# Time Series with SQL Server 2022

DATA SATURDAYS

1nn0va community

# Sponsors

altitudo

beanTech
IT moves your business

fluentis
the ERP ready to go live!

QUANTUMDATIS

[stesi]
Powered by Innovation

With the support of:

CREDEM CREDEMTEL

‹packt›

DATA
SATURDAYS

inn0va
community

# Gianluca Sartori

Founder at Quantumdatis

Data Platform MVP

Works with SQL Server since version 7

DBA @ Scuderia Ferrari

Blog: [spaghettidba.com](spaghettidba.com)
Twitter: [@spaghettidba](@spaghettidba)

With the support of:

# AGENDA

1. What is Time Series data?
2. Options on the market: InfluxDB
3. Storing Time Series data
4. Retention Policies & Downsampling
5. Querying Times Series data

# What is Time Series data?

DATA SATURDAYS

1nn0va community

# Definitions

What is **time series data**?

> *“Time series data (or time-stamped data) is a collection of observations for a single object (entity) at different time intervals.”*

What is a **time series database**?

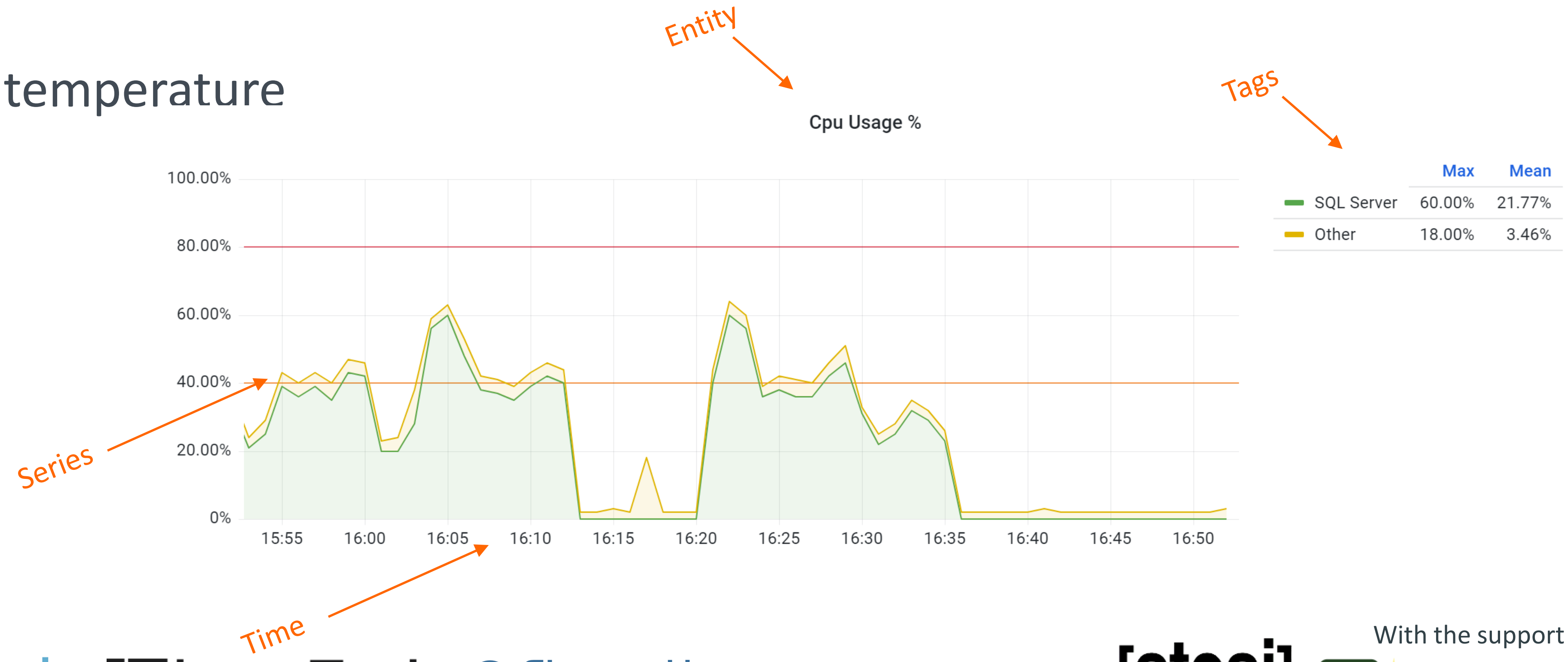
> *“A time series database (TSDB) is a database optimized for **time series data** and for measuring change over time.”*

# Time Series Data

Time series data is obtained by performing repeated measurements over time

**Examples:**

- Atmospheric temperature
- Stock prices
- CPU usage %
- Emails/sec
- Sensor data



| | Max | Mean |
|---|---|---|
| 🟩 SQL Server | 60.00% | 21.77% |
| 🟨 Other | 18.00% | 3.46% |

