

A quack of all trades:

DuckDB, a versatile analytical
use to keep in your toolkit

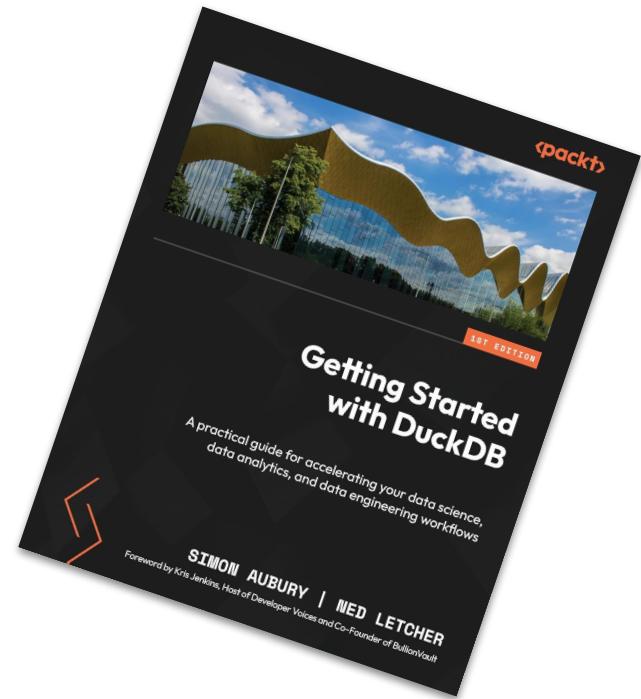


Ned Letcher

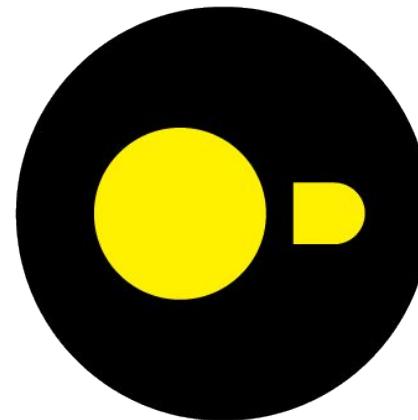


Simon Aubury



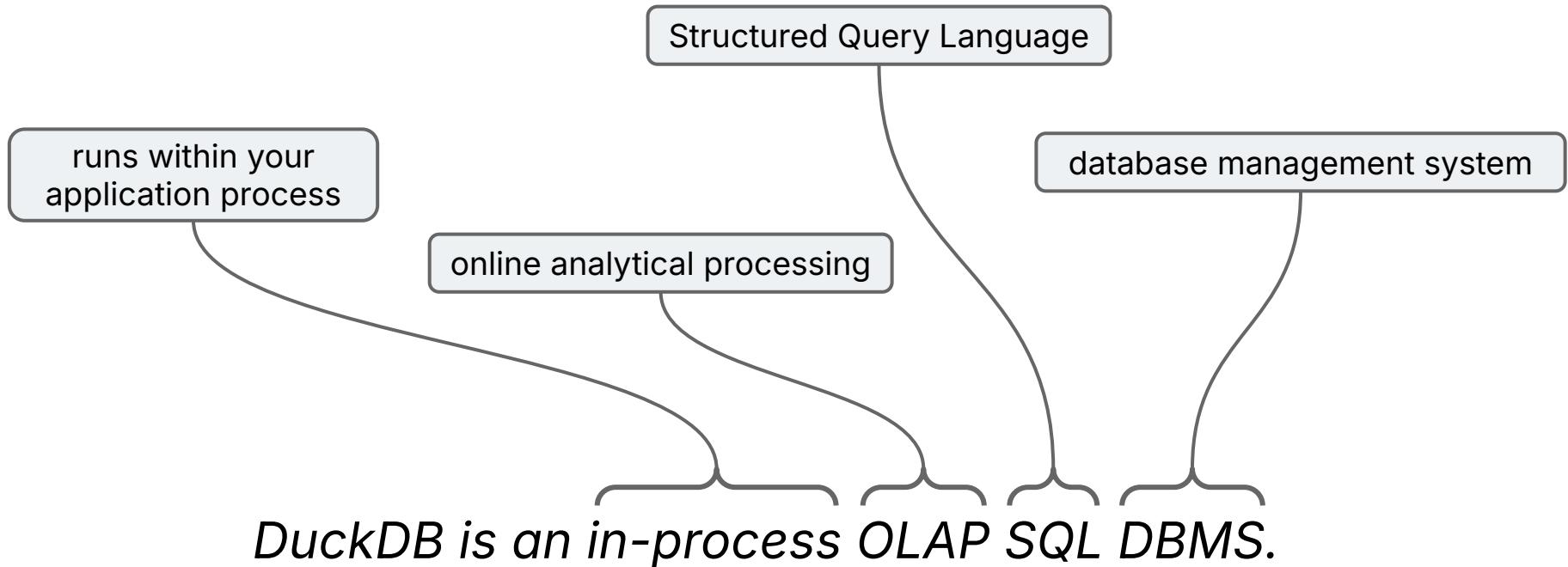


OK, so what is DuckDB?



*DuckDB is a fast in-process
analytical database*

OK, so what is DuckDB?



Comparing DuckDB

in-process



standalone



OLTP



DuckDB is
MIT-licensed in
perpetuity!



OLAP

I'm old and cynical ...

The image shows four screenshots from the [r/dataengineering](#) subreddit, illustrating a cynical perspective on the hype around DuckDB.

- Post 1:** [What is the hype around duckDB that I don't seem to understand?](#)

When we have moved to data lakes and data lake house based architecture, why should I care about an OLAP DB? At this point in the data eco system? I seem to have missed the memo on this one lol
- Post 2:** [Is anyone using duckDB at work?](#)

I'm a big fan and have been exploring adding it as our data warehouse given we're at a small scale. Does anyone use it at work/production environment? If so, how's your experience been with it?
- Post 3:** [\[deleted\] What is DuckDB Used For?](#)

Sorry for the noob question if it sounds ignorant. Here is the situation. I joined a new company, and inherited a lot of legacy codes (ETL jobs, written in python) from another colleague who already left before I got to talk to him. Throughout his code, he made extensive use of duckdb just to query pandas dataframes, basically, something like this.

