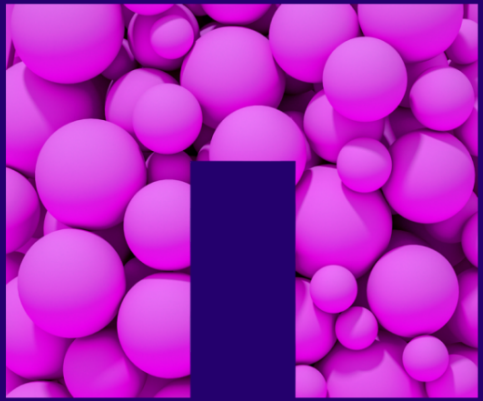
NetApp

PostgreSQL and DuckDB

Supercharging ad-hoc Data Analysis and ETL



Josef Machytka <josef.machytka@netapp.com>
NetApp Open Source Services
2025-01-09 - PostgreSQL MeetUp for All

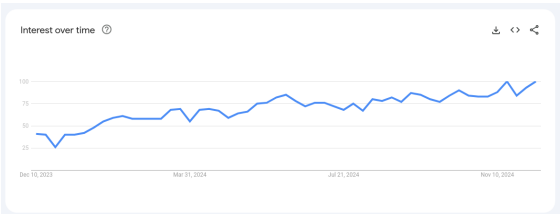
Josef Machytka

- Professional Service Consultant - PostgreSQL specialist at NetApp Open Source Services / 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
- 2 years of practical experience with different LLMs / AI including their architecture and principles.
- From Czechia, living now 11 years in Berlin.

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

Hype around DuckDB

- Significant hype around DuckDB in recent months



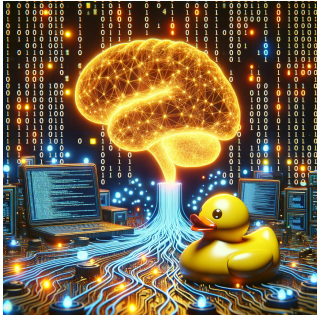
(Search for "duckdb" by Google trends)

How DuckDB handles data not fitting into memory? 4 min read · Nov 13, 2024 · View story	600 Views	354 Reads
Extending DuckDB ETL Capabilities with Python 3 min read · Nov 24, 2024 · View story	532 Views	315 Reads
Easy Cross-Database Selects with DuckDB 2 min read · Nov 26, 2024 · View story	358 Views	175 Reads
DuckDB Database File as a New Standard for Sharing ... 4 min read · Dec 30, 2024 · View story	326 Views	235 Reads
DuckDB as a Rudimentary Data Migration Tool 3 min read · Nov 30, 2024 · View story	241 Views	153 Reads
Using DuckDB as an Intelligent ETL tool for PostgreSQL 4 min read · Nov 2, 2024 · View story	203 Views	128 Reads

(Stats of my DuckDB related articles on medium.com)

Table of contents

- Big Data vs Small Data
- Data Formats
- Meet DuckDB
- Connect DuckDB and PostgreSQL
- Data Analysis and ETL
- Rudimentary Migration Tool

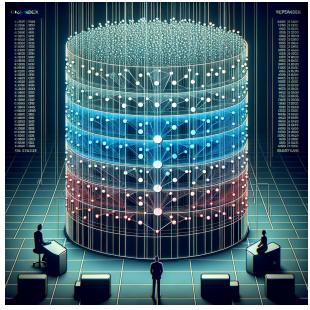


All AI images used in slides
created by the author
using DeepDreamGenerator

Big Data vs Small Data

Big Data

- Data sets too large or complex for traditional processing at that time
- Characterized by 5 V's:
- *Volume*: Large data size, low density of useful information
- *Velocity*: High speed of data in and out, crucial for real-time data
- *Variety*: Different data types - structured, unstructured, semi-structured
- *Veracity*: Data uncertainty, including noise, outliers, and errors
- *Value*: Potential to generate significant value from data



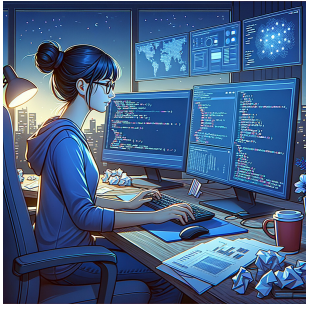
Big Data is (Mostly) Dead

- The boom of Big Data appears to be mostly over
- Companies often collected large data sets without clear usage plans
- Only a small percentage of companies truly need to store Big Data
- Some raw data can be deleted after a short period of time
- Companies mostly need aggregated results
- Useful data sets typically have only a few dozen GBs
- Hardware is advancing rapidly, instances can have TBs of memory
- Refer to the article Big Data is Dead

