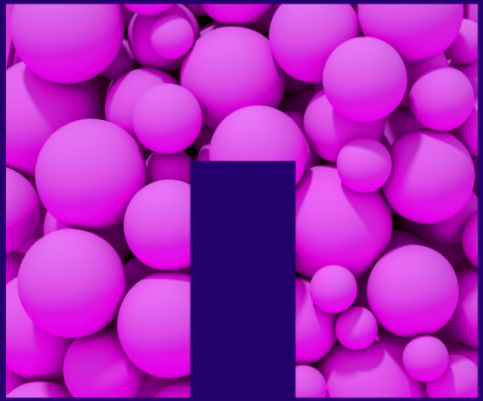
NetApp

PostgreSQL and DuckDB

Combining the best of both Worlds for quicker Data Analysis and ETL processing



Josef Machytka <josef.machytka@netapp.com>
NetApp Deutschland GmbH
2024-10-29 - Prague PostgreSQL MeetUp

Josef Machytka

- Professional Service Consultant - PostgreSQL specialist at NetApp Open Source Services (former credativ).
- 30+ years of experience with different databases.
- PostgreSQL (12y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y).
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 1.5+ years of practical experience with different LLMs / AI including their architecture and principles.
- Originally from Czechia, currently living in Berlin for 11 years.

- LinkedIn: www.linkedin.com/in/josef-machytka.
- ResearchGate.com: https://www.researchgate.net/profile/Josef-Machytka
- Academia.edu: https://netapp.academia.edu/JosefMachytka

Table of contents

Hype around DuckDB

Hype around DuckDB

- Lately there is a lot of hype around DuckDB.
- How great and powerful it is, how it solves (almost) all problems.
- On one PG conference I have seen poster comparing DuckDB to PostgreSQL.
- Of course DuckDB was described as much better then PostgreSQL.
- So I was very curious and started to test DuckDB.
- And I was prepared to really deeply scrutinize DuckDB.
- So, originally I thought this talk will be all about comparing performance.

Hype around DuckDB

- Small explanation for the proper context of my interest in DuckDB.
- In my previous job I was running Data Warehouse on PostgreSQL with almost 200 TB of data.
- And later Data Warehouse on Google BigQuery with almost 3 PB of data.
- Out data pipeline collected daily from 700 GB to 1.5 TB of data.
- These have been aggregated into more then 100 different metrics for clients dashboards.
- Our monthly billing processed between 20 and 40 TB of data.
- So, I am not overly impressed by 1 or 2 GB of data or 100 millions of rows in a table.
- Which the usual amount of data discussed in those hype-articles about DuckDB.

Hype around DuckDB

- But the more I tested, the more I realized, DuckDB is not a competitor to any database.
- DuckDB simply fills the gap where other databases are weak or do not care at all.
- It is still in early production version - 1.1.2.
- It is already very powerful, but still missing some useful features.
- It is definitely a very handy tool for ad-hoc data analysis.
- And safely the fastest database for analytical queries on small datasets I could find.
- But data sets are limited by memory, so it is not a database for big data.
- So I had to re-assess my intentions and testing use cases.

Hype around DuckDB

- With this change I also had to re-focus of this talk.
- I will talk mainly about cooperation between DuckDB and PostgreSQL.
- I will only briefly touch performance, because DuckDB fills its special niche.
- Articles comparing DuckDB to PostgreSQL are not fair to both databases.
- DuckDB is not a competitor to PostgreSQL.
- Currently it is more or less a very powerful ETL tool.
- And can be used even as a simple database migration tool.
- I love DuckDB after testing it, but PostgreSQL remains my favorite database.

What is Data Analysis

Data Analysis

- Data Analysis is the discovery, interpretation, and communication of patterns in data.
- Goal is to support decision-making processes and provide actionable insights.
- Analysis is usually performed with some delay after data collection.
- Typically in batches - hourly, daily, weekly, or monthly.
- Data Analysis answers questions like ”What happened?”, ”Why did it happen?”, ”What will happen?”, etc.
- Majority of Data Analysis involves relatively simple statistical methods.

Functions used for Data Analysis

- Data Analysis uses various functions; SQL standard defines many of them.
- Aggregations - AVG, MIN, MAX, SUM, COUNT, standard deviation, etc.
- Joins - INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, etc.
- Window Functions - RANK, DENSE_RANK, ROW_NUMBER, LAG, LEAD, etc.
- Filtering - WHERE, HAVING, etc.
- Grouping - GROUP BY, etc.
- Sorting - ORDER BY, etc.
- Transformation - CASE, COALESCE, etc.

Big Data vs Small Data

Big Data

- Refers to data sets too large or complex for traditional processing at that time.
- Challenges include data capture, storage, analysis, search, sharing, and transfer.
- Characterized by 3Vs: volume, velocity, and variety.
- Some companies collect a lot of data without sometimes knowing what to do with it.
- PostgreSQL is capable of handling Big Data to some extent.

