
Integrating DuckDB
with Node.js as a REST API for FileMaker
Marcel Moré
engageU
Power of the Present - Force of the Future

Integrating DuckDB · Agenda

- Why DuckDB?
- Building a REST API with node.js
- Connecting from FileMaker
- Demo
- ETL Process
- Feeding Dashboards

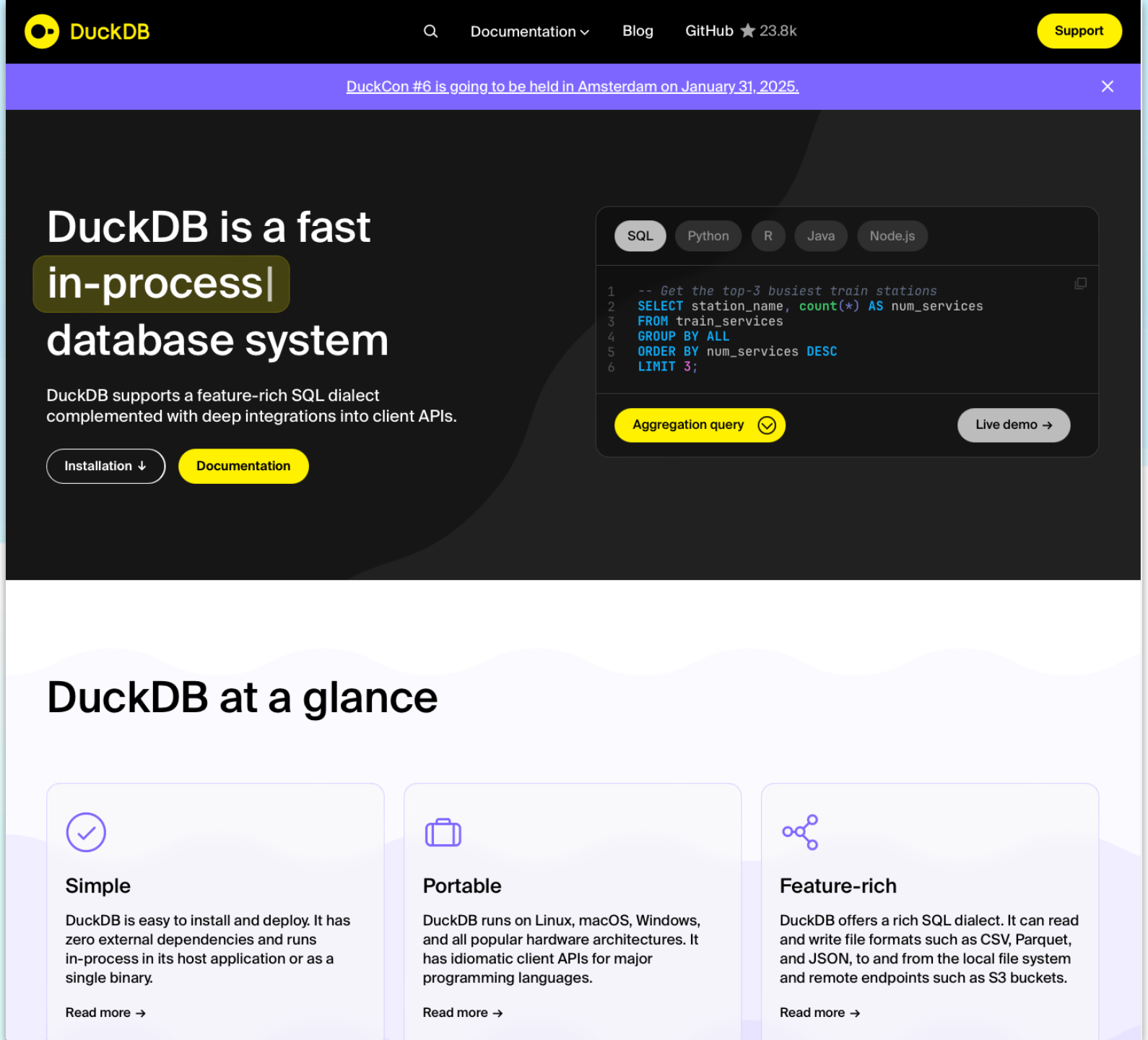
About me ...

- Self-employed as developer and information designer for 36 years
- FileMaker enthusiast since Version 2
- Specializes in customized business solutions for clients in industry, trade and marketing
- Host of FileMaker Usergroup Braunschweig (Germany)
- Author and Speaker for FileMaker related topics

Marcel Moré

@ mmore
@ dateimacher

Why DuckDB ?

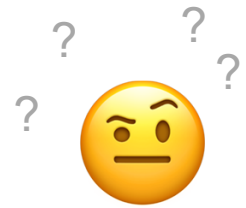


What is DuckDB?

- RAM based
- in-process Database
- OLAP Tool
- single node
- open source
- portable
- bring the engine to where the data is

What it's not...

- »classic« Database
- client server
- data storage
- multi user
- vendor specific
- bring the data to the engine



Why DuckDB

- small footprint
- blazingly fast! 💫🔥
- available on all platforms
- interesting feature set 🧐
- can access online and offline data sources
- multiple formats supported
- data analysis & ETL 📈