Small Data Manifesto

- DuckDB Team wrote The Small Data Manifesto
- Small Data can be more useful and accessible than Big Data
- Easier to understand, process, and share
- Sufficient for many use cases and decision-making processes

- Complex systems for Big Data lead to resource competition
- Modern notebooks are powerful but often underutilized



Data Formats

Relational Data are not Predominant Anymore

- 15-20 years ago, data was mainly stored in relational databases
- Data analysis was performed on classical database tables using SQL
- The digital world was ”slower,” allowing time to transform data
- Data was stored in relational tables for structured querying

- Today, real-time processing is the new norm
- Modern technologies utilize diverse data formats
- Data analysis must handle these diverse formats directly

Most Common Data File Formats

- *CSV/TSV* - Comma (Tab) Separated Values - row oriented, each line is a record
- *JSON* - JavaScript Object Notation - key-value pairs, supports nested structures
- *Parquet* - columnar storage format, optimized for efficient reading
- *Avro* - row oriented, schema-based, binary format, supports rich data structures
- *ORC* - Optimized Row Columnar - columnar storage format, optimized for read-heavy workloads

Meet DuckDB

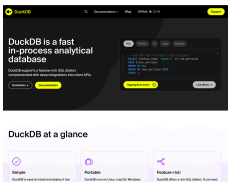
Meet DuckDB

- Created at the National Research Institute for Mathematics and Computer Science, Amsterdam
- Non-profit DuckDB Foundation ensures long-term maintenance and development
- Open-source, column-oriented, in-memory relational database
- Single-node database, intended for embedding in applications, like SQLite
- Designed for heavy parallel analytical workloads
- Columnar-vectorized query processing engine

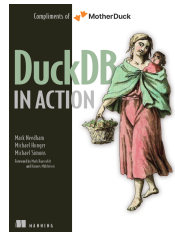
- API support for major languages: Go, Python, R, Java, C++, Rust
- Extremely portable, runs on all architectures, no dependences

DuckDB follows PostgreSQL conventions

- SQL dialect closely follows PostgreSQL conventions
- Uses SQL parser from PostgreSQL
- New functionality can be added via custom extensions
- Uses Multi-Version Concurrency Control (MVCC) for transactions



DuckDB Main Webpage - duckdb.org



MotherDuck Webpage - motherduck.com

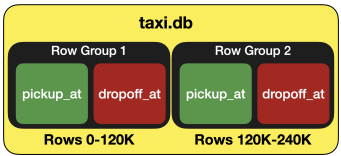
Single-node database

- Single-node database
- Not designed for distributed environments
- Lacks commands CREATE USER / ROLE, GRANT, REVOKE
- Designed mainly for datasets fitting into memory
- Successfully tested with datasets larger than memory

(Picture from Udemy course DuckDB - The Ultimate Guide)

DuckDB Storage

- Single-file database format with columnar storage and lightweight compression
- Layout similar to Apache Parquet but columns split into fixed-size blocks
- Default block size 256 KB - default_block_size parameter (min 16 KB)
- Allows in-place ACID updates, deletes, and adding/dropping columns



(Image from the article Lightweight Compression in DuckDB)

DuckDB Data Structures

- Supports all traditional database data types
- Includes multiple synonyms from older databases
- Python structures: Enums, Lists/Arrays, Maps, and Structs
- Built-in functions for operations with these objects
- Works directly with Pandas, NumPy, and Polars data frames
- Memory variables: SET VARIABLE, SELECT getvariable('name')
- Allows user-defined data types
- Implements UNION data type ("one of these types")

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)

(Documentation DuckDB Data Types)

DuckDB CLI

- Starting ”duckdb” command or container, we run both engine and command line client
- CLI has ”dot commands” - starting with ”.” - similar to psql ”\” commands
- .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)
- .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	[]	v1.1.3	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	[]	v1.1.3	STATICALLY_LINKED	
inet	false	false		Adds support for IP-related data types and functions	[]			
jemalloc	true	true	(BUILT-IN)	Overwrites system allocator with JEMalloc	[]	v1.1.3	STATICALLY_LINKED	
json	true	true	(BUILT-IN)	Adds support for JSON operations	[]	v1.1.3	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	[]	v1.1.3	STATICALLY_LINKED	
postgres_scanner	false	true	/home/josef/.duckdb/extensions/v1.1.3/linux_amd64…	Adds support for connecting to a Postgres database	[postgres]	03eaed7	REPOSITORY	core
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	[]	v1.1.3	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 (FDW) for external data sources
- Access DuckDB database files using `duckdb_fdw`
- Version 1.13 supports basic DML operations