Small Data Manifesto

- DuckDB and MotherDuck Team wrote The Small Data Manifesto.
- They argue Small Data can be more useful and more accessible than Big Data.
- Modern notebooks and smartphones can process data very quickly.
- These devices are underutilized while we build complex systems for Big Data.
- Small Data is easier to understand, process, and share.
- It is sufficient for many use cases and decision-making processes.
- DuckDB is designed to handle Small Data that fits in memory.

Data Formats for Data Analysis

Most Common Data Formats

- 15-20 years ago, data was mainly stored in relational databases.
- Data analysis was done on classical database tables using SQL.
- Digital world was ”slower”, almost no real-time processing.
- There was enough time to transform data and store it in relational tables.

- Today digital world is very different, real-time processing is new norm.
- We have new technologies and they use different data formats.
- Data analysis must now be able to use these diverse formats.

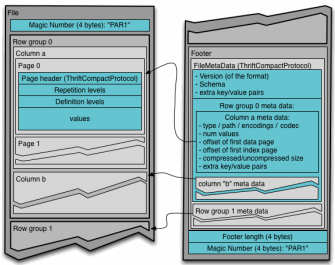
Single File Data Formats

- CSV / TSV - Comma (Tab) Separated Values - row oriented, each line is a record.
- JSON - JavaScript Object Notation - key-value pairs, nested structures.
- Parquet - columnar storage format, optimized for reading.

- YAML - YAML Ain’t Markup Language - human-readable data serialization format, easy to read and write.
- Avro - row oriented, schema-based, binary format.
- ORC - Optimized Row Columnar - columnar storage format, optimized for reading.
- Protocol Buffers - Google’s data interchange format.

Parquet Data Format

- Parquet is a columnar storage format optimized for reading.
- Columns values are stored sequentially, one column after another in row groups.
- Repository: apache/parquet-format.
- Implements mainly numeric data types INT32, INT64, FLOAT, DOUBLE + BOOLEAN.
- Strings are stored as BYTE_ARRAY with STRING annotation.



(Image from the apache/parquet-format repository)

Data Format Frameworks

- Created for managing and processing large data sets.
- Designed to handle different data types and large volumes of data.

- Apache Arrow
- Cross-language development platform for in-memory processing of large data sets.
- Defines a standardized, language-independent columnar in-memory format.
- Idea is to use one in-memory data storage over multiple processes (zero-copy reads).

Data Format Frameworks

- Apache Iceberg
- High-performance format for huge analytic tables on distributed Data Lakes and Lakehouses.
- Supports versioning, partitioning, full schema evolution to track changes to the table over time.
- Implements time travel to query historical data and verify changes between updates.

- Delta Lake
- Open-source storage format for Data Lakehouse architecture, developed by Databricks.
- Focuses mainly on Parquet data format.
- Provides ACID transactions, scalable metadata handling, limited schema evolution, and data versioning.
- Optimized for Apache Spark, but can be used with other engines.

How PostgreSQL handles these Data Formats

- Both implement these data formats as extensions.
- PostgreSQL access these data formats through *foreign data wrappers*.
- We have to create *foreing server* and *foreign tables*.
- For external data files we have to define structure of the foreign table.
- Advantage is that we can use foreign objects as regular tables in our queries and code.

How DuckDB handles these Data Formats

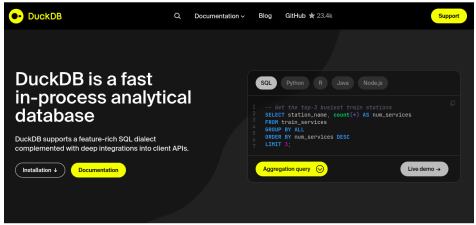
- DuckDB defines *table functions* API.
- Extensions implement it to call external data files as tables in FROM clause.
- Either directly: SELECT * FROM '/data/yellow_tripdata_2024-01.parquet';
- Or through a function (if file does not have expected name): SELECT * FROM read_parquet('mydata.parq');
- We do not need to know structure of data file, DuckDB reads it automatically.
- If structure is not known, DuckDB quickly analyzes the file and determines the structure.
- Advantage is amazing simplicity and speed of usage.

Meet DuckDB

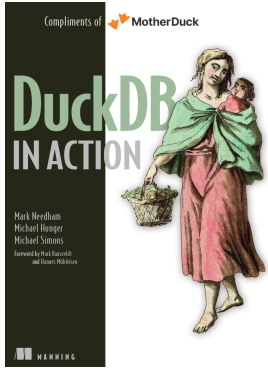
Meet DuckDB

- DuckDB is a an open-source column-oriented, in-memory relational database.
- Designed for analytical workloads, for datasets which fit into memory.
- Single-node database, intended for embedding in applications, like SQLite.
- Repository: github.com/duckdb/duckdb, current version 1.1.2.
- SQL dialect closely follows PostgreSQL conventions.
- New functionality can be added via custom extensions.
- If we start DuckDB without any parameters, it starts in-memory database.
- To persist data, we have to specify database file on startup or use ATTACH command.

