Time series databases with TimeScaleDB and InfluxDB

check out our GitHub repository

TEAM MEMBERS

Kristóf Balázs: 000612294
Stefanos Kypritidis : 000606810
Nishant Sushmakar : 000606016
Olha Baliasina : 000603977

01

Time Series Databases

Introduction | Application | Tools

02

Database Management Systems

TimescaleDB | InfluxDB | Differences | Grafana

03

Tool Comparison

TSBS | Custom Benchmark

04

Conclusion & Insights

Tool Suitability | Key Takeaways | Exercises

01 **TIME SERIES DATABASES**

Introduction | *Application* | *Tools*

02 **DATABASE MANAGEMENT SYSTEMS**

TimescaleDB | *InfluxDB* | *Differences* | *Grafana*

03 **TOOL COMPARISON**

TSBS | *Custom Benchmark*

04 **CONCLUSION & INSIGHTS**

Tool Suitability | *Key Takeaways* | *Exercises*

Introduction

WHAT

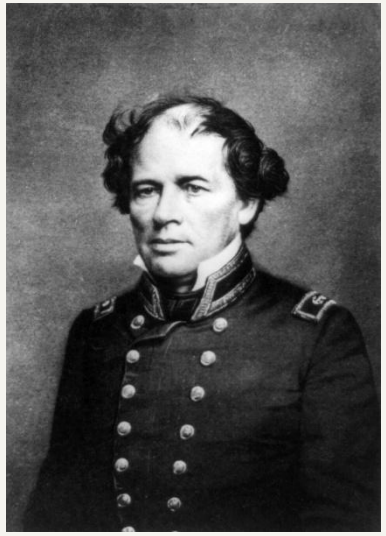
Time series databases are designed for storing and accessing data where each data point is associated with a specific time. This type of data is common in many applications such as sensor readings, stock prices and log monitoring

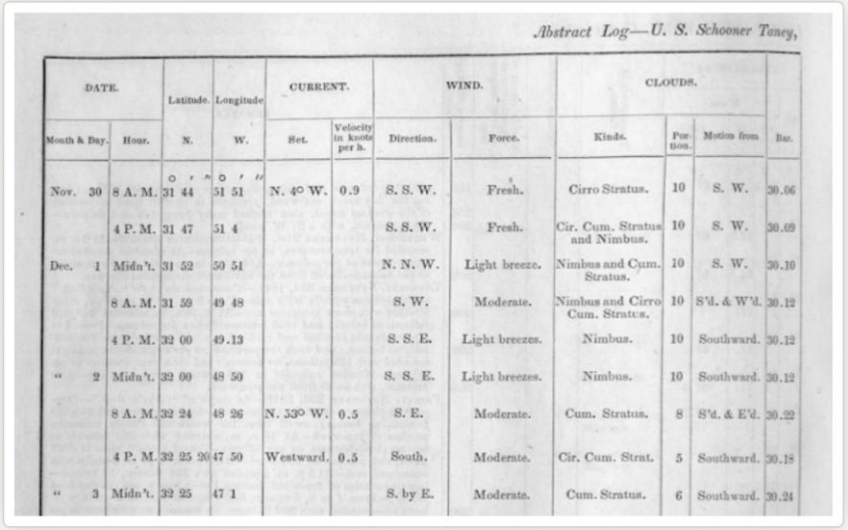
WHY

Time series data can reveal trends, patterns, and correlations that would not be apparent from single measurements

The primary focus of a TSDB is to optimize queries that retrieve data within a specific time range, making these databases suitable for applications where time-based analysis is crucial

Time Series is an Old Idea





Abstract Log—U. S. Schooner Taney,

DATE.		Latitude.	Longitude.	CURRENT.		WIND.		CLOUDS.			
Month & Day.	Hour.	N.	W.	Set.	Velocity in knots per h.	Direction.	Force.	Kinds.	Portions.	Motion from	Bar.
		° ′ ″	° ′ ″								
Nov. 30	8 A. M.	31 44	51 51	N. 4° W.	0.9	S. S. W.	Fresh.	Cirro Stratus.	10	S. W.	30.06
	4 P. M.	31 47	51 4			S. S. W.	Fresh.	Cir. Cum. Stratus and Nimbus.	10	S. W.	30.09
Dec. 1	Midn't.	31 52	50 51			N. N. W.	Light breeze.	Nimbus and Cum. Stratus.	10	S. W.	30.10
	8 A. M.	31 59	49 48			S. W.	Moderate.	Nimbus and Cirro Cum. Stratus.	10	S'd. & W'd.	30.12
	4 P. M.	32 00	49.13			S. S. E.	Light breezes.	Nimbus.	10	Southward.	30.12
" 2	Midn't.	32 00	48 50			S. S. E.	Light breezes.	Nimbus.	10	Southward.	30.12
	8 A. M.	32 24	48 26	N. 33° W.	0.5	S. E.	Moderate.	Cum. Stratus.	8	S'd. & E'd.	30.22
	4 P. M.	32 25 20	47 50	Westward.	0.5	South.	Moderate.	Cir. Cum. Strat.	5	Southward.	30.18
" 3	Midn't.	32 25	47 1			S. by E.	Moderate.	Cum. Stratus.	6	Southward.	30.24

- Matthew Fontaine Maury, known as the Pathfinder of the Seas, was a sailor who, due to a leg injury, turned to scientific research in various subject which included meteorology, astronomy, oceanography etc.
- In Mid-19th century ship captains and science officers traditionally kept logbooks detailing speed, location, and ocean and weather observations.
- Maury saw the collective value of ship logs and aimed to share it with captains. As head of the US Navy's Depot of Charts and Instruments, he began extracting wind and current data from years of logs.
- He analysed this time series data to recommend optimal shipping routes based on prevailing winds and currents
- In 1848, Maury sent one of his Wind and Current Charts to Captain Jackson, who used the evidence-based route to Rio de Janeiro. Captain Jackson saved 17 days on the outbound voyage and even more on the return trip
- Maury encouraged more regular and systematic time series data collection by creating the "Abstract Log for the Use of American Navigators."
- Maury's work is an early example of crowdsourced, open-source, big data project, since he relied on data from many ships and encouraged more to collect data in a standardized way

Sources:

Application of Time Series Databases

- **Trend Analysis**: Detect long-term increases, decreases, or stability in values to guide decisions like capacity planning and inventory management.
- **Seasonality Detection**: Identify recurring patterns, such as holiday sales spikes or peak web traffic times, to improve forecasting and resource allocation.
- **Forecasting**: Predict future behaviour using methods from moving averages to machine learning for applications like stock prices, weather, or equipment failure.
- **Event Detection**: Spot anomalies, such as spikes in latency or dips in yield, to enable early diagnosis and proactive intervention.

