1000 Days of DuckDB
Pareto Principle Still Holds

Hannes Mühleisen
Mark Raasveldt
DuckDB

- In-process OLAP DBMS
- Full SQL support
- No external dependencies
- APIs for C, C++, CLI, Python, R, Java, Node.JS, …
- Extensively tested
- MIT License

www.duckdb.org
D-Day 2018-07-13

Fridays are traditionally reserved for long shots at DA
1000 Days of DuckDB

- ~ 6,500 Commits
- ~ 660 Pull Requests Merged
- ~ 600 Issues Closed
- ~ 2,600 Stars, ~ 166 Forks
- ~ 225,000 LOC (Amalgamation)
DuckDB Adoption

- Python: ~10,000 installations/month
- node.js: ~ 4,000 installations/month
- Java: ~2,500 installations/month
- R: ~2,000 installations per month
- + GH releases/clones, smaller platforms, …

~ 20,000 installations/month total
1. DuckDB – An embeddable SQL database like SQLite, but supports Postgres features (duckdb.org)
   301 points by pcr910303 7 hours ago | flag | hide | 58 comments

2. ▲ Raspberry Pi – UASP, Trim, and Boot Performance via USB (jeffgeerling.com)
   90 points by geerlingguy 4 hours ago | flag | hide | 8 comments
Major Milestones

• **Inter-Pipeline Parallelism** for scans, aggregations etc.

• **Vector format** with compressed execution support and nested types

• OLAP-optimized **Multi-Version Concurrency Control**

• Lock-Free **Buffer Manager**

• **Statistics propagation**

• **Comprehensive SQL** including

  • Subquery folding

  • Recursive Common Table Expressions

  • Collations, Decimals, Window Functions, 100s of Scalar/Aggregate Functions…
Vector Format

- Vectors are core unit of data in DuckDB
  - “Vectorized engine”, “Vector volcano”
- Subset of a logical column of data
- Table: List of lists of vectors.
- Should support compression
- Should support nested types
Vector <> Vector

• Need compressed storage
  • Need to process compressed data

• Naive option: Decompress on Scan
  • Drawback: data & engine explosion

• Better idea: Operate on compressed data directly

• Challenge:
  • Many different compression methods
  • Many operators/functions
  • Implementation effort / code size explosion
Compression + Canonicalisation

• Vectors can be stored in many encodings
  • Constants, plain, Dictionary, RLE, Bit packed, arbitrary expressions, ...

• Database operators are free to implement each case
  • Will be done for performance-critical ops

• But not every SQL function needs to, it can canonicalise the vector into common format
  • What is the canonical format?
Canonical Vector Format

• Naive approach: Decompress into plain C-Style Array
  • Drawback: Slow, large intermediate

• Better: Use “Orrification” canonicalisation into
  • Define Mask (where are the NULLs)
  • Selection List (Offsets into data)
  • Data (plain values)

• Free for flat and constant vectors
**Canonicalisation example: RLE**

<table>
<thead>
<tr>
<th>Define</th>
<th>Offsets</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>true</td>
<td>2</td>
<td>42</td>
</tr>
<tr>
<td>true</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>true</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>true</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

10, 1
42, 5
43, 4
NULL, 1
11, 1

Paper coming!
OLAP MVCC

• Modified HyPer*

• Assume changes commit immediately write to in-memory table data

• Keep undo version per tuple
  • Huge storage overhead for batch updates
  • Retrieval needs version check on every tuple

• Keep undo versions per column chunk

• OLAP updates bulky, column subset

DELETE
FROM orders
WHERE order_time < DATE '2010-01-01'

UPDATE people
SET age=NULL
WHERE age=-99;
OLAP MVCC Benchmarks

- Example: Many single-column updates on table with single column
- Row-based MVCC unfit for use case

```
UPDATE tbl
SET i=i+1
```

### SINGLE COLUMN

<table>
<thead>
<tr>
<th>system</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyper</td>
<td>24,0</td>
</tr>
<tr>
<td>monetdb</td>
<td>13,2</td>
</tr>
<tr>
<td>sqlite</td>
<td>8,9</td>
</tr>
<tr>
<td>duckdb</td>
<td>0,72</td>
</tr>
</tbody>
</table>
OLAP MVCC Benchmarks

- Example: Many single-column updates on table with 100 columns
- Row-based MVCC unfit for use case

```
UPDATE tbl
SET i = i + 1
```

<table>
<thead>
<tr>
<th>system</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyper</td>
<td>149,3</td>
</tr>
<tr>
<td>monetdb</td>
<td>13,2</td>
</tr>
<tr>
<td>sqlite</td>
<td>155,3</td>
</tr>
<tr>
<td>duckdb</td>
<td>0,72</td>
</tr>
</tbody>
</table>

Paper coming!
Next-Gen System Goals

- Environment-Awareness
  - We are not the only one wanting to achieve something on this hardware
  - Allow checkpointing/resuming during queries
- Self-Driving (Hello, Andy)
  - No DBA
- Robustness
  - Environment breaks
  - e.g. CRC-during-scan, appends update checksum
Next-Gen System Goals

- **Make it work first!**
  - Environment-Awareness
    - We are not the only one wanting to achieve something on this hardware
    - Allow checkpointing/resuming during queries
  - Self-Driving (Hello, Andy)
    - No DBA
  - Robustness
    - Environment breaks
    - e.g. CRC-during-scan, appends update checksum
Next-Gen System Goals

- Environment-Awareness
  - We are not the only one wanting to achieve something on this hardware
  - Allow checkpointing/resuming during queries

  - Self-Driving (Hello, Andy)
    - No DBA
    - Robustness
      - Environment breaks
      - e.g. CRC-during-scan, appends update checksum

Pointless in Isolation (Except for Paper)
Next Steps

• Compressed Storage

• Cardinality Estimation

• Native UDFs for Python, R, etc.

• Finalize on-disk representation
  • Blocking release v1.0.0

• Make basic functionality **boring**

• Back to **interesting** issues from back in 2018
Very Special Thanks

- Peter Boncz (History Teacher)
- Till Döhmen (Automatic CSV Parsing)
- Orri Erling (Vectors)
- Denis Hirn (Recursive CTEs)
- Pedro Holanda (Indexing & more)
- Andre Kohn (WASM Compilation)
- Manuel Rigger (SQLancer)
- Richard Wesley (Aggregates)