DuckDB Resources



(Image from the article DuckDB Main Webpage - duckdb.org)



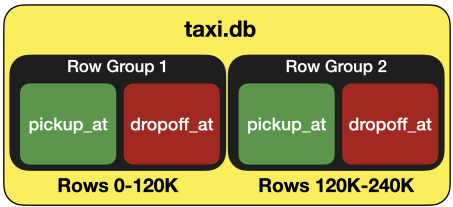
(Image from the article MotherDuck Webpage - motherduck.com)

DuckDB vs PostgreSQL

- It is not a competitor to PostgreSQL, lacks many advanced features.
- DuckDB is a single-node database, not designed for distributed environments.
- Lacks core commands like CREATE USER, CREATE ROLE, GRANT, REVOKE, etc.
- Allows only simple programming using CREATE MACRO command.
- Minimal support for transactions and parallel write access.
- Data size is limited by available memory.
- Spilling to disk is supported but considered an exception use case.
- But can be used alongside PostgreSQL for enhanced performance.

DuckDB Storage

- DuckDB uses a single-file database format with columnar storage and lightweight compression.
- Tables are split into Row Groups of 120K rows, stored in columnar chunks (Column Segments).
- Storage layout is similar to Apache Parquet but columns are split into fixed-size blocks.
- This allows in-place ACID updates, deletes, and adding/dropping columns.
- Storage formats differ between DuckDB versions; backward compatibility is not fully guaranteed.



(Image from the article Lightweight Compression in DuckDB)

DuckDB Data Structures

- DuckDB supports all traditional database data types, with their multiple synonyms.
- DuckDB implements data structures Enums, Lists / Arrays, Maps, and Structs.
- Implements also functions for operations with these objects.
- For example for Maps - map_entries(), map_keys(), map_values(), map_contains(), etc.
- It also has memory variables - SET VARIABLE, SELECT getvariable('name'), etc.

General-Purpose Data Types

The table below shows all the built-in general-purpose data types. The alternatives listed in the aliases column can be used to refer to these types as well, however, note that the aliases are not part of the SQL standard and hence might not be accepted by other database engines.

Name	Aliases	Description
`BIGINT`	`INT8`, `LONG`	signed eight-byte integer
`BIT`	`BITSTRING`	string of 1s and 0s
`BLOB`	`BYTEA`, `BINARY`, `VARBINARY`	variable-length binary data
`BOOLEAN`	`BOOL`, `LOGICAL`	logical boolean (true/false)
`DATE`		calendar date (year, month day)
`DECIMAL(prec, scale)`	`NUMERIC(prec, scale)`	fixed-precision number with the given width (precision) and scale, defaults to `prec = 18` and `scale = 3`
`DOUBLE`	`FLOAT8`,	double precision floating-point number (8 bytes)
`FLOAT`	`FLOAT4`, `REAL`	single precision floating-point number (4 bytes)

(See in the article DuckDB Data Types in docu)

DuckDB CLI

- If we run ”duckdb” command, or start container, we run both engine and command line client.
- Here we can run SQL commands, see results, and use some special commands.
- CLI has ”dot commands” - starting with ”.” - for special operations.
- .help - List all available dot commands.
- .help CMD - Show help for specific command CMD.
- .excel - Display the output of next command in spreadsheet
- .mode MODE - Set the output mode to MODE (box, line, table, csv, json, markdown, etc.)
- .system CMD ARGS... - Run CMD ARGS... in a system shell
- .tables TABLE - List names of tables matching LIKE pattern TABLE
- .timer on|off - Turn SQL timer on or off
- .exit or .quit - Exit the DuckDB shell