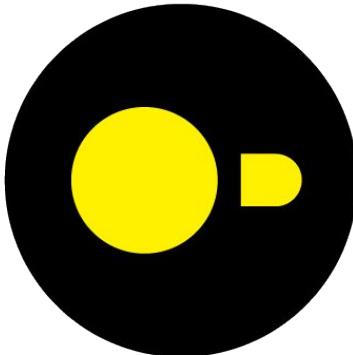
```
import duckdb, pandas as pd
df:pd.DataFrame
    df[df['value']>20] # Pandas form
    df[duckdb.FILTER('value')>20] # DuckDB value>20;
```
- Post 4:** [\[Out of the Loop\] What's the hype with DuckDB now?](#)

I've seen many posts here mentioning it. I think it's a different approach but what's its leverage against a stack built around BigQuery or plain Postgres for example?

Each post includes a timestamp, author, and a note indicating it is an archived post. The bottom of the image shows standard Reddit interaction buttons: upvote (47), downvote, comment (51), and share.

<https://www.reddit.com/r/dataengineering>

DuckDB is Versatile



Data analysis



Data pipelines



Data lake querying



Data warehousing

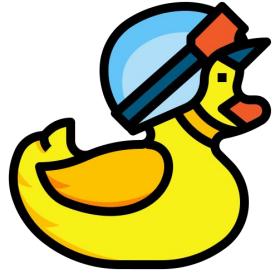


Data cubes



Data apps





DuckDB for data engineers



Demo CSV files

pizza_n.csv

```
food_name,color,calories,is_healthy
```

```
Margherita,mixed,60, FALSE
```

```
Chicago,yellow,70, FALSE
```

```
Pep,Hawaiian,yellow,60, FALSE
```

```
Hawaiian,yellow,60, FALSE
```

```
Neapolitan,red,45, FALSE
```

```
BBQ,red,62, FALSE
```

```
Veggie,mixed,32, TRUE
```

fast_food.csv

```
food_name,color,calories
```

```
burger,mixed,60
```

```
fries,yellow,35
```

```
sandwhich,white,22
```

salad.csv

```
food_name,calories,is_healthy,color
```

```
sushi,60,TRUE,mixed
```

```
pho,70,TRUE,yellow
```

