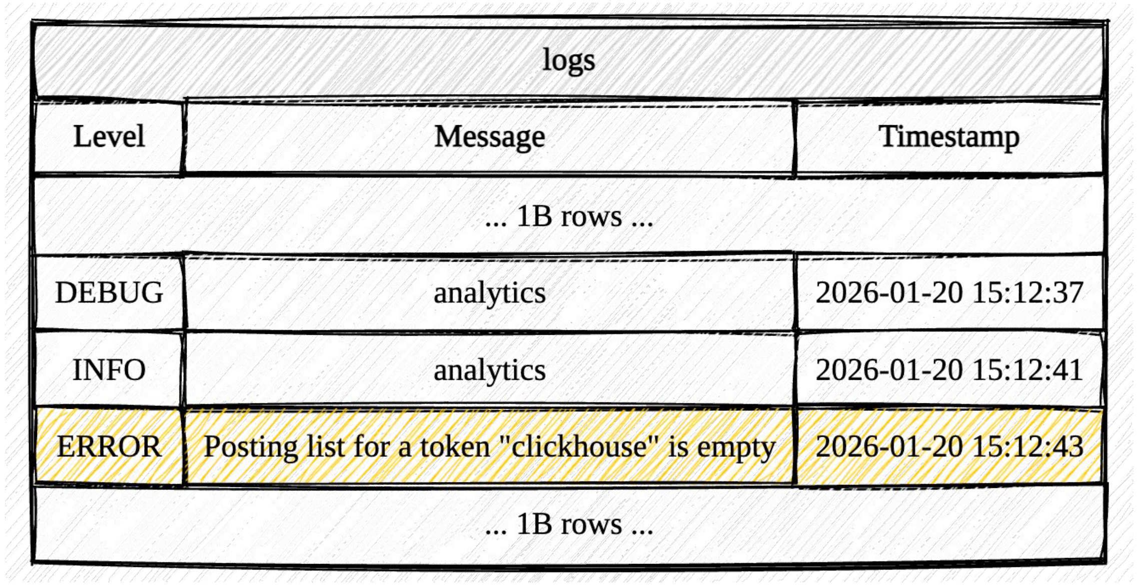
ClickHouse

01

Use case

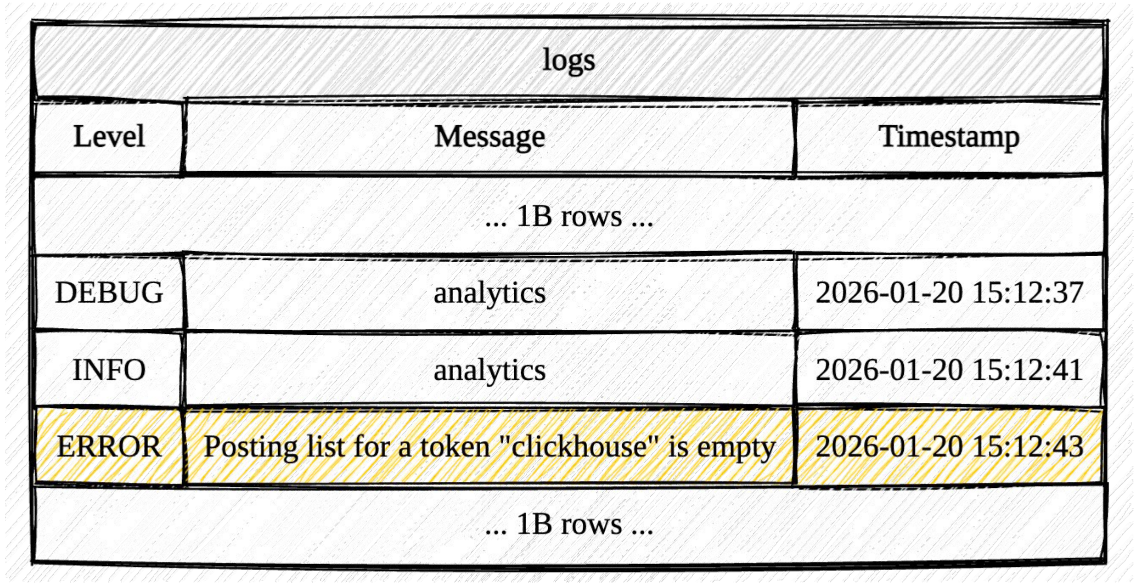
What is the problem?

Example use case: Observability



logs		
Level	Message	Timestamp
... 1B rows ...		
DEBUG	analytics	2026-01-20 15:12:37
INFO	analytics	2026-01-20 15:12:41
ERROR	Posting list for a token "clickhouse" is empty	2026-01-20 15:12:43
... 1B rows ...		