Why DuckDB

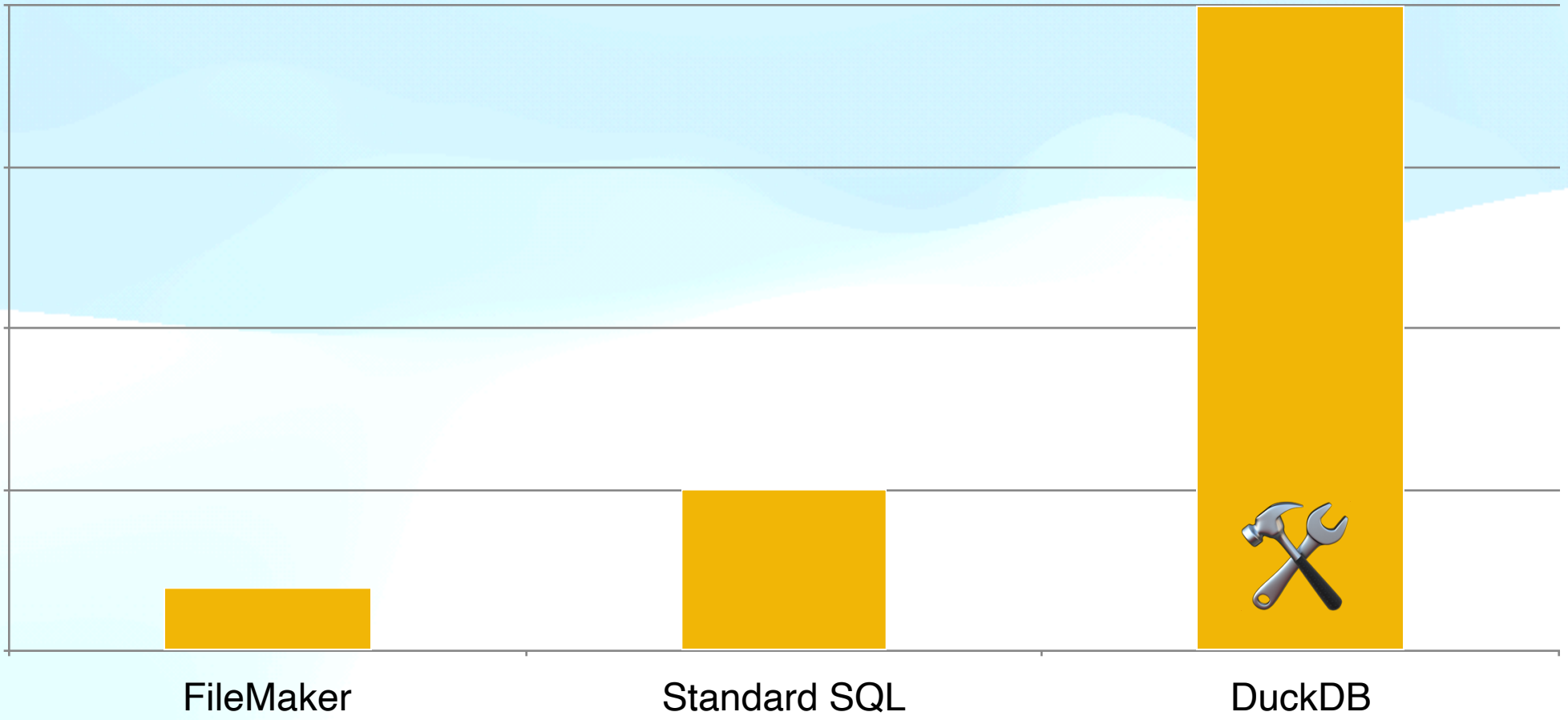
- load and process data
- CSV, Parquet, JSON, Excel
- tolerant parser
- column optimized storage
- modify, filter, aggregate
- extended SQL
- Swiss Army Knife for Data!

SQL Power

- friendly SQL
- shortcuts
- wildcards
- RegEx
- nesting, unnesting, lists, arrays, JSON
- CTEs, macros, subqueries
- rich function set for date, time, strings, ...
- lots of extentions (e.g. geo spatial data)