DuckDB Extensions

```
D SELECT * FROM duckdb_extensions();
```

extension_name varchar	loaded boolean	installed boolean	install_path varchar	description varchar	aliases varchar[]	extension_version varchar	install_mode varchar	installed_from varchar
arrow	false	false		A zero-copy data integration between Apache Arrow and DuckDB	[]			
autocomplete	true	true	(BUILT-IN)	Adds support for autocomplete in the shell	[]		STATICALLY_LINKED	
aws	false	false		Provides features that depend on the AWS SDK	[]			
azure	false	false		Adds a filesystem abstraction for Azure blob storage to DuckDB	[]			
delta	false	false		Adds support for Delta Lake	[]			
excel	false	false		Adds support for Excel-like format strings	[]			
fts	true	true	(BUILT-IN)	Adds support for Full-Text Search Indexes	[]	v1.1.2	STATICALLY_LINKED	
httpfs	false	false		Adds support for reading and writing files over a HTTP(S) connection	[http, https, s3]			
iceberg	false	false		Adds support for Apache Iceberg	[]			
icu	true	true	(BUILT-IN)	Adds support for time zones and collations using the ICU library	[]	v1.1.2	STATICALLY_LINKED	
inet	false	false		Adds support for IP-related data types and functions	[]			
jemalloc	true	true	(BUILT-IN)	Overwrites system allocator with JEMalloc	[]	v1.1.2	STATICALLY_LINKED	
json	true	true	(BUILT-IN)	Adds support for JSON operations	[]	v1.1.2	STATICALLY_LINKED	
motherduck	false	false		Enables motherduck integration with the system	[md]			
mysql_scanner	false	false		Adds support for connecting to a MySQL database	[mysql]			
parquet	true	true	(BUILT-IN)	Adds support for reading and writing parquet files	[]	v1.1.2	STATICALLY_LINKED	
postgres_scanner	false	false		Adds support for connecting to a Postgres database	[postgres]			
shell	true	true		Adds CLI-specific support and functionalities	[]		STATICALLY_LINKED	
spatial	false	false		Geospatial extension that adds support for working with spatial data and functions	[]			
sqlite_scanner	false	false		Adds support for reading and writing SQLite database files	[sqlite, sqlite3]			
substrait	false	false		Adds support for the Substrait integration	[]			
tpcds	false	false		Adds TPC-DS data generation and query support	[]			
tpch	true	true	(BUILT-IN)	Adds TPC-H data generation and query support	[]	v1.1.2	STATICALLY_LINKED	
vss	false	false		Adds indexing support to accelerate Vector Similarity Search	[]			

24 rows 9 columns

Implemented JOINs

- Usual Joins: LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN (Cartesian product)
- LATERAL JOIN - join with a subquery that can reference columns from the left table.
- NATURAL JOIN - i.e. join done using attributes with the same names.
- SEMI JOIN - returns rows from the left table that have at least one matching row in the right table.
- ANTI JOIN - returns rows from the left table that have no matching rows in the right table.
- POSITIONAL JOIN - always FULL OUTER JOIN, used for ordered data, joins on row number.
- ASOF JOIN - used for time series data, joins on the nearest timestamp.

Connect DuckDB and PostgreSQL

Connect PostgreSQL to DuckDB

- PostgreSQL uses foreign data wrappers to connect to other databases to work with remote data.
- There is a foreign data wrapper for accessing DuckDB database file - `duckdb_fdw`.
- But it allows us to work only with tables already stored in DuckDB database file.
- Here we do not have the ability to use DuckDB as a processing engine.

- For using DuckDB as a processing engine, we need different extensions.
- Currently 2 extensions available: `pg_duckdb` and `pg_analytics` - I will talk about them.

Connect DuckDB to PostgreSQL

- But we can connect from DuckDB to PostgreSQL - it works ”out of the box”.
- DuckDB core contains PostgreSQL extension, it is auto-loaded on first use.
- PostgreSQL database can be connected using ATTACH command.
- In the ATTACH command we specify local synonym for remote database and schema name.
- Login credentials can be added directly into ATTACH command or stored as SECRET.
- PostgreSQL tables are then directly accessible using local synonym and remote table name.
- No need to create any local FDW objects.

Connect DuckDB to PostgreSQL

- Create SECRET for PostgreSQL connection (will not persist if DuckDB runs only in memory):

```
D INSTALL POSTGRES;
D LOAD POSTGRES;
D CREATE PERSISTENT SECRET postgres_test (TYPE POSTGRES, DATABASE duckdb_test, USER 'postgres', HOST 'postgres_container', PORT 5432, PASSWORD 'postgres');
```

Success
boolean
true

```
D .mode line
D SELECT * FROM duckdb_secrets();
         name = postgres_test
         type = postgres
     provider = config
   persistent = true
      storage = local_file
        scope = []
secret_string = name=postgres_test;type=postgres;provider=config;serializable=true;scope;dbname=duckdb_test;host=postgres_container;password=redacted;port=5432;user=postgres
```


Connect DuckDB to PostgreSQL

- Attach PostgreSQL database from DuckDB using SECRET:
- ATTACH ” AS pg_duckdb_test (TYPE POSTGRES, SECRET postgres_test, SCHEMA ’public’);
- We can now see all tables, including those from PostgreSQL database: SHOW ALL TABLES
- And we can directly use PostgreSQL tables from DuckDB using local synonym and remote table name.
- We can now also create remote PostgreSQL tables from DuckDB using CREATE TABLE command.
- Which we will use for easy import of external data into PostgreSQL.

DuckDB as a Intelligent ETL Tool

Import of big CSV file

- Other data formats are getting more and more popular, but this is still a very common task.
- 100 million data set from Kaggle - CSV data for Japan's 100 million customs trade statistics since 1988.
- Total size 4.5 GB (1.1 GB zip), 113,607,322 records, 8 columns, no header.
- File structure only vaguely described on Kaggle as 8 INT columns, no names.