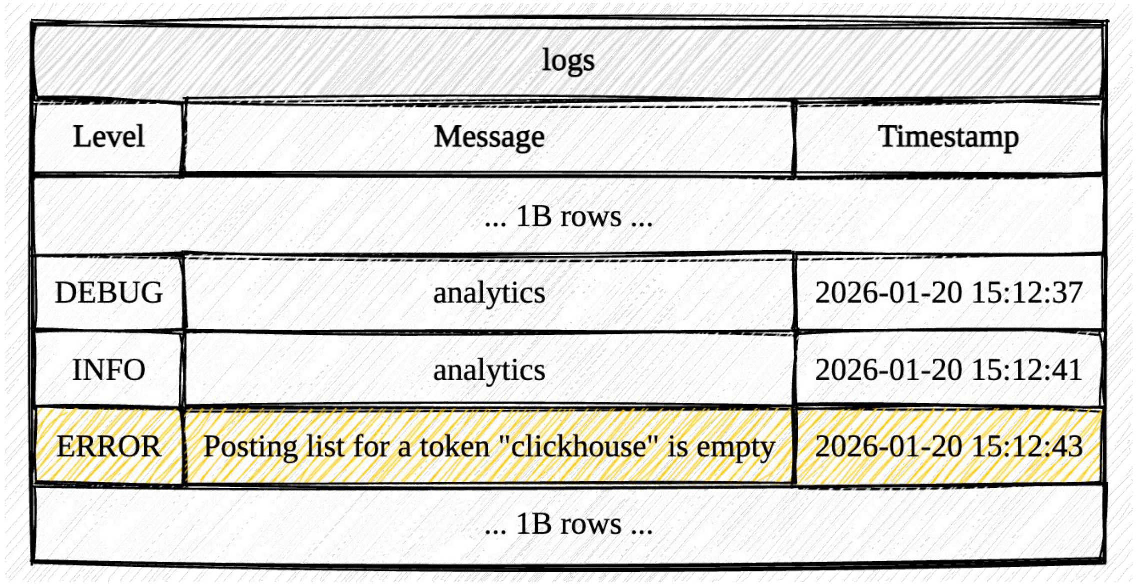
Example use case: Observability



logs		
Level	Message	Timestamp
... 1B rows ...		
DEBUG	analytics	2026-01-20 15:12:37
INFO	analytics	2026-01-20 15:12:41
ERROR	Posting list for a token "clickhouse" is empty	2026-01-20 15:12:43
... 1B rows ...		

```
SELECT count() FROM logs WHERE hasAllTokens(message, ['token', 'is', 'empty']);
```

Example use case: Observability

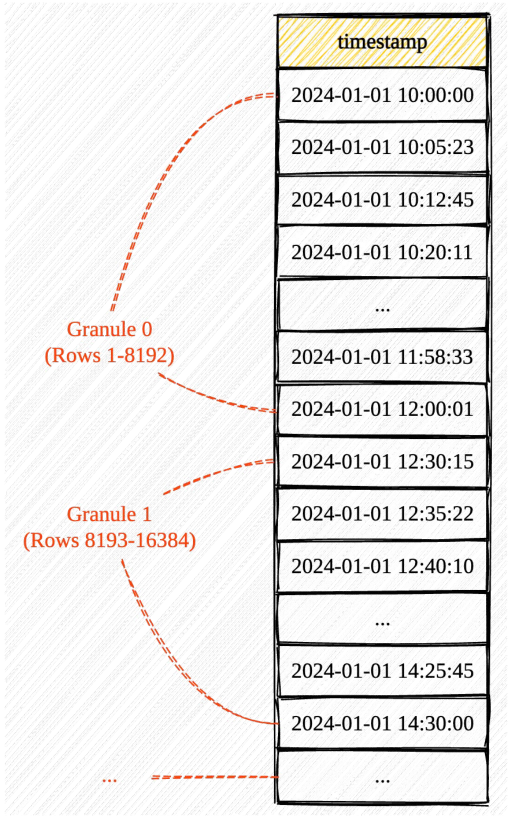


logs		
Level	Message	Timestamp
... 1B rows ...		
DEBUG	analytics	2026-01-20 15:12:37
INFO	analytics	2026-01-20 15:12:41
ERROR	Posting list for a token "clickhouse" is empty	2026-01-20 15:12:43
... 1B rows ...		

SLOW! FULL TABLE SCAN!

ClickHouse

What is a granule?



- A granule represents the smallest indivisible data unit processed by the scan and index lookup operators in ClickHouse.

- The rows of a part are further logically divided into groups of 8192 records, called granules.