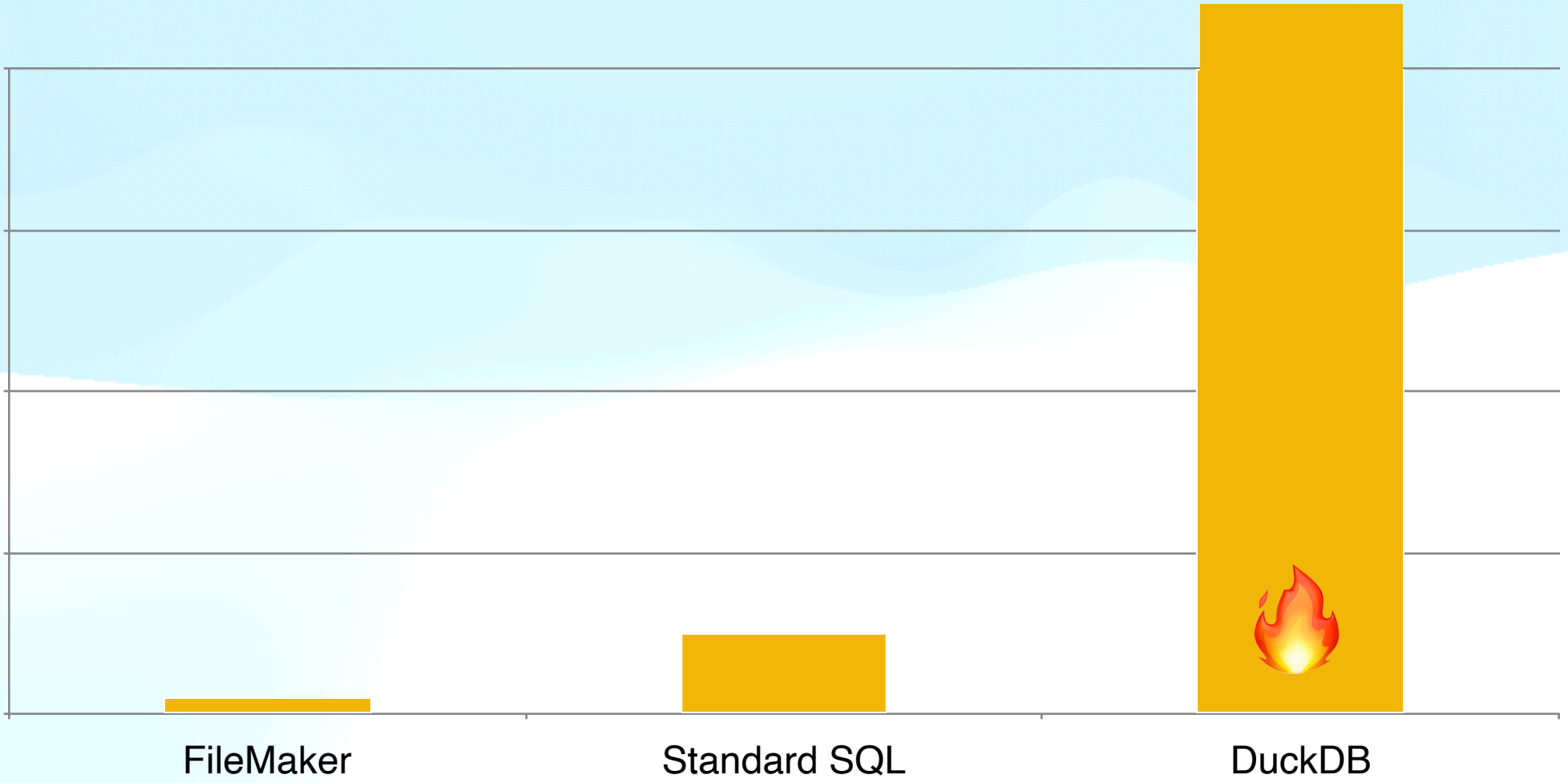
SQL Power

~ 600 Functions



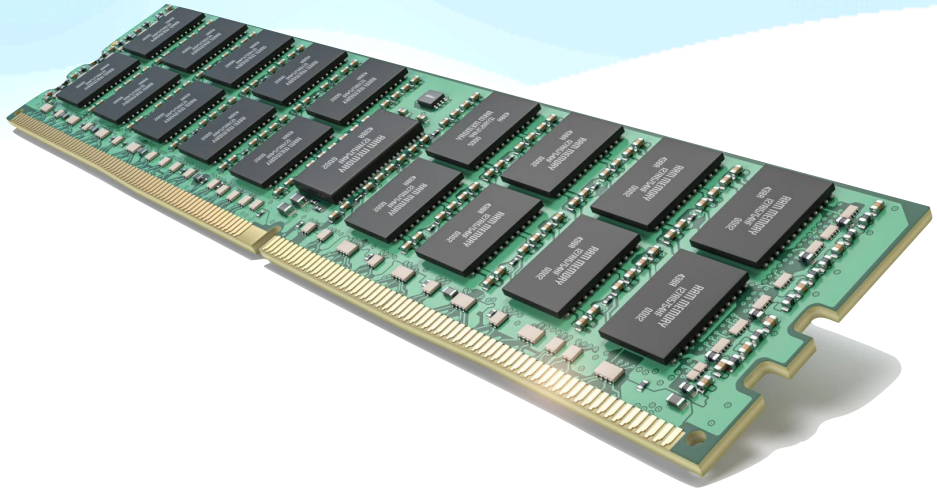
Speed / Performance

1000x ~ 10.000x



Speeeeeeeeed !!!!!!

- RAM-based
- hardware optimized
- multi-threading
- vectorized
- streaming
- columnar storage format
- best-in-class query optimizer
- lots of nifty tricks

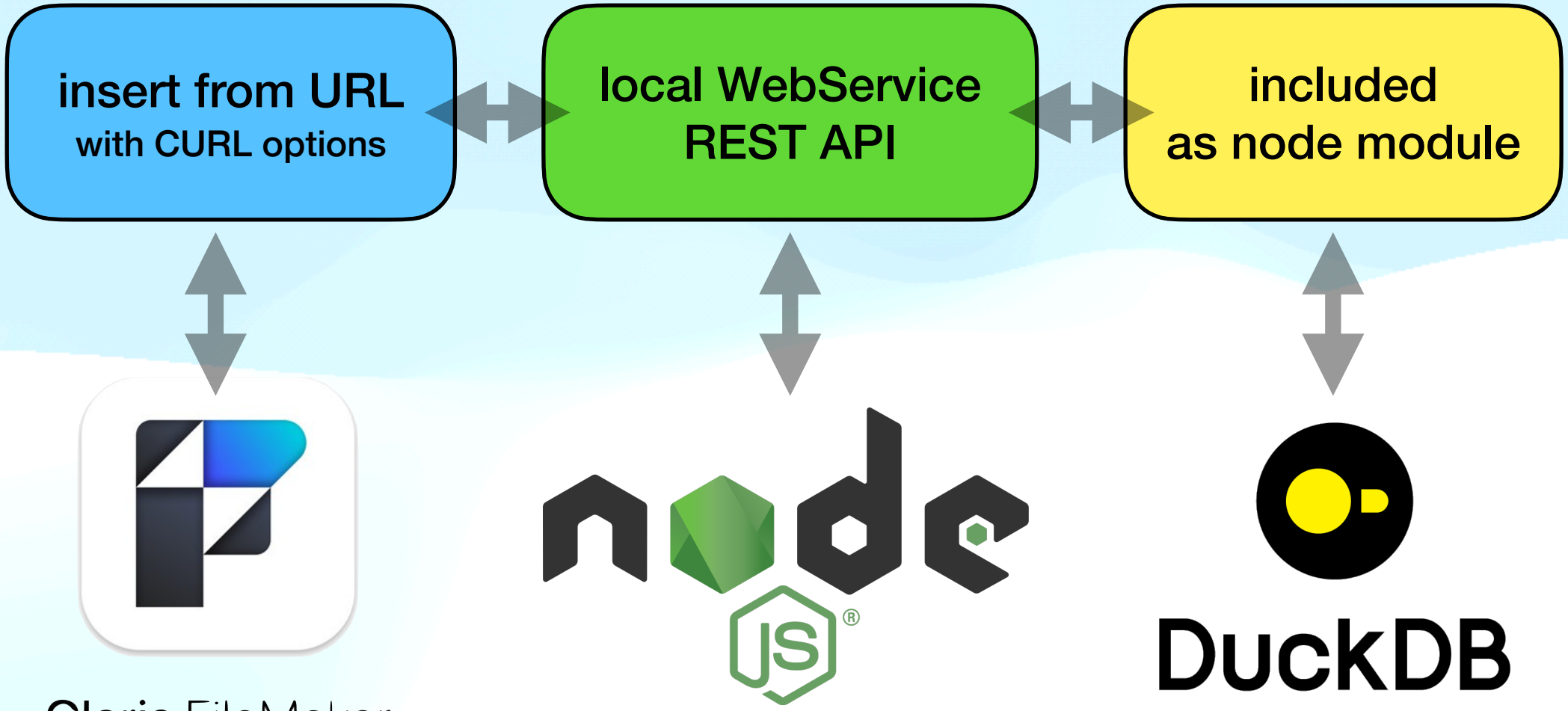


How to make use of it ?

