DuckDB
an Embeddable Analytical RDBMS

Mark Raasveldt & Hannes Mühleisen
About Me

- Mark Raasveldt
- PhD Student @ CWI in Amsterdam
- Database Architectures group
- Supervised by Hannes Mühleisen and Stefan Manegold
- Me & Hannes made DuckDB
Motivation for Using Database Systems

Data Science and Database Systems

DuckDB: Systems Overview
Motivation for Using Database Systems
Why **should** people use relational database systems?

This is a strange question in our field (DBMS research)

**Obviously** everyone should use RDBMSs!

But for many people it is not so obvious

**So why should you actually use a RDBMS?**
Database systems offer ACID properties
  - Consistency, reliability
  - Integrity checks
  - Advanced query optimizers
  - Fast and flexible query execution (SQL)
  - Takes care of data layout for you (in theory)
The Bad

- Schema needs to be defined beforehand
  - Annoying at start of a project when there are many schema changes

- Database systems are **difficult to setup**
- Even PostgreSQL will take you an hour if you are new
- ...and then you still need to learn SQL!
Database systems are **expensive**

Oracle costs **$17.5K per processor**

For the standard edition!

<table>
<thead>
<tr>
<th>Database Products</th>
<th>Named User Plus</th>
<th>Software Update License &amp; Support</th>
<th>Processor License</th>
<th>Software Update License &amp; Support</th>
<th>Prices in USA (Dollar)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle Database</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Edition 2</td>
<td>350</td>
<td>77.00</td>
<td>17,500</td>
<td>3,850.00</td>
<td></td>
</tr>
<tr>
<td>Enterprise Edition</td>
<td>950</td>
<td>209.00</td>
<td>47,500</td>
<td>10,450.00</td>
<td></td>
</tr>
<tr>
<td>Personal Edition</td>
<td>460</td>
<td>101.20</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Mobile Server</td>
<td>-</td>
<td>-</td>
<td>23,000</td>
<td>5,060.00</td>
<td></td>
</tr>
<tr>
<td>NoSQL Database Enterprise Edition</td>
<td>200</td>
<td>44</td>
<td>10,000</td>
<td>2,200.00</td>
<td></td>
</tr>
<tr>
<td><strong>Enterprise Edition Options:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multitenant</td>
<td>350</td>
<td>77.00</td>
<td>17,500</td>
<td>3,850.00</td>
<td></td>
</tr>
<tr>
<td>Real Application Clusters</td>
<td>460</td>
<td>101.20</td>
<td>23,000</td>
<td>5,060.00</td>
<td></td>
</tr>
<tr>
<td>Real Application Clusters One Node</td>
<td>200</td>
<td>44.00</td>
<td>10,000</td>
<td>2,200.00</td>
<td></td>
</tr>
<tr>
<td>Active Data Guard</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>Partitioning</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>Real Application Testing</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>Advanced Compression</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>Advanced Security</td>
<td>300</td>
<td>66.00</td>
<td>15,000</td>
<td>3,300.00</td>
<td></td>
</tr>
<tr>
<td>Label Security</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>Database Vault</td>
<td>230</td>
<td>50.60</td>
<td>11,500</td>
<td>2,530.00</td>
<td></td>
</tr>
<tr>
<td>OLAP</td>
<td>460</td>
<td>101.20</td>
<td>23,000</td>
<td>5,060.00</td>
<td></td>
</tr>
<tr>
<td>Advanced Analytics</td>
<td>460</td>
<td>101.20</td>
<td>23,000</td>
<td>5,060.00</td>
<td></td>
</tr>
</tbody>
</table>
Motivation

- These problems lead to the rise of NoSQL systems
  - Thankfully (almost) everyone now realizes this was a bad idea*
- But these problems are still valid
- Lead to many people using inferior* technology

* In my completely unbiased opinion as RDBMS researcher
Data Science and Database Systems
Data science seems like a prime target for RDBMS

After all, it has “data” in the name!

Data scientists work with data

Thus they need to manage that data!

Yet, many data scientists do not use RDBMS...
Instead of using RDBMS, they have invented their own solutions.

They manage data using flat files:
- CSV files, binary, HDF5, parquet...

They created their own libraries for DBMS ops:
- dplyr, pandas, DataFrames
Flat File Management - what is the problem?

- Manually managing files is cumbersome
- Loading and parsing e.g. CSV files is inefficient
- File writers typically do not offer resiliency
  - Files can be corrupted
  - Difficult to change/update
- It does not scale!
The reason people use it:

```r
# load a CSV file into a DataFrame
df <- read.csv("input.csv", sep="|")
# write a CSV file to a DataFrame
write.csv(df, sep="|")
```
Start by using flat files because they are easy

But then never switch!