- DuckDB as an analytical and ETL engine: extensions `pg_duckdb` and `pg_analytics`
- Run DuckDB queries and ETL tasks inside PostgreSQL
- Suitable for Data Lakehouse architecture - Apache Iceberg & Delta Lake integration
- Extensions are in early versions (0.2.x), not ready for production

Connect DuckDB to PostgreSQL

- Use the ATTACH command to connect to a remote PostgreSQL database
- Login credentials can be included in the ATTACH command or stored as a SECRET
- PostgreSQL tables accessible with local synonym, and remote schema nad table names
- Perform SQL operations on PostgreSQL tables directly from DuckDB
- Database can be attached also as READ_ONLY

```
INSTALL POSTGRES; LOAD POSTGRES;

-- direct attach with credentials
ATTACH 'dbname=postgres user=postgres password=postgres host=localhost port=5436' as pgdb (TYPE POSTGRES);

-- create secret for PostgreSQL connection
CREATE SECRET local_pg (TYPE POSTGRES, HOST 'localhost', PORT 5436, DATABASE postgres, USER 'postgres', PASSWORD 'postgres');

ATTACH '' AS pgdb (TYPE POSTGRES, SECRET local_pg);

-- work with PostgreSQL tables
SELECT * FROM pgdb.schema.table_name;

CREATE TABLE pgdb.schema.new_table AS SELECT * FROM local_duckdb_table;

CREATE TABLE local_duckdb_table AS SELECT * FROM pgdb.schema.table_name;

-- detach PostgreSQL database
DETACH pgdb;
```


Cross Databases Joins

- DuckDB can join tables from different databases of various versions easily.
- It can also join tables with external data files seamlessly.
- PostgreSQL can achieve the same, but it is more complex and requires more steps.

```
D ATTACH 'host=localhost port=5433 user=postgres password=postgres dbname=test' AS pg13 (TYPE POSTGRES, SCHEMA 'public');
D ATTACH 'host=localhost port=5434 user=postgres password=postgres dbname=postgres' AS pg14 (TYPE POSTGRES, SCHEMA 'public');
D ATTACH 'host=localhost port=5435 user=postgres password=postgres dbname=orders' AS pg15 (TYPE POSTGRES, SCHEMA 'public');
D SELECT
      u.username,
      u.email,
      o.order_date,
      o.total_amount,
      p.product_name,
      od.quantity,
      p.price,
      (od.quantity * p.price) AS total_price
  FROM
      pg13.users u
  JOIN
      pg15.orders o ON u.user_id = o.user_id
  JOIN
      pg15.order_details od ON o.order_id = od.order_id
  JOIN
      pg14.products p ON od.product_id = p.product_id
  ORDER BY
      u.username, o.order_date;
```

username varchar	email varchar	order_date date	total_amount decimal(10,2)	product_name varchar	quantity int32	price decimal(10,2)	total_price decimal(18,2)
Alice	alice@example.com	2024-11-20	150.00	Mouse	2	20.00	40.00
Alice	alice@example.com	2024-11-20	150.00	Keyboard	1	50.00	50.00
Bob	bob@example.com	2024-11-21	250.00	Monitor	1	200.00	200.00
Charlie	charlie@example.com	2024-11-22	300.00	Laptop	1	1000.00	1000.00

Data Analysis and ETL

Testing Data

- Testing tables: `probe_measurements` and `user_data`
- Data generated using Python: 10 GB + 100 GB CSV files for both tables

- `probe_measurements`: PG 15 GB, 143M rows / PG 144 GB, 1.4B rows
- `user_data`: PG 11 GB, 40M rows / PG 111 GB, 400M rows

- Tables fitting into memory are easy to process remotely
- Large tables should be copied to DuckDB for efficient processing

```
CREATE TABLE probe_measurements (
    measurement_id BIGSERIAL PRIMARY KEY,
    probe_id INTEGER NOT NULL,
    measurement_time TIMESTAMP NOT NULL,
    temperature NUMERIC(5,2) NOT NULL,
    pressure NUMERIC(6,2) NOT NULL,
    humidity NUMERIC(5,2) NOT NULL,
    voltage NUMERIC(5,2) NOT NULL,
    current NUMERIC(5,2) NOT NULL,
    resistance NUMERIC(10,2) NOT NULL,
    sample_rate INTEGER NOT NULL);
```

```
CREATE TABLE user_data (
    user_id BIGSERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    signup_date TIMESTAMP NOT NULL,
    last_login TIMESTAMP,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    account_balance NUMERIC(12,2) NOT NULL DEFAULT 0.00,
    country_code CHAR(2) NOT NULL,
    favorite_number INTEGER,
    profile_text TEXT,
    checksum UUID NOT NULL);
```


