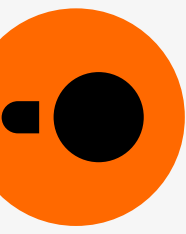


- 2 PM Welcome to DuckCon!
- 2:05 State of the Duck v4
- 2:35 Hugging a Duck (Polina)
- 3:00 Building a Data Lake using DuckDB (Subash)
- 3:25 Break
- 3:40 The Duck(DB) Feather in Your Parquet Cap (Niger)
- 4:05 Lightning Talks (Rik, Arjen, Remco, Kshitij, Mike, Ian)
- 4:40 Drinks and Snacks (sponsored by Rill Data)
- ~6:40 End

State of the Duck

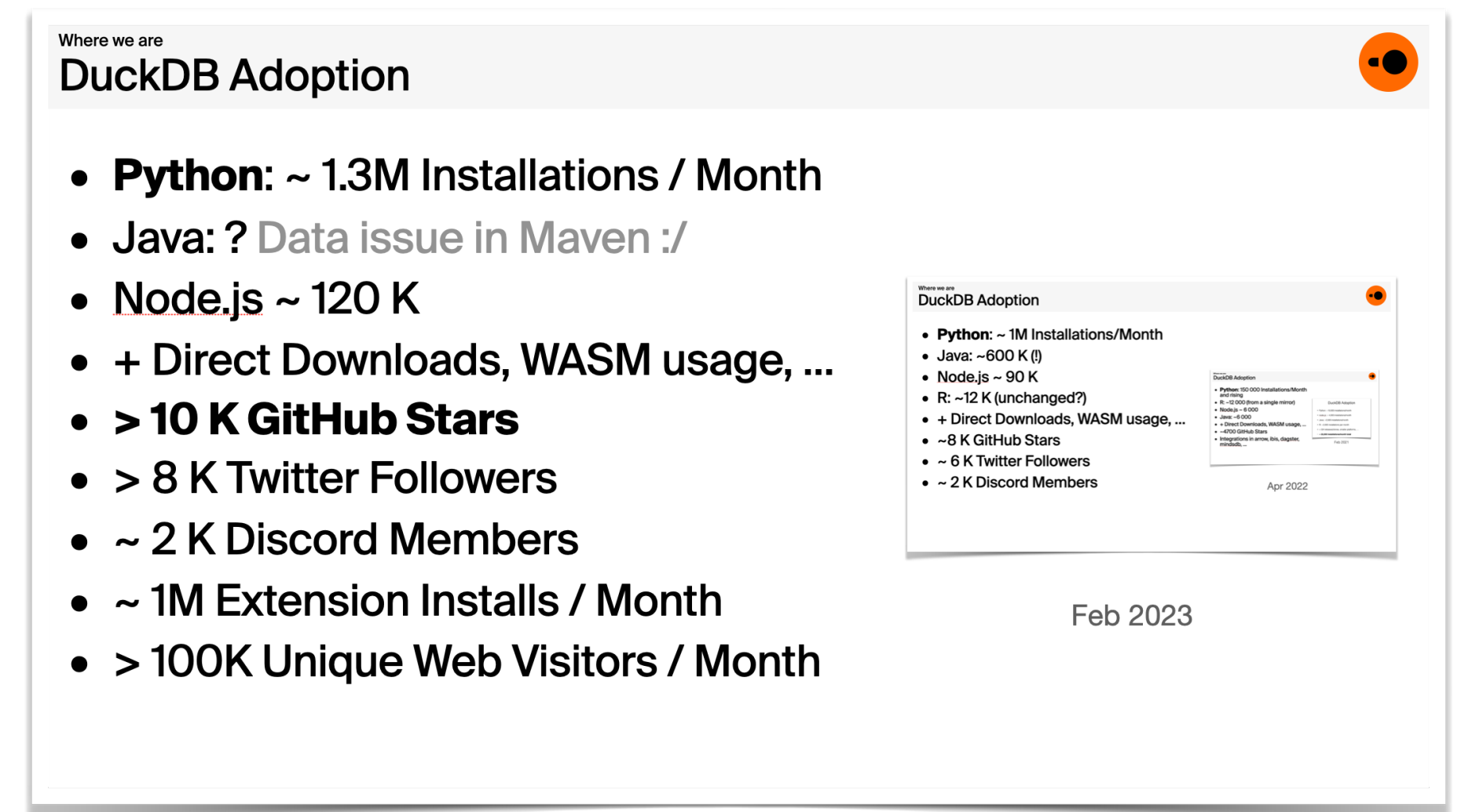


- **Where we are**
 - Adoption
- Where we're going
 - Support Policy
 - Sneak Peek 0.10.0
- Q&A