- But there was a problem in column number 5 - numeric values were clearly formatted as strings.
- And some rows were not convertible to integer.
- Other tools analyzed data but marked all columns are integers and later reported errors in data import.
- DuckDB was able to recognize the problem, and set column 5 to VARCHAR and imported all data.
- Question of course is how optimal is this behavior in broader context.

Import of big CSV file

- In DuckDB we can directly select data from CSV file: SELECT * FROM '/data/custom_1988_2020.csv';.
- We can directly import data into DuckDB from CSV file:
  CREATE TABLE custom AS SELECT * FROM '/data/custom_1988_2020.csv';.
- We do not need to check file structure or create table manually.
- DuckDB will automatically create table with 8 columns and import data.
- We can check structure of the data file using DESCRIBE SELECT * FROM '/data/custom_1988_2020.csv';.
- Select automatically recognizes if CSV file has header or not.

```
D DESCRIBE SELECT * FROM '/data/custom_1988_2020.csv';
```

column_name varchar	column_type varchar	null varchar	key varchar	default varchar	extra varchar
column0	BIGINT	YES			
column1	BIGINT	YES			
column2	BIGINT	YES			
column3	BIGINT	YES			
column4	VARCHAR	YES			
column5	BIGINT	YES			
column6	BIGINT	YES			
column7	BIGINT	YES			

```
Run Time (s): real 0.032 user 0.018639 sys 0.013101
```


Import of big CSV file

- We can also immediately produce some statistics:
  SUMMARIZE SELECT * FROM '/data/custom_1988_2020.csv';.
- We can put these stats into table:
  CREATE TABLE custom_stats AS
  SELECT * FROM (SUMMARIZE SELECT * FROM '/data/custom_1988_2020.csv');.

```
D summarize select * from '/data/custom_1988_2020.csv';
100% ▕████████████████████████████████████████▏
```

column_name varchar	column_type varchar	min varchar	max varchar	approx_unique int64	avg varchar	std varchar	q25 varchar	q50 varchar	q75 varchar	count int64	null_percentage decimal(9,2)
column0	BIGINT	198801	202012	368	200512.6231617008	928.6562707338801	199729	200544	201314	113607322	0.00
column1	BIGINT	1	2	2	1.3918669433239588	0.4881672296763373	1	1	2	113607322	0.00
column2	BIGINT	103	703	260	193.28352292962244	121.02486459809657	106	123	222	113607322	0.00
column3	BIGINT	100	900	146	313.31866009794747	179.77390078079418	104	300	428	113607322	0.00
column4	VARCHAR	000000011	970600000	17334						113607322	0.00
column5	BIGINT	0	1255000000000	924549	47876.754848697165	29145919.848440725	0	0	8	113607322	0.00
column6	BIGINT	0	1885789678	1767602	127166.98737165902	4571867.453589752	98	717	6760	113607322	0.00
column7	BIGINT	30	183270402	1394313	32488.302375721876	376908.2117641157	720	2478	10072	113607322	0.00

```
Run Time (s): real 11.671 user 219.519242 sys 1.136306
```


Import of big CSV file into PostgreSQL

- On cloud we can use some ETL tools to load data from external sources.
- Like AWS Glue, Google Dataflow, Azure Data Factory, etc.
- Which generally means to create some very simple pipeline and run it.
- It means some clicking and setting up, but if we know what to do, it is generally easy.

Import of big CSV file into PostgreSQL

- DB GUIs pgAdmin4 or DBeaver have a dialog for creating table from external data file.
- But both are not very user-friendly, require some manual work.
- And import is slow, in DBeaver even extremely slow - few minutes at least.
- There are also some third-party tools for loading data into PostgreSQL, usually commercial.
- There have been extensions ”pgfutter” or ”pgcsv” for loading CSV files.
- But they are not maintained anymore for several years.

Import of big CSV file into PostgreSQL

- To get data into PostgreSQL we usually have to do some manual work.
- For import we have to manually create table with 8 columns and import data using COPY command.
- Total time spend was several minutes - import 63 seconds, but other manual work took few minutes.
- Other work included looking for description on Kaggle, checking file structure, creating table etc.
- To just select data from CSV file we have to use file_fdw.
- It means to create foreign server, user mapping, foreign table, and then select data.