Tokenizers

```
2026.01.31::12:57:01 {0bd92626-e789-49b2-aac5-e35084b7bc08} <Debug>
```

- `splitByNonAlpha = ['2026', '01', '31', '12', '...', '0bd92626']`
- `splitByString(['::', ' ']) = ['2026.01.31', '12:57:01', '0bd92626-e789-49b2-aac5-e35084b7bc08']`
- `ngrams(3) = ['202', '026', '26.', '6.1', '...', 'ug>']`
- `array = ['2026.01.31:12:57:01 {0bd92626-e789-49b2-aac5-e35084b7bc08} <Debug>']`

ClickHouse

ClickHouse

02

Solution: Full-text search

Building blocks

The new text index

```
CREATE TABLE table (
  ...
  text String,
  INDEX idx_text(text) TYPE text(tokenizer =
    splitByNonAlpha |
    splitByString(['::', ' ']) |
    ngrams(N) |
    sparseGrams(min, max) |
    array
  )
ENGINE = MergeTree
ORDER BY `time`;
```

New functions

- Two new functions have been introduced
  - **hasAnyTokens**, finds columns containing any search tokens

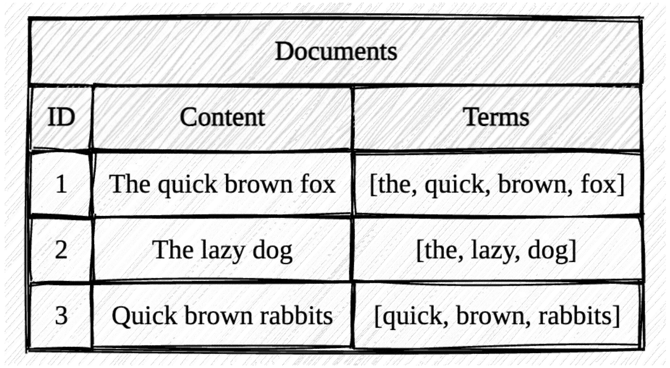
```
SELECT count() FROM hackernews
WHERE hasAnyTokens(text, ['clickhouse', 'fosdem']);
```

  - **hasAllTokens**, finds columns containing all search tokens

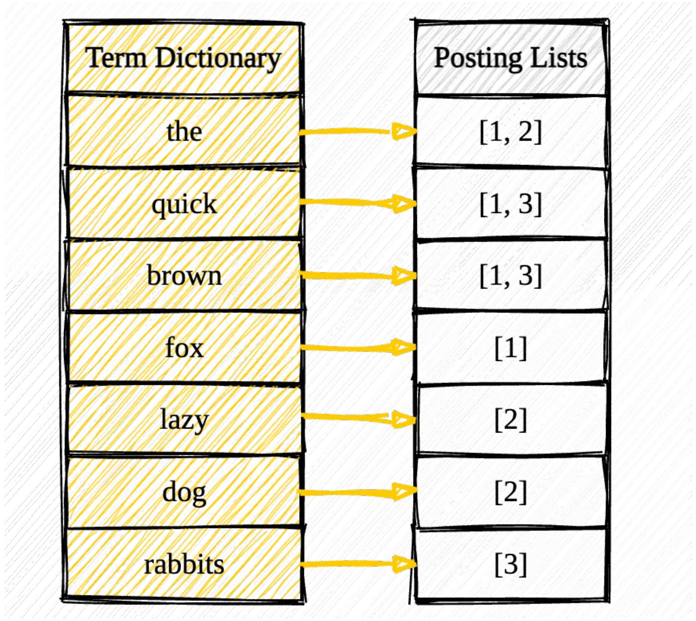
```
SELECT count() FROM hackernews
WHERE hasAllTokens(text, ['clickhouse', 'fosdem']);
```

Inverted index

- Inverted index (**token → documents**)

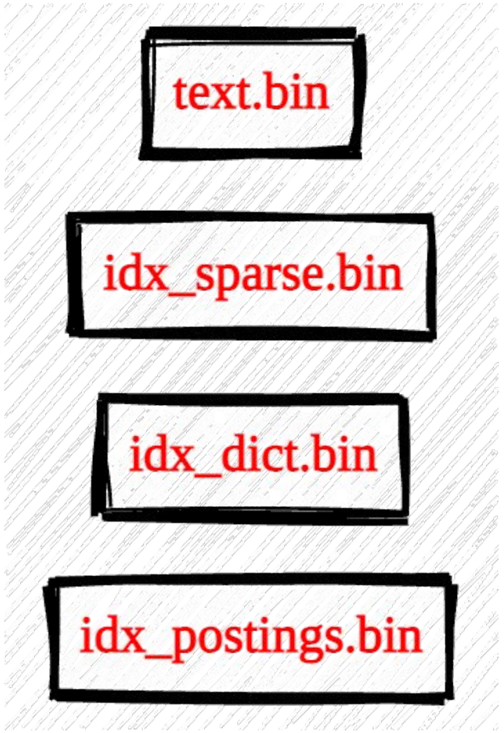


Documents		
ID	Content	Terms
1	The quick brown fox	[the, quick, brown, fox]
2	The lazy dog	[the, lazy, dog]
3	Quick brown rabbits	[quick, brown, rabbits]