Tools

☐ include secondary database models — 44 systems in ranking, December 2024

Rank Dec 2024	Rank Nov 2024	Rank Dec 2023	DBMS	Database Model	Score Dec 2024	Score Nov 2024	Score Dec 2023
1.	1.	1.	InfluxDB	Time Series, Multi-model	21.23	-0.24	-6.88
2.	↑ 3.	2.	Prometheus	Time Series	7.12	+0.20	-1.21
3.	↓ 2.	3.	Kdb	Multi-model	6.99	-0.07	-1.29
4.	4.	4.	Graphite	Time Series	4.81	-0.10	-0.65
5.	5.	5.	TimescaleDB	Time Series, Multi-model	3.74	+0.07	-1.55
6.	6.	↑ 10.	QuestDB	Time Series, Multi-model	2.98	+0.07	+0.67
7.	7.	7.	Apache Druid	Multi-model	2.88	+0.17	-0.45
8.	8.	↓ 6.	DolphinDB	Multi-model	2.54	-0.05	-1.37
9.	9.	↓ 8.	TDengine	Time Series, Multi-model	2.11	-0.16	-1.09
10.	10.	↑ 11.	GridDB	Time Series, Multi-model	1.94	+0.02	-0.30
11.	11.	↓ 9.	RRDtool	Time Series	1.64	-0.07	-1.18
12.	12.	12.	OpenTSDB	Time Series	1.48	-0.03	-0.57
13.	13.	13.	Fauna	Multi-model	1.48	+0.04	-0.28
14.	14.	↑ 15.	Apache IoTDB	Time Series	1.40	-0.01	+0.08
15.	15.	↓ 14.	VictoriaMetrics	Time Series	1.24	-0.04	-0.22
16.	16.	↑ 17.	Amazon Timestream	Time Series	1.23	+0.01	+0.04
17.	17.	↑ 20.	eXtremeDB	Multi-model	0.99	+0.05	+0.04
18.	18.	↓ 16.	M3DB	Time Series	0.99	+0.05	-0.20
19.	19.	↓ 18.	CrateDB	Multi-model	0.66	+0.01	-0.35
20.	20.	↓ 19.	KairosDB	Time Series	0.62	+0.05	-0.34
21.	↑ 22.	↑ 22.	ITTIA	Time Series, Multi-model	0.42	+0.04	-0.26
22.	↓ 21.	↑ 24.	Raima Database Manager	Multi-model	0.37	-0.06	-0.21
23.	23.	23.	CnosDB	Time Series	0.32	+0.03	-0.30
24.	24.	↑ 37.	Alibaba Cloud TSDB	Time Series	0.27	+0.03	+0.15
25.	25.	25.	Axibase	Time Series	0.22	+0.02	-0.15
26.	26.	↑ 31.	IBM Db2 Event Store	Multi-model	0.19	+0.01	-0.02
27.	27.	↑ 28.	Riak TS	Time Series	0.17	+0.01	-0.14
28.	28.	↑ 29.	Quasardb	Time Series	0.14	+0.01	-0.15
29.	29.	↑ 33.	GreptimeDB	Time Series	0.13	+0.02	-0.04
30.	↑ 38.	↓ 21.	Heroic	Time Series	0.11	+0.09	-0.72
31.	↑ 33.	↓ 26.	Machbase Neo	Time Series	0.09	+0.03	-0.26
32.	↓ 31.	↓ 27.	Bangdb	Multi-model	0.08	+0.02	-0.25
33.	↓ 30.		OpenMLDB	Time Series, Multi-model	0.08	+0.00	
34.	↓ 32.	↑ 35.	SiteWhere	Time Series	0.06	0.00	-0.09
35.	↓ 34.	↑ 39.	Warp 10	Time Series	0.06	0.00	-0.04
36.	↓ 35.	36.	Blueflood	Time Series	0.06	0.00	-0.08
37.	↓ 36.	↓ 32.	Tigris	Multi-model	0.05	0.00	-0.14
38.	↑ 41.	↑ 40.	SiriDB	Time Series	0.04	+0.04	-0.03
39.	↓ 37.	↓ 30.	ArcadeDB	Multi-model	0.04	+0.00	-0.17
40.	↓ 39.		ReductStore	Time Series	0.02	+0.02	
41.	41.	↑ 42.	Newts	Time Series	0.02	+0.02	+0.02
42.	↓ 41.	↓ 34.	openGemini	Time Series	0.02	+0.02	-0.14
43.	↓ 40.	↓ 38.	Hawkular Metrics	Time Series	0.01	+0.01	-0.10
44.	↓ 41.	↓ 41.	NSDb	Time Series	0.00	±0.00	-0.04

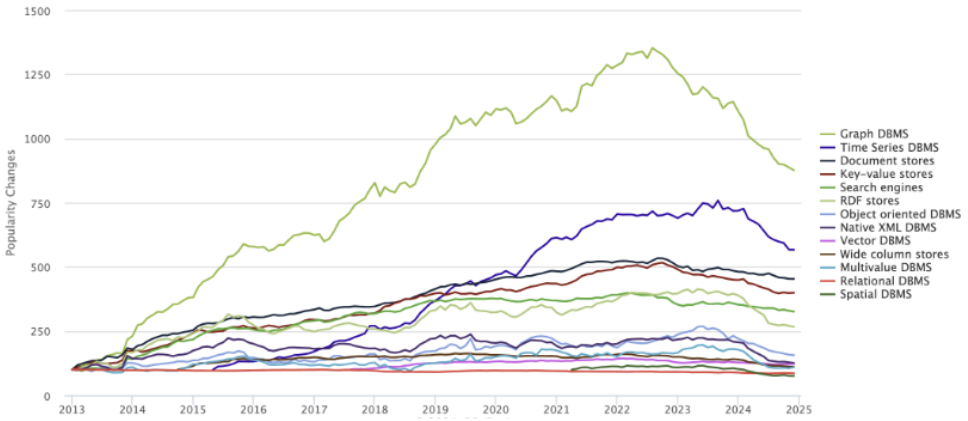


Figure 1.1: DBMS popularity (since 2013)

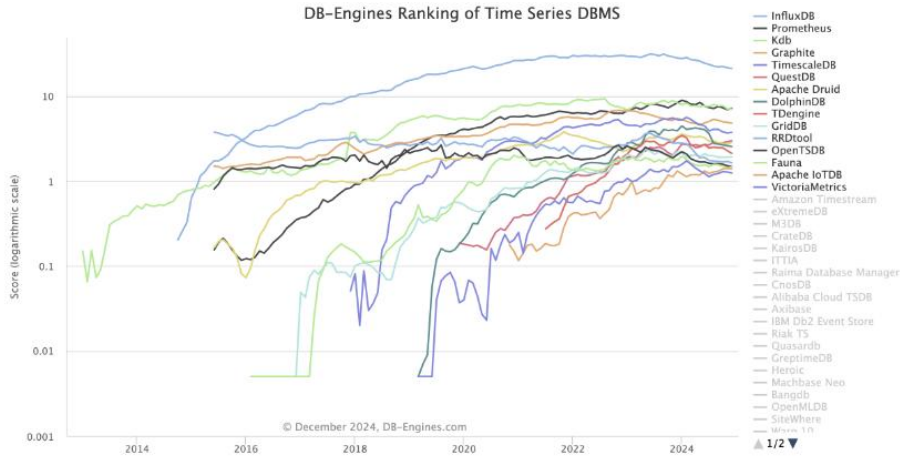


Figure 1.2: Time series DBMS ranking (since 2013)

Sources:

DATABASE MANAGEMENT SYSTEMS

01 Time Series Databases

Introduction | Application | Tools

02 Database Management Systems

TimescaleDB *| InfluxDB | Differences | Grafana*

03 Tool Comparison

TSBS | Custom Benchmark

04 Conclusion & Insights

Tool Suitability | Key Takeaways | Exercises

WHAT IS TIMESCALE?