Import of big CSV file using DuckDB

- But we can directly import data into PostgreSQL using DuckDB.
- We attach PostgreSQL database to DuckDB.
- Either using secret or with credentials in ATTACH command.
- And we can create table in PostgreSQL as select from external data file.

```
D ATTACH 'dbname=duckdb_test user=postgres host=postgres_container port=5432 password=postgres' AS pg_duckdb_test (TYPE postgres, SCHEMA 'public');
Run Time (s): real 0.627 user 0.394838 sys 0.082223
D CREATE TABLE pg_duckdb_test.custom_1988_2020 AS SELECT * FROM '/data/custom_1988_2020.csv';
100% [redacted]
Run Time (s): real 88.547 user 36.957726 sys 2.746653
```


Import of big CSV file using DuckDB

- And what about structure of the PostgreSQL table created this way?
- DuckDB handles missing header in CSV file by creating columns as column0, column1, etc.

```
duckdb_test=# \dS+ custom_1988_2020
                                    Table "public.custom_1988_2020"
 Column  |       Type        | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
---------+-------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 column0 | bigint            |           |          |         | plain    |             |              |
 column1 | bigint            |           |          |         | plain    |             |              |
 column2 | bigint            |           |          |         | plain    |             |              |
 column3 | bigint            |           |          |         | plain    |             |              |
 column4 | character varying |           |          |         | extended |             |              |
 column5 | bigint            |           |          |         | plain    |             |              |
 column6 | bigint            |           |          |         | plain    |             |              |
 column7 | bigint            |           |          |         | plain    |             |              |
Access method: heap
```


Import of big CSV file - results

- If we would want to look at this use cases as a competition, DuckDB is a clear winner.
- Import through DuckDB is definitely fastest and easiest when we count all manual work.
- This operation can be also easily automated using very simple script.
- We just need to ATTACH PostgreSQL database, and CREATE TABLE in PostgreSQL AS SELECT from external data file.
- DuckDB even automatically recognizes if CSV file has header or not.
- I love it, I do not need to care about ”bureaucracy”, I can immediately start working with data.

```
D select * from '/data/DataScience_salaries_2024.csv' limit 2;
```

work_year int64	experience_level varchar	employment_type varchar	job_title varchar	salary int64	salary_currency varchar	salary_in_usd int64	employee_residence varchar	remote_ratio int64	company_location varchar	company_size varchar
2021	MI	FT	Data Scientist	30400000	CLP	40038	CL	100	CL	L
2021	MI	FT	BI Data Analyst	11000000	HUF	36259	HU	50	US	L

```
D select * from '/data/custom_1988_2020.csv' limit 2;
```

column0 int64	column1 int64	column2 int64	column3 int64	column4 varchar	column5 int64	column6 int64	column7 int64
198801	1	103	100	000000190	0	35843	34353
198801	1	103	100	120991000	0	1590	4154

Import from other data formats

- Out of the box DuckDB supports CSV, Parquet, JSON, Excel files.
- Support for Apache Iceberg and Delta Lake is currently in development - ”experimental”.
- Import from all these data formats functions exactly the same way.
- Some data formats provide metadata about table structure, so DuckDB uses it.
- If structure is unknown, DuckDB is able to quickly analyze data to determine structure.

- DuckDB also directly supports Hive partitioning for Parquet files.
- Hive partitioning is a strategy for splitting data into multiple files based on partition key.
- Files are stored in sub-directories with partition key and value in the directory name.

Work with Cloud Storage

- DuckDB can directly read data from cloud storage of all main cloud providers.
- The main use case is to read or write CSV or Parquet files.
- Extensions contain functions which allow to automatically fetch credentials from cloud provider.
- The best integration seems to have with AWS S3.

- This also means we can this way migrate data between PostgreSQL and BigQuery.
- BigQuery exports data to GCS or can read data from GCS.
- Using DuckDB we can exchange data between GCS and PostgreSQL both ways.

DuckDB as a very intelligent ETL tool

- We can use DuckDB SQL functions to manipulate data before import.
- This way we can utilize DuckDB as a very intelligent ETL tool.
- In the Radar extension we can find list of third-party extensions for DuckDB.
- Support for other data formats seems to be in active development and progressing.
- Which can make DuckDB even more appealing tool for ETL and data analysis.
- This is a clear niche not covered by other databases.

Database Migration Tool

Migration between Databases

- Migration of data between databases is a common task these days.
- There are many tools available for this task.
- But generally it is not trivial task to migrate data between different databases.
- Especially if we have to migrate data with very special database types.
- In that case some transformation of data is necessary.

