

A quack of all trades:

DuckDB, a versatile analytical
engine to keep in your toolkit



Ned Letcher

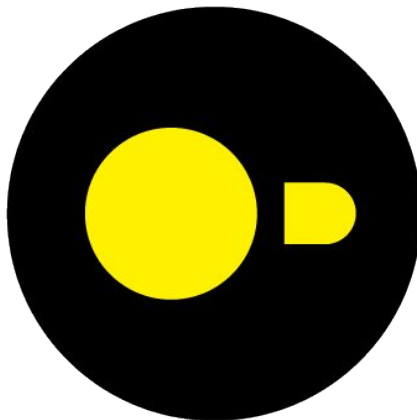


Simon Aubury





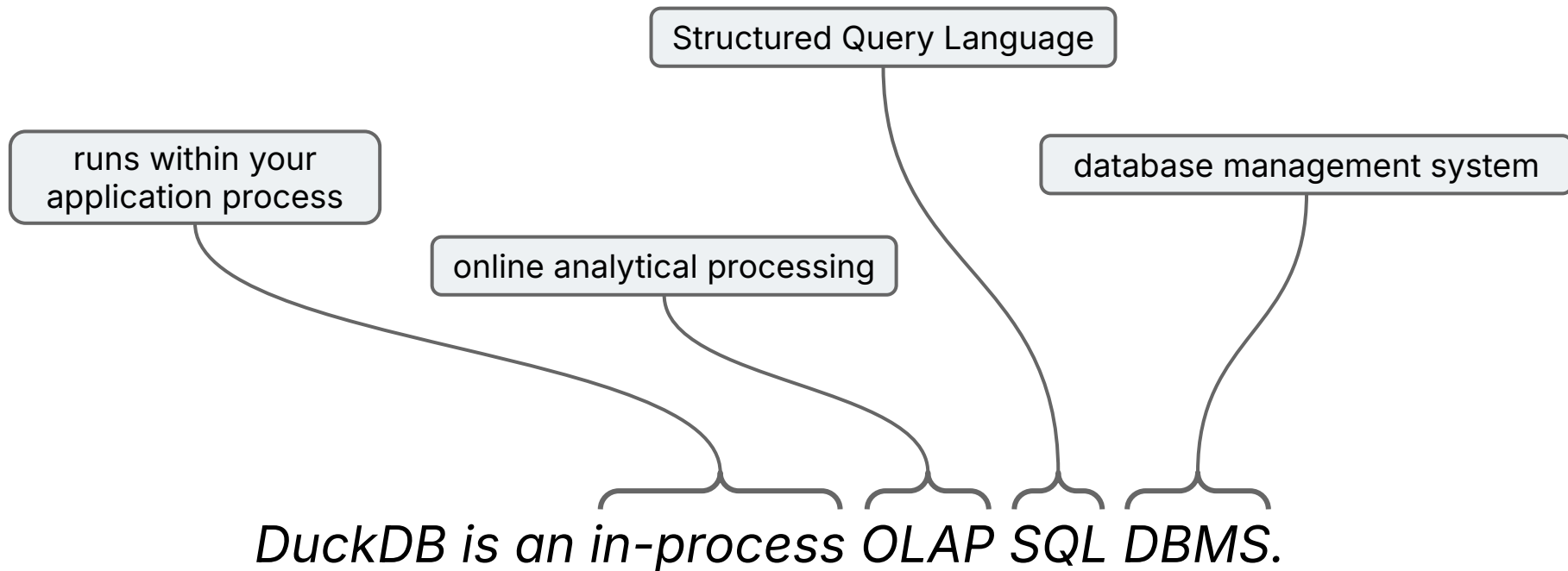
OK, so what is DuckDB?



DuckDB is a fast in-process analytical database



OK, so what is DuckDB?



Comparing DuckDB

in-process



DuckDB is
MIT-licensed in
perpetuity!