we are living in FileMaker

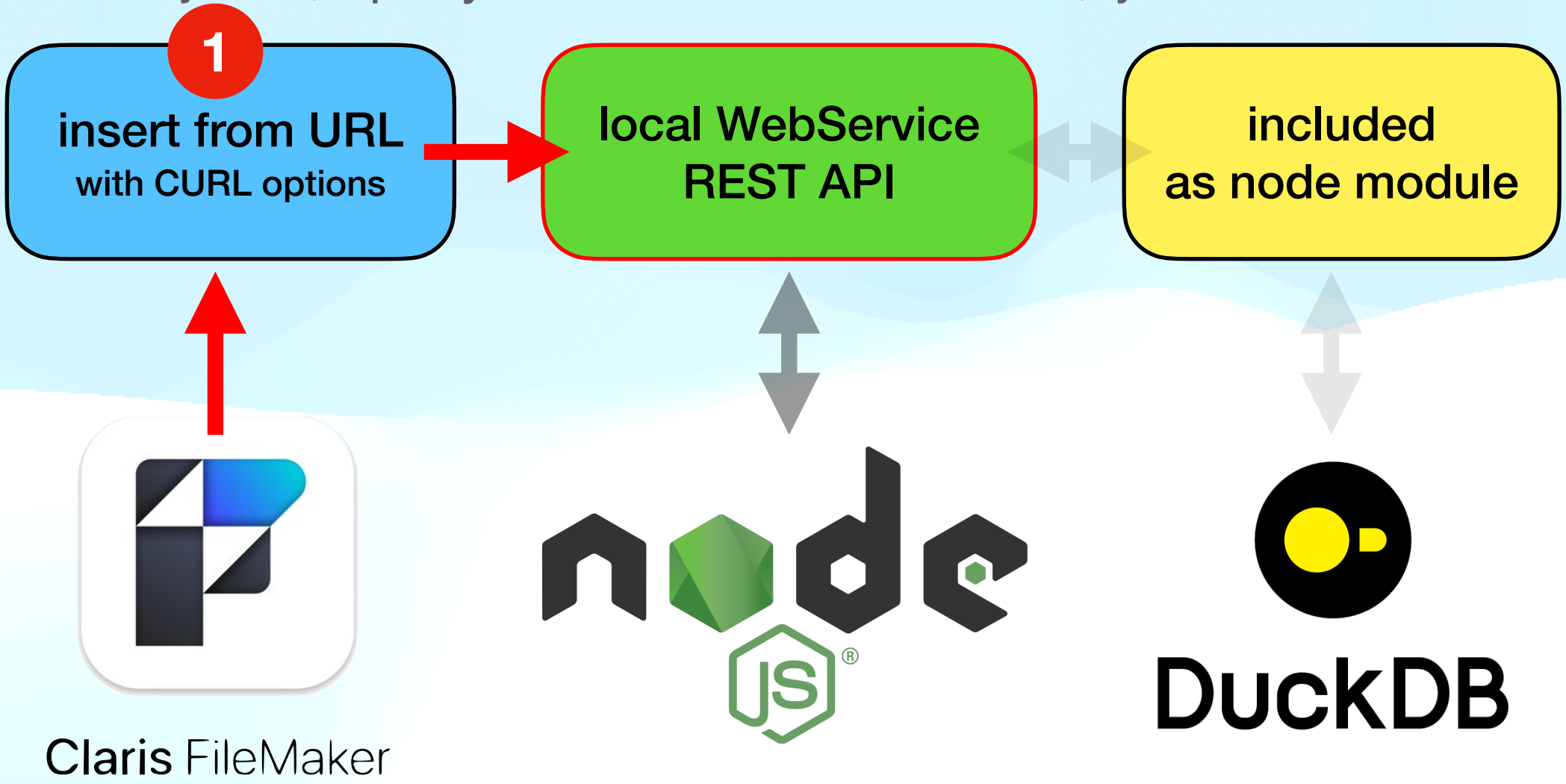
Architecture

Claris FileMaker



http://127.0.0.1:3000/query

{"format":"json","query":"SELECT 'hello world';"}



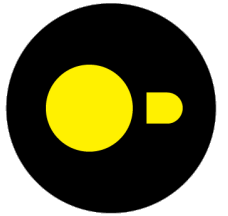
Input:
SELECT 'hello world';

insert from URL
with CURL options

local WebService
REST API

2
included
as node module

Claris FileMaker



DuckDB

result:

[{"'hello world'":"hello world"}]

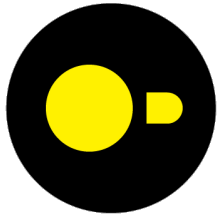
3

insert from URL
with CURL options

local WebService
REST API

included
as node module

Claris FileMaker

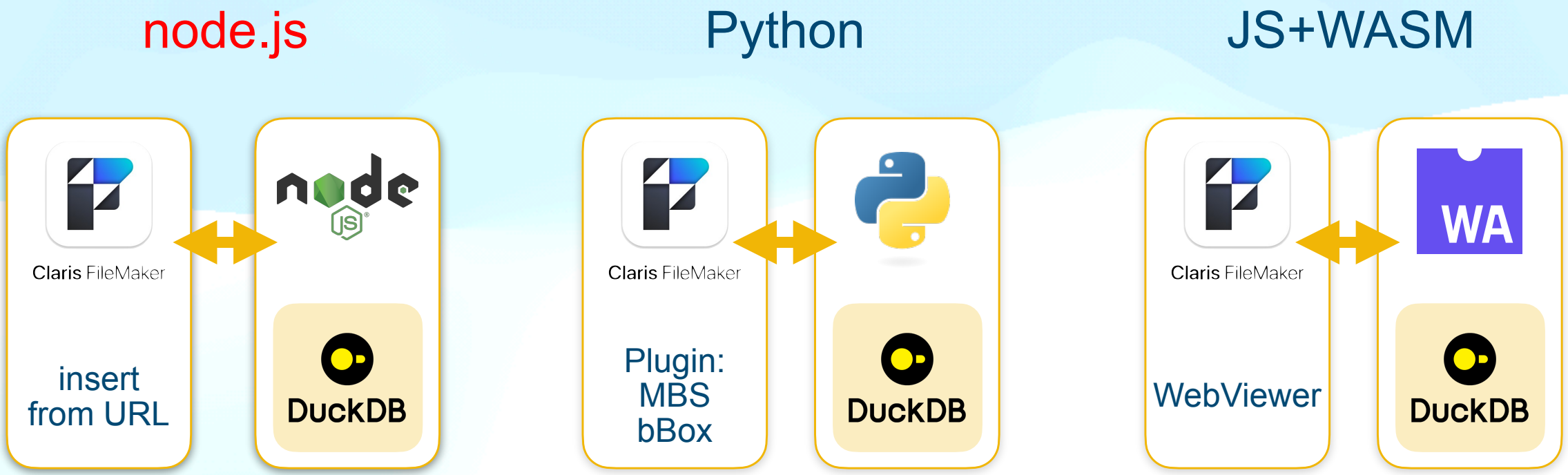


DuckDB

Why node.js ?