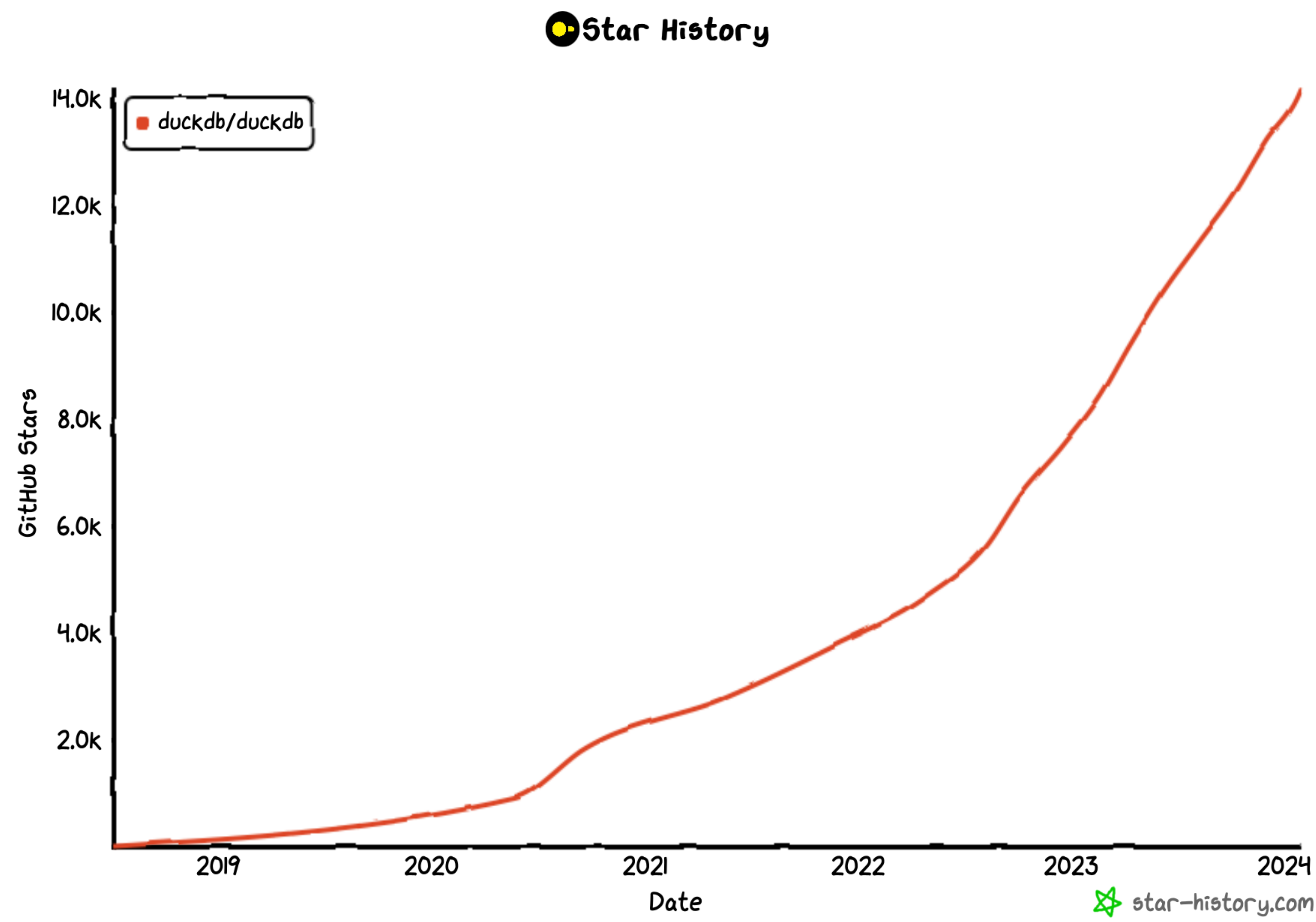
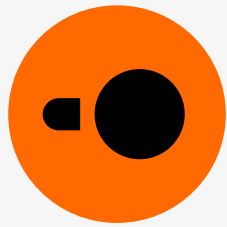
DuckDB Adoption - January 2024

- > 2 M Downloads / Month
- > 5 M Extension Installs / Month
- > 500K Unique Web Visitors / Month
- > 14 K GitHub Stars
- > 11 K Twitter Followers
- > 6 K LinkedIn Followers
- > 4 K Discord Members
- ^ DB-Engines Ranking
Overall: 82, Relational: 45