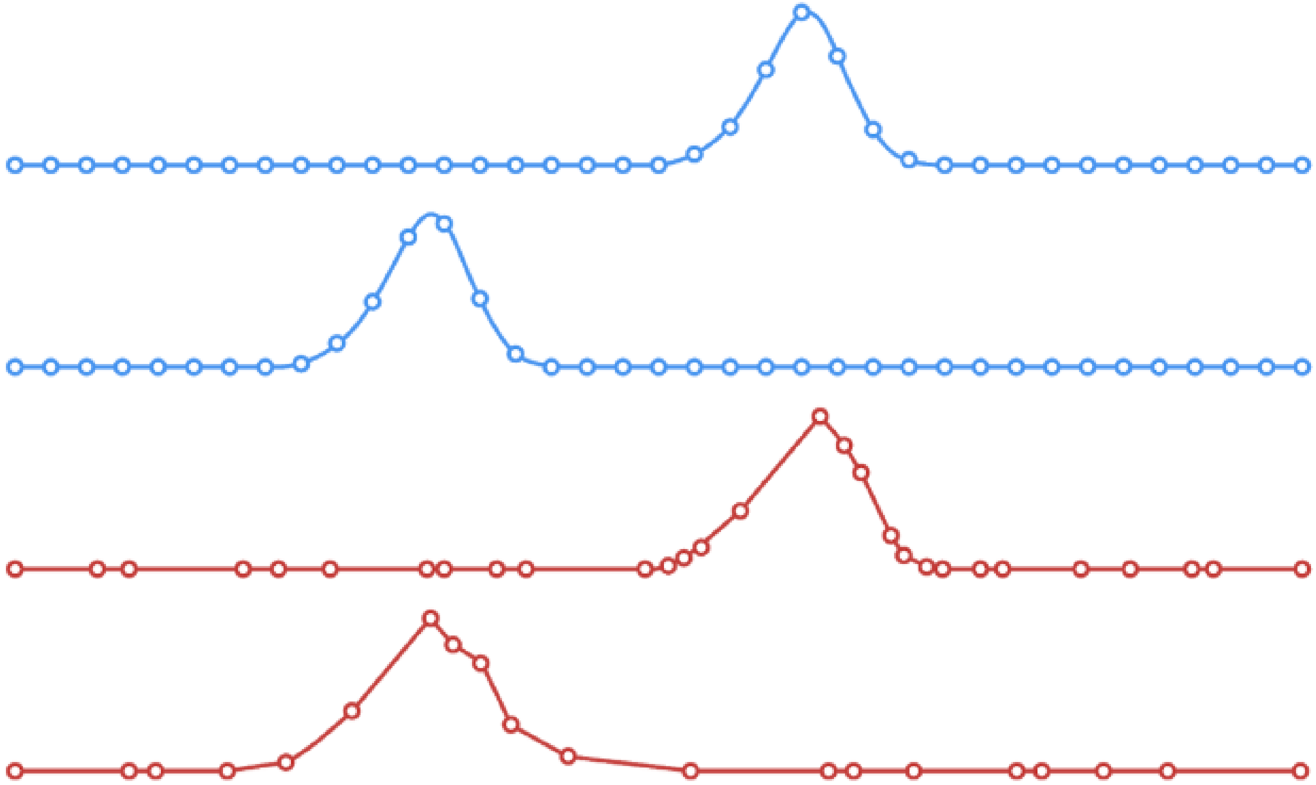
# Time Series Data

**Metrics (Regular)**

Measurements
gathered at regular
time intervals

**Events (Irregular)**

Measurements
gathered at irregular
time intervals



# Time Series Data

## How is this different from relational data?

- Continuous Stream of data in time order
- Bulk uploads of large sets of data
- High volume data
- Data is append-only – no updates
- Delete large volumes of data when it goes out of scope
- Downsampling and aggregating high resolution data to save space