Open-source relational time series database built atop of PostgreSQL

History

- Launched on April 4, 2017
- 3,400 stars on Github in the first year
- Currently over 18,000 stars

SQL Compatibility

- Allows users to write queries with familiar constructs, easing onboarding, and promoting adoption in SQL-proficient organizations.

Scalability and Performance

- Efficient handling of high write loads
- Automated partitioning
- Parallelization of queries
- Advanced indexing strategies

Key Feautures

- Hypertables
- Compression
- Data retention
- Continuous aggregates

Sources:docs.timescale

ARCHITECTURE EXAMPLE

1) Aggregate data for further processing

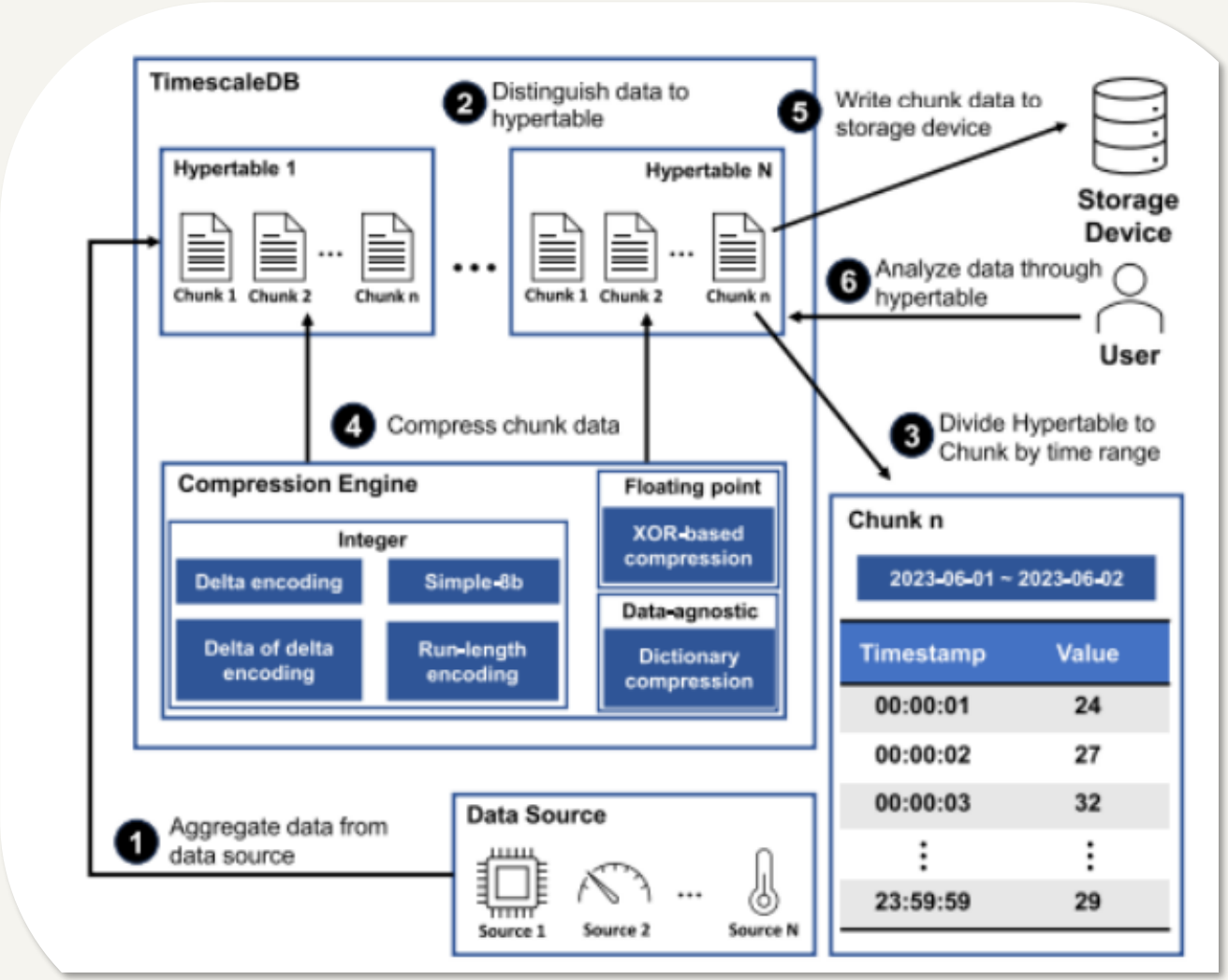
2) Record data in appropriate Hypertable

3) Divide Hypertable to chunks based on time range

4) Compress chunk data

5) Write chunks to storage device

6) Analyze data using timescale functionality



Sources: Analyzing I/O Characteristics of Time-Series Data Using High Performance Storage Devices

HYPERTABLES

Hypertables consist of chunks that partition data along one or two dimensions

Partition dimensions: Time interval and, optionally, another partition key such as a device identifier

PostgreSQL: Hypertables appear and behave like standard PostgreSQL tables

Automatic handling: Complexity of partitioning, storage optimization, and indexing happens behind the scenes

Optimization: Automatically segmenting data into manageable intervals (**chunk_size**) aligned with the dataset's temporal characteristics

Relational Modeling: Non-time-series data can remain in regular PostgreSQL tables, preserving the flexibility and power of relational modeling

```
SELECT create_hypertable('weather_conditions',
                         by_range('timestamp_column', INTERVAL '1 day'));
```

Sources: docs.timescale

CONTINUOUS AGGREGATES

A hypertable type that is continuously refreshed in the background, incrementally updating as new data is added or old data is modified.

Performance benefits: Outperform regular materialized views as they do not rebuild from scratch with each update. Pre-computing aggregates, allowing for much faster query execution compared to raw data queries.

Easy use: Query them like regular tables, enabling efficient data access while also supporting advanced features

Build on top: Continuous aggregates can be built on top of other continuous aggregates, providing flexibility for complex data analysis

Real-time aggregates: Combining pre-aggregated data with recent, unaggregated raw data, offers both high performance and up-to-date results

Restrictions: Direct updates or deletes are not possible and data lag when not combined with raw data

Refresh policy: how and when the materialized data is updated → **Data Freshness**

Sources: docs.timescale

CONTINUOUS AGGREGATES SYNTAX

```
CREATE MATERIALIZED VIEW conditions_summary_daily
WITH (timescaledb.continuous) AS
SELECT location_name,
    time_bucket(INTERVAL '1 day', timestamp_column) AS bucket,
    AVG(temperature),
    MAX(temperature),
    MIN(temperature)
FROM weather_conditions
GROUP BY location_name, bucket;
```

Continuous Aggregate Creation

Calculates daily mean, min, max of temperature for each location using **time_bucket()**

Automatic Refresh Policy

Include data from one month before the current time until one day before the current time and refreshes every hour

```
SELECT add_continuous_aggregate_policy('conditions_summary_daily',
    start_offset => INTERVAL '1 month',
    end_offset => INTERVAL '1 day',
    schedule_interval => INTERVAL '1 hour');
```

COMPRESSION

Compression operates on chunk level

How: Groups multiple records of the same chunk into a single row replacing numerous rows with array-like structures within the columns of these rows.

Advantages: Disk space usage minimization and optimization of query performance

Structure: Compressed data is stored in a column-order structure managed by timescaleDB

Optimization: The parameter **chunk_time_interval** and strategies like ordering and segmenting data can maximize compression.