Term Dictionary	Posting Lists
the	[1, 2]
quick	[1, 3]
brown	[1, 3]
fox	[1]
lazy	[2]
dog	[2]
rabbits	[3]

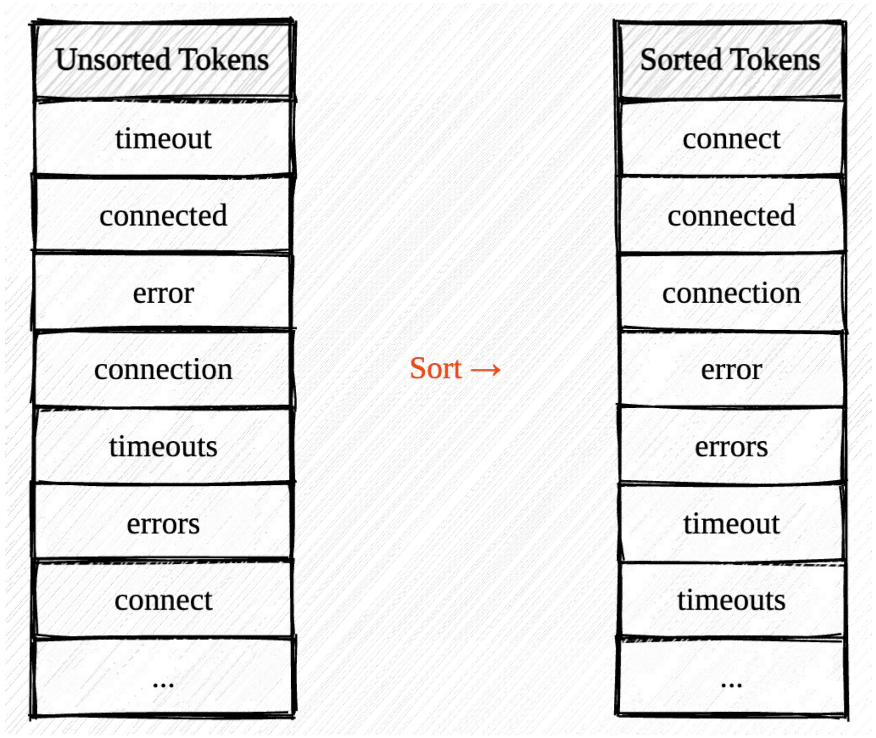
How the data is organized



- While building an index, we maintain a hash table to store a **token → posting list** mapping.

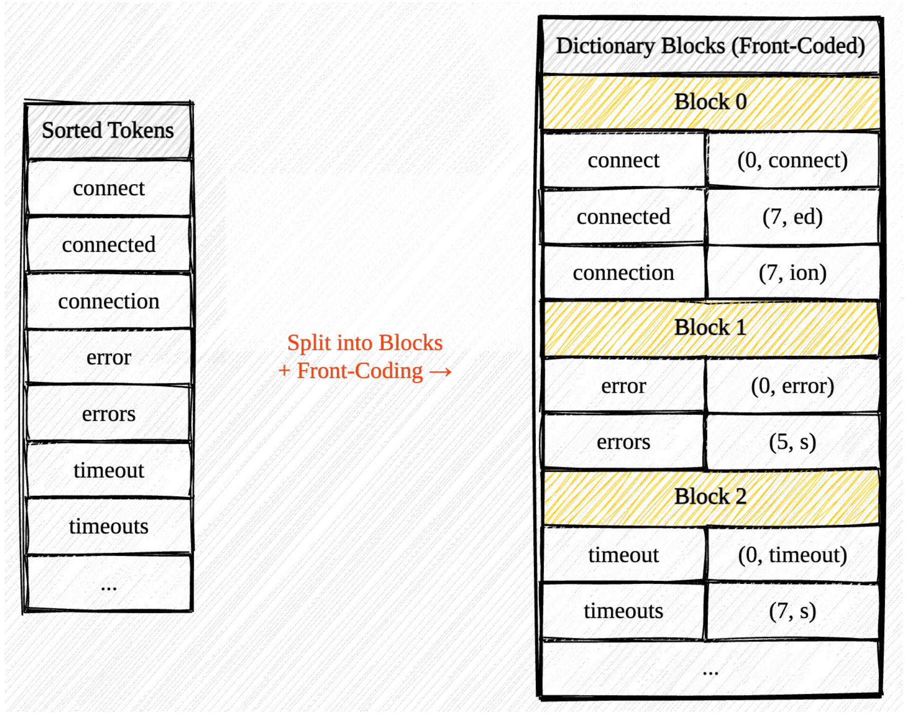
- When flushed to disk, an inverted index consists of **three files** on disk next to the column data file.