June 2023

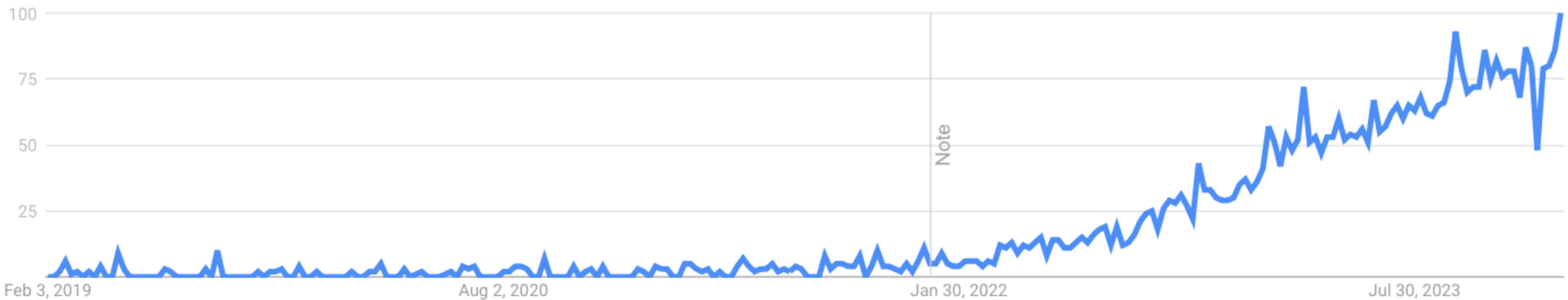
DuckDB Adoption - January 2024



DuckDB Adoption - January 2024

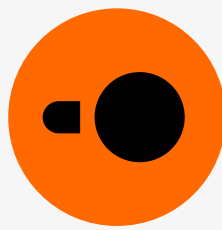


Interest over time 



Google Trends

Where We Are



github.com

☰ davidgasquez / awesome-duckdb 🔍 + ⌚ 🔗 📁 👤

<> Code ⌚ Issues 🔗 Pull requests ⌚ Actions 📁 Projects 🛡 Security 📈 Insights

📄 main awesome-duckdb / README.md 🔍 Go to file t ⋮

davidgasquez

 add web clients ✓ 5ec129e · last week ⌚ History

167 lines (136 loc) · 13.9 KB

Preview Code Blame Raw 📄 ⬇ ✎ ⋮

Awesome DuckDB awesome


A curated list of awesome DuckDB libraries, tools and resources.

[DuckDB](#) is an in-process SQL OLAP database management system.

Contents

- [Resources](#)