# Should I use a relational db, like SQL Server?

Relational databases do not deal very well with time series data

→No built-in storage optimizations

→No easy retention policies enforcement

→No time series specific query capabilities

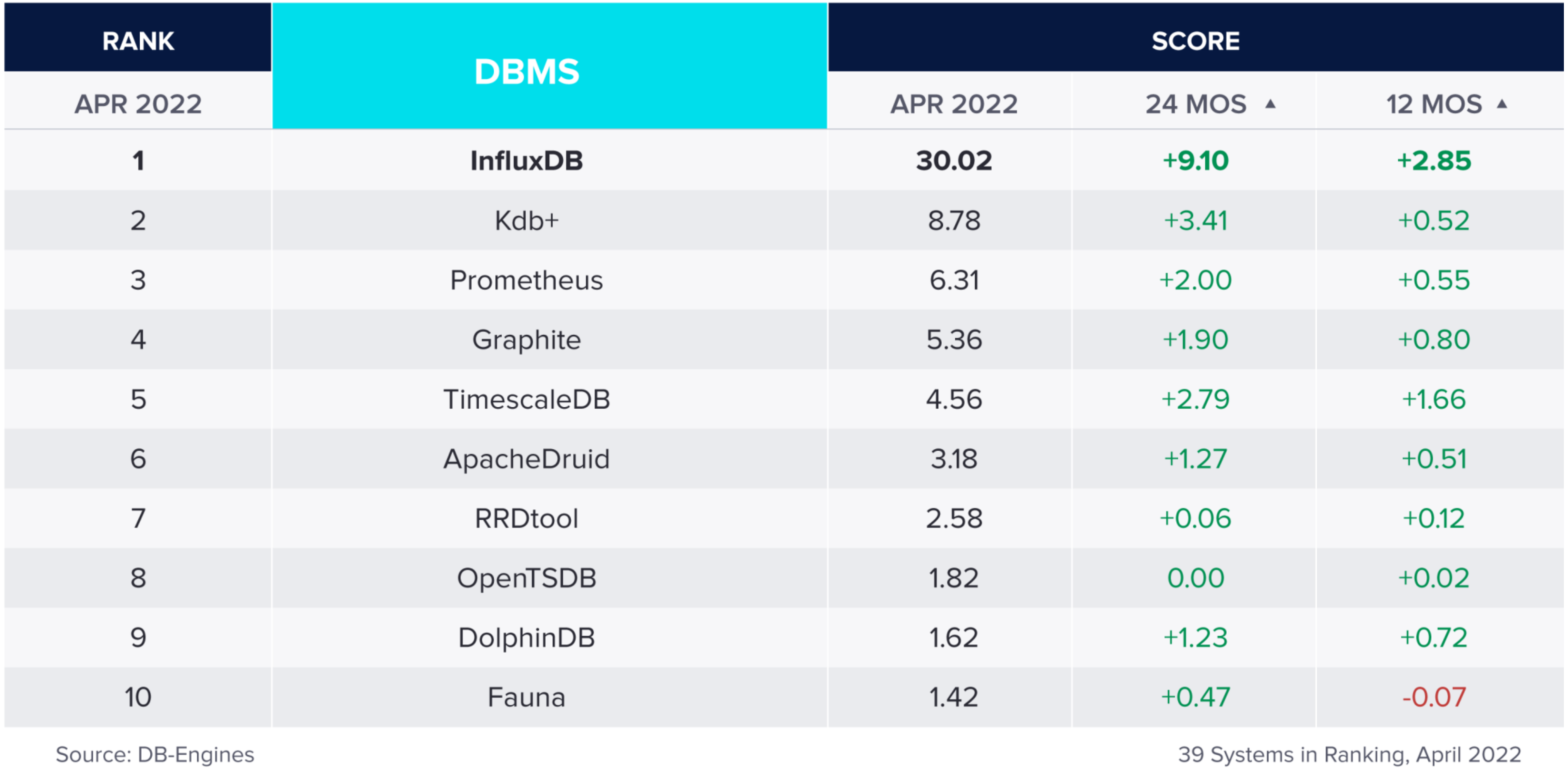
- Azure SQL Edge is designed for time series data
  See https://docs.microsoft.com/en-us/azure/azure-sql-edge/
- SQL Server 2022 gets some of that: is it any good?

# Options on the market: InfluxDB

DATA SATURDAYS

1nn0va community

# Time Series Database Options



| RANK | DBMS | SCORE | | |
|---|---|---|---|---|
| APR 2022 | | APR 2022 | 24 MOS ▲ | 12 MOS ▲ |
| **1** | **InfluxDB** | **30.02** | **+9.10** | **+2.85** |
| 2 | Kdb+ | 8.78 | +3.41 | +0.52 |
| 3 | Prometheus | 6.31 | +2.00 | +0.55 |
| 4 | Graphite | 5.36 | +1.90 | +0.80 |
| 5 | TimescaleDB | 4.56 | +2.79 | +1.66 |
| 6 | ApacheDruid | 3.18 | +1.27 | +0.51 |
| 7 | RRDtool | 2.58 | +0.06 | +0.12 |
| 8 | OpenTSDB | 1.82 | 0.00 | +0.02 |
| 9 | DolphinDB | 1.62 | +1.23 | +0.72 |
| 10 | Fauna | 1.42 | +0.47 | -0.07 |

Source: DB-Engines

39 Systems in Ranking, April 2022

With the support of:

# InfluxDB

Most popular time series database

Designed and optimized for time series data

- Uses optimized TSM storage engine with columnar compression
- Built-in retention policies

Open Source with permissive licensing (MIT)

Written in GO

Cross platform: *windows, linux, macOS, docker, kubernetes, raspberryPI*

# InfluxDB

**Different SKUs**

- *Cloud*
  → Hosted by influxdata
- *OSS 2.X*
  → Current supported version
  → More powerful
- *OSS 1.X*
  → Currently deprecated
  → **Easier to use for relational people**

# InfluxDB design principles

**Time series data:**

Series are stored in **measurements**

Measurements can have **fields** and **tags**

**Series** are sets of values of a single field, measured over time

Key of a series = `[measurement name, time, tags, field name]`

**Example:** <u>CPU Usage</u>

`CPUHistory` ← measurement name
`Time`
`ServerName` ← tag
`CPUPercent` ← field

# Series – An Example

SQL Server CPU usage percent from `sys.dm_os_ring_buffers`

| time | SQL Server CPU | Other CPU |
|---|---|---|
| 08:00 | 10 | 3 |
| 08:01 | 15 | 2 |
| 08:02 | 18 | 6 |
| 08:03 | 73 | 1 |

# Storing Time Series Data

DATA SATURDAYS

1nn0va community

# Series – An Example

influxdb

In InfluxDB this is saved in a **measurement** (let’s call it **CPUHistory)** with **two series:**

| time | SQL Server CPU | Other CPU |
|---|---|---|
| 08:00 | 10 | 3 |
| 08:01 | 15 | 2 |
| 08:02 | 18 | 6 |
| 08:03 | 73 | 1 |