- low barrier
- multi platform
- easy to deploy
- easy to maintain
- can run locally
- can run on another machine
- REST API mechanism allows microservice architecture

other options to integrate with FM



Building a REST API

with only a few lines of code !

REST API
with node.js

Step 1: create a local WebService

```
const express = require('express');

// Init express for API Server

const app = express ();
app.use(express.json());

const PORT = 3000;

app.listen(PORT, () => {
  console.log("Server Listening on PORT:", PORT);
});
```

REST API
with node.js

Step 2: create a DuckDB database

```
//---- setting up DuckDB ----//

const duckdb = require('duckdb');

// Init in-memory Database
const db = new duckdb.Database(':memory:');
```

REST API
with node.js

Step 3: define POST request handler

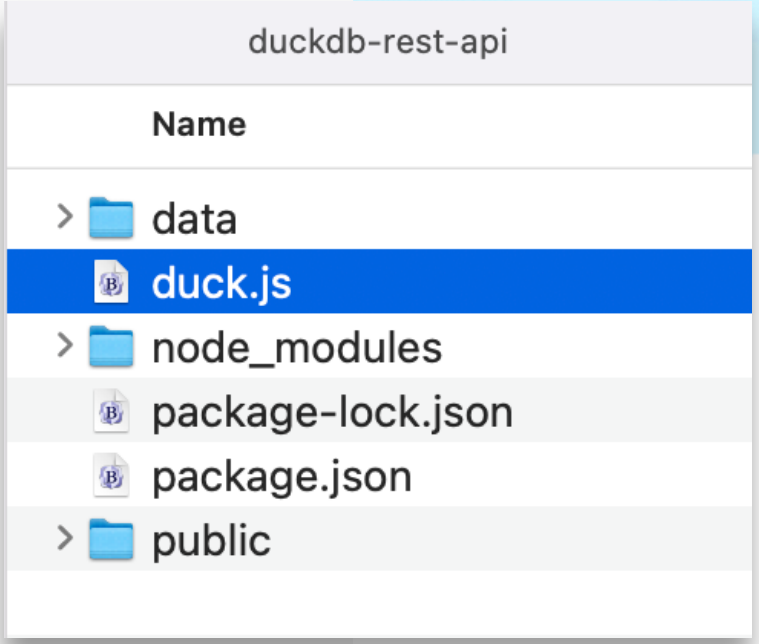
```
//---- POST Request handler ----//

// Query
app.post('/query', (req, res) => {

    // extracting parameters from body
    const sqlQuery = req.body.query;

    // call DuckDB and get result
    db.all(sqlQuery, (err, result) => {
    if (err) {
      return res.status(400).json({ error: err.message });
    }

    // return result as JSON
    res.json(result);

});
```

REST API
with node.js

complete node.js script available ...

~/WWW/nodejs/duckdb-rest-api/duck.js

```
/* duck.js v1.5 2024-06-04                */
/*                                        */
/* simple REST API interface for DuckDB   */
/* © 2024 Marcel Moré                     */
/* http://blog.marcel-more.de             */



/*
Send POST request

Header: "Content-Type: application/json"
Body  : JSON object
{
  "format": "json",
  "query": "SELECT 'hello world';"
}


format: text|json|html|html-css
query : valid query-text for DuckDB

response will be formatted according to 'format' option:
text: result as raw text
json: result as JSON object
html: result as HTML table
html-css: HTML table with nice CSS formatting (experimental)


Sample POST request:
URL : http://127.0.0.1:3000/query
body: {"format":"json","query":"SELECT 'hello world';"}

Sample result:
[{"'hello world'":"hello world"}]
*/
```



EngageU
Power of the Present - Force of the Future

Connecting from FileMaker

FileMaker
talks to node.js

Step 1: put parameters into JSON

```
# Parameter: Query
Set Variable [ $Query ; Value: Get ( ScriptParameter ) ]
# get Format from global field
Set Variable [ $Format ; Value: Lower ( DuckDB API::Format ) ]

# Build complete JSON for REST API call: query, format
Set Variable [ $JSONdata ;
Value: JSONSetElement ( "" ;
   ;[ "query" ; $Query ; JSONString ]
   ;[ "format" ; $Format ; JSONString ] ) ]
) ]
```