 **DuckDB**

Documentation ▾ Blog GitHub ★ 14.2k

Support

DuckDB is a fast in-process analytical database

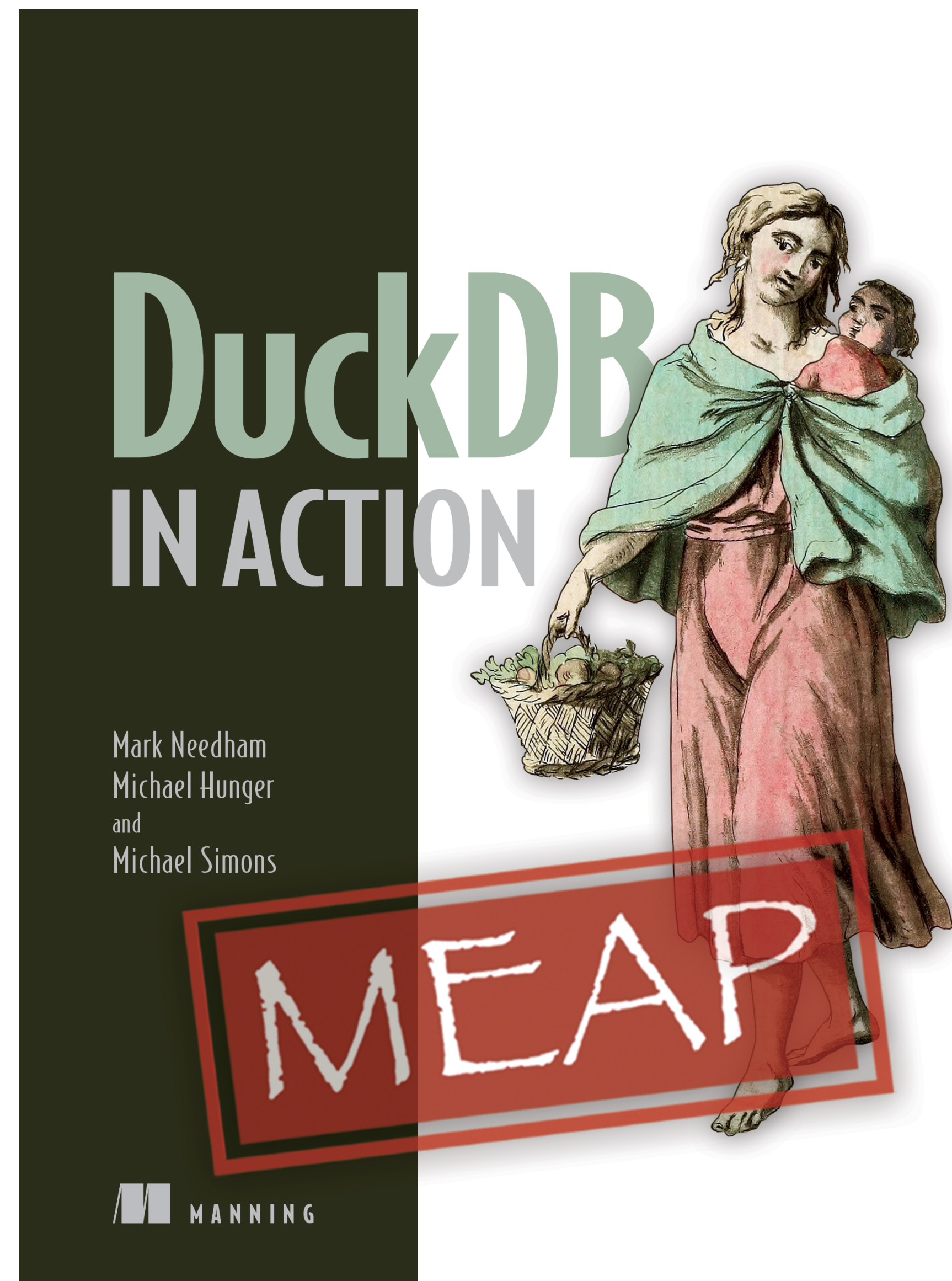
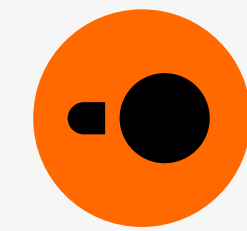
DuckDB supports a feature-rich SQL dialect complemented with deep integrations into client APIs

Installation ▾ Documentation

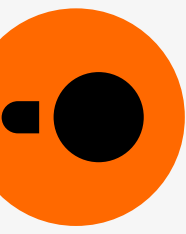
SQL Python R Java Node.js

```
1  -- Get the top-3 busiest train stations
2  SELECT station_name, count(*) AS num_services
3  FROM train_services
4  GROUP BY ALL
5  ORDER BY num_services DESC
6  LIMIT 3;
```

Aggregation Query ✓ Live Demo →



Where We Are





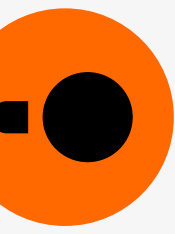
- Where we are
 - Adoption
- **Where we're going**
 - Support Policy
 - Sneak Peek 0.10.0
- Q&A



- Many users, many questions
- DuckDB [Labs] not VC-Funded, no huge DevRel team
- Need to prioritise support effort
 - No guarantees for response time
 - No corporate freeloading
 - No obscure APIs (e.g. Julia, Swift)
 - No internals
 - No weird platforms



- **DuckDB Labs now offers commercial support contracts**
 - Privileged access to DuckDB core team
 - Higher priority issues
 - Guaranteed response time
 - Private issue tracker
 - Private datasets

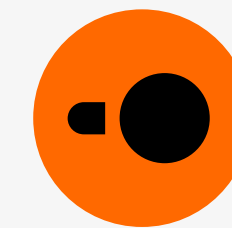


Where we're going

- Release 0.10.0 “Fusca” coming soon (early Feb)
- Release **1.0.0** planned for first half of 2024
 - No new features between 0.10.0 and 1.0.0
 - “Snow Leopard Duck”
 - Series of bugfix-only releases 0.10.1, 0.10.2, ...
- Focus on stability and robustness
- Backwards-compatibility of database file format

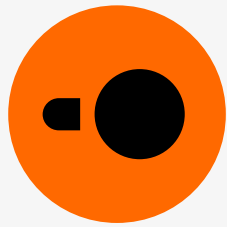






- **1.0: Storage format stabilization**
- **Goals**
- Stability
- Able to **extend and improve** the format
- Backwards compatibility **guaranteed**
 - **Always** able to read older DuckDB files
- Forwards compatibility **best effort**
 - Able to read newer DuckDB files if no new features are used

v0.10.0 - Backwards Compatibility!