**[CPUHistory, SQL Server CPU]**

| time | SQL Server CPU |
|---|---|
| 08:00 | 10 |
| 08:01 | 15 |
| 08:02 | 18 |
| 08:03 | 73 |

**[CPUHistory, Other CPU]**

| time | Other CPU |
|---|---|
| 08:00 | 3 |
| 08:01 | 2 |
| 08:02 | 6 |
| 08:03 | 1 |

# Series – An Example

influxdb

If we add **tags**, we will get **one series for each tag value:**

| time | **ServerName** | SQL Server CPU | Other CPU |
|---|---|---|---|
| 08:00 | ACCOUNTING | 10 | 3 |
| 08:00 | CRM | 37 | 7 |
| 08:01 | ACCOUNTING | 15 | 2 |
| 08:01 | CRM | 48 | 5 |
| 08:02 | ACCOUNTING | 18 | 6 |
| 08:02 | CRM | 41 | 9 |
| 08:03 | ACCOUNTING | 73 | 1 |
| 08:03 | CRM | 28 | 7 |

[CPUHistory, **ACCOUNTING**, SQLServerCPU]

| time | SQL Server CPU |
|---|---|
| 08:00 | 10 |
| 08:01 | 15 |
| 08:02 | 18 |
| 08:03 | 73 |

[CPUHistory, **ACCOUNTING**, OtherCPU]

| time | Other CPU |
|---|---|
| 08:00 | 3 |
| 08:01 | 2 |
| 08:02 | 6 |
| 08:03 | 1 |

[CPUHistory, **CRM**, OtherCPU]

| time | Other CPU |
|---|---|
| 08:00 | 7 |
| 08:01 | 5 |
| 08:02 | 9 |
| 08:03 | 7 |

[CPUHistory, **CRM**, SQLServerCPU]

| time | SQL Server CPU |
|---|---|
| 08:00 | 37 |
| 08:01 | 48 |
| 08:02 | 41 |
| 08:03 | 28 |

# InfluxDB – Storage Engine

- Series are stored independently
- We get one series for each combination of tag values → **cardinality**
- Does this look familiar? [Hint: columnstore indexes in SQL Server]

- Storage engine called **TSM** (Time series Structured Merge tree)
  - derived from **LSM** (Log-Structured Merge tree) → **Cassandra**
  - stores data in columns (series)
  - groups writes in memory and writes to disk in append mode
  - data on disk is always appended, never updated
  - series are compressed at regular intervals

# SQL Server 2022 – Storage Engine

Big Heaps or Clustered Indexes are problematic

- Huge in size
- Hard to query
- Hard to maintain

Alternatives?

- Data compression
- Clustered Columnstore Indexes

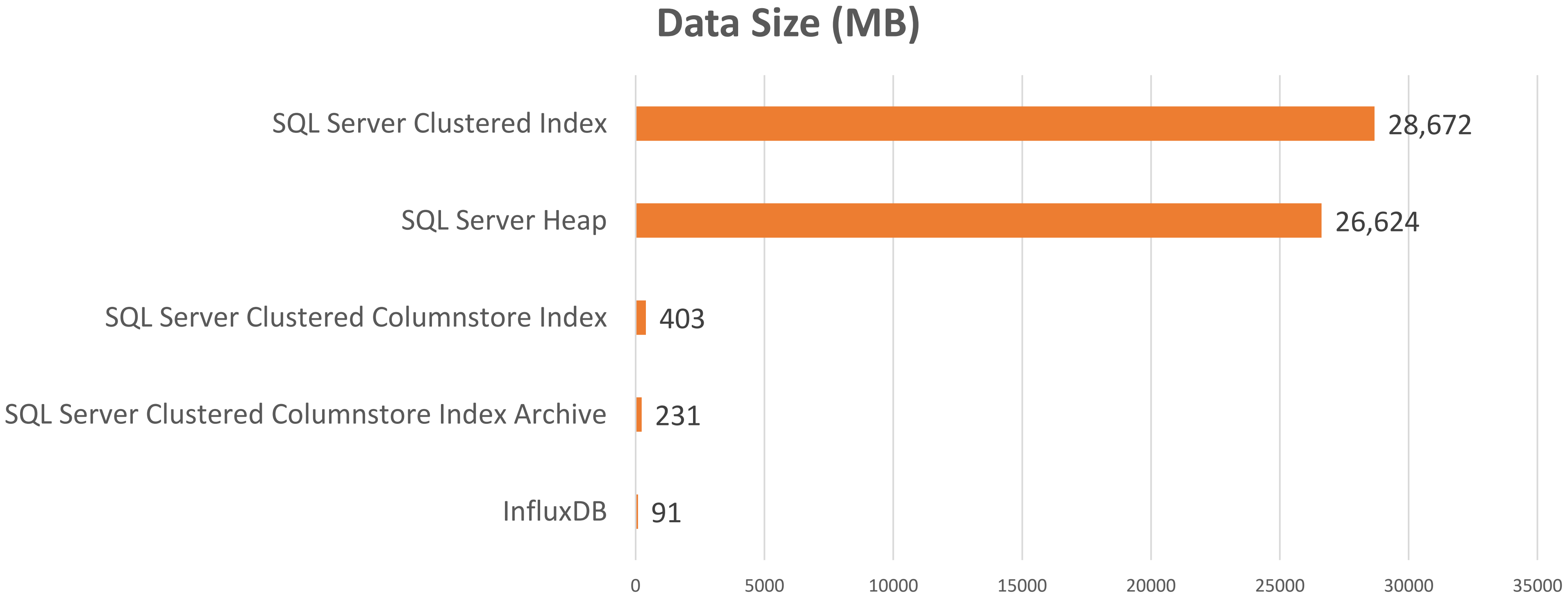
# SQL Server 2022 – Columnstore Indexes

- Each column is stored independently
- Compresses data very efficiently
- Suitable when lots of repeating data exists
- Good for bulk operations
- Don’t play well with UPDATEs and DELETEs