Direct Work with External Data Files

- DuckDB implements *table functions* API for external data sources
- API allows querying data files as tables in the FROM clause
- Direct access: SELECT * FROM 'mydata.parquet';
- Function-based access: SELECT * FROM read_parquet('mydata.parq', params...);
- Automatically analyzes the structure of data files
- Supports CSV, Parquet, JSON, Excel files, and HTTP requests (cloud storage)
- Supports Apache Iceberg and Delta Lake external tables

Direct Import of Data Files into PostgreSQL with DuckDB

- Create PostgreSQL table directly from external data file using DuckDB
- Comparable speed to direct import into PostgreSQL
- No need for manual PostgreSQL table creation
- DuckDB automatically recognizes structure

```
ATTACH 'dbname=postgres user=postgres password=postgres host=localhost port=5436' as pgdb
(TYPE POSTGRES, SCHEMA 'public');

-- table is directly created in PostgreSQL
CREATE TABLE pgdb.probe_measurements AS SELECT * FROM 'probe_measurements_100gb.csv';

-- insert of data into existing PostgreSQL table
INSERT INTO pgdb.probe_measurements SELECT * FROM 'probe_measurements_100gb.csv';
```


Import of legacy DBF file into PostgreSQL

- Combining DuckDB with Pandas/Polars to read DBF file and write to PostgreSQL

```
import pandas as pd
import duckdb
from dbfread import DBF

table = DBF('dbase_sample_data.dbf', load=True)

dataframe = pd.DataFrame(iter(table))

conn = duckdb.connect(database=':memory:')

conn.execute("""
    ATTACH 'dbname=duckdb_test user=postgres password=postgres host=localhost port=5432'
    AS pg (TYPE POSTGRES, SCHEMA 'public')""")

conn.execute("CREATE TABLE IF NOT EXISTS pg.dbase_sample_data AS SELECT * FROM dataframe")

conn.close()
```


Conversion of data formats

- DuckDB can convert CSV, JSON, and other formats to Parquet
- Parquet format is de facto standard for Data Lakes/ Lakehouses
- Apache Iceberg, Delta Lake, Apache Hudi are all built on top of Parquet
- Can be use also to export data into other formats

```
-- export to Parquet
COPY 'user_data.csv' TO 'user_data.parquet' (format parquet);

COPY (SELECT col1, col2, sum(col3) AS col3_sum FROM dataframe GROUP BY ALL)
    TO 'legacy_dbf_data.parquet' (format parquet);

COPY pgdb.probe_measurements TO 'pgdb_probe_measurements.parquet' (format parquet);

INSTALL avro FROM community; LOAD avro;
COPY (SELECT * FROM read_avro('some_avro_file.avro') ) TO 'avro_data.parquet' (format parquet);

-- export to CSV, JSON
COPY pgdb.user_data TO 'user_data.csv' (format csv);

COPY pgdb.user_data TO 'user_data.json' (format json);
```


Data Analysis Commands

- DuckDB has commands: `SUMMARIZE`, `HISTOGRAM`, `PIVOT`

- `SUMMARIZE` - statistics for all columns of table/data file/data frame
- `HISTOGRAM` - creates a histogram of a column
- `PIVOT` - pivots a table based on a column

- Advanced aggregations and window functions

SUMMARIZE command

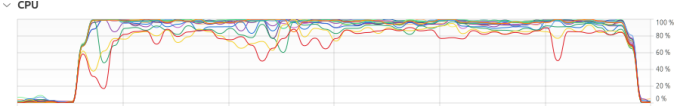
```
D summarize probe_measurements;
100% ▕████████████████████████████████████████████████████████████▏
```

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)
measurement_id	BIGINT	1	1413016000	1527453031	706508000.5	407902584.12096883	353257459	706307700	1059392748	1413016000	0.00
probe_id	BIGINT	1	1000	1232	500.5051361102776	288.67566940333309	251	500	750	1413016000	0.00
measurement_time	TIMESTAMP	2024-12-18 23:39:37	2024-12-30 16:20:48	1109062			2024-12-21 22:39:00.23257	2024-12-24 20:02:28.896443	2024-12-27 17:27:23.33628	1413016000	0.00
temperature	DOUBLE	-50.0	50.0	9552	-2.7343037870909387e-05	28.867546460060938	-24.996470102538034	-0.0011956080965514972	24.990232430893594	1413016000	0.00
pressure	DOUBLE	900.0	1100.0	10127	999.9984155742529	57.73468372202689	949.9449746301059	1000.0098645416814	1049.9773157086229	1413016000	0.00
humidity	DOUBLE	0.0	100.0	7761	50.00012248850552	28.867509878920046	25.807864495794472	50.00865217644044	75.02619651710863	1413016000	0.00
voltage	DOUBLE	0.0	5.0	555	2.500065617607927	1.4433790933627602	1.2505033211815084	2.5001711268049593	3.7460747999880537	1413016000	0.00
current	DOUBLE	0.0	10.0	1177	5.000136832894723	2.886672093004417	2.499375000540707	5.001974925063086	7.499162047293135	1413016000	0.00
resistance	DOUBLE	0.0	10000.0	1012389	5000.003452122564	2886.755067209653	2502.1300079043476	5000.173776842059	7500.563493510334	1413016000	0.00
sample_rate	BIGINT	1	1000	1232	500.5025258864726	288.67384154007715	251	500	751	1413016000	0.00