- **v0.10.0** -> backwards compatible with v0.9.2
- Can read and write to files created with v0.9.2

```
$ > duckdb v092.db
v0.9.2
D CREATE TABLE lineitem AS FROM lineitem.csv;
$ > build/debug/duckdb v092.db
v0.10.0
D SELECT l_orderkey, l_partkey, l_comment FROM lineitem LIMIT 1;
```

l_orderkey int64	l_partkey int64	l_comment varchar
1	155190	to beans x-ray carefull

- Beta storage stabilization
 - **v1.0 onwards:** always support backwards compatibility

v0.10.0 - Partial Forwards Compatibility



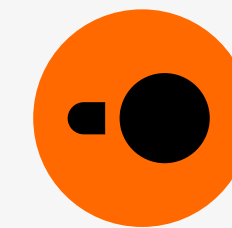
- **v0.10.0** *partially* forwards compatible with v0.9.2
- v0.9.2 can read **some** files created by v0.10.0

```
$ > build/debug/duckdb v010.db
v0.10.0
CREATE TABLE lineitem AS FROM lineitem.csv;

$ > duckdb v010.db
v0.9.2
SELECT l_orderkey, l_partkey, l_comment FROM lineitem LIMIT 1;
```

l_orderkey int64	l_partkey int64	l_comment varchar
1	155190	to beans x-ray carefull

- Files cannot be read if they contain:
 - New compression method (ALP)
 - New types (ARRAY/UHUGEINT)



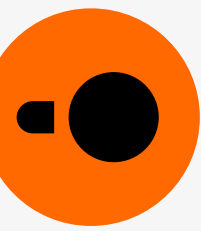
- ATTACH for Postgres/MySQL
- Connect to a running database
- Query as-if they are DuckDB tables

```
ATTACH 'dbname=postgrescanner' AS pgdb (TYPE postgres);  
SELECT c_name, c_address FROM pgdb.customer LIMIT 3;
```