-- Mixed CSV schemas – this will adapt as the schema changes

```
SELECT *  
FROM read_csv('./data_food/*.csv', union_by_name=true);
```

Names & types

food_name varchar	calories int64	is_healthy boolean	color varchar
sushi	60	true	mixed
pho	70	true	yellow
burger	60		mixed
fries	35		yellow
sandwhich	22		white
Margherita	60	false	mixed
Chicago	70	false	yellow
Pepperoni	55	false	red
Hawaiian	60	false	yellow
Neapolitan	45	false	red
BBQ	62	false	red
Veggie	32	true	mixed
salad	50	true	green
yogurt	20	true	white

Anywhere



Amazon S3

```
-- Reading remote files with the httpfs extension
CREATE OR REPLACE SECRET mysecret (
    TYPE S3,
    REGION 'us-east-1',
    ENDPOINT 's3.amazonaws.com'
);

SELECT *
FROM read_parquet('s3://duckdb-s3-bucket-public/countries.parquet')
WHERE name SIMILAR TO '.*Republic.*';
```

```
SELECT json_extract(hourly, '$.temperature_2m[1]')
from read_json('https://api.open-meteo.com/v1/forecast?
latitude=-33.8678&longitude=151.2073&hourly=temperature_2m&forecast_days=1');
```

{ REST API }



```
INSTALL sqlite_scanner;
LOAD sqlite_scanner;

ATTACH './data_iMessage/chat.db' as imessage_chat_sqlite (TYPE sqlite);

SELECT text
FROM imessage_chat_sqlite.message;
```

A mini lakehouse w., Parquet



Projection pushdown
into the Parquet file
itself for columns

Hive partitioning filters
on the partition keys

Filter pushdown for
Parquet zone-maps

```
SELECT count(*), max(ride_amt)
FROM read_parquet('s3://mybucket/taxi/*/*/*.parquet',
    hive_partitioning = true)
WHERE year = 2024 AND month = 9
AND passengers > 4;
```

DuckDB is extensible



Microsoft Azure
Blob Storage



Full text search



Hugging Face



{JSON}



Spatial / H3



PRQL

Spatial / PostGIS

Complex remote

Remote file

<https://data.melbourne.vic.gov.au>

The screenshot shows the City of Melbourne Open Data portal. A search bar at the top has the URL "data.melbourne.vic.gov.au/explore/dataset/cafes-and-restaurants-with-seating-capacity/expor..." entered. Below the search bar, the page title is "CITY OF MELBOURNE OPEN DATA". The main content area displays a dataset titled "Café, restaurant, bistro seats". It shows "60,055 records" and includes a "Filters" section. The filters include dropdown menus for "Census year" (2017, 2016, 2018, 2019, 2020, 2014), "Block ID" (85, 15, 108, 64, 58, 109), and "CLUE small area" (Melbourne (CBD), Carlton). There are also sections for "Geographic file formats" (GeoJSON, Shapefile, KML, FlatGeobuf, GPX) and "Data analysis file formats" (Parquet).

Geo-json

<https://geojson.io>

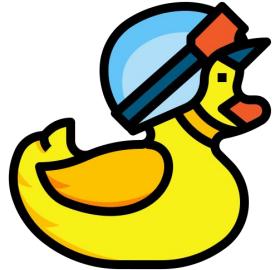
The screenshot shows the geojson.io interface. At the top, it says "geojson.io" and "powered by mapbox". Below that is a search bar with the text "186 Exhibition Street, Melbourne Central". On the left is a sidebar with options like "Open", "Save", "New", "Meta", "File", "Help", "D/JSON", "Table", and "?Help". The main area is a map of Melbourne, specifically the central business district. A large red polygon is drawn over several buildings, representing a specific geographic area. The map also shows streets like Little Bourke St, Bourke St, and Flinders St.