**DEMO!**

# SQL Server 2022 – Columnstore Indexes

Data size for SQL Server performance counters, 2 instances, 15 days of data, 82 million rows



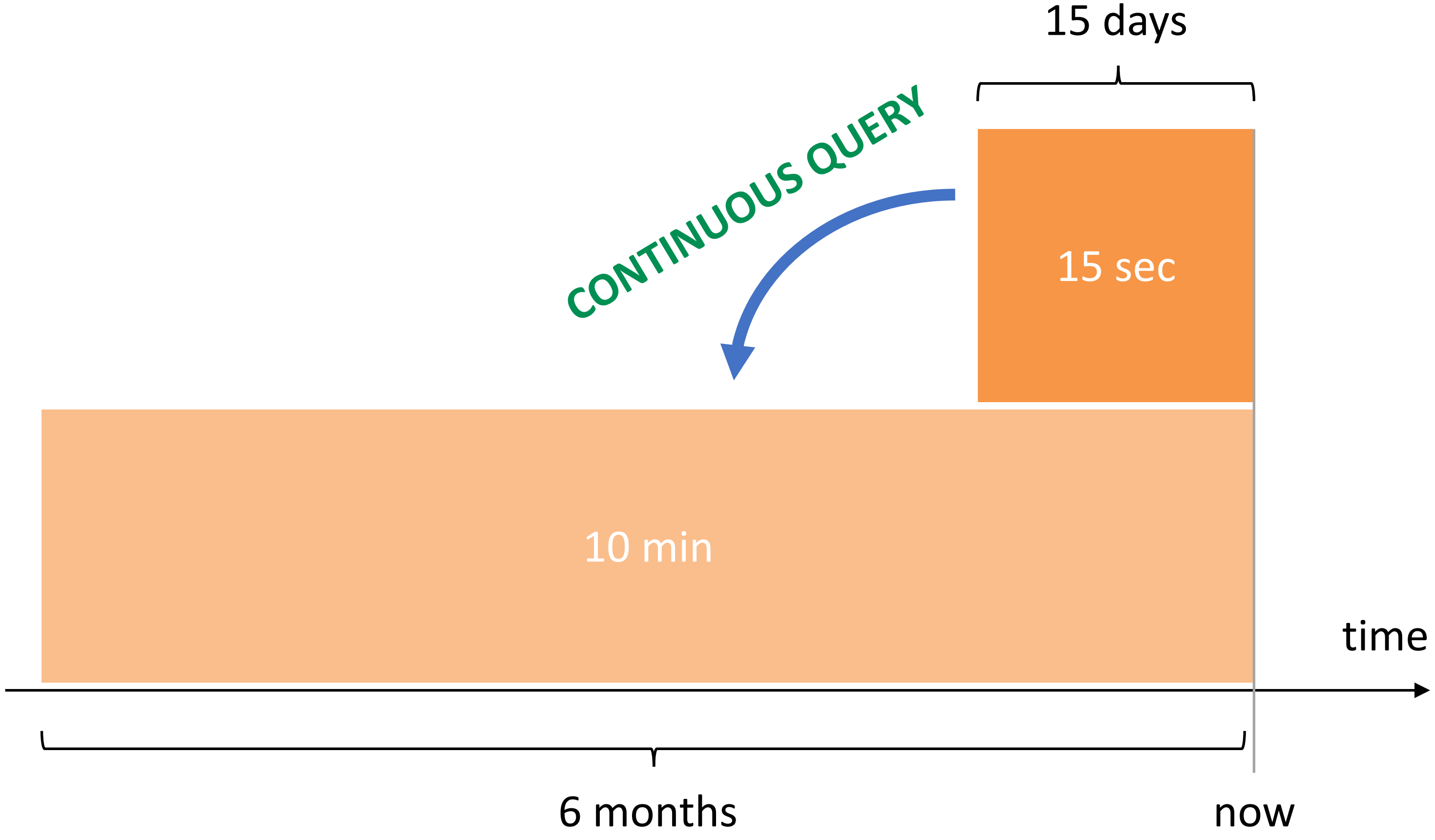
# Retention Policies & Downsampling

DATA SATURDAYS

1nn0va community

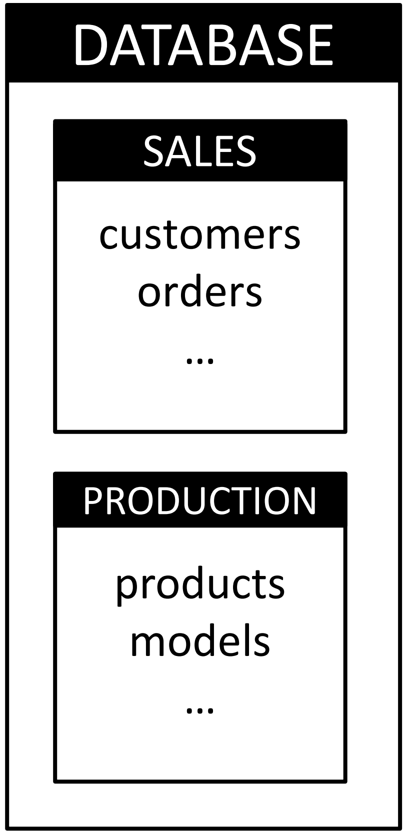
# Retention Policies & downsampling influxdb