Compression algorithms: Delta encoding, delta-of delta encoding, simple-8b, and run-length encoding, XOR-based compression, Dictionary Compression

Automatic selection: TimescaleDB automatically applies the optimal compression algorithm, eliminating the need for user intervention or concern

Sources: docs.timescale

COMPRESSION EXAMPLE

Timestamp	Device ID	Device Type	CPU
12:00:01	A	SSD	70.11
12:00:01	B	HDD	69.70
12:00:02	A	SSD	70.12
12:00:02	B	HDD	69.69
12:00:03	A	SSD	70.14
12:00:03	B	HDD	69.70

Timestamp	Device ID	Device Type	CPU	Disk IO
[12:00:01, 12:00:01, 12:00:02, 12:00:02, 12:00:03, 12:00:03]	[A, B, A, B, A, B]	[SSD, HDD, SSD, HDD, SSD, HDD]	[70.11, 69.70, 70.12, 69.69, 70.14, 69.70]	[13.4, 20.5, 13.2, 23.4, 13.0, 25.2]

Sources: docs.timescale

COMPRESSION SYNTAX

```
ALTER TABLE weather_conditions
SET (
    timescaledb.compress,
    timescaledb.compress_orderby='timestamp_column'
);
```

Enabling compression

Ordering the data by timestamp will improve compression ratio and performance of your queries

Automatic compression policy

Compress chunks that are older than seven days

```
SELECT add_compression_policy('weather_conditions',
                              INTERVAL '7 days');
```

DATA RETENTION

Facilitates the removal of outdated data

Old data: In time-series analyses, data often loses relevance as it ages

Automatic data retention: Streamline the deletion of outdated data

Manual control: Enables users to selectively remove specific data chunks

Data downsampling: Integrates retention policies with continuous aggregates to create summarized datasets and remove the old raw data

Automatic retention policy

Removes data older than 2 months. This policy is recurring and runs daily (default)

```
SELECT add_retention_policy('weather_conditions',
                            INTERVAL '2 months');
```

FINE-TUNING CHUNK SIZE

chunk_time_interval is a crucial setting

Why: Defines the size of each chunk and should be defined when creating a hypertable

Default: The default is 7 days

Advantages: Optimizes efficient data management and query performance for specific application

For best compression: Use **larger** chunks that store more rows → larger chunk_time_interval

For dropping raw data as quickly as possible after materialization: Smaller chunks that align more closely with the time_bucket interval defined in the continuous aggregate

For efficient utilization of server resources: Chunk_size that **balances** other priorities while ensuring that the active chunks from all hypertables together use approximately **25%** of the PostgreSQL memory allocation

Sources: timescale.com/blog

QUERY OPTIMIZATION

Indexes

Hypertables: Automatic creation of a B-tree index on the time column after creation of hypertable

PostgreSQL compatibility: all of PostgreSQL's standard indexing mechanisms can be used

Unique indexes on hypertable: Ensuring uniqueness and speeding up queries but must include all the partitioning columns while non-partitioning columns can be included.

```
CREATE UNIQUE INDEX idx_location_time
ON weather_conditions(location_name, timestamp_column);
```

Chunk skipping

Basic querying: a query does not scan the entire table but only the relevant chunks

Where to use: On a non-partitioning **column x** which is correlated with the partitioning column and commonly used in WHERE filter clauses.

Functionality: Tracks the min and max values for **the column x** of every chunk, storing them in the **chunk_column_stats** table. When column x is in WHERE clause condition, timescaleDB uses **chunk_column_stats** to exclude irrelevant chunks, speeding up the query.

```
SELECT enable_chunk_skipping('weather_conditions',
                             'temperature');
```

Sources: docs.timescale

DATABASE MANAGEMENT SYSTEMS

01 Time Series Databases

Introduction | Application | Tools

02 Database Management Systems

*TimescaleDB | **InfluxDB** | Differences | Grafana*

03 Tool Comparison

TSBS | Custom Benchmark

04 Conclusion & Insights

Tool Suitability | Key Takeaways | Exercises

WHAT IS INFLUX?

Open-source No-SQL time series database

Purpose

- designed to handle high-write loads and large-scale data storage

Usecases

- Ideal for metrics, events, logs, and real-time analytics

Key Characteristics

- Time-stamped data
- Optimized for time-series workloads

Multiple versions

- OSS (v1, v2, v3)
- Enterprise
- Clustered
- Cloud

Sources:docs.influxdata.com

KEY FEATURES

Schema Flexibility

High-Write Throughput

InfluxQL: SQL-like Query Language

Ecosystem and Integrations

DATA MODEL AND SERIES REPRESENTATION

InfluxDB

Database:
American_Manufacturing

Measurement:
Power Generators

	Tags			Fields	
	Location	**Model**	**Unit Serial No**	**Temperature**	**State**
Series #1	Houston	Siemens A10	A1230FL1	92.34094	ON
	Houston	Siemens A10	A1230FL1	93.49348	ON
Series #2	Denver	Caterpillar S9	Q1293832	78.34434	OFF
Series #3	Denver	Caterpillar S9	Q9238754	77.19948	ON

Line protocol

provides the measurement, tag set, field set, and timestamp of a data point

measurement | tag set | field set | timestamp

```
measurement,tag1=val1,tag2=val2 field1="v1",field2=1i 0000000000000000000
```

STORAGE ENGINE

Time-Structured Merge Tree (TSM)

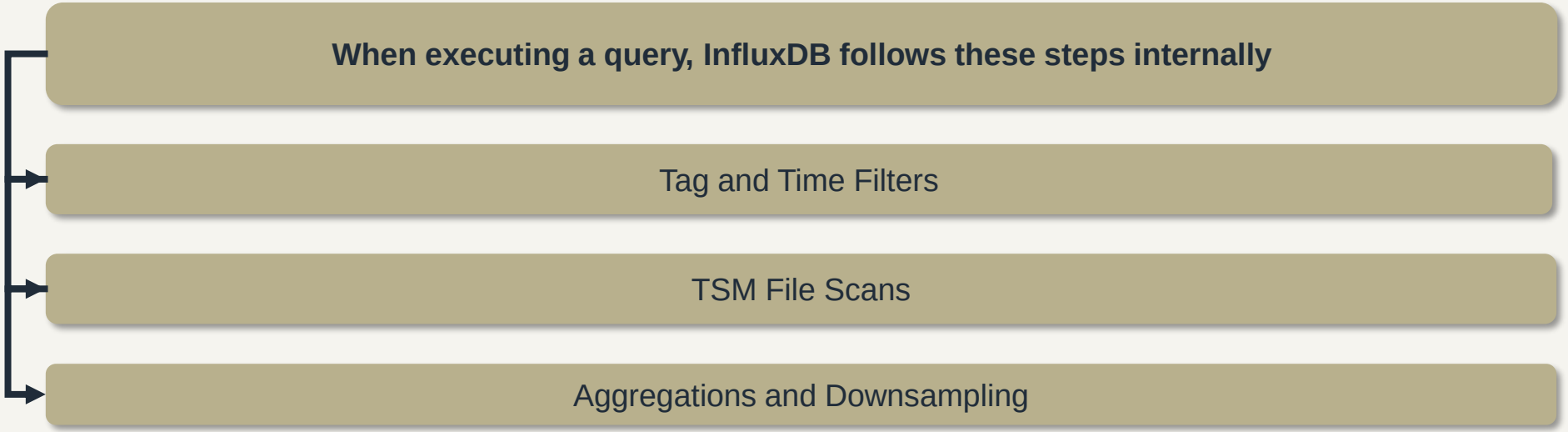
What: custom-built storage engine tailored for time-series data