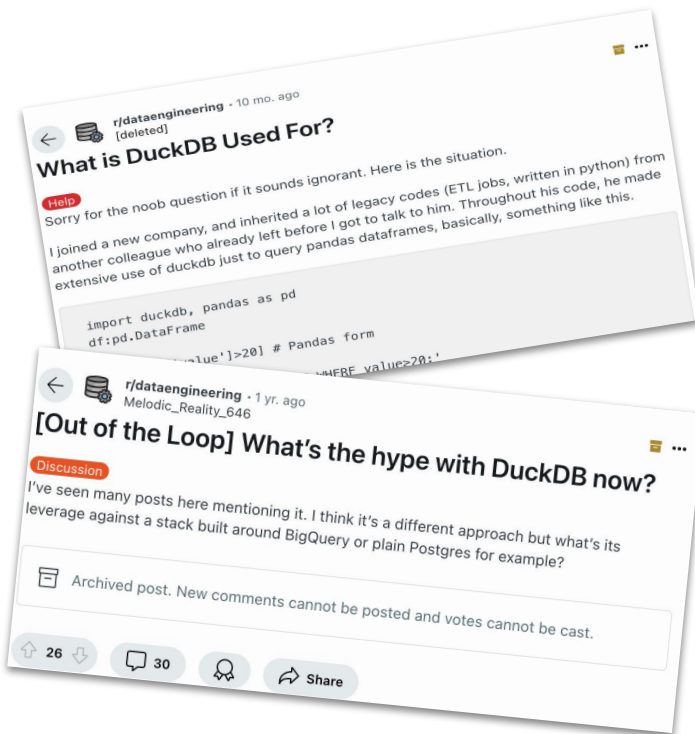
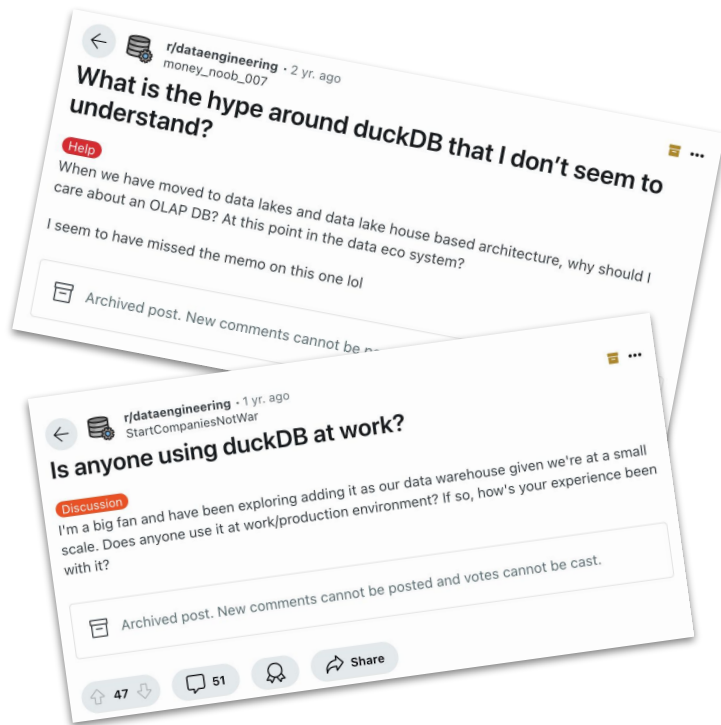
standalone



OLTP

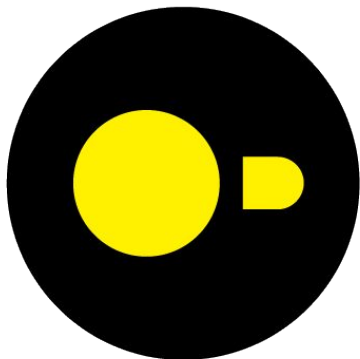
OLAP

I'm old and cynical ...



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

DuckDB is Versatile



Data analysis



Data pipelines



Data lake querying



Data warehousing

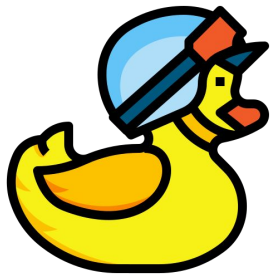


Data cubes



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
```

```
Pepperoni,mixed,55,FALSE  
Hawaiian,yellow,60,FALSE  
Neapolitan,red,45,FALSE
```

```
food_name,color,calories,is_healthy  
BBQ,red,62,FALSE  
Veggie,mixed,32,TRUE
```

fast_food.csv