Migration from MySQL to PostgreSQL

- I used a very simple MySQL table but with some special data types.

```
-- Create the table
CREATE TABLE special_data_types (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    status ENUM('active', 'inactive', 'pending') NOT NULL,
    permissions SET('read', 'write', 'execute') NOT NULL,
    small_number TINYINT NOT NULL,
    medium_number MEDIUMINT NOT NULL,
    description TEXT,
    data BLOB,
    created_at DATE NOT NULL
);

-- Insert 10 rows of data
INSERT INTO special_data_types (name, status, permissions, small_number, medium_number, description, data, created_at) VALUES
('Alice', 'active', 'read,write', 5, 1000, 'Alice description', 'Alice data', '2023-01-01'),
('Bob', 'inactive', 'read', 10, 2000, 'Bob description', 'Bob data', '2023-02-01'),
('Charlie', 'pending', 'write,execute', 15, 3000, 'Charlie description', 'Charlie data', '2023-03-01'),
('David', 'active', 'read,write,execute', 20, 4000, 'David description', 'David data', '2023-04-01'),
('Eve', 'inactive', 'execute', 25, 5000, 'Eve description', 'Eve data', '2023-05-01'),
('Frank', 'pending', 'read,write', 30, 6000, 'Frank description', 'Frank data', '2023-06-01'),
('Grace', 'active', 'read', 35, 7000, 'Grace description', 'Grace data', '2023-07-01'),
('Hank', 'inactive', 'write,execute', 40, 8000, 'Hank description', 'Hank data', '2023-08-01'),
('Ivy', 'pending', 'read,write,execute', 45, 9000, 'Ivy description', 'Ivy data', '2023-09-01'),
('Jack', 'active', 'execute', 50, 10000, 'Jack description', 'Jack data', '2023-10-01');
```


Migration from MySQL to PostgreSQL

- And this is the whole migration:
  ATTACH MySQL database
  ATTACH PostgreSQL database
  CREATE TABLE in PostgreSQL AS SELECT FROM MySQL

```
D ATTACH 'host=mysql_container port=3306 user=root password=root database=duckdb_test' AS mysql_duckdb_test (TYPE MYSQL);
D SELECT * FROM mysql_duckdb_test.special_data_types;
```

id int32	name varchar	status varchar	permissions varchar	small_number int8	medium_number int32	description varchar	data blob	created_at date
1	Alice	active	read,write	5	1000	Alice description	Alice data	2023-01-01
2	Bob	inactive	read	10	2000	Bob description	Bob data	2023-02-01
3	Charlie	pending	write,execute	15	3000	Charlie description	Charlie data	2023-03-01
4	David	active	read,write,execute	20	4000	David description	David data	2023-04-01
5	Eve	inactive	execute	25	5000	Eve description	Eve data	2023-05-01
6	Frank	pending	read,write	30	6000	Frank description	Frank data	2023-06-01
7	Grace	active	read	35	7000	Grace description	Grace data	2023-07-01
8	Hank	inactive	write,execute	40	8000	Hank description	Hank data	2023-08-01
9	Ivy	pending	read,write,execute	45	9000	Ivy description	Ivy data	2023-09-01
10	Jack	active	execute	50	10000	Jack description	Jack data	2023-10-01

10 rows 9 columns

```
D ATTACH 'host=postgres_container port=5432 user=postgres password=postgres dbname=duckdb_test' as pg_duckdb_test (TYPE POSTGRES, SCHEMA 'public');
D CREATE TABLE pg_duckdb_test.special_data_types as SELECT * FROM mysql_duckdb_test.special_data_types;
```


Migration from MySQL to PostgreSQL

- And how does the PostgreSQL table looks like?

```
duckdb_test=# \dS+ special_data_types
                                        Table "public.special_data_types"
    Column     |       Type        | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
---------------+-------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 id            | integer           |           |          |         | plain    |             |              |
 name          | character varying |           |          |         | extended |             |              |
 status        | character varying |           |          |         | extended |             |              |
 permissions   | character varying |           |          |         | extended |             |              |
 small_number  | smallint          |           |          |         | plain    |             |              |
 medium_number | integer           |           |          |         | plain    |             |              |
 description   | character varying |           |          |         | extended |             |              |
 data          | bytea             |           |          |         | extended |             |              |
 created_at    | date              |           |          |         | plain    |             |              |
Access method: heap

duckdb_test=# select * from special_data_types ;
 id |  name   |  status  |    permissions     | small_number | medium_number |     description     |            data            | created_at
----+---------+----------+--------------------+--------------+---------------+---------------------+----------------------------+------------
  1 | Alice   | active   | read,write         |            5 |          1000 | Alice description   | \x416c6963652064617461     | 2023-01-01
  2 | Bob     | inactive | read               |           10 |          2000 | Bob description     | \x426f622064617461         | 2023-02-01
  3 | Charlie | pending  | write,execute      |           15 |          3000 | Charlie description | \x436861726c69652064617461 | 2023-03-01
  4 | David   | active   | read,write,execute |           20 |          4000 | David description   | \x4461766964206461746      | 2023-04-01
  5 | Eve     | inactive | execute            |           25 |          5000 | Eve description     | \x4576652064617461         | 2023-05-01
  6 | Frank   | pending  | read,write         |           30 |          6000 | Frank description   | \x4672616e6b2064617461     | 2023-06-01
  7 | Grace   | active   | read               |           35 |          7000 | Grace description   | \x4772616365206461746      | 2023-07-01
  8 | Hank    | inactive | write,execute      |           40 |          8000 | Hank description    | \x48616e6b2064617461       | 2023-08-01
  9 | Ivy     | pending  | read,write,execute |           45 |          9000 | Ivy description     | \x4976792064617461         | 2023-09-01
 10 | Jack    | active   | execute            |           50 |         10000 | Jack description    | \x4a61636b2064617461       | 2023-10-01
(10 rows)
```


DuckDB as Database Migration Tool

- This is just a very trivial and always working example.
- Default data types conversions may not fit all use cases.
- But huge amount of simple, straight forward migrations can be done this way.
- And we can use DuckDB SQL functions to transform data during migration.