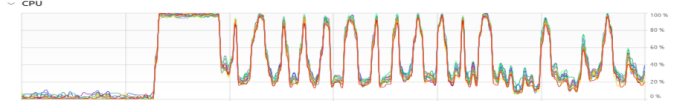
10 rows 12 columns

```
Run Time (s): real 257.893 user 4876.925018 sys 11.167547
```

Local DuckDB table:



Remote PostgreSQL table:



HISTOGRAM command

```
select
    key_value['key'] as bucket_max,
    key_value['value'] as data_points
from (
    select unnest(map_entries(histogram(round(humidity/10)*10))) as key_value
    from probe_measurements where humidity is not null);
```

bucket_max double	data_points uint64
0.0	70589516
10.0	141280714
20.0	141311460
30.0	141304167
40.0	141291430
50.0	141299513
60.0	141312375
70.0	141298756
80.0	141312497
90.0	141287019
100.0	70728553
11 rows	2 columns

```
Run Time (s): real 4.541 user 89.779258 sys 0.040516
```


PIVOT command

- Can be very memory intensive, depends on the number of columns and rows

```
PIVOT (
    SELECT probe_id, measurement_time, temperature
    FROM probe_measurements
    WHERE measurement_time BETWEEN '2024-12-01' AND '2024-12-30')
ON measurement_time::date
USING AVG(temperature) AS avg_temperature GROUP BY probe_id;
```

```
100% ▕████████████████████████████████████████████████████████████▏
100% ▕████████████████████████████████████████████████████████████▏
```