At CWI:

Genetics researchers asked us how they could speed up their data loading

...their data was 1TB of CSV files

...that they loaded every time they ran an analysis

Our answer: use a RDBMS!
- `dplyr, pandas, DataFrames` - what is the problem?

- For those unfamiliar: these libraries are basically query execution engines

```
SELECT SUM(l_quantity) FROM lineitem GROUP BY l_returnflag, l_linestatus;
```

```
dplyr
lineitem %>% group_by(l_returnflag, l_linestatus) %>% summarise(sum_qty=sum(l_quantity))
```

```
SELECT * FROM part JOIN partsupp ON (p_partkey=ps_partkey) WHERE p_size=15 AND p_type LIKE '%BRASS';
```

```
dplyr
part %>% filter(p_size == 15, grepl(".*BRASS$", p_type)) %>% inner_join(partsupp, by=c("p_partkey" = "ps_partkey"))
```
dplyr, pandas, **DataFrames** - what is the problem?

The problem is that they are *very poor query engines*!

- Materialize huge intermediates
- **No** query optimizer
  - Not even for basics like filter pushdown
- No support for out of memory computation
- No support for parallelization
- Unoptimized implementations for joins/aggregations
Data scientists need the functionality RDBMSs offer

But they opt not to use RDBMSs

Often this leads to problems down the road

- When the data gets bigger...
- When a power outage corrupts their data...

Can we save these lost souls and unite them with the RDBMS?
DuckDB
an Embeddable Analytical RDBMS
Problem: Databases are difficult to use

What is the easiest to use database?
SQLite is an embedded database

- No external server management
- It has bindings for every language
- Database is stored in a single **file** (not directory)

* https://www.sqlite.org/famous.html
SQLite is great

- It is public domain and very easy to use
- It is secretly the most used RDBMS in the world
  - Runs on every phone, browser and OS*
- It even runs inside airplanes!

* https://www.sqlite.org/famous.html
SQLite has one problem: designed for OLTP

- Row store (basically a giant B-tree)
- Tuple-at-a-time processing model
- Does not utilise memory to speed up computation
- Query optimizer is very limited

Great for OLTP, not so good for analytics
DuckDB: The SQLite for Analytics

Core Features

- Simple installation
- Embedded: no server management
- Single file storage format
- Fast analytical processing
- Fast transfer between R/Python and RDBMS
Why “Duck” DB?

- Ducks are amazing animals
- They can fly, walk and swim
- They are resilient
- They can live off anything
- Also Hannes used to own a pet duck
DuckDB Internals

- Column-storage database
- Vectorized processing model
- MVCC for concurrency control
- ART index, used also for maintaining key constraints
- Combination of both cost/rule based optimizer

- We use the PostgreSQL parser
- Bindings for C/C++, Python and R
DuckDB uses a typical pipeline for query processing.
Life of a Query

```
SELECT COUNT(*)
FROM lineitem, orders
WHERE l_orderkey=o_orderkey
  AND o_orderstatus='X'
  AND l_tax > 50;
```
SELECT COUNT(*)
FROM lineitem, orders
WHERE l_orderkey=o_orderkey
AND o_orderstatus='X'
AND l_tax > 50;
SELECT COUNT(*)
FROM lineitem, orders
WHERE l_orderkey = o_orderkey
  AND o_orderstatus = 'X'
  AND l_tax > 50;
SELECT COUNT(*)
FROM lineitem, orders
WHERE  l_orderkey=o_orderkey
AND  o_orderstatus='X'
AND  l_tax > 50;
SELECT COUNT(*)
FROM lineitem, orders
WHERE l_orderkey=o_orderkey
AND o_orderstatus='X'
AND l_tax > 50;
SELECT COUNT(*)
FROM lineitem, orders
WHERE l_orderkey=o_orderkey
  AND o_orderstatus='X'
  AND l_tax > 50;
Query Execution

DuckDB uses a vectorized pull-based model ("vector volcano")

Each operator calls “GetChunk” on its child operators to fetch an input chunk (= set of vectors)

Scans fetch data from the base tables
DuckDB

HT Build: Vectors flow from right side into HT
After build is completed, chunks flow from left side to root aggregate.
DuckDB is free and open-source

Currently in pre-release (v0.1)

We have a website: [www.duckdb.org](http://www.duckdb.org)

Source Code: [https://github.com/cwida/duckdb](https://github.com/cwida/duckdb)

Feel free to try it

And send us a bug report if anything breaks!
Lessons Learned for Building a RDBMS

Use an existing SQL parser

- Writing a robust parser is difficult!
- PostgreSQL parser saved us so much time

Write many, many tests

- Also steal tests from other systems!

Read all of Thomas Neumann's papers