- Default **retention policy** → ∞
- Multiple retention policies, different retention periods
- **Continuous queries** aggregate and downsample the data to long term retention policy



# Databases, retention policies, buckets…

## SQL Server



DATABASE

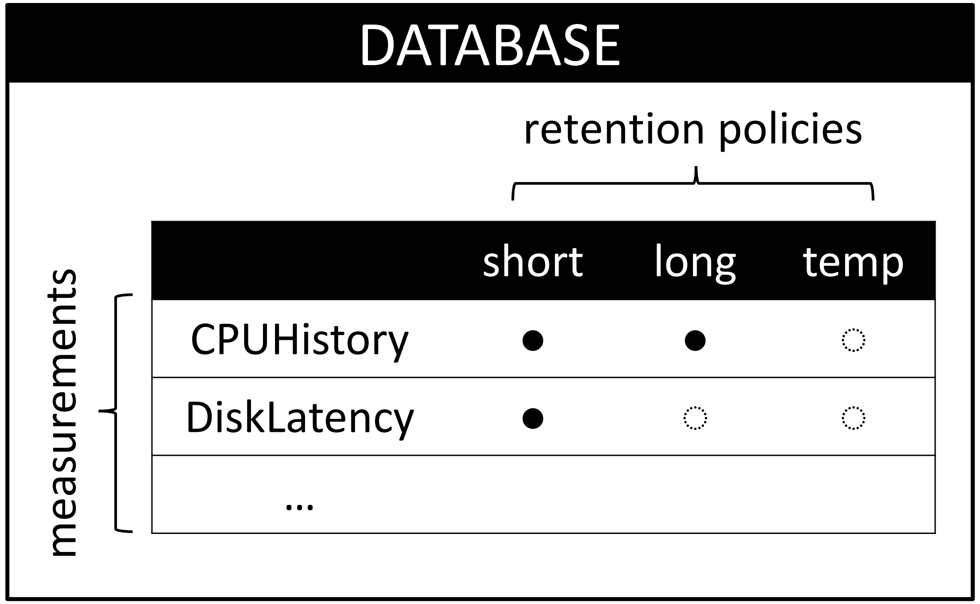
SALES

customers
orders
…

PRODUCTION

products
models
…

## InfluxDB 1.X



DATABASE

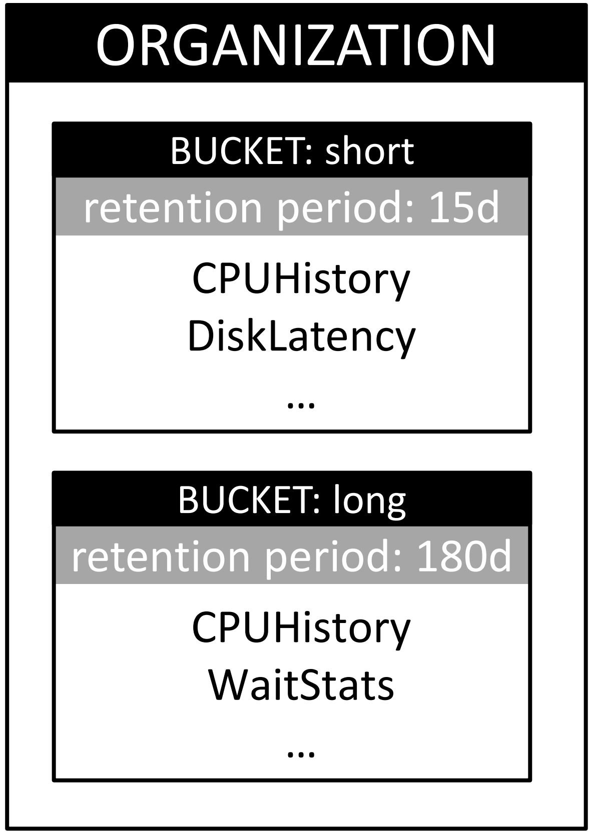
retention policies

measurements

| | short | long | temp |
|---|---|---|---|
| CPUHistory | ● | ● | ◌ |
| DiskLatency | ● | ◌ | ◌ |
| … | | | |

Measurement exists on all retention policies, you can write to whichever you need

## InfluxDB 2.X



ORGANIZATION

BUCKET: short

retention period: 15d

CPUHistory
DiskLatency
…

BUCKET: long

retention period: 180d

CPUHistory
WaitStats
…

Measurement belongs to a single bucket

# Retention Policies in Azure SQL Edge

Azure SQL Edge is a specialized SQL Server Edition for IoT applications

Designed with telemetry and Time Series data in mind

Collects data at the edge, uploads to the cloud

Supports DATA_DELETION:

```sql
ALTER TABLE [dbo].[sqlserver_performance]
SET (
        DATA_DELETION = ON (
                FILTER_COLUMN = [time],
                RETENTION_PERIOD = 1 month
        )
)
```