probe_id int64	2024-12-18_avg_tem… double	2024-12-19_avg_tem… double	2024-12-20_avg_tem… double	2024-12-21_avg_tempera… double	2024-12-22_avg_temperature double	2024-12-23_avg_temperature double	2024-12-24_avg_temperature double	2024-12-25_avg_temperature double	2024-12-26_avg_temperature double
1	-0.8933333333333334	0.008709499040937409	-0.09530979815314192	0.019285515229009003	0.003220654581248706	0.030594762176715293	0.049906470232352514	-0.06471281550048891	-0.09223871894888122
2	-1.8681481481481481	-0.011747858418378…	-0.19133368175211105	-0.06358527768649369	0.11331738802791523	-0.05397422663520763	0.038526346907102126	0.04089322668593745	0.03326487629381347
3	0.5514285714285714	0.03776647082605725	0.27300376356349165	0.14048107877274832	0.04094226047252625 5	0.1567063639133907	0.017888198249105095	0.07009628430843685	0.06914989786961741
4	0.15520325203251978	-0.08069218717719157	0.08583476862546635	-0.061396022144761045	0.07696832505 09632	-0.06567259049996733	0.08130447964208755	-0.017244264661 5069	0.14819615052958746
5	0.5163888888888886	-0.045780490736447…	0.019536422214398 72	-0.015513540087672686	-0.027953410029353885	-0.043304115665817676	-0.10062423870835925	0.044650833762686366	-0.039236870293297824
6	1.0165486725663717	0.029071114059955 56	0.08107406316960285	0.017795231646764467	0.016278338778338834	-0.009425380660713982	0.011424546126673877	0.05181201463163142	-0.060968557886722965
7	0.07568807339449515	0.14391917325689887	0.05512631347262192	-0.09983962752198666	0.12788043033959542	0.023149875470 03965	0.09259896659081773	0.03175231268862381	-0.11223802652937381
8	2.8346	0.11482386883886428	0.0952209829294477	0.06277199528672446	-0.04637323943661974	0.049361163872150 63	0.08528180870022256	-0.11501692331647842	0.007993704195563745
9	1.545505617977528	0.09855607778879746	-0.03765412237862679	0.04697267741617682	0.09843688621186 3	0.003490696664563633 5	-0.07644541693058163	0.03566836608831525	0.022228149866832705
10	-0.6838000000000005	-0.2049289678612912	-0.0840726499222777	-0.05535888615888627	-0.10882201709066891	-0.008429370902406003	-0.0665516675689418	-0.08402811202388617	0.0457466794647765 8
11	-3.7764347826086957	-0.10416217933853793	0.07367137134759218	0.10342240982852607	-0.04006488872117565	-0.13273488624141	0.11913655242331092	0.03771748698163031	-0.12711236120664257
12	3.520606060606061	0.21009271603441998	0.10893850718864242	-0.055756935746621085	-0.19829790403058312	-0.13930400294394235	-0.11725233229675706	-0.048178956687104875	0.13915531668488107
13	-0.44719626168224325	-0.04059024910542245	0.05239493131063224	-0.021405589594050686	-0.0011683645848713153	-0.0732388614165868	-0.07923209493561773	-0.12060021576383678	0.003944801758528082
14	-2.0013392857142853	0.07316068336305559	-0.05005799448684292	-0.022370493094058825	-0.021022986905570914	-0.026031002546695 79	-0.015378131950200886	-0.0021882640586795158	-0.00900708346671255
15	-2.2092391304347827	-0.005144966824154…	-0.056626768137670…	0.0021320367634987407	-0.06956876133226599	0.05630958832126371	0.06073610769104472	-0.05806551749624292	-0.12771957966412417
16	-5.104608695652174	-0.18033826238143624	0.012928384872972373	-0.05957034285854526	-0.1248524541908897	0.14371292281716902	0.028998356352574653	-0.14699142960146863	-0.13006180449531746
17	4.781382978723403	0.06598824767887339	0.16312828213207278	-0.07717688773880818	0.08023023145652645	-0.038632613255204624	0.12279314909542 57	-0.0638324246771877	-0.019261811588852335
18	0.09867256637168134	-0.002597352359666…	0.03803206675119608	0.05646905735349647	-0.026712692468430676	-0.03410211253184339	0.03625656149113234	-0.11965561067606904	0.010702908332310832
19	2.0930851063829787	-0.011483227106449…	-0.0293494908417101	-0.1291783334152051	-0.0692980734635137	0.14444814353188123	-0.05976907327498298	0.09958658140646381	-0.11974303370236905
20	0.6865853658536589	-0.10475527495400905	-0.02643816057244097	0.0533247916955696	-0.0009146426206459616	0.11571361173925065	0.022602637675702048	-0.09403047193741465	0.002290704743752001
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
981	6.274818181818179	-0.05899642646971043	-0.1570124146036877	-0.014281849483521776	-0.04045517241379281	-0.003816195425344581	0.07853187768643646	0.09794127029569331	0.19274950988032052
982	4.146236559139786	-0.006210066473466…	-0.12709462668479055	0.07320853030092006	0.09433868171772686	0.02740642765600864	0.05038462171323293	0.021300503299589747	-0.06984338305689465
983	-2.2424489795918374	0.12922897296554808	-0.035789563729683…	-0.1103674216725650 8	0.04802378729050938	-0.08357195982117317	0.21014926105554138	0.010941278560493987	0.07014688635375733
984	-4.861891891891892	-0.014959834176213…	-0.06395120195650247	0.10667740909053824	-0.03976035553809516	-0.002612354711966745 4	-0.05507349622054567	0.05915719751511917	0.09380629357452414
985	-1.8353846153846152	0.003296385458702747	0.09753330664989415	0.143567116869926 9	-0.04384037874561036	0.018234836639296325	0.07940076101632507	-0.11911601216362022	0.010995028956736944
986	-4.523727272727273	0.02192584379754014	0.018245244860027404	-0.02690612438702915	-0.12688271250880853	-0.0670026116928517 2	0.02006191975855365	0.02310269357509417	0.0663664031718091
987	0.9406930693069301	-0.17302650062481162	-0.12358152747873492	-0.04427029467290447	-0.03428757090778281	0.06328289824187537	0.05533612412753531 6	-0.046453115373557 34	-0.012468752299127486
988	-0.969724770642202	-0.028599005785152 37	-0.032572301558772…	0.024569387822198623	0.0984674091057978	-0.03651586726799885	0.09136342015483562	0.05416257389536166 5	0.027020663024277823
989	3.5601886792452837	0.19396788557085634	0.09489573405199349	-0.1713985185185187	-0.035478911486491786	-0.1703186132014444	-0.07490180823533273	0.019859791986471836	0.1493699100572366
990	-4.490602409638554	0.01268443877331072	0.13265936654099988	-0.1605713371668563	-0.10797332809391674	-0.0009497930450082465	0.04845249586885043	0.009286847312391022	0.24088746501693964
991	-2.127849462365591 5	-0.06413519809512394	0.0363411325318931	0.052666699241017564	-0.0505648194429647 3	0.0523471131412049 7	0.08047357791714015	0.018696163203577074	0.01148671021093994
992	-0.5800917431192661	-0.026366627901968…	0.06159776005763623	0.07686053072739 9	-0.09336459025538801	-0.0893642137574084 7	-0.08328302411570292	-0.0237359240268411	-0.028371934995292142
993	3.4088990825688077	-0.03593746508409891	-0.03272355366839503	-0.05592654793308554	-0.018977401637137 21	-0.07400251823265844	0.13042357742231586	-0.04385997567207952	0.019602749319999 08
994	1.0148872180451125	0.0610259921455148	-0.020181514913658…	0.09304260496105286	0.13016825248526817	0.06138583308777758	-0.0036500535774170563	-0.03546506292606164	0.03636593096012162
995	-2.299230769230769	0.20774707867194434	0.0441590812000921	-0.05555994590366966	0.002148434936675540 4	0.026034839909312 09	0.019166988044735343	-0.08367477792555333	0.04658873062796706
996	1.4535245901639344	0.02414919136423424	0.04559683800928663	-0.13891605347874628	0.03920472066397636	-0.0035686485692604194	-0.061708895541975 72	-0.17864845687067693	0.06099960777263887 4
997	-2.591138211382115	-0.08115191341321976	-0.06748401691747823	-0.12257804855989413	-0.07097760925112996	-0.10919452675471404	-0.17215660661030743	0.0323177023430053	-0.04465563971340 8
998	2.2800000000000002	-0.06084394631148937	0.13064412371134018	-0.13457757095103903	0.090096591467208	0.1222070553274169	-0.08088398611799709	0.04015635482973587	0.039341899834064886
999	1.070101010101009	0.17764822066501784	0.017009660013448336	-0.10710369312385247	-0.07052009436965798	-0.03042672010249160 3	0.03832793897132945	0.04332702004673473 5	0.045120938167013916
1000	-0.646165413533835	0.029340945011098043	0.04084012128165192 4	-0.009584006728454192	0.09046170359202921	-0.07955311517295559	-0.05940255483169583	0.08800570335991628	0.1088689106056012 4