```
food_name,color,calories  
burger,mixed,60  
fries,yellow,35  
sandwich,white,22
```

salad.csv

```
food_name,calories,is_healthy,color  
sushi,60,TRUE,mixed  
pho,70,TRUE,yellow
```

Names & types

```
-- Mixed CSV schemas - this will adapt as the schema changes  
SELECT *  
FROM read_csv('./data_food/*.csv', union_by_name=true);
```

food_name varchar	calories int64	is_healthy boolean	color varchar
sushi	60	true	mixed
pho	70	true	yellow
burger	60		mixed
fries	35		yellow
sandwich	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



```
-- Reading remote files with the https 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');
```



```
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



DELTA LAKE



Full text search



dbt



Hugging Face



PostgreSQL



Spatial / H3



PRQL

Spatial / PostGIS

<https://duckdb.org/docs/extensions/overview.html>

Complex remote

Remote file

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

The screenshot shows the 'Café, restaurant, bistro seats' dataset page on the City of Melbourne Open Data portal. It displays 60,055 records and various filters. The 'Flat file formats' section lists CSV, JSON, and Excel. The 'Geographic file formats' section lists GeoJSON, Shapefile, KML, FlatGeobuf, and GPX. The 'Data analysis file formats' section lists Parquet.

Geo-json

<https://geojson.io>

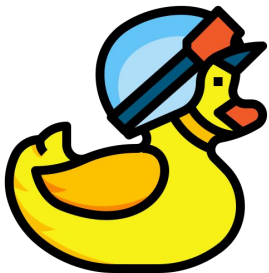
The screenshot shows the geojson.io web application. On the left is a map of Melbourne with a red polygon highlighting a specific area. On the right is a JSON editor showing the GeoJSON data for this area, including coordinates and properties.

-- Files don't need to be local

```
SELECT *
FROM read_csv(
  'https://data.melbourne.vic.gov.au/api/explore/v2.1/catalog/ops?dataset=cafes-and-restaurants-with-seating-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
```

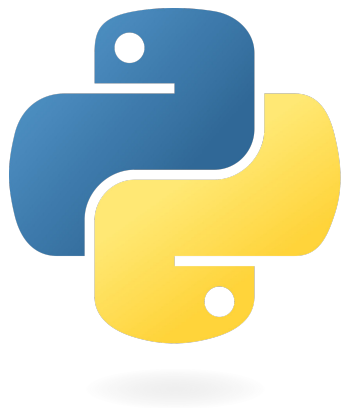
Trading name	Business address	Seating type	Number of seats
Shimbashi Japanese Soba & Sake Bar	17 Liverpool Street MELBOURNE 3000	Seats - Indoor	24
Shimbashi Japanese Soba & Sake Bar	17 Liverpool Street MELBOURNE 3000	Seats - Indoor	24
Butchers Diner	Ground 10 Bourke Street MELBOURNE VIC 3000	Seats - 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 -l ./data  
174502796.json  
17450310.json  
17451211.json  
17451321.json  
...
```

*In-memory
data
wrangling*

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