# Retention Policies in SQL Server 2022

SQL Server 2022 gets new Time Series features…

```
ALTER TABLE [dbo].[sqlserver_performance]
SET (
        DATA_DELETION = ON (
                FILTER_COLUMN = [time],
                RETENTION_PERIOD = 1 month
        )
)
```

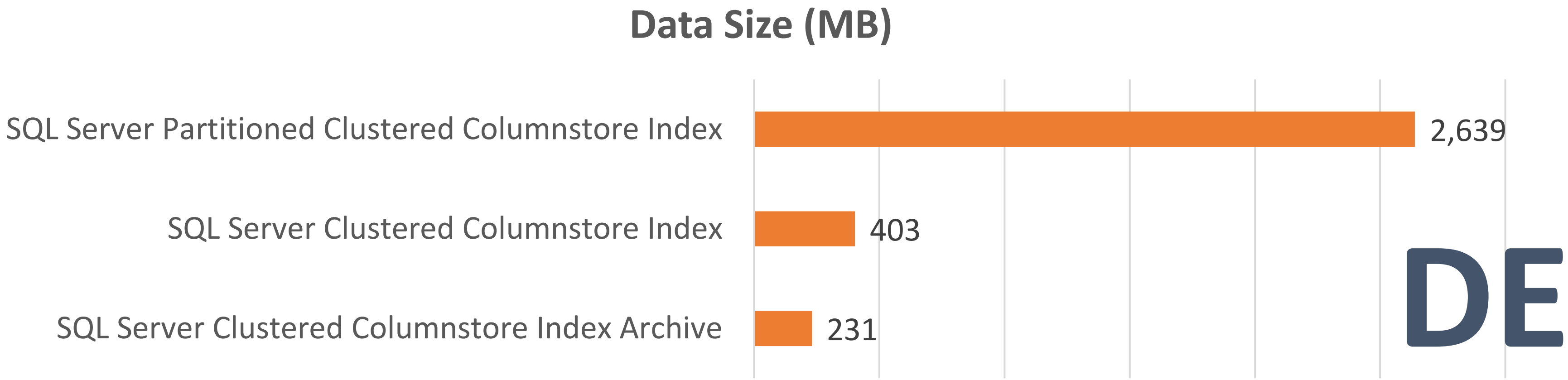
```
Msg 102, Level 15, State 1, Line 117
Incorrect syntax near 'DATA_DELETION'.
```

… does not get
this one though ☹

With the support of:

# Retention Policies in SQL Server 2022

- SQL Agent Job → DELETE
- Lots of rows → DELETE takes a long time, uses a lot of transaction log
- Columnstore Indexes don't like DELETEs
- Partitioning allows switching data in and out with metadata-only operations
- Different partitions can have different compression settings
- Reduces efficiency of columnstore compression (each partition is independent)



DEMO!

With the support of:

# Querying Time Series data

DATA
SATURDAYS

1hn0va
community

# Querying data in InfluxDB

InfluxDB supports 2 query languages:

**InfluxQL** → similar to SQL, born with version 1, still available in v2

```
SELECT MEAN(SQLServerCPU) AS avg_cpu
FROM CPUHistory
WHERE time > now() -1d
GROUP BY server_name
```

# Querying data with Flux

**Flux** → doesn’t look like SQL (more like kusto for Azure Data Explorer)

Default in v2, also available in v1

Supports «advanced» query features like joins…

```
from(bucket:"demo/autogen" )
     |> range(start: -1d)
     |> filter(fn: (r) => r._measurement == "CPUHistory")
     |> filter(fn: (r) => r._field == "SQLServerCPU")
     |> mean()
     |> group(columns: ["server_name"])
     |> yield(name: "avg_cpu")
```

With the support of:

# InfluxQL: SELECT syntax

```
SELECT <fields>
    FROM <retention_policy>.<measurement>
    [ INTO <retention_policy>.<measurement> ]
    [ WHERE <search_condition> ]
    [ GROUP BY <tags> [ FILL(<fill_expression>) ] ]
    [ ORDER BY <order_by_expression> ]
    [ LIMIT <number> ]
    [ OFFSET <number> ]
    [ SLIMIT <number> ]
    [ SOFFSET <number> ]
    [ TZ(<timezone_clause>) ]
```

With the support of:

# GROUP BY TIME

InfluxQL allows grouping by time:

```
SELECT MEAN(value) AS avg_lazy_writes_sec
FROM "sqlserver_performance"
WHERE counter = 'Lazy writes/sec'
    AND sql_instance = 'SQLCSRV04:SQL2017'
    AND time > now() - 1d
GROUP BY time(1h)
```

# GROUP BY TIME

```
… GROUP BY time(1h)
```

| time | lazy_writes_sec |
|---|---|
| 16-10-2022 10:00:00 | 398091 |
| … | … |
| 16-10-2022 10:59:59 | 395644 |
| 16-10-2022 11:00:00 | 432211 |
| … | … |
| 16-10-2022 11:59:59 | 452325 |
| 16-10-2022 12:00:00 | 456541 |
| … | … |
| 16-10-2022 12:59:59 | 433505 |
| 16-10-2022 13:00:00 | 456654 |
| … | … |
| 16-10-2022 13:59:59 | 456788 |
| 16-10-2022 14:00:00 | 417545 |
| … | … |