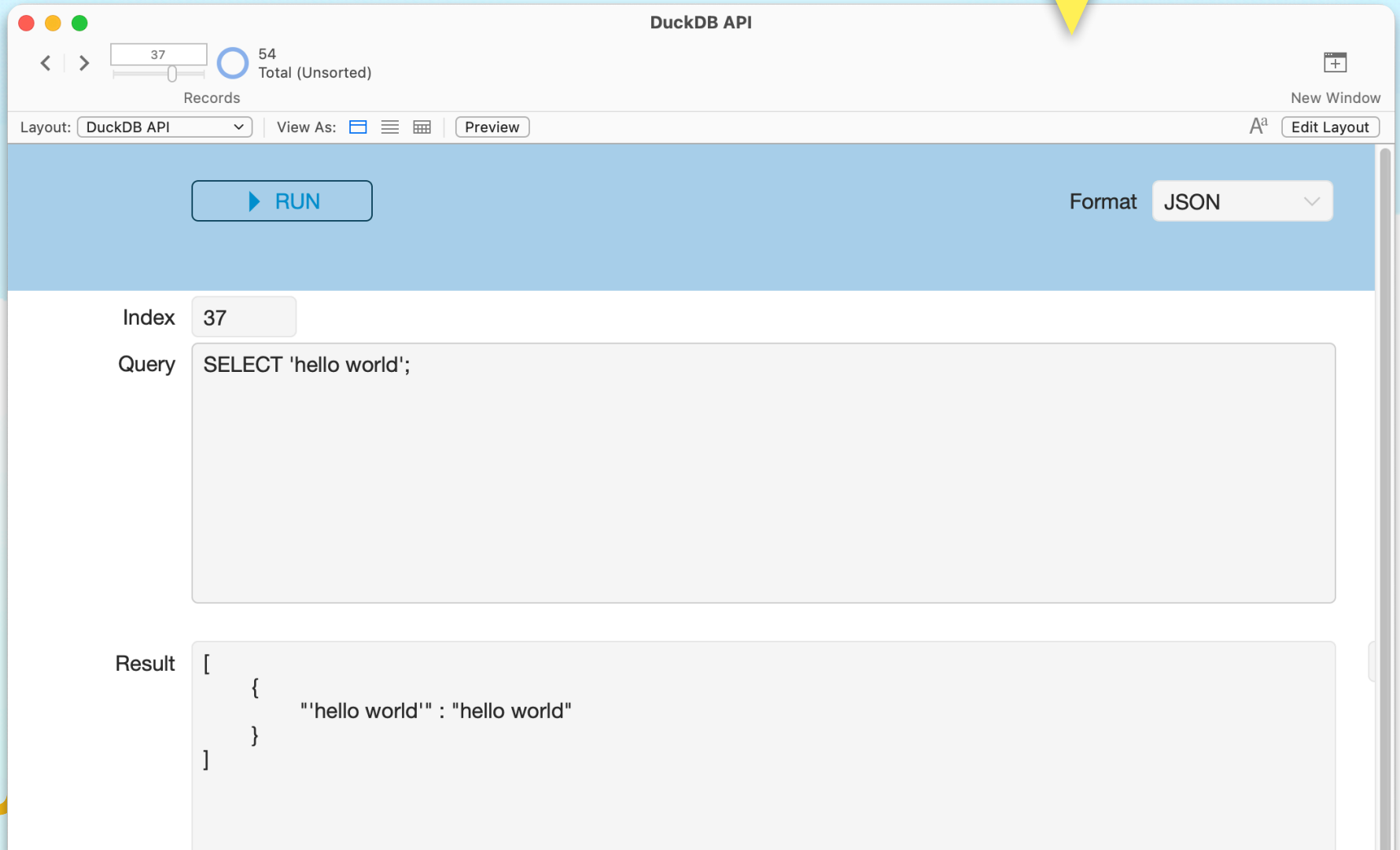
FileMaker
talks to node.js

Step 2: call WebService via CURL

```
# Call node.js WEB API → WebService must be running on local host port 3000

# cURL options: Timeout = 10 Sec
Set Variable [ $curl ; Value: "-m 100 -H \"Content-Type: application/json\""
 & " -d @$JSONdata" ]
Set Variable [ $url ; Value: "http://127.0.0.1:3000/query" ]
Insert from URL [ Select ; With dialog: Off
; Target: $Response ; $URL ; cURL options: $curl ]
```

FileMaker
talks to node.js

Step 3: display result



Demo

DuckDB API

38
38
Gesamt (Unsortiert)
Datensätze
Neues Fenster

Layout: DuckDB API | Anzeige: | Seitenansicht | Layout bearbeiten

RUN

Format JSON

Index 38

Query

```
CREATE OR REPLACE TABLE Results AS SELECT
 *,
 epoch ( strptime ( '1970-01-01 ' || rpad ( Duration, 15,'0' ) , '%Y-%m-%d %H:%M:%S,%f' ) ) as Duration_ms
FROM read_csv('data/Pathfinder_Results.tab',
    delim = '\t',
    header = false,
    decimal_separator = ',',
    ignore_errors = true,
    columns = {
        'Index': 'INT',
        'Nodes': 'INT'
```

Result


```
[
    {
        "avg(Delta_Optimized_Percent)" : 0.0726345577402552,
        "avg(Delta_PartOptimized_Percent)" : 0.040060901601516,
        "sum(Delta_Optimized)" : 2274188.62000004,
        "sum(Delta_PartOptimized)" : 1481485.61000002,
        "sum(Duration_ms)" : 997.155917000003,
        "sum(Len)" : 12509633.259999,
        "sum(Len_Optimized)" : 10235444.6399998,
        "sum(Len_PartOptimized)" : 11028147.6499996,
        "sum(Nodes)" : "470070"
    }
]
```

Performance

130.000 Lines of CSV Data

Simple Scenario

Pathfinder_Results.tab

~/WWW/nodejs/duckdb-rest-api/data/Pathfinder_Results.tab

1	1475567	14	298,35	247,72	50,63	,1697000167588403	202,31	96,04	,3219038042567454	00:00:00,038314	
2	1475568	1	91,35	91,35	0	0	91,35	0	0	00:00:00,001768	
3	1475569	1	47,96	47,96	0	0	47,96	0	0	00:00:00,001497	
4	1475570	1	31,49	31,49	0	0	31,49	0	0	00:00:00,001501	
5	1475571	1	24,94	24,94	0	0	24,94	0	0	00:00:00,001105	
6	1475572	4	82,83	82,83	0	0	82,83	0	0	00:00:00,005329	
7	1475573	1	35,46	35,46	0	0	35,46	0	0	00:00:00,001744	
8	1475574	2	112,37	112,37	0	0	112,37	0	0	00:00:00,00423	
9	1475575	15	241,93	199,57	42,36	,1750919687512917	183,38	58,55	,2420121522754516	00:00:00,012616	
10	1475576	2	38,43	38,43	0	0	38,43	0	0	00:00:00,003692	
11	1475577	3	72,79	76,3	-3,51	-,0482209094655859	72,79	0	0	00:00:00,003778	
12	1475578	23	386,7	250,52	136,18	,352159296612361	215,38	171,32	,4430307732092061	00:00:00,037169	
13	1475579	2	35,1	37,71	-2,61	-,0743589743589744	35,1	0	0	00:00:00,003725	
14	1475580	9	222,88	207,8	15,08	,0676597272074659	191,55	31,33	,1405689160086145	00:00:00,006467	
15	1475581	2	29,07	29,07	0	0	29,07	0	0	00:00:00,002233	
16	1475582	3	69,78	74,68	-4,9	-,0702206936084838	68,55	1,23	,0176268271711092	00:00:00,002259	
17	1475583	9	246,12	193,46	52,66	,2139606695920689	167,84	78,28	,3180562327320007	00:00:00,006808	
18	1475584	2	29,27	29,27	0	0	29,27	0	0	00:00:00,003055	
19	1475585	12	356,48	248,13	108,35	,3039441202872531	204,97	151,51	,4250168312387792	00:00:00,012348	
20	1475586	5	153,97	111,53	42,44	,2756381113203871	106,39	47,58	,3090212379034877	00:00:00,003948	
21	1475587	3	135,56	122,68	12,88	,095013278253172	119,84	15,72	,1159634110357037	00:00:00,002458	
22	1475588	4	84,92	83,4	1,52	,0178991992463495	76,88	8,04	,0946773433820066	00:00:00,002432	