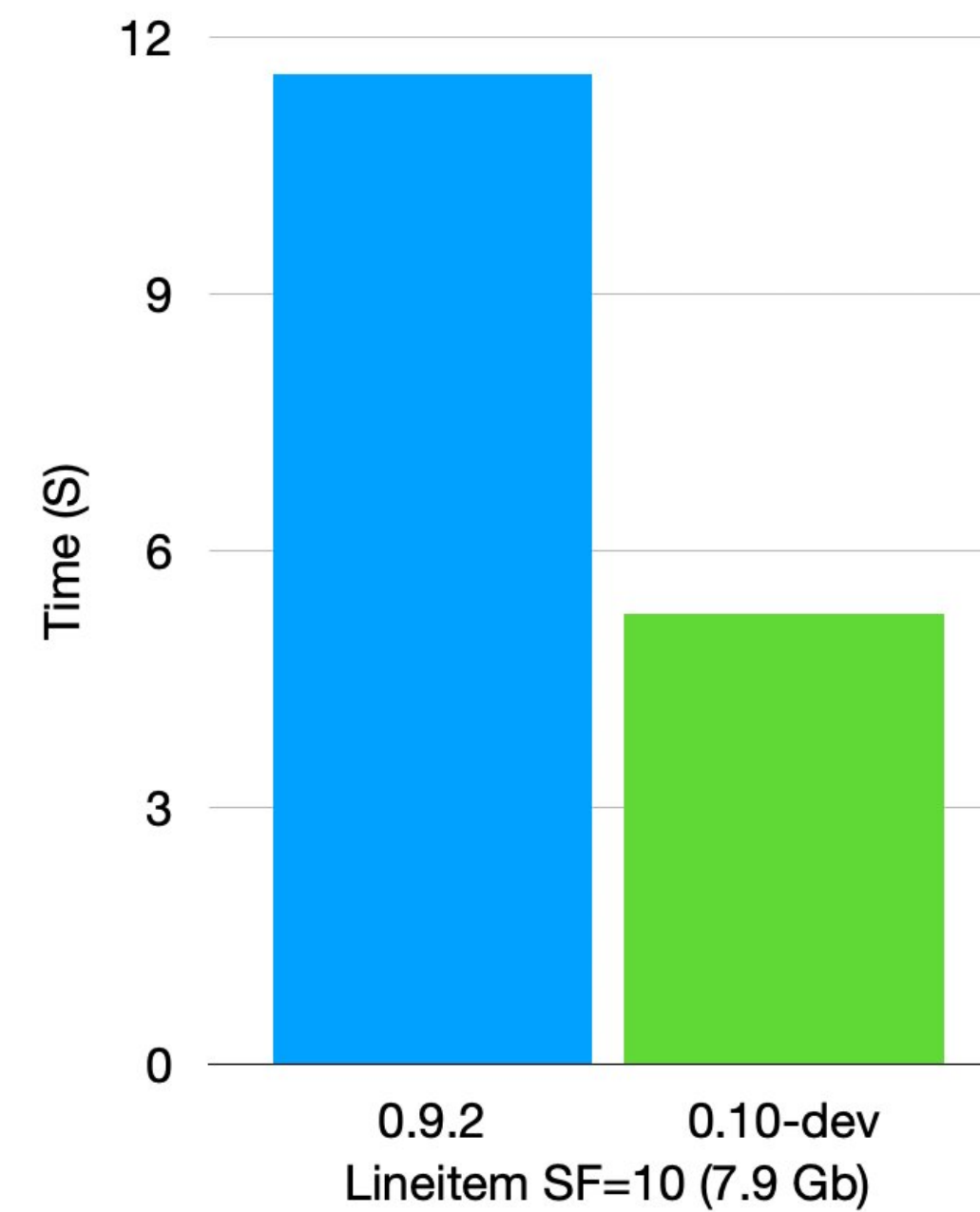
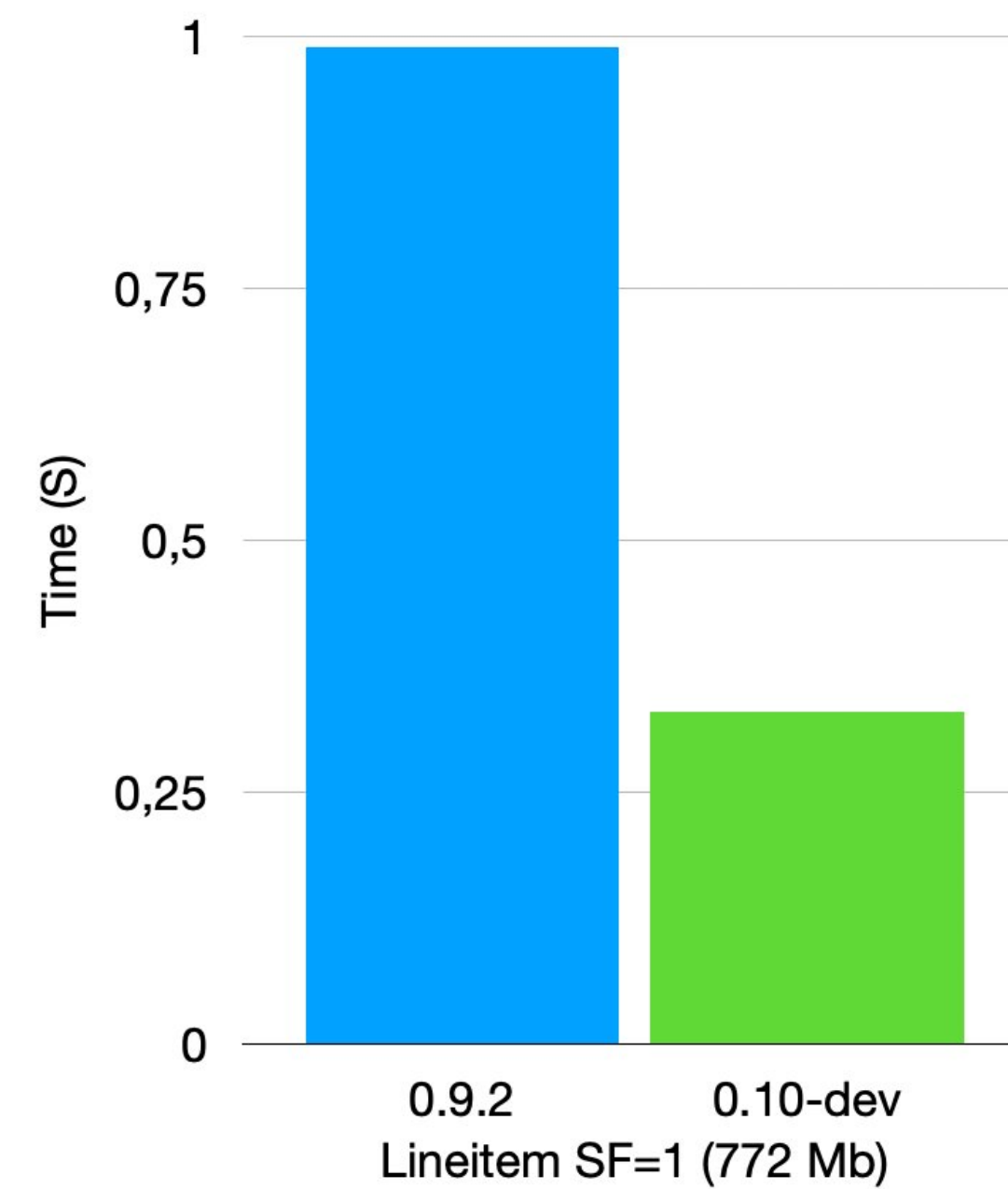
c_name varchar	c_address varchar
Customer#000000001	j5JsirBM9PsCy001m
Customer#000000002	487LW1dovn6Q4dMVymKwwLE90Kf3QG
Customer#000000003	fkRGN8nY4pkE

- Support many operations:
 - INSERT INTO, COPY
 - DELETE, UPDATE
 - Transactions, ALTER TABLE, ...