Dictionary blocks & sparse index



- First, all terms are sorted alphabetically

Dictionary blocks & sparse index



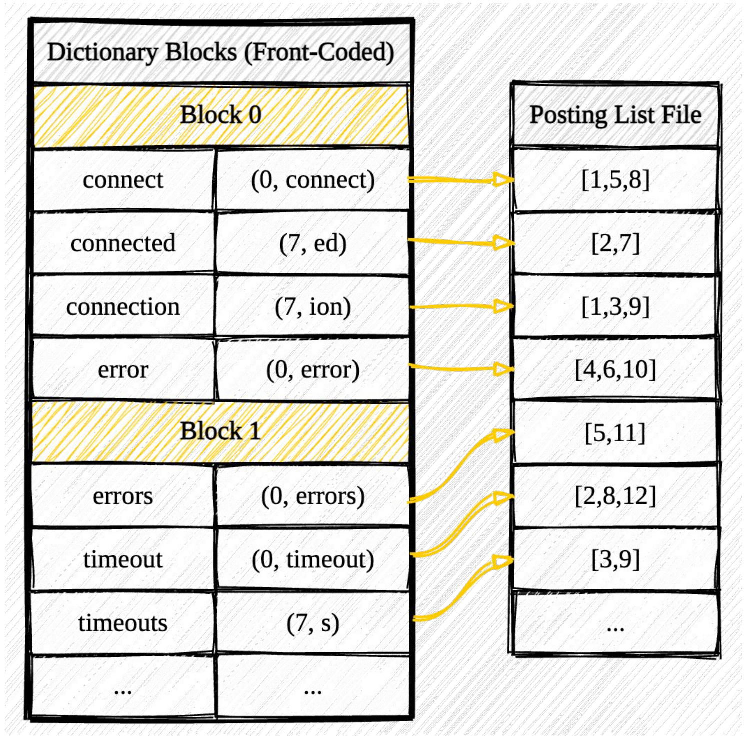
Sorted Tokens
connect
connected
connection
error
errors
timeout
timeouts
...

Dictionary Blocks (Front-Coded)	
Block 0	
connect	(0, connect)
connected	(7, ed)
connection	(7, ion)
Block 1	
error	(0, error)
errors	(5, s)
Block 2	
timeout	(0, timeout)
timeouts	(7, s)
...	

- First, all terms are sorted alphabetically
- Sorted terms are splitted into blocks:
    - Dictionary blocks are compressed by the front-coding compression

ClickHouse

Dictionary blocks & sparse index



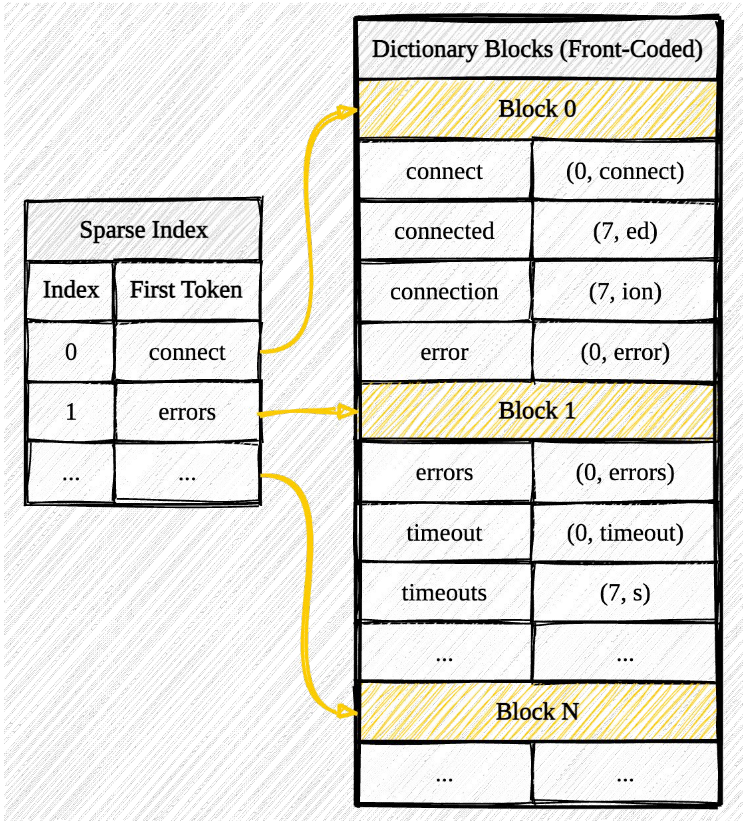
- First, all terms are sorted alphabetically

- Sorted terms are splitted into blocks:
  - Dictionary blocks are compressed by the front-coding compression

  - Each term stores to an offset of its postings in the posting list file

ClickHouse

Dictionary blocks & sparse index



Sparse Index

Index	First Token
0	connect
1	errors
...	...

