Data Blocks:
Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation

Harald Lang¹, Tobias Mühlbauer², Florian Funke³,
Peter Boncz⁴, Thomas Neumann¹, Alfons Kemper¹

¹ Technical University of Munich, ² Tableau Software,
³ Snowflake Computing, ⁴ Centrum Wiskunde & Informatica
Data Blocks:
Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation

Reduce the memory footprint of in-memory OLTP&OLAP database systems

Retain high transaction throughput and high query performance
Basic Assumptions

Large portion of the memory is used for cold data.

Definition: "infrequently updated (still in memory)"
Basic Assumptions

Mostly point accesses through index

Some on cold data
Basic Assumptions

Mostly point accesses through index

Some on cold data
Compression of Cold Data

Mostly point accesses through index
Some on cold data

OLTP

Index

Hot
uncompressed

Cold

compressed
Data Blocks

Secondary Storage

OLAP
The HyPer Approach

Chunked Relation

Cold chunks are transformed into compressed Data Blocks

\[ \text{chunk}_n \]
\[ \text{chunk}_1 \]
\[ \text{chunk}_0 \]

\text{e.g., 128K tuples / chunk}
Data Block Format

- Compressed **columnar** storage format
- Designed for cold data (mostly read)
- **Fast scans** *and* **fast point-accesses**
- Novel index structure
Compression Schemes

- Lightweight compression only
  - Single value, byte-aligned truncation, ordered dictionary
  - All compressed values remain **byte-addressable**! (1, 2 or 4 byte “codes”)
- Efficient predicate evaluation, decompression and point accesses
- Optimal compression chosen based on the actual value distribution
  - Improves compression ratio, amortizes lightweight compression schemes and redundancies caused by blockwise compression
Intra-Block Indexing

**Small Materialized Aggregates** (SMAs) similar to „ZoneMaps“
- Materialization of min/max values of each column
- Used to **skip entire blocks** during scans
Intra-Block Indexing

Novel **Positional SMAs** (PSMAs)

- Fuzzy index on unordered data
- Used to **narrow the scan range** within a block
- Improve scan performance
Intra-Block Indexing

Novel **Positional SMAs** (PSMAs)

- Fuzzy index on unordered data
- Used to **narrow the scan range within** a block
- Improve scan performance

---

![Diagram of Intra-Block Indexing](image.png)
Challenge for JIT-compiling Systems (like HyPer)

- The variety of physical Data Blocks representations either result in
  - multiple code paths → exploding compile times
  - or in interpretation overhead → performance drop at runtime
Vectorization to the Rescue

- Integrate vectorized scan into the tuple-at-a-time JIT query engine
- Specialized scan functions for each compression scheme
- Greatly reduces interpretation overhead
- Fast compile times (independent of the number of storage layouts)
- Comparable runtimes (in many cases faster, due to SIMD)
Evaluation Results

TPC-H (SF100)

- Memory footprint: 60% of the original size
- Query performance improvement: 30% (geomean)
- Compilation times reduced by 50%

TPC-C (5 Warehouses)

- Transaction throughput only slightly decreased (1%)

Byte- vs. Bit-Level Storage (BitWeaving/H)

- Faster predicate evaluation: 1.8x
- Much faster access to individual tuples: 3x
- Space/time trade-off
Summary

The **Data Block** storage format ...

- greatly saves scarce memory resources
- improves performance on a variety of query workloads
- retains high transaction throughput
- integrates well with JIT-compiling query engines
Summary

The Data Block storage format ...

- greatly saves scarce memory resources
- improves performance on a variety of query workloads
- retains high transaction throughput
- integrates well with JIT-compiling query engines

For more details, please join the poster session at 3:30 – 5:00pm (Grand Ballroom A)

You can see Data Blocks in action at the demo session on Tuesday or Thursday, 3:30 – 5:00pm (Garden Room): “High-Performance Geospatial Analytics in HyPerSpace”
Bonus Slides
Positional SMAs

- Supports predicates of type:

  \[ \text{COLUMN} \ op \ \text{constant}, \ \text{where} \ \ op \in \{=, <, \leq, \geq, >\} \]

  \[ \text{COLUMN} \ \text{between} \ a \ \text{and} \ b \]

- Considers only the most significant non-zero byte

- Concise: \( \text{sizeof}(T) \times 2^k \)

- Higher accuracy for small values

- Works best in combination with compression/truncation
SIMD Scan

**Initial predicate**
- Data: Unaligned and aligned data
- Read offset: 11
- Predicates evaluated

**Additional predicates**
- Match positions: 1, 3, 14, 17, 18, 20, 21, 25, 26, 29, 31
- Read offset

**Steps**

**Produce a match vector**
- Lookup: 0, 1, 2, 3, 4, 5, 6, 7
- Precomputed positions table
- Add global scan position and update match vector
- Write offset

**Reduce a match vector**
- Filter: 11, 14, 15, 17
- Precomputed positions table
- Shuffle match vector
- Store