Query: read CSV + calculate + aggregate

```
CREATE OR REPLACE TABLE Results AS SELECT
    *,
    epoch ( strptime ( '1970-01-01 ' || rpad ( Duration, 15,'0' ) , '%Y-%m-%d %H:%M:%S,%f' ) ) as Duration_ms
FROM read_csv('data/Pathfinder_Results.tab',
    delim = '\t',
    header = false,
    decimal_separator = ',',
    ignore_errors = true,
    columns = {
        'Index': 'INT',
        'Nodes': 'INT',
        'Len': 'DOUBLE',
        'Len_PartOptimized': 'DOUBLE',
        'Delta_PartOptimized': 'DOUBLE',
        'Delta_PartOptimized_Percent': 'DOUBLE',
        'Len_Optimized': 'DOUBLE',
        'Delta_Optimized': 'DOUBLE',
        'Delta_Optimized_Percent': 'DOUBLE',
        'Duration': 'VARCHAR'
    });


SELECT
    SUM( Len ),
    SUM( Len_PartOptimized ),
    SUM( Len_Optimized ),
    SUM( Delta_PartOptimized ),
    SUM( Delta_Optimized ),
    AVG( Delta_PartOptimized_Percent ),
    AVG( Delta_Optimized_Percent ),
    SUM( Duration_ms ),
    SUM( Nodes )
FROM Results;
```

Performance

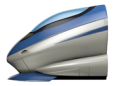
Aggregation on 130.000 Lines of Data

	FileMaker native	DuckDB
Aggregation (sec.)	1,3774	0,0014
Factor	984	1

1000 times faster!!! 😯

Performance

Aggregation on 130.000 Lines of Data with I/O penalty

	FileMaker native	DuckDB
Aggregation (sec.)	1,3774	0,0014
+ **Export Data as CSV**	Bottleneck 0,9217 	
+ **Import Data from CSV**		0,0397 
Factor	23	1
= **Total time** (sec.)	1,3774	0,9628
Factor	1,4	1

 still faster!

engageU
Power of the Present - Force of the Future

5.5 Million Lines of Data

~ 350 MBytes of CSV

Complex Scenario

Really complex scenario:

- unpacking nested groups of labels
- normalizing multiline keys
- aggregating on missing relational data
- generating data on if-then-else logic
- calculation of statistical weights upon aggregated results of the whole dataset
- normalizing the weights upon subsets of the data in several dimensions
- extrapolation of the whole dataset by a cross product of labels, weights and several measures

FileMaker native	DuckDB
12 Tables, **25** TOs	**6** CSV Files
86 Helper Fields	**1** Query
10 Custom Functions	**27** SELECT Statements
56 Scripts	**360** Lines of Code
1524 Script-Steps	

Performance

FileMaker native	DuckDB
10 Hours	**4** Seconds

pure magic!!!

EngageU
Power of the Present - Force of the Future

read more on my blog post

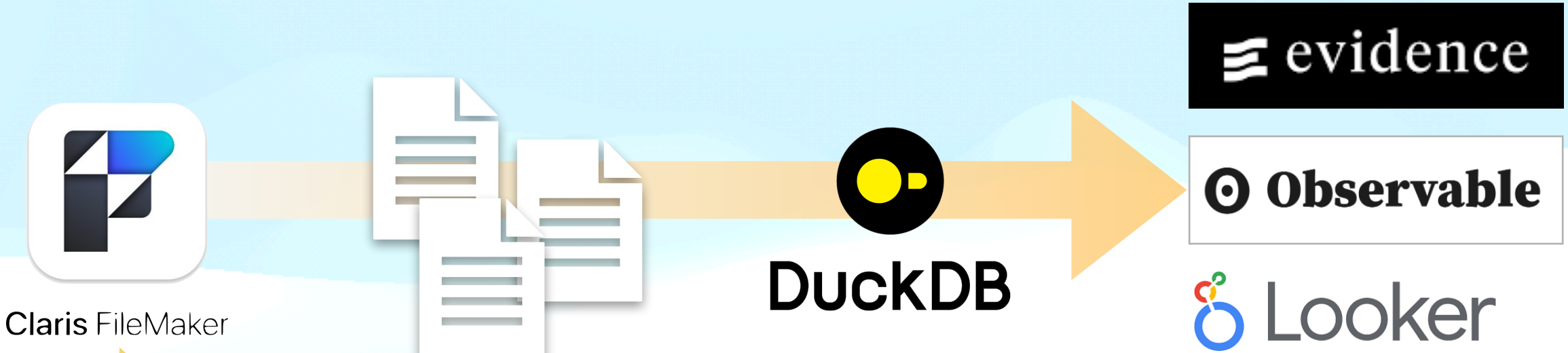
Use cases

what to do next...