```
$ pip install duckdb
```

*Loading into
a persistent
database*

```
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()
```

id varchar	name varchar	filename 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

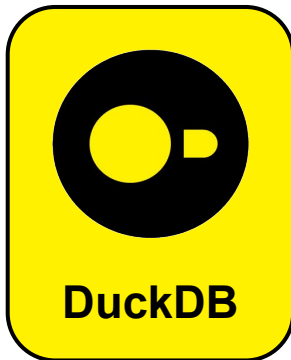
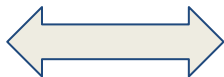
```
conn.sql("SELECT mean(length(name)) FROM relation")
```

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

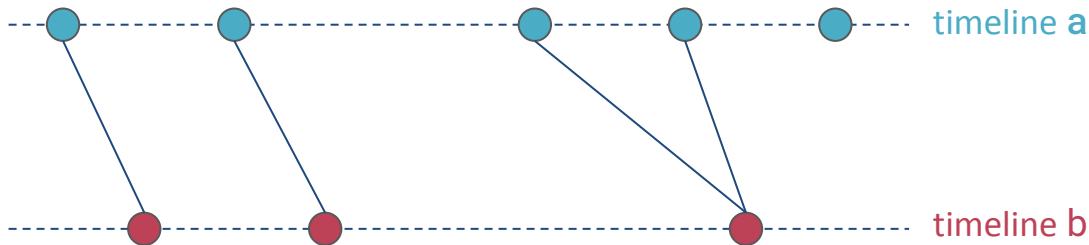
Time series data

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 displays the Harlequin SQL IDE interface, which is divided into three main sections:

- Data Catalog:** A tree view on the left showing the database structure. It includes a 'my db' folder containing a 'main sch' schema. Under 'main sch', there are several tables and views: 'enriched_users t', 'mean_name_length v', 'mediate_name_length v', and 'users t'. The 'users t' table is expanded, showing columns 'filename s', 'id s' (highlighted in blue), and 'name s'.
- Query Editor:** A text area on the right where SQL queries are written. It contains the following query:

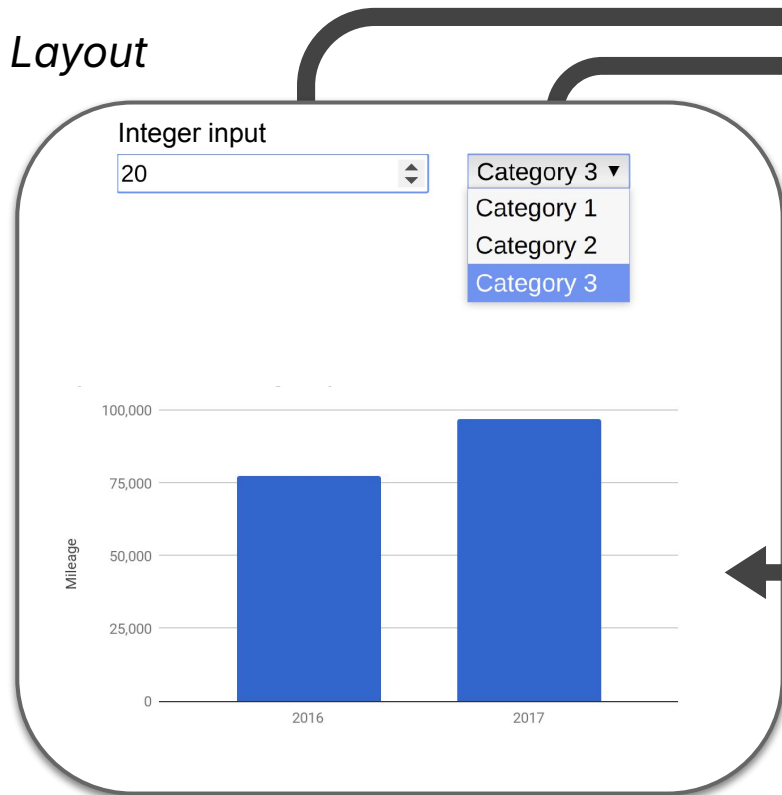
```
1 SELECT name, count(name) AS count
2 FROM users
3 GROUP BY name
4 ORDER BY count DESC
```
- Query Results:** A table at the bottom right showing the results of the query. It has columns 'name s' and 'count ##'. The results are as follows:

name s	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





In conclusion ...

Why add DuckDB to your data toolkit?



A building block for
data infrastructure &
data products



Data analysis



Data pipelines



Data lake querying



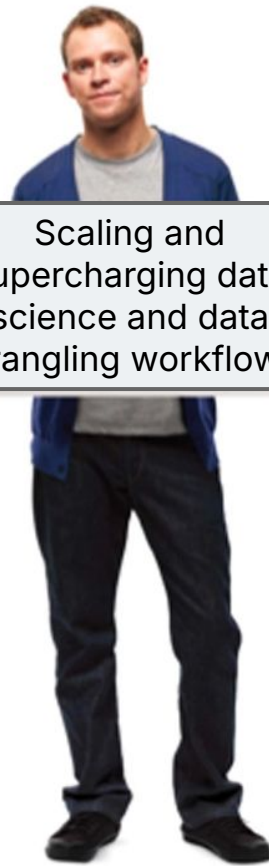
Data warehousing



Data cubes



Data apps



Scaling and
supercharging data
science and data
wrangling workflows

Thank you!



<https://packt.link/byKYt>