Why: improve write and query performance, durability, and resource efficiency

Key aspects:
- Shard-Based Partitioning
- Write-Ahead Log (WAL) and Caching
- TSM Files (Immutable, Compressed Storage)
- Compression and Encoding

Indexing:
- Tag-Based Indexing
- In-Memory Index Structures

QUERY EXECUTION



LIFECYCLE MANAGEMENT AND RETENTION POLICIES

Retention Policies

Define how long data stays in the database before being automatically expired

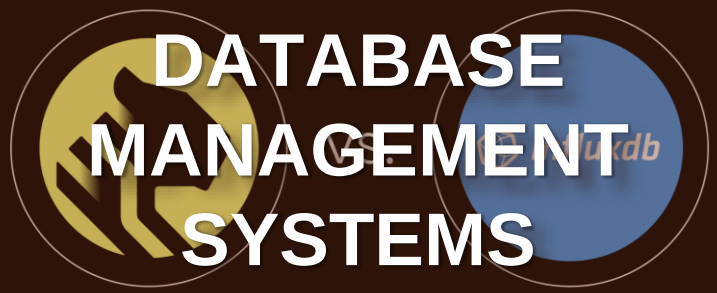
Continuous Queries

InfluxQL queries that run automatically and periodically on realtime data and store query results in a specified measurement

RP duration	Shard group interval
< 2 days	1 hour
[2 days, 6 months]	1 day
> 6 months	7 days

```
CREATE CONTINUOUS QUERY <cq_name> ON <database_name>
BEGIN
  SELECT <function[s]> INTO <destination_measurement> FROM
<measurement> [WHERE <stuff>] GROUP BY time(<interval>)[,
<tag_key[s]>]
END
```

DATABASE MANAGEMENT SYSTEMS



01 **Time Series Databases**

Introduction | Section2 | Section3

02 **Database Management Systems**

*TimescaleDB | InfluxDB | Differences | **Grafana***

03 **Tool Comparison**

TSBS | Custom Benchmark

04 **Conclusion & Insights**

Tool Suitability | Key Takeaways | Exercises

GRAFANA WITH INFLUXDB & TIMESCALEDB

	TIMESCALEDB	INFLUXDB
Integration	**Built-in** data source plugin	**PostgreSQL** data source plugin
	Using **docker-compose**	
Language	**SQL** with **Timescale functionalities**	**InfluxQL** and **Flux**
Features	**Grafana Alerting**: **monitors query results** and can **trigger alerts based on thresholds**, **trends**, or **patterns** in time-series data that we specify	
	Downsampling, aggregation, and continuous queries	
	time_bucket(): data to fixed intervals	**GROUP BY time()**: data to fixed intervals
	Streaming data visualization	
	Geospatial data visualization	

Sources: grafana.com; docs.influxdata.com; docs.timescale.com

01 **Time Series Databases**

Introduction | Section2 | Section3

02 **Database Management Systems**

TimescaleDB | InfluxDB | Differences | Grafana

03 **Tool Comparison**

TSBS *| Custom Benchmark*

04 **Conclusion & Insights**

Tool Suitability | Key Takeaways | Exercises

TIME SERIES BENCHMARK SUITE

Written in **Go**, TSBS offers a **toolkit for benchmarking the performance of time-series databases**.

Made by the **creators of Timescale**

2 MAIN PERFORMANCE METRICS

- **Bulk data loading**
- **Query execution performance**

3 USE CASES

- **IoT**
- **Dev ops**
- **CPU only**

The IoT use case is modeled around a **fleet management system**, and it **simulates data streams from trucks** in a **fictional logistics company.**

Data challenges: out-of-order events, batch ingestion from offline devices, and missing entries.

Metrics examples: fuel levels, geolocation, load capacity, and operational status.

Data is produced **deterministically**: pseudo-random number generator (PRNG).

SUPPORTED DATABASE TECHNOLOGIES:

Akumuli
Cassandra
ClickHouse
CrateDB
InfluxDB
MongoDB
QuestDB
SiriDB
TimescaleDB
Timestream
VictoriaMetrics

Sources: github.com; timescale.com

TIME SERIES BENCHMARK SUITE

Written in **Go**, TSBS offers a **to**

SAMPLE DATA FOR THE IOT USE CASE

Made by the **creators of Timescale**

2 MAIN PERFORMANCE METRICS

3 USE CASES

SUPPORTED DATABASE TECHNOLOGIES:

- CrateDB
- **InfluxDB**
- MongoDB
- QuestDB
- SiriDB
- **TimescaleDB**
- Timestream
- VictoriaMetrics

TAGS TABLE

ID	Name	Fleet	Driver	Model	Version	Load Cap.	Fuel Cap.	Fuel Cons.
1	truck_0	South	Trish	H-2	v2.3	1500	150	12
2	truck_2	North	Derek	F-150	v1.5	2000	200	15
3	truck_3	East	Albert	F-150	v2.0	2000	200	15
4	truck_6	East	Trish	G-2000	v1.0	5000	300	19
5	truck_10	North	Albert	F-150	v2.3	2000	200	15

DIAGNOSTICS TABLE

Time	Tags ID	Fuel State	Load	Status	Add. Tags
2016-01-01 01:00	19035	1	0	0	
2016-01-01 01:00	18568	1	0	0	
2016-01-01 01:00	21277	1	0	0	
2016-01-01 01:00	18558	1	0	0	
2016-01-01 01:00	22636	1	0	0	

The IoT use case is modeled around a **fleet management system**, and it **simulates data streams from trucks** in a **fictional logistics company**

Data challenges: o

Metrics exa

Data is prod

READINGS TABLE

Time	Tags ID	Lat.	Long.	Elev.	Vel.	Head.	Grade	Fuel Cons.	Add. Tags
2016-01-01 01:00	1785	14.07	110.77	152	0	69	0	25	
2016-01-01 01:00	4274	52.31	4.72	124	0	221	0	25	
2016-01-01 01:00	9	66.68	105.76	222	0	252	0	25	
2016-01-01 01:00	1875	6.78	166.86	1	0	112	0	25	
2016-01-01 01:00	458	81.93	56.12	236	0	335	0	25	

Sources: github.com; timescale.com

TSBS – QUERIES & SPECIFICATIONS

There are **12 query options** for benchmarking the IoT use case, **each designed to showcase a different scenario**.

EVERY QUERY TESTED

Query Type	Description
last-loc	Fetch real-time (i.e., last) location of each truck.
low-fuel	Fetch all trucks with low fuel (less than 10%).
high-load	Fetch trucks with high current load (over 90% load capacity).
stationary-trucks	Fetch all trucks that are stationary (low average velocity in the last 10 minutes).
long-driving-sessions	Get trucks which haven't rested for at least 20 minutes in the last 4 hours.
long-daily-sessions	Get trucks which drove more than 10 hours in the last 24 hours.
avg-vs-projected-fuel-consumption	Calculate average vs. projected fuel consumption per fleet.
avg-daily-driving-duration	Calculate average daily driving duration per driver.
avg-daily-driving-session	Calculate average daily driving session per driver.
avg-load	Calculate average load per truck model per fleet.
daily-activity	Get the number of hours a truck has been active (vs. out-of-commission) per day per fleet.
breakdown-frequency	Calculate breakdown frequency by truck model.