-- Files don't need to be local

```
SELECT *
FROM read_csv(
  'https://data.melbourne.vic.gov.au/api/explore/v2.1/catalog/capacity/exports/csv?lang=en&timezone=Australia%2FSydney&use_')
LIMIT 10;
```

```
SELECT *
FROM cafes
WHERE st_within(
    st_point(longitude, latitude),
    (
        SELECT geom
        FROM st_read('data_geo/DEB_melborne_boundary_region.geojson')
    )
)
AND "Industry (ANZSIC4) description" = 'Cafes and Restaurants'
AND "Seating type" = 'Seats - Indoor'
AND "Number of seats" < 30
```

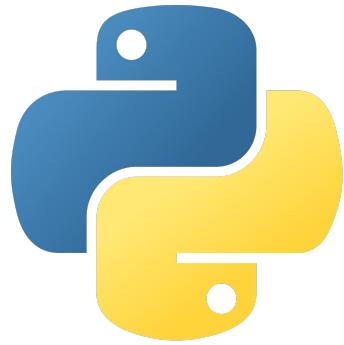
Trading name	Business address	Seating type	Number of seats
Shimbashi Japanese Soba & Sake Bar	17 Liverpool Street MELBOURNE 3000	Seats - Indoor	24
Shimbashi Japanese Soba & Sake Bar	17 Liverpool Street MELBOURNE 3000	Seats - Indoor	24
Butchers Diner	Ground 10 Bourke Street MELBOURNE VIC 3000	Seats - Indoor	20
Rice Paper Scissors Asian Kitchen	19 Liverpool Street MELBOURNE 3000	Seats - Indoor	25



DuckDB for data scientists



You can use DuckDB with the tools you already use



Python client



R client

Streamlined user experience

```
$ ls -1 ./data  
174502796.json  
17450310.json  
17451211.json  
17451321.json  
...
```

```
$ pip install duckdb
```

*In-memory
data
wrangling*

*Loading into
a persistent
database*

```
import duckdb  
duckdb.sql(  
    """  
        SELECT id, name, filename  
        FROM read_json('data/*.json', filename=true)  
    """  
) .to_parquet("users.parquet")
```

```
conn = duckdb.connect("my.duckdb")  
  
conn.sql(  
    """  
        CREATE OR REPLACE TABLE users AS  
        SELECT id, name, filename  
        FROM read_json('data/*.json', filename=true)  
    """  
)  
  
conn.sql("SELECT count(*) FROM users")
```

Friendly SQL

```
SELECT * EXCLUDE (city)
FROM addresses;
```

```
SELECT * REPLACE (lower(city) AS city)
FROM addresses;
```

```
SELECT COLUMNS('number\d+')
FROM addresses;
```

Column selection expressions

```
SELECT [
    lower(x) FOR x IN ['Hello', '', 'World']
    IF len(x) > 0
] AS strings;
```

List comprehensions

```
FROM tbl
SELECT i, s;
```

```
FROM tbl;
```

FROM-first syntax

```
SELECT
    42 AS x,
    ['a', 'b', 'c',] AS y,
    'hello world' AS z,
    ;
```

Trailing commas

Performance

DuckDB allows you to scale the size of your workloads on your laptop further, before you need a distributed compute solution.

- Vectorized query execution model
- Multi-core processing
- Out-of-core processing
- Compression
- Lazy evaluation
- Query optimiser



Lazy evaluation

- **relations**: query representations
- Evaluated as needed
- Compose with **replacement scans**
- Enables query optimization

```
relation = conn.sql("FROM users USING SAMPLE 100000")
```

```
relation.show()
```

<code>id</code> varchar	<code>name</code> varchar	<code>filename</code> varchar
735065622	JJ Gato	data/12144232_following.json
10793512	Laura Bright, Ph.D.	data/12687952_following.json
133026272	Stuff to Blow Your Mind	data/123085589_following.json
.	.	.
.	.	.
.	.	.
2236341775	FGI France	data/1206374840_following.json
330556991	Spencer Muhlstock	data/12369372_following.json
67380876	Shailesh Shukla	data/1260881050005315586_following.json

?-rows (>9999 rows, 6 shown)

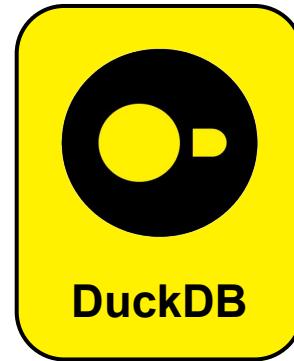
3 columns

```
conn.sql("SELECT mean(length(name)) FROM relation")
```

```
mean(length("name"))  
double
```

```
14.01666
```