1000 rows (40 shown)

Run Time (s): real 16.434 user 322.946683 sys 0.043173

Direct work with data file

- Functions like SUMMARIZE, HISTOGRAM, PIVOT can run on external data files
- DuckDB uses parallel readers to process large data files efficiently
- Enables real-time data analysis without the need for data import

```
D summarize from 'probe measurements 100gb.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)
measurement_id	BIGINT	1	1413016000	1527453031	706580000.5	407902584.12895525	353460881	706262751	1059619539	1413016000	0.00
probe_id	BIGINT	1	1000	1232	500.5053361102776	288.67566940331545	250	501	750	1413016000	0.00
measurement_time	TIMESTAMP	2024-12-18 23:39:37	2024-12-30 16:20:48	1109862			2024-12-21 22:32:59.337439	2024-12-24 20:00:33.590275	2024-12-27 17:28:59.862436	1413016000	0.00
temperature	DOUBLE	-50.0	50.0	9552	-2.7343037807649e-05	28.867546460680487	-24.978619831225555	0.0010965803637129083	24.990570133714 63	1413016000	0.00
pressure	DOUBLE	900.0	1100.0	10127	999.9904155742769	57.734683722824424	950.0124387587778	1000.0158339945617	1049.9635827052468	1413016000	0.00
humidity	DOUBLE	0.0	100.0	7761	50.000122480585086	28.867509878920785	24.976093831 4025	49.98795528933654	75.00055998729647	1413016000	0.00
voltage	DOUBLE	0.0	5.0	555	2.5000656617687933	1.4433709933626929	1.2510427633778147	2.4994305489855044	3.7492250822997946	1413016000	0.00
current	DOUBLE	0.0	10.0	1177	5.000136832894714	2.886672093004267 3	2.501413570725295	4.999466466550448	7.498089966103736 5	1413016000	0.00
resistance	DOUBLE	0.0	10000.0	1012389	5000.003452122519	2886.7550672095927	2501.0407087575295	4999.707653837607	7498.754915867427	1413016000	0.00
sample_rate	BIGINT	1	1000	1232	500.5025258864726	288.6738415400651	250	501	750	1413016000	0.00

10 rows 12 columns

Run Time (s): real 306.288 user 5808.848000 sys 47.870835