HARDWARE SPECIFICATIONS

Processor	Intel® Core™ i5-13500H, 12C (4P + 8E) / 16T, P-core 2.6, 4.7GHz, E-core 1.9 / 3.5GHz, 18MB
RAM	32.0 GB LPDDR5-5200
GPU	NVIDIA GeForce RTX 3050 6GB GDDR6 Laptop GPU
Storage	1TB SSD M.2 2242 PCIe 4.0x4 NVMe

SOFTWARE SPECIFICATIONS

Software	Version
PostgreSQL	13.17
TimescaleDB	2.6.1
InfluxDB	1.11.8
Ubuntu	22.04.5 LTS
Go	1.18.1
Python	3.10.12

Sources: github.com; timescale.com

TSBS SETUP

DATA GENERATION
10 days worth of data
Log interval: **1 second**
Scale factor: **number of trucks** tracked

```
./tsbs_generate_data --use-case="iot" --seed=123 --scale=SF \
--timestamp-start="2016-01-01T00:00:00Z" --timestamp-end="2016-01-10T00:00:00Z" \
--log-interval="1s" --format="timescaledb" > ../tmp/data_ts_sfX.txt
```

DATA LOADING
8 concurrently inserting clients
Batch size: **10,000** (default)
Chunk time: **12 hours** (default)

```
cat ../tmp/data_ts_sfX.txt | ./tsbs_load_timescaledb \
--host="localhost" --port=5432 --user="postgres" --pass="xxx" \
--db-name="benchmark" --workers=8
```

Custom bash scripts for **generating the queries**

Automate testing every query

```
python3 generate_run_script.py \
-b 500 \
-d timescaledb \
-e '--postgres="host=localhost
user=postgres password=xxx
dbname=benchmark sslmode=disable"' \
-f ./queries.txt \
-l ../tmp \
-n 500 \
-o ../tmp/queries_timescaledb_sfX \
-s localhost \
-w 8 >
../tmp/query_test_timescaledb.sh
```

500 runs on each query

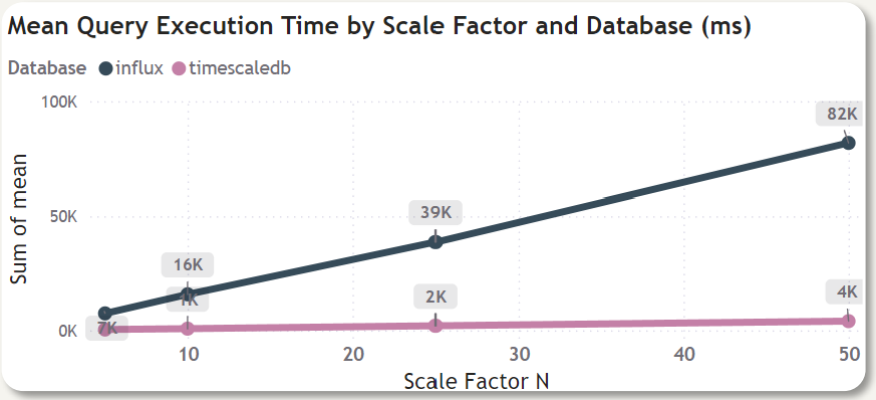
SF: 5, 10, 25, 50

Aggregate the results

Transform and visualize in Power BI

TSBS RESULTS

AVG-LOAD



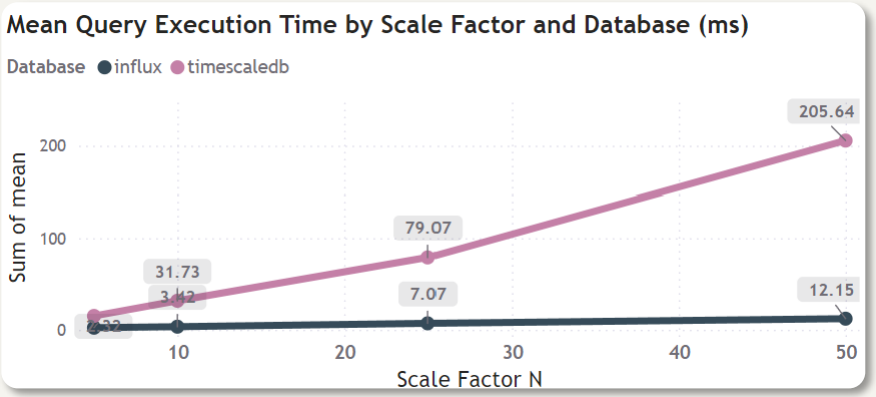
TimescaleDB - avg-load Query

```
SELECT t.fleet, t.model, t.load_capacity,
    avg(d.avg_load / t.load_capacity)
    AS avg_load_percentage
FROM tags t
INNER JOIN (
    SELECT tags_id, avg(current_load)
    AS avg_load
    FROM diagnostics
    GROUP BY tags_id
) d ON t.id = d.tags_id
GROUP BY fleet, model, load_capacity;
```

InfluxDB - avg-load Query

```
SELECT mean("ml") AS mean_load_percentage
FROM (
    SELECT "current_load" / "load_capacity"
    AS "ml"
    FROM "diagnostics"
    GROUP BY "name", "fleet", "model"
)
GROUP BY "fleet", "model";
```

LONG-DRIVING-SESSIONS



TimescaleDB - long-driving-sessions Query

```
WITH driving_sessions AS (
    SELECT time_bucket('10 minutes', time)
    AS ten_minutes, tags_id
    FROM readings
    WHERE velocity > 1
    GROUP BY ten_minutes, tags_id
),
long_driving_sessions AS (
    SELECT time_bucket('4 hours', ten_minutes)
    AS session,
           tags_id, count(*) AS periods
    FROM driving_sessions
    GROUP BY session, tags_id
)
SELECT t.name, t.driver
FROM tags t
INNER JOIN long_driving_sessions d
ON t.id = d.tags_id
WHERE d.periods > calculated_value;
```

InfluxDB - long-driving-sessions Query

```
SELECT "name", "driver"
FROM (
    SELECT count(*) AS ten_min
    FROM (
        SELECT mean("velocity")
        AS mean_velocity
        FROM "readings"
        WHERE "fleet" = 'random fleet'
              AND time > 'start time'
              AND time <= 'end time'
        GROUP BY time(10m), "name", "driver"
    )
    WHERE "mean_velocity" > 1
)
WHERE ten_min > calculated_value;
```

01 **Time Series Databases**

Introduction | Section2 | Section3

02 **Database Management Systems**

TimescaleDB | InfluxDB | Differences | Grafana

03 **Tool Comparison**

TSBS | ***Custom Benchmark***

04 **Conclusion & Insights**

Tool Suitability | Key Takeaways | Exercises

Why Custom Benchmark ?

- **Bias Toward Specific Products**: Many time-series benchmarks are created by companies promoting their own products, leading to biased results that favour certain databases.

- **Lack of Flexibility**: Existing benchmarks are often rigid and can't be customized to test unique workloads, data models, or specific use cases effectively.