- export FileMaker data as CSV and process it with DuckDB
- import Excel or CSV data, get results as JSON
- building a Pivot table from FileMaker data
- read from external data sources in the cloud
- mix and process data sources for analytics
- use **DuckDB as ETL interface**, prepare data for other systems
- process data for a **tool pipeline to display dashboards** in a FileMaker Webviewer

ETL & Dashboards

Architecture



CSV Files

master data & transactions
only needed columns!
fast enough export (mins)
repeat daily

ETL

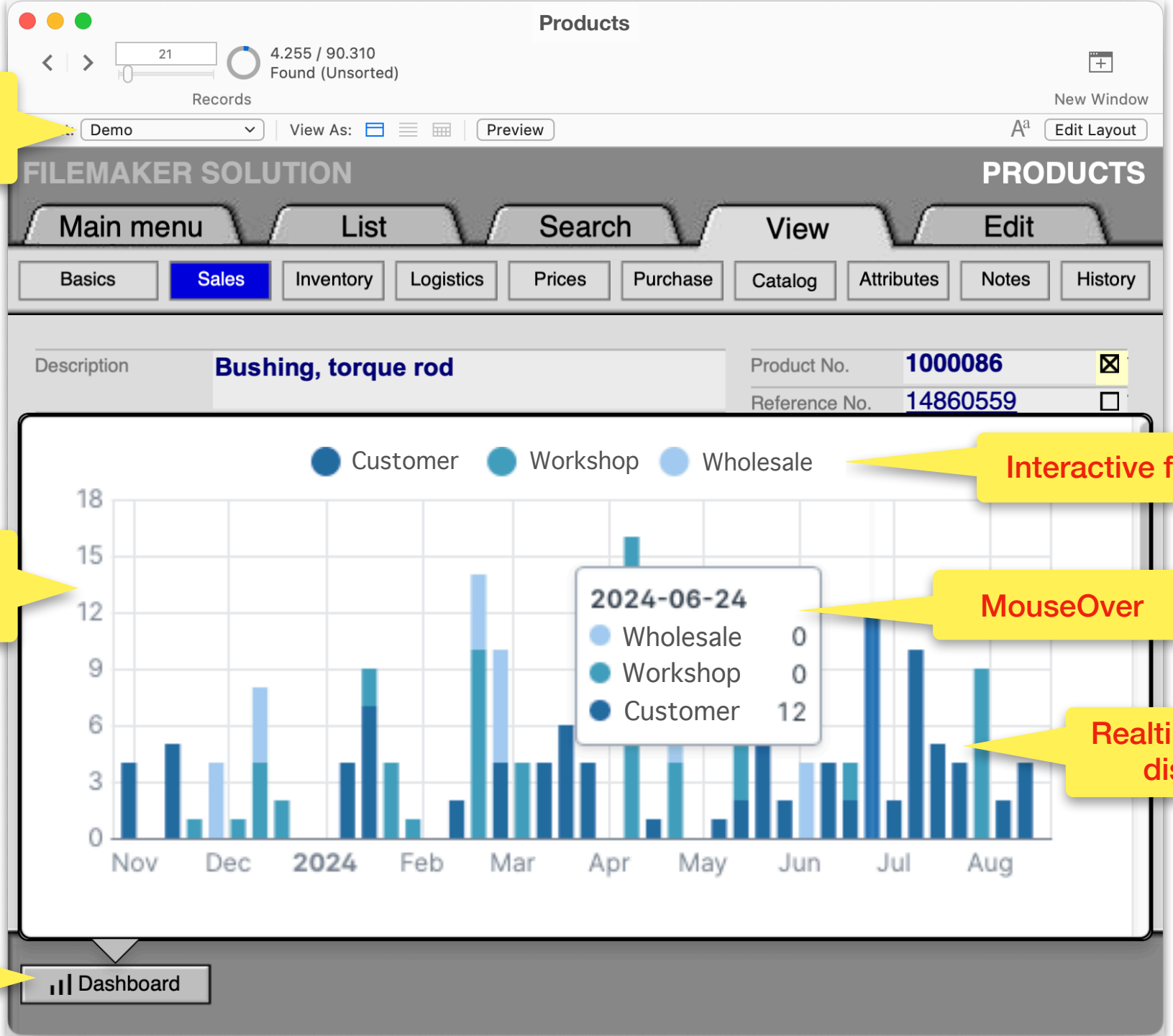
prepare for analysis
normalize
extend
mix

Dashboards

push to analytics app
local or cloud
lot of options!
display in Webviewer

EngageU
Power of the Present - Force of the Future

Demo



FileMaker solution

WebViewer

Popover button

Interactive filter

MouseOver

Realtime chart display

evidence

Summary

Summary

- just what I learned about DuckDB within a few days....
- nice tool for data crunching / analytics
- goes beyond regular SQL databases
- easy to install
- relatively easy to make practical use of

...and to show you some benefits if you want to dive deeper

Summary

- this session is meant to open a door
- node.js for REST API as a template
- FileMaker demo for own experiments
- available for download
- have fun!

DuckDB
API.fmp12

one more thing...

WASM

it even runs
in the Browser!!!

WebAssembly

https://shell.duckdb.org/

https://duckdb.org/2021/10/29/duckdb-wasm.html

further reading

References: DuckDB

- https://duckdb.org/
- https://duckdb.org/docs/
- https://duckdb.org/news/
- https://github.com/duckdb/duckdb
- https://www.manning.com/books/duckdb-in-action
- https://www.oreilly.com/library/view/duckdb-up-and/9781098159689/
- https://github.com/davidgasquez/awesome-duckdb

References: DuckDB (explanations)

- Implementing Hardware-Friendly Databases with DuckDB co-creator, Hannes Mühleisen
  https://youtu.be/pZV9FvdKmLc?&t=135
- Software Engineering Daily: DuckDB with Hannes Mühleisen
  https://softwareengineeringdaily.com/2024/08/08/duckdb-with-hannes-muhleisen/
- DuckCon #5: DuckDB – Overview and latest developments
  https://youtu.be/xX6qnP2H5wk
- Data Talks on the Rocks 5 - Hannes Mühleisen, DuckDB
  https://youtu.be/a-RmhY5RPVg

References: Dashboards

- **https://evidence.dev/** 😊👍
- https://observablehq.com/
- https://idl.uw.edu/mosaic/
- https://www.rilldata.com/
- https://streamlit.io/
- https://lookerstudio.google.com/
- https://www.claris.com/studio/
- https://evidence.dev/blog/business-intelligence-tools

open source · duckdb included

References: node.js

- https://nodejs.org/en/download/prebuilt-installer
- https://hashstudioz.com/blog/install-nodejs-npm-on-window-and-mac/
- https://blog.postman.com/how-to-create-a-rest-api-with-node-js-and-express/
- https://duckdb.org/docs/api/nodejs/overview.html

thank you!

EngageU

Power of the Present - Force of the Future

Q&A

EngageU
Power of the Present - Force of the Future