- Again this is a niche not covered by other databases.
- DuckDB currently supports MySQL, PostgreSQL, and SQLite.
- Extensions for some other databases are in development.
- Over time DuckDB can become a very powerful tool for data migrations.

PostgreSQL Extensions integrating DuckDB

PostgreSQL Extension pg_duckdb

- There are PostgreSQL extensions integrating DuckDB, why not use them?
- Advantage would be direct access to DuckDB functionality from PostgreSQL.
- Their usage requites adding them into shared_preload_libraries.
- Unfortunately these extensions are currently only in early development versions.
- They also do not cover all features and implement older version of DuckDB.
- However they look promising and could be very useful in the near future.

PostgreSQL Extension pg_duckdb

- Extension pg_duckdb is a fairly new project.
- Currently in the very first version 0.1.0.
- Integrates DuckDB and its functions through special schema ”duckdb”.
- But so far implements only a subset of data types for tables.
- Can work only with CSV and Parquet files, and integrates Iceberg extension.
- Documentation explicitly warns, that ”other extensions may work, but is at your own risk.”
- Development is progressing quite quickly, but it is currently in very early stage.

PostgreSQL Extension pg_analytics

- Second project is pg_analytics extension.
- It is a project of the ParadeDB team.
- ParadeDB aims to be ”Elasticsearch alternative built on Postgres”.
- Extension original name was pg_lakehouse, they renamed it to pg_analytics.
- Currently in the version 0.2.1.
- Problems are similar - currently implements older DuckDB 1.0.0, functionality is limited.
- Looks very promising, and development is progressing quickly, but still in early stages.

How to automate DuckDB tasks

Automating DuckDB tasks

- So far we have seen only how to work with DuckDB manually.
- But in real life we need to automate tasks.
- So how to automate ETL tasks which we would want to run in DuckDB?

- Fortunately DuckDB is designed as embedded database for multiple programming languages.
- Of course, solution always depends on the specific environment and requirements.
- Let’s just look at some options.

Automation via Bash scripts

- Bash scripts may not be ”sexy” anymore, but let’s take about efficiency.
- DuckDB CLI can run external SQL scripts or separate SQL commands.
- Encapsulation into a Bash script is very simple.
- Command: duckdb -c ”select count(*) from ’custom_1988_2020.csv’;”
- Script: duckdb -c ”.read myscript.sql”
- Bash control structures can enhance the logic, DuckDB macros are still very basic.
- Script can be scheduled using cron or other scheduler like Apache Airflow.
- This is a very simple and quickly build solution.
- All other solutions will be more complex and require more time to implement.

Automation using dbt (Data Build Tool)

- Data Build Tool (dbt) is quite popular open source tool for automating ETL tasks.
- Written in Python, uses Jinja2 templating language.
- Templates can be used to generate even complex SQL commands and actions automatically.
- Jinja templates allow usage of control structures like loops and conditions in dbt macros.
- Dbt macros can significantly enhance the logic, because DuckDB macros are very basic.
- But generally dbt is not intended primarily as a ETL tool.

Using DuckDB with Apache Airflow

- Apache Airflow is an open-source workflow management platform.
- It is a very powerful environment for automating complex scheduled ETL tasks.
- Includes a web-based UI for managing workflows and monitoring tasks.
- Written in Python, implements ”configuration as code” concept.
- Integration with DuckDB is currently in development.
- But Airflow can run either Bash scripts or dbt code which can use DuckDB.

Summary

Summary

- DuckDB is not a competitor to any database, including PostgreSQL.
- DuckDB is a very powerful complementary tool for all databases.
- It covers the niche where other databases are weak or do not care at all.
- Some ”cheap” hype-articles are misleading and rather hurting the DuckDB reputation.
- DuckDB is a great tool, but it is not a ”silver bullet” for all problems.
- And definitely cannot replace big databases in their typical use cases.
- Is not even intended to be such replacement.

Summary

- DuckDB currently still lacks features that would make it even more useful.
- Many extensions are in ”experimental” phase, and not recommended for production.
- But already working features are very powerful.
- Currently the best use case seems to be in-memory ETL processing.
- Data migrations between MySQL, PostgreSQL, and SQLite are also very easy.
- Integration into data pipelines is still in development.
- I hope DuckDB will continue to grow and improve.
- It is already a great tool and can be even better.

THANK YOU

- Questions?