Integration with data tooling



*In-memory data
formats*

*Alternative DuckDB
query interfaces*

Versatility over data types

High-cardinality data

Multidimensional data

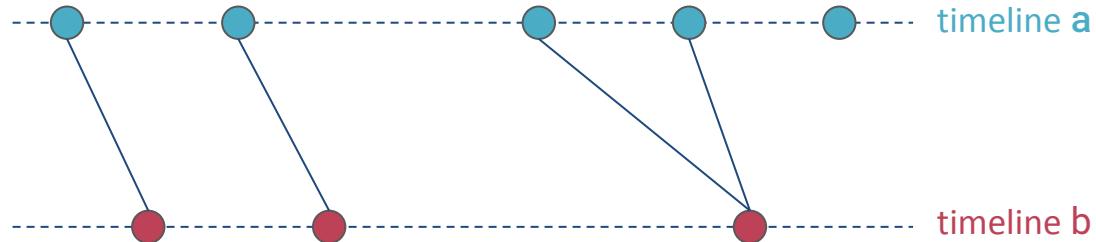
Time series data

Semi-structured data

Spatial data

Vector data

Text data



```
SELECT a.id, a.when, price  
FROM a  
ASOF JOIN b  
    ON a.id = b.id  
    AND a.when >= b.when;
```

Data management

ACID transactions

Concurrency control

Integrity constraints

Indexes & query optimisation

Views & CTEs

Data catalogue

The screenshot shows the Harlequin SQL IDE interface. On the left, the Data Catalog displays a database structure with a schema named 'main sch' containing two tables: 'enriched_users t' and 'users t'. The 'enriched_users t' table has columns 'filename s', 'id s', 'name s', and 'name_length ##'. It also contains two views: 'mean_name_length v' and 'mediate_name_length v'. The 'users t' table has columns 'filename s', 'id s', and 'name s'. In the center, the Query Editor contains the following SQL code:

```
1 SELECT name, count(name) AS count
2 FROM users
3 GROUP BY name
4 ORDER BY count DESC
```

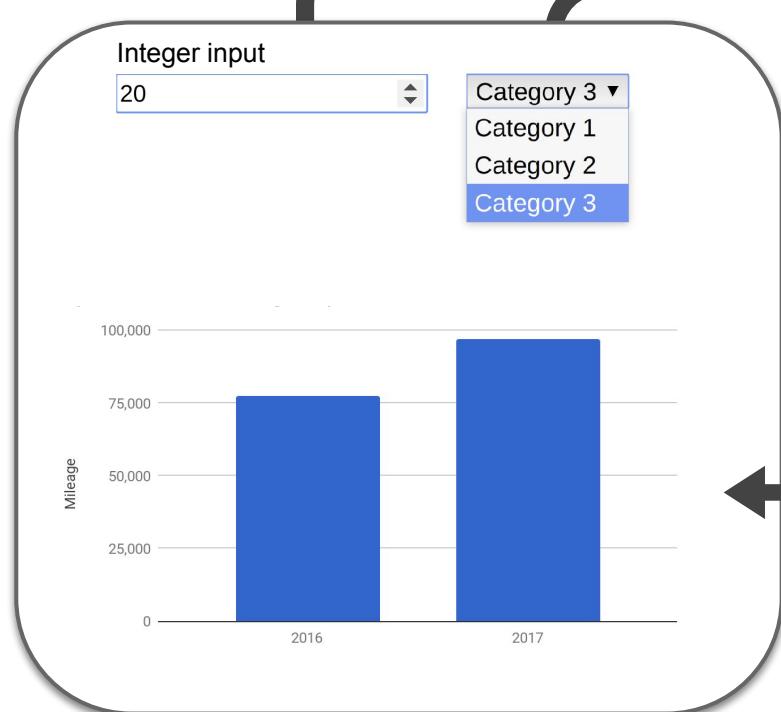
On the right, the Query Results window shows the output of the query, listing names and their counts. The results are as follows:

name	count
.	2201
Chris	870
Alex	856
David	820
Elon Musk	808
Mike	727
Matt	723
Barack Obama	685
James	674