Dictionary Blocks (Front-Coded)

Block 0	
connect	(0, connect)
connected	(7, ed)
connection	(7, ion)
error	(0, error)
Block 1	
errors	(0, errors)
timeout	(0, timeout)
timeouts	(7, s)
...	...
Block N	
...	...

- First, all terms are sorted alphabetically

- Sorted terms are splitted into blocks:
  - Dictionary blocks are compressed by the front-coding compression.
  - Each term stores to an offset of its postings in the posting list file.

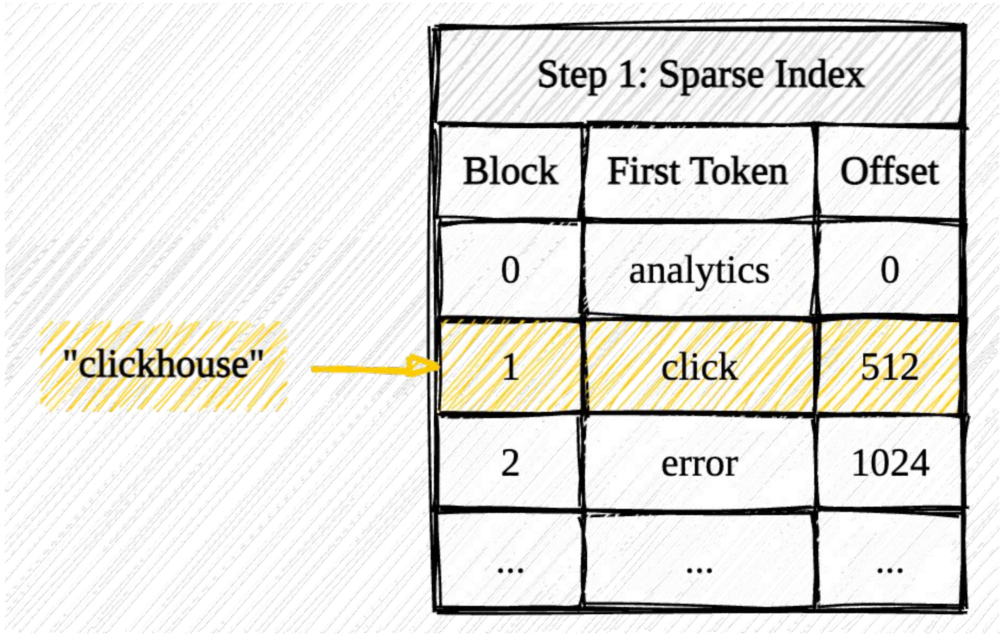
- The sparse index points to the beginning of blocks.

ClickHouse

ClickHouse

03 Demo

What happened!?

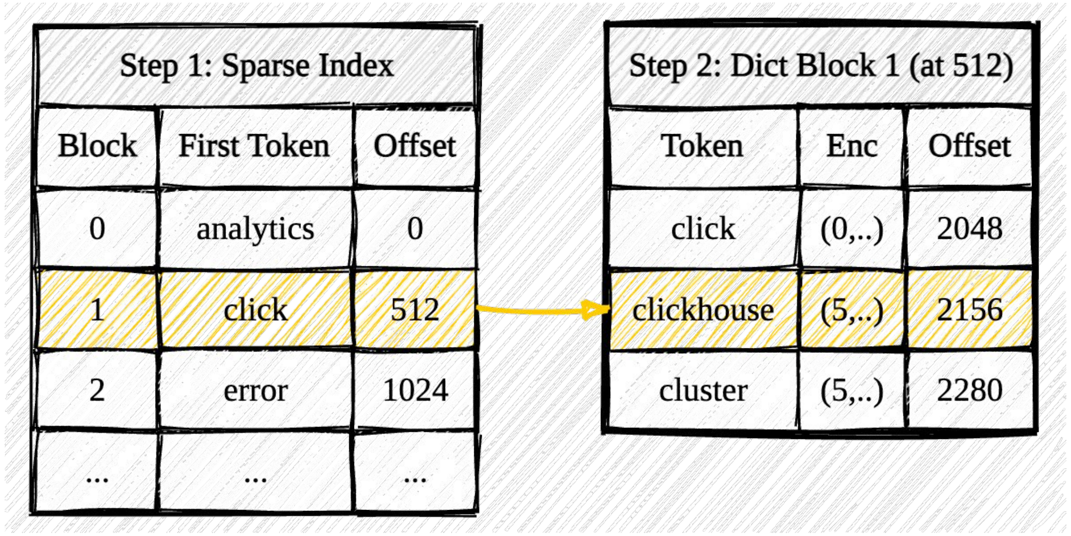

"clickhouse"

Step 1: Sparse Index

Block	First Token	Offset
0	analytics	0
1	click	512
2	error	1024
...	...	...

- Binary search on sparse index

What happened!?