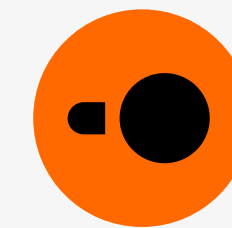
v0.10.0 - CSV Reader Rework



- Major CSV reader overhaul
- Large performance improvement!



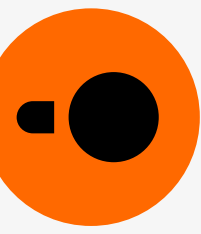
- Many small improvements
- Better error reporting, bug fixes



- New type: fixed-length arrays

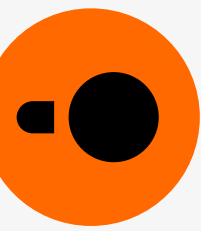
```
CREATE TABLE vectors(v DOUBLE[3]);  
INSERT INTO vectors VALUES ([1, 2, 3]);
```

- Similar to lists, but every entry **must** be the same size
 - Useful for operating on vectors
- New functions:
 - array_cross_product
 - array_dot_product
 - array_cosine_similarity
 - ... and more!



v0.10.0 - Temporary Memory Manager

- DuckDB supports out-of-core **operators**
 - Aggregates, joins, sorting, ...
- Previously operators **did not communicate**
 - Problem when running complex queries!
- v0.10 -> Temporary Memory Manager
 - Communicate memory usage
- Improved memory utilization for complex queries
- Larger-than-memory execution for complex queries



v0.10.0 - Secret Manager

- DuckDB previously managed secrets (S3, ...) through settings
- v0.10 introduces **Secrets** and the **Secret Manager**

```
CREATE SECRET (  
  TYPE S3,  
  KEY_ID 'mykey',  
  SECRET 'mysecret',  
  REGION 'myregion',  
);
```

- **Multi-secret management/scoping**
- **More secure**
- **Persistent**
 - Secrets can be persisted to the DuckDB folder (~/.duckdb/stored_secrets)
 - Automatically available when using DuckDB

v0.10.0 - ALP Compression



- ALP floating point compression was added in v0.10
- Contributed by Azim, Leonardo and Peter from the CWI
- Replaces Chimp/Patas compression
- ALP is faster at both compression and decompression
- ALP achieves a higher compression ratio

Compression	Load	Query	Size
ALP	0.434s	0.02s	184MB
Patas	0.603s	0.08s	275MB
Uncompressed	0.316s	0.012s	489MB

60 million 8-byte floats

ALP: Adaptive Lossless floating-Point Compression

Azim Afrozeh
CWI
Amsterdam, The Netherlands

Leonardo Kuffö
CWI
Amsterdam, The Netherlands

Peter Boncz
CWI
Amsterdam, The Netherlands

ABSTRACT

IEEE 754 doubles do not exactly represent most real values, introducing rounding errors in computations and [de]serialization to text. These rounding errors inhibit the use of existing lightweight compression schemes such as Delta and Frame Of Reference (FOR), but recently new schemes were proposed: Gorilla, Chimp128, PseudoDecimals (PDE), Elf and Patas. However, their compression ratios are not better than those of general-purpose compressors such as Zstd; while [de]compression is much slower than Delta and FOR.

We propose and evaluate ALP, that significantly improves these previous schemes in both speed and compression ratio (Figure 1). We created ALP after carefully studying the datasets used to evaluate the previous schemes. To obtain speed, ALP is designed to fit *vectorized execution*. This turned out to be key for also improving the compression ratio, as we found in-vector commonalities to create compression opportunities. ALP is an adaptive scheme that uses a strongly enhanced version of PseudoDecimals [31] to losslessly encode doubles as integers if they originated as decimals, and otherwise uses vectorized compression of the doubles' front bits. Its high speeds stem from our implementation in scalar code that auto-vectorizes, using building blocks provided by our Fast-Lanes library [6], and an efficient two-stage compression algorithm that first samples row-groups and then vectors.

KEYWORDS

lossless compression, floating point compression, lightweight compression, vectorized execution, columnar storage, big data formats

ACM Reference Format:

Azim Afrozeh, Leonardo Kuffö, and Peter Boncz. 2024. ALP: Adaptive Lossless floating-Point Compression. In *Proceedings of The 2024 International Conference on Management of Data (SIGMOD '24)*. ACM, New York, NY, USA, 14 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 INTRODUCTION

Data analytics pipelines manipulate floating-point numbers (64-bit *doubles*) more frequently than classical enterprise database workloads, which typically rely on fixed-point *decimals* (systems often store these as 64-bit integers). Floating-point data is also a natural fit in scientific and sensor data; and can have a temporal component, yielding *time series*.

Analytical data systems and big data formats have adopted columnar compressed storage [4, 12, 37, 41, 50, 51], where the compression in storage is either provided by general-purpose or

Figure 1: Compression performance for all schemes (on Intel Ice Lake). Each dot is one dataset. ALP is 1-2 orders of magnitude faster in [de]compression than all competing schemes, while providing an excellent compression ratio. The only one to achieve a compression ratio similar to ALP is Zstd, but it is slow and block-based (one cannot skip through compressed data). Elf is inferior to Zstd on all performance metrics. The evaluation framework is presented in Section 4.

lightweight compression. Lightweight methods, also called "encodings", exploit knowledge of the type and domain of a column. Examples are Frame Of Reference (FOR), Delta-, Dictionary-, and Run Length Encoding (RLE) [20, 44, 46]. The first two are used on high-cardinality columns and encode values as the addition of a small integer with some fixed base value (FOR) or the previous value (Delta). These encodings also *bit-pack* the small integers into just the necessary bits. However, with IEEE 754 doubles [1], additions introduce rounding errors, making Delta and FOR unusable for raw floating-point data. General-purpose methods used in big data formats are gzip, Zstd, Snappy and LZ4 [13, 14, 26]. LZ4 and snappy trade more compression ratio for speed, gzip the other way round, with Zstd in the middle. The drawback of general-purpose methods is that they tend to be slower than lightweight encodings in [de]compression; also, they force decompression of large blocks for reading anything, preventing a scan from *pushing down* filters that could *skip* compressed data.

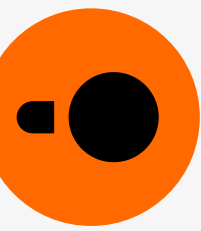
Recently though, a flurry of new floating-point encodings were proposed: Gorilla [38], Chimp and Chimp128 [29], PseudoDecimals (PDE) [31], Patas [24] and Elf [28]. A common idea in these is to use the XOR operator with a previous value in a stream of data; as combining two floating-point values at the bit-pattern level using XOR provides somewhat similar functionality to additions, without the problem of rounding errors. Chimp does an XOR with the immediate previous value, whereas Chimp128 XORs with one value that may be 128 places earlier in the stream – at the cost of storing a 7-bit offset to that value. After the XOR, most bits are 0, and the Chimp variants only store the bit sequence that is non-zero. Patas, introduced in DuckDB compression [24], is a version of Chimp128 that stores non-zero *byte*-sequences rather than bit-sequences. Whereas

SIGMOD '24, June 09–15, 2024, Santiago, Chile
© 2024 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of The 2024 International Conference on Management of Data (SIGMOD '24)*, <https://doi.org/XXXXXXX.XXXXXXX>.



v0.10.0 - Vacuuming Deletes & Parallel Checkpointing

- **Storage improvements**
- During checkpoint, deleted rows are now automatically vacuumed
 - Free up space in on-disk file
 - Speed up queries on tables (defragmentation)
- Checkpoint is now multi-threaded
 - Parallel vacuuming



- Improved multiline editing
- Multiline history
- Multiline becomes the default mode

```
D SELECT
  l_returnflag,
  l_linestatus,
  sum(l_quantity) AS sum_qty,
  sum(l_extendedprice) AS sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
  avg(l_quantity) AS avg_qty,
  avg(l_extendedprice) AS avg_price,
  avg(l_discount) AS avg_disc,
  count(*) AS count_order
FROM
  lineitem
WHERE
  l_shipdate <= CAST('1998-09-02' AS date)
GROUP BY
  l_returnflag,
  l_linestatus
ORDER BY
  l_returnflag,
  l_linestatus;
```

- Faster text rendering



- Extension Ecosystem
- Optimiser Improvements
 - Partition/Sorting Awareness
 - Cardinality Estimation
- Lakehouse Data Formats



- Where we are
 - Adoption
- Where we're going
 - Support Policy
 - Sneak Peek 0.10.0
- **Q&A**

**Drinks and
snacks
sponsored by**

Rill

www.rilldata.com