Harlequin SQL IDE

Powering data apps

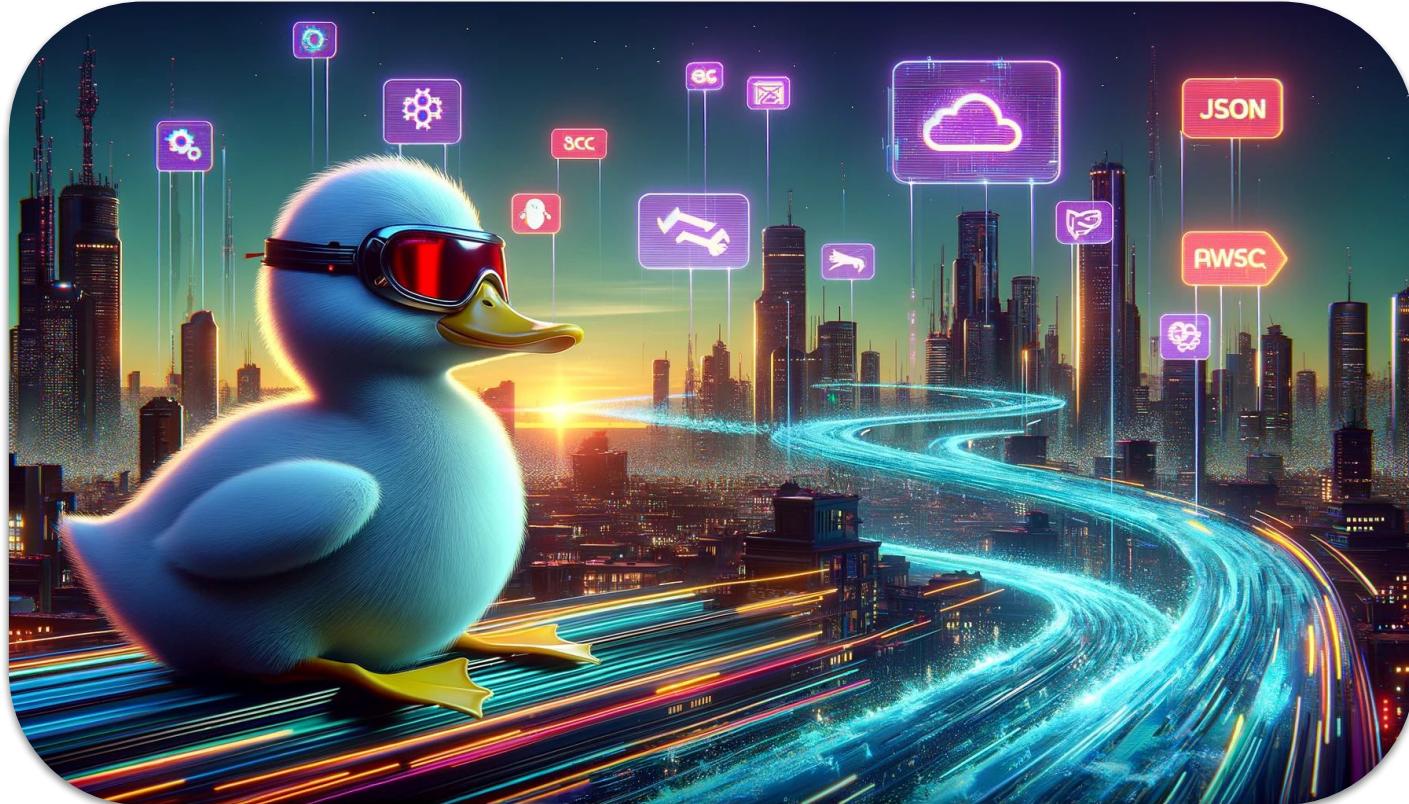
Layout



```
def filter_data(filter1, filter2):  
    rel = conn.sql(<SQL query>)  
    return rel.fetchall()
```

Callback function

 Streamlit plotly | Dash Shiny Panel



In conclusion ...

Why add DuckDB to your data toolkit?



A building block for
data infrastructure &
data products



Data analysis



Data pipelines



Data lake querying



Data warehousing



Data cubes



Data apps



Scaling and
supercharging data
science and data
wrangling workflows



Thank you!



<https://packt.link/byKYt>



Getting Started with DuckDB

A practical guide for accelerating your data science,
data analytics, and data engineering workflows

SIMON AUBURY | NED LETCHER

Foreword by Kris Jenkins, Host of Developer Voices and Co-Founder of BullionVault