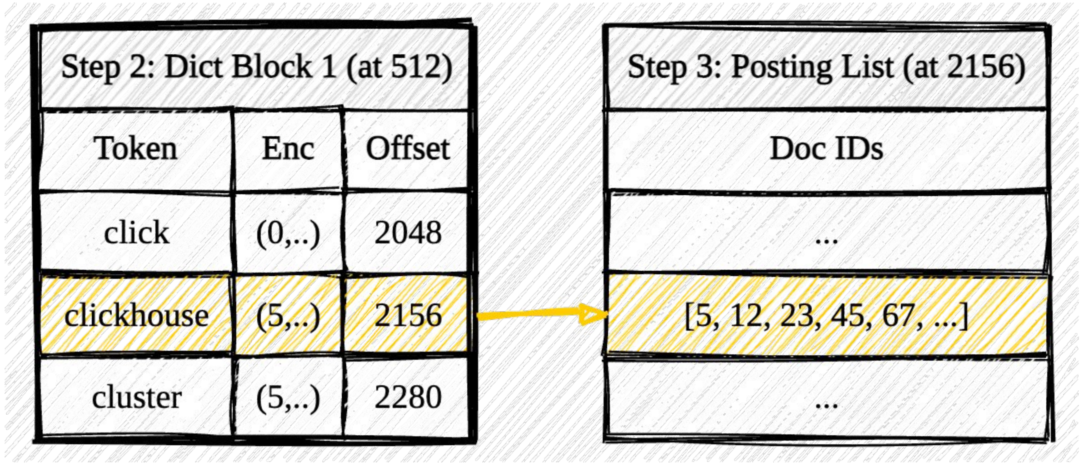


Step 1: Sparse Index		
Block	First Token	Offset
0	analytics	0
1	click	512
2	error	1024
...	...	...

Step 2: Dict Block 1 (at 512)		
Token	Enc	Offset
click	(0,..)	2048
clickhouse	(5,..)	2156
cluster	(5,..)	2280

- Binary search on sparse index
- Search token in a dictionary block

What happened!?



Step 2: Dict Block 1 (at 512)

Token	Enc	Offset
click	(0,..)	2048
clickhouse	(5,..)	2156
cluster	(5,..)	2280

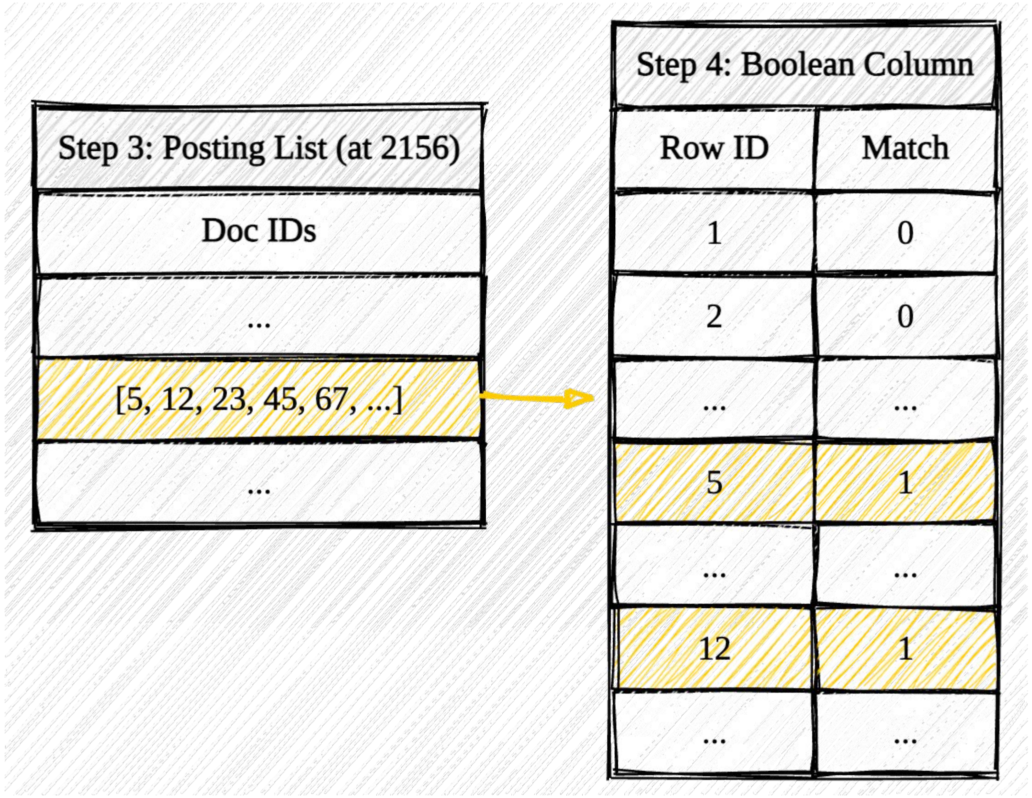
Step 3: Posting List (at 2156)

Doc IDs
...
[5, 12, 23, 45, 67, ...]
...