10:00

11:00

12:00

13:00

…

| time | avg_lazy_writes_sec |
|---|---|
| 16-10-2022 10:00:00 | 398,091.00 |
| 16-10-2022 11:00:00 | 400,187.50 |
| 16-10-2022 12:00:00 | 406,095.00 |
| 16-10-2022 13:00:00 | 411,167.76 |
| 16-10-2022 14:00:00 | 417,963.40 |
| 16-10-2022 15:00:00 | 425,683.27 |
| 16-10-2022 16:00:00 | 433,001.40 |
| 16-10-2022 17:00:00 | 441,937.02 |
| 16-10-2022 18:00:00 | 449,937.73 |
| 16-10-2022 19:00:00 | 450,737.81 |
| 16-10-2022 20:00:00 | 453,467.40 |
| 16-10-2022 21:00:00 | 455,703.76 |
| 16-10-2022 22:00:00 | 457,270.19 |
| 16-10-2022 23:00:00 | 458,743.47 |
| 17-10-2022 00:00:00 | 461,533.64 |
| … | … |

# Introducing DATE_BUCKET

New DATE_BUCKET function in SQL Server 2022!!

```
SELECT DATE_BUCKET(hour, 1, time) AS time,
       AVG(value) AS avg_lazy_writes_sec
FROM [dbo].[sqlserver_performance_cci_partitioned]
WHERE time >= '2022-10-01'
    AND time < '2022-10-02'
    AND counter = 'Lazy writes/sec'
    AND sql_instance = 'SQLCSRV04:SQL2017'
GROUP BY DATE_BUCKET(hour, 1, time)
```

DEMO!

With the support of:

# FILL

InfluxQL allows filling intervals with data:

```
SELECT MEAN(value) AS avg_lazy_writes_sec
FROM "sqlserver_performance"
WHERE counter = 'Lazy writes/sec'
    AND sql_instance = 'SQLCSRV04:SQL2017'
    AND time > now() - 1d
GROUP BY time(1h) FILL(NULL)
```

# FILL

```
… FILL(NULL|VALUE)
```

| time | value |
|---|---|
| 16-10-2022 10:35:45 | 398091 |
| 16-10-2022 10:36:01 | 398091 |
| 16-10-2022 12:16:15 | 406095 |
| 16-10-2022 12:16:30 | 406095 |
| 16-10-2022 13:06:45 | 411167 |
| 16-10-2022 13:07:01 | 411167 |

No data for 11:00

→

| time | avg_lazy_writes_sec |
|---|---|
| 16-10-2022 10:00:00 | 398,091.00 |
| 16-10-2022 11:00:00 | NULL |
| 16-10-2022 12:00:00 | 406,095.00 |
| 16-10-2022 13:00:00 | 411,167.76 |

# Introducing GENERATE_SERIES

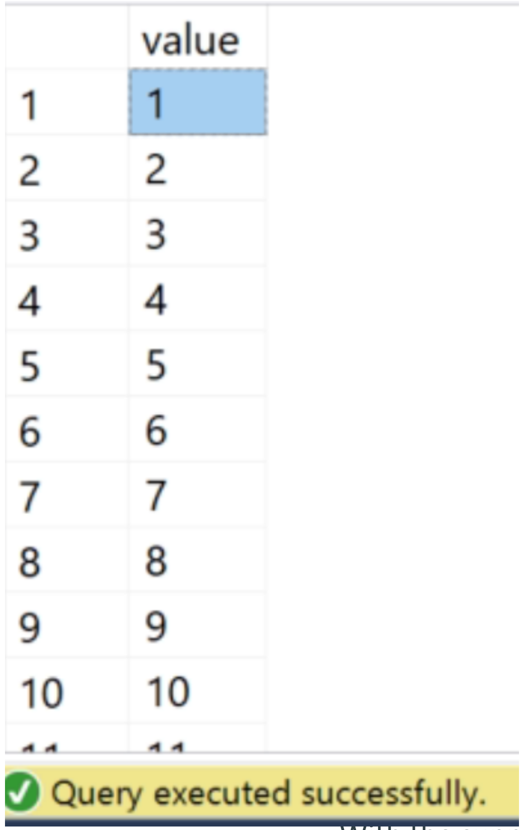
New GENERATE_SERIES function in SQL Server 2022!!

```
SELECT * FROM GENERATE_SERIES(1, 100);
```

Generates a set of values from argument 1 to argument 2

Can be combined with DATE_BUCKET

DEMO!



| | value |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |

Query executed successfully.

# Conclusions

# Is SQL Server 2022 good for time series data?

- Brings innovative T-SQL syntax to handle time series data
- Already has adequate storage capabilities
- Performance is acceptable
- Not as capable as a proper Time Series database like InfluxDB
  - Requires managing some things manually (retention policies, downsampling)
- Offers T-SQL syntax and 360° support for relational data

## THANK YOU!

## Got questions? Email me!

Gianluca Sartori
Quantumdatis, owner

gianluca.sartori@quantumdatis.com
@spaghettidba