Custom Benchmark - Dataset

Table Name	Description
ACC	Data includes the `Timestamp` as a datetime value, with accelerometer data for the `X`, `Y`, and `Z` orientations.
BVP	Refers to blood volume pulse; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
Dexcom	Refers to interstitial glucose concentration; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
EDA	Refers to electrodermal activity; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
TEMP	Refers to skin temperature; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
IBI	Refers to interbeat interval; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
HR	Refers to heart rate; data includes `Timestamp` as a datetime value and `Value` as the measurement recorded at the time.
Food Log	Refers to food items consumed by the participant throughout the study; data includes: • `date`: as a date value • `time_of_day`: as a time value • `time_begin` and `time_end`: as datetime values • `logged_food`: as a string value • `amount`: as a numeric value • `unit`: as a string value • `searched_food`: as a string value • `calorie`: as a numeric value • `total_carb`, `dietary_fiber`, `sugar`, `protein`, and `total_fat`: as numeric values

Source: sphysionet.org

Custom Benchmark – Setup (TimeScale DB)


```
SELECT create_hypertable('accelerometer_data','ts',chunk_time_interval => INTERVAL '1 days');
```

```
ALTER TABLE accelerometer_data SET (timescaledb.compress,
timescaledb.compress_segmentby = 'participant_id',
timescaledb.compress_orderby='ts DESC');
SELECT add_compression_policy('accelerometer_data', INTERVAL '2 weeks');
```

Custom Benchmark – Setup (Influx DB)

Measurement	Tags	Fields
demographics	ID, Gender	HbA1c
accelerometer_data	participant_id	acc_x, acc_y, acc_z
blood_volume_pulse	participant_id	bvp
interstitial_glucose	participant_id, event_type, source_device_id	event_subtype, device_info, glucose_value, insulin_value, carb_value, duration, glucose_rate_change, transmitter_time
electrodermal_activity	participant_id	eda
heart_rate_data	participant_id	hr
ibi_data	participant_id	ibi
temperature_data	participant_id	temp

Table 3.7: InfluxDB Database Schema: Measurements, Tags, and Fields

Custom Benchmark – QUERIES & SPECIFICATIONS

There were **9 queries** designed for the benchmark

Query Name	Description and Explanation
Highest Heart Rates for Each Day	Identifies the top three highest heart rates for each day within a specific week, grouped by participants. Useful for monitoring peak activity or potential stress events.
Rate of Change in Blood Volume Pulse (BVP)	Calculates the rate of change in BVP over five-second intervals, grouped by participant. Indicates cardiovascular changes during stress or exercise.
Hourly Glucose Levels with Statistics	Computes hourly mean and standard deviation of glucose levels, grouped by participants. Provides insight into blood sugar trends and abnormalities.
Minimum Daily Average Electrodermal Activity (EDA)	Finds the lowest daily average EDA values, grouped by participants. Identifies calm or inactive periods based on stress-related skin conductance.
Maximum Movement	Calculates the peak movement magnitude for each participant from accelerometer data. Highlights the highest physical activity levels.
Temperature Difference	Determines the range between the highest and lowest temperatures for each participant. Useful for detecting environmental changes or thermoregulation.
Count of Interbeat Intervals (IBI) Above Threshold	Counts IBIs greater than one second, grouped by participants. Key for understanding heart rate variability related to stress or health conditions.
Daily Glucose Level Changes	Captures daily non-negative differences in maximum glucose values, grouped by participants. Tracks daily glucose trends and responses to dietary or medical adjustments.
Median Heart Rate Over Six-Hour Intervals	Calculates the median heart rate over six-hour intervals, grouped by participants. Summarizes central cardiovascular activity while minimizing outlier effects.

HARDWARE SPECIFICATIONS

Processor	Apple M3 Pro
RAM	18.0 GB
Storage	512GB SSD

SOFTWARE SPECIFICATIONS

Software	Version
PostgreSQL	14.2
TimescaleDB	2.6.1
InfluxDB	1.11.8
MacOS	Sonoma 14.6.1
Python	3.13

Sources: github.com; timescale.com

Custom Benchmark – EXAMPLE

InfluxDB does not have IN operator, which is needed for scaling

InfluxDB Query - Glucose Statistics

```
SELECT MEAN("glucose_value") AS "mean_g",
       STDDEV("glucose_value") AS "std_g"
FROM "interstitial_glucose"
WHERE ({list_of_participants})
  AND time >= '2020-02-15T00:00:00Z'
  AND time < '2020-02-25T23:59:59Z'
GROUP BY time(1h)
```

Raw Query

```
queries_dir = config.QUERIES_PATH
list_of_participants = " OR
".join([f'"participant_id"=\'{id}\'' for id in
range(1,scale_factor+1)])
with open(os.path.join(queries_dir,f"query_2.txt"),
'r') as file:
    raw_query = file.read()
if "{list_of_participants}" in raw_query:
    final_query =
raw_query.format(list_of_participants=list_of_particip
ants)
else:
    final_query = raw_query
```

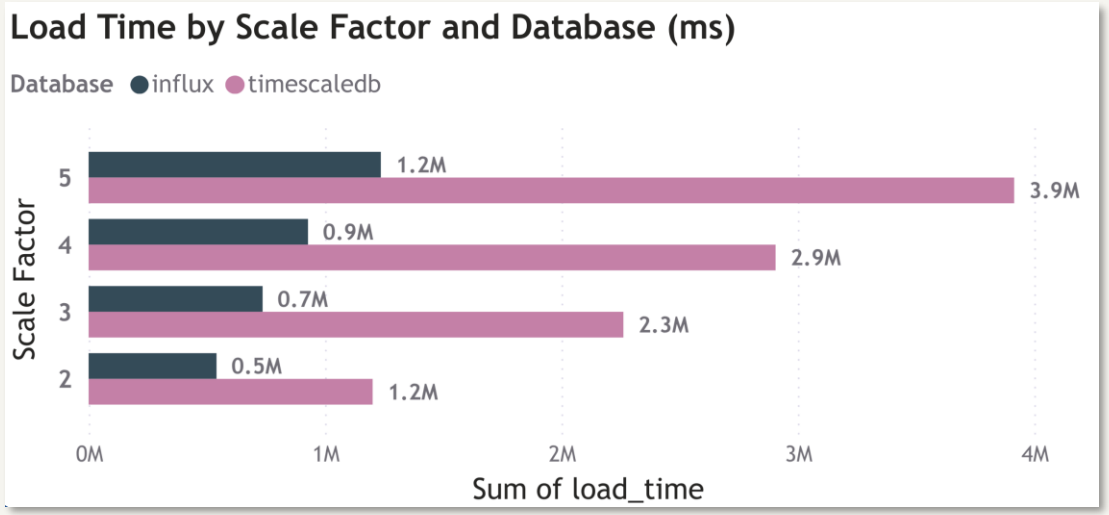
Condition Integration (Python)

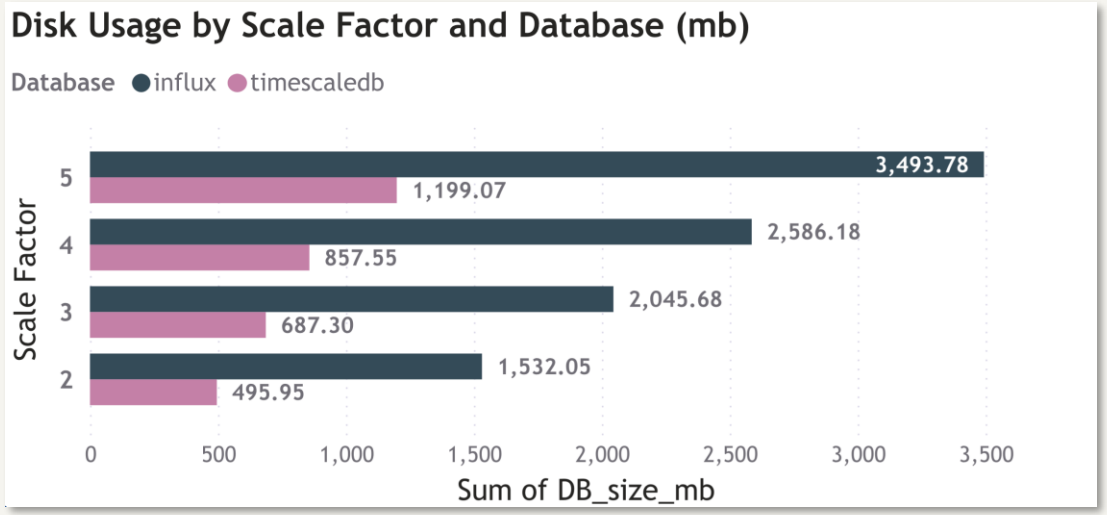
InfluxDB Query - Glucose Statistics

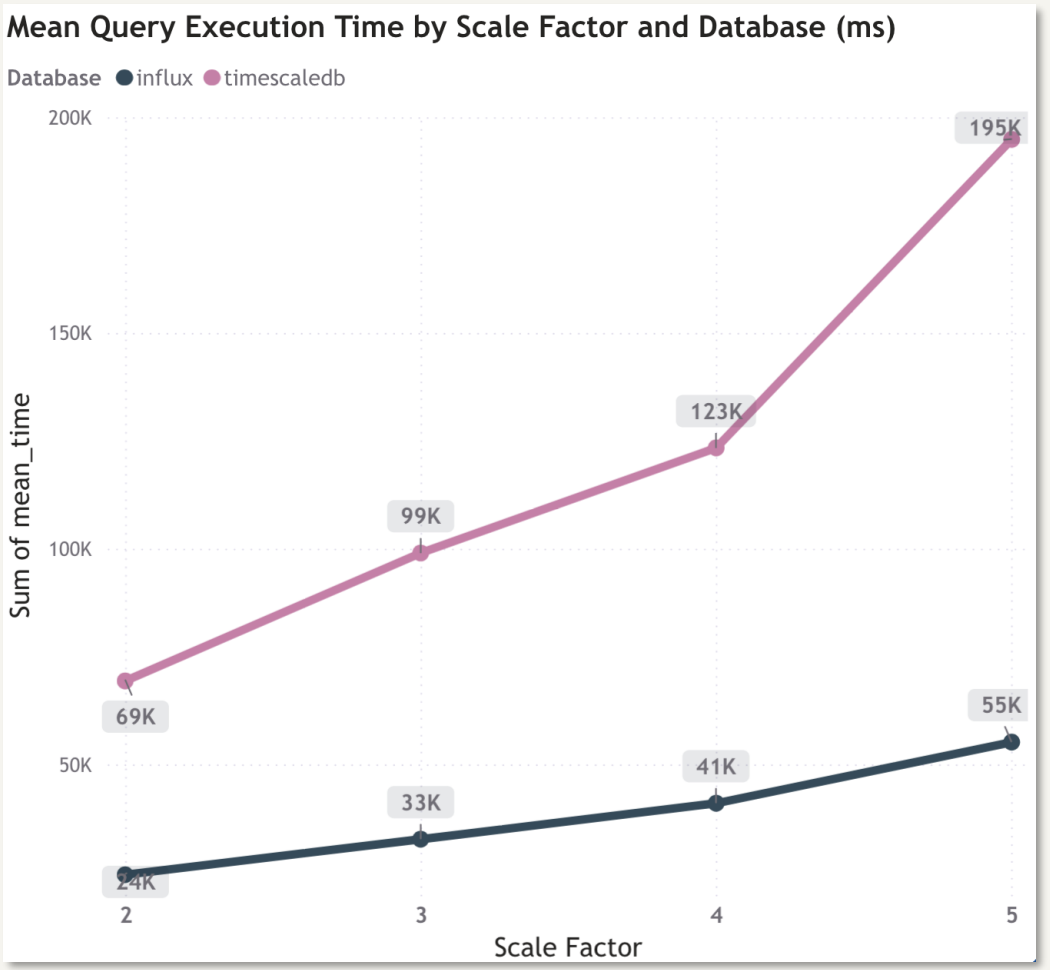
```
SELECT MEAN("glucose_value") AS "mean_g",
FROM "interstitial_glucose"
WHERE ("participant_id"='1'
       OR "participant_id"='2'
       OR "participant_id"='3')
    AND time >= '2020-02-15T00:00:00Z'
    AND time < '2020-02-25T23:59:59Z'
GROUP BY time(1h)
```

Final Query

Custom Benchmark - Results

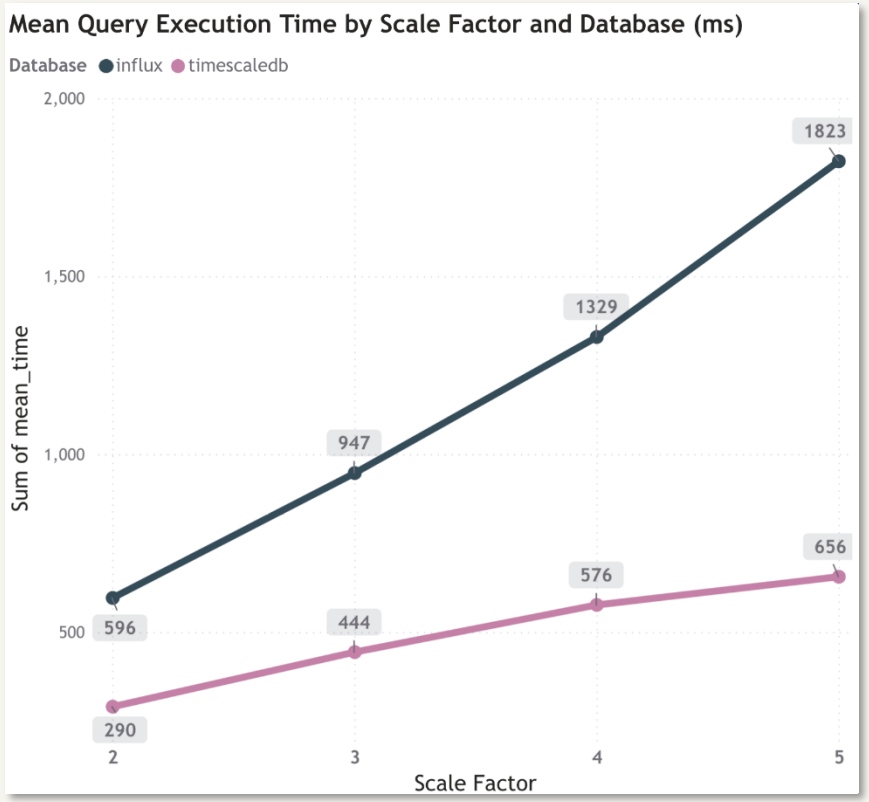






Custom Benchmark - Results

Query-5



TimescaleDB - Query-5