```
D PIVOT (
    SELECT probe_id, measurement_time, temperature
    FROM 'probe measurements 100gb.csv'
    WHERE measurement_time BETWEEN '2024-12-01' AND '2024-12-30')
  ON measurement_time::date
  USING AVG(temperature) AS avg_temperature GROUP BY probe_id;
100% ▕████████████████████████████████████████▏
100% ▕████████████████████████████████████████▏
```

probe_id int64	2024-12-18_avg_tem… double	2024-12-19_avg_tem… double	2024-12-20_avg_tem… double	2024-12-21_avg_tem… double
17	4.781302970723404	0.06598024767887 33	0.16312828213207278	-0.07717680773800788
915	4.841684210526315	-0.009986179062481…	-0.00843102160116 2…	0.17954731868961 14
143	-1.9576033057851239	0.1455837585253132	0.07717094975725977	-0.12720375882512 11
9	1.5455056179775282	0.09855607778879723	-0.03765412237862668	0.04697267741617673
693	-1.7113131313131328	-0.01376376134934 4…	0.09752592871959302	0.13564543425474062

Rudimentary Migration Tool

Migration from MySQL to PostgreSQL

- Example: Migrating a simple MySQL table with 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

- Steps for 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

- Result: PostgreSQL table after migration

```
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)
```


Migration from MS SQL to PostgreSQL

- Combining DuckDB with Polars to read MS SQL table and write it to PostgreSQL

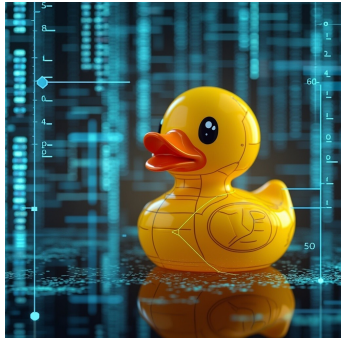
```
import pyodbc
import polars as pl
import duckdb

conn_str = "DRIVER={ODBC Driver 17 for SQL Server};SERVER=localhost,1433;DATABASE=master;UID=sa;PWD=..."
connection = pyodbc.connect(conn_str)

query = "SELECT * FROM dbo.small_mssql_table"
df = pl.read_database(query, connection)
connection.close()

duckdb_conn = duckdb.connect(database=':memory:')
duckdb_conn.execute("""
    ATTACH 'dbname=duckdb_test user=postgres password=postgres host=localhost port=5432'
    AS pg (TYPE POSTGRES, SCHEMA 'public')""")

duckdb_conn.execute("CREATE TABLE pg.small_mssql_table AS SELECT * FROM df")
duckdb_conn.close()
```


Migration from MS SQL to PostgreSQL - Batch Processing

```
import pyodbc
import polars as pl
import duckdb

source_table_name = 'dbo.big_mssql_table'
target_table_name = 'pg.big_mssql_table'

batch_size = 1000
conn_str = "DRIVER={ODBC Driver 17 for SQL Server};SERVER=localhost,1433;DATABASE=master;UID=sa;PWD=..."
connection = pyodbc.connect(conn_str)

mssql_cursor = connection.cursor()
query = f"SELECT * FROM {source_table_name}"
mssql_cursor.execute(query)
rows = mssql_cursor.fetchmany(batch_size)

schema = {column[0]: pl.datatypes.DataType.from_python(column[1]) for column in mssql_cursor.description}
df = pl.DataFrame([dict(zip(schema.keys(), row)) for row in rows], schema=schema, orient="row")

duckdb_conn = duckdb.connect(database=':memory:')
duckdb_conn.execute("""
    ATTACH 'dbname=duckdb_test user=postgres password=postgres host=localhost port=5432'
    AS pg (TYPE POSTGRES, SCHEMA 'public')""")
duckdb_conn.execute(f"CREATE TABLE {target_table_name} AS SELECT * FROM df")

while rows:
    rows = mssql_cursor.fetchmany(batch_size)
    if rows:
        df = pl.DataFrame([dict(zip(schema.keys(), row)) for row in rows], schema=schema, orient="row")
        duckdb_conn.execute(f"INSERT INTO {target_table_name} SELECT * FROM df")

mssql_cursor.close()
connection.close()
duckdb_conn.close()
```


Summary

- Powerful complementary tool for PostgreSQL
- Uses PostgreSQL SQL parser and follows its SQL conventions
- Directly works with Parquet, CSV, JSON, and other data formats
- Intelligent ETL tool for data imports and exports
- Simplifies data format conversions
- Enables easy cross-database joins
- Provides SUMMARIZE stats, HISTOGRAM, PIVOT on tables or data files
- Rudimentary migration tool for table snapshots
- Database file easily shared across different platforms
- Direct use of Python Pandas/Polars data frames extends capabilities



THANK YOU

- **Questions?**
- Josef Machytka <josef.machytka@netapp.com>

NetApp