- Binary search on sparse index
- Search token in the dictionary block
- If present, read the posting list

ClickHouse

What happened!?



Step 3: Posting List (at 2156)
Doc IDs
...
[5, 12, 23, 45, 67, ...]
...

Step 4: Boolean Column	
Row ID	Match
1	0
2	0
...	...
5	1
...	...
12	1
...	...

- Binary search on sparse index
- Search token in the dictionary block
- If present, read the posting list
- Fill the result column with doc IDs

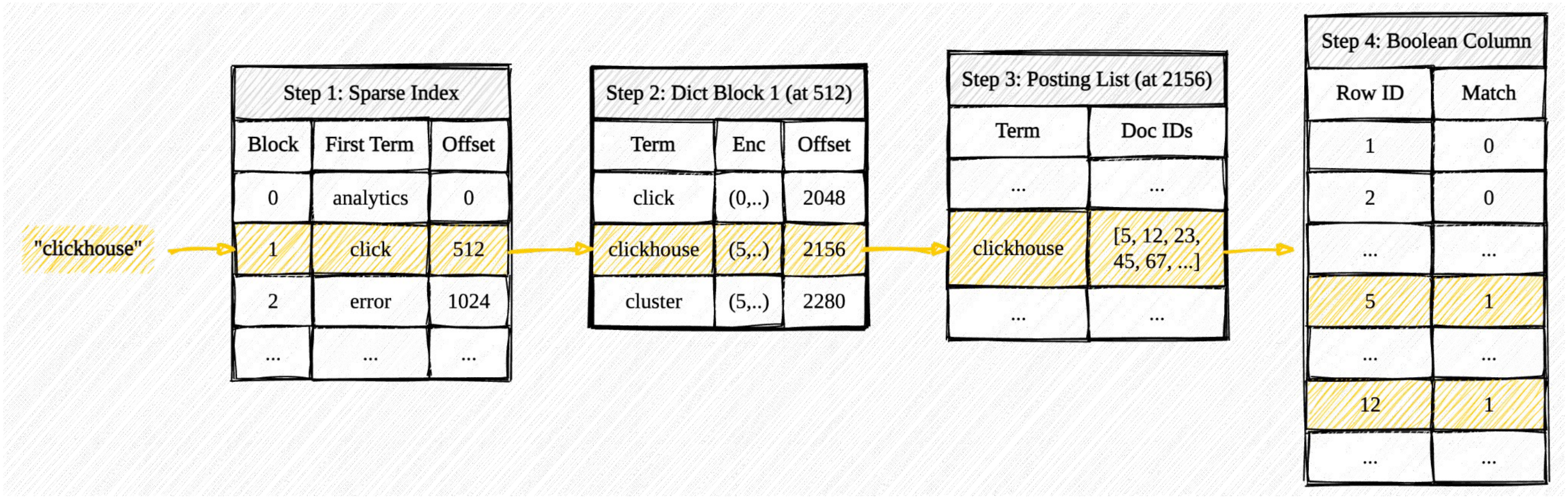
Text index optimization

- Use the virtual boolean column instead of the original filter condition

```
SELECT count() FROM hackernews
WHERE hasAnyTokens(text, ['clickhouse']);
```

```
-- _text_virt_column is filled during optimization.
SELECT count() FROM hackernews
PREWHERE _text_virt_column = 1;
```

Putting it all together



Step 1: Sparse Index

Block	First Term	Offset
0	analytics	0
1	click	512
2	error	1024
...	...	...

Step 2: Dict Block 1 (at 512)

Term	Enc	Offset
click	(0,..)	2048
clickhouse	(5,..)	2156
cluster	(5,..)	2280

Step 3: Posting List (at 2156)

Term	Doc IDs
...	...
clickhouse	[5, 12, 23, 45, 67, ...]
...	...

Step 4: Boolean Column

Row ID	Match
1	0
2	0
...	...
5	1
...	...
12	1
...	...

Summary - I loved it! Can I try it myself?

- ClickHouse DB has a state-of-the-art full-text search now
  - Full-text search BETA version is available on ClickHouse OSS since v25.12

- We offer a private preview for our ClickHouse Cloud customers

- More features are coming
  - Phrase search (a.k.a positional queries)

  - Scoring (BM25)

Your search ends here

Thanks! Questions?

ClickHouse

ClickHouse

Connect with ClickHouse

slack

ClickHouse Community Dinner

Try ClickHouse for your use case

- ClickHouse Cloud
- Download open source

Learn

- Academy / certifications
- Blogs / YouTube

Engage with our community

- Community Slack
- Monthly Community calls
- Meetups / events

We are Hiring. Come Work with Us!

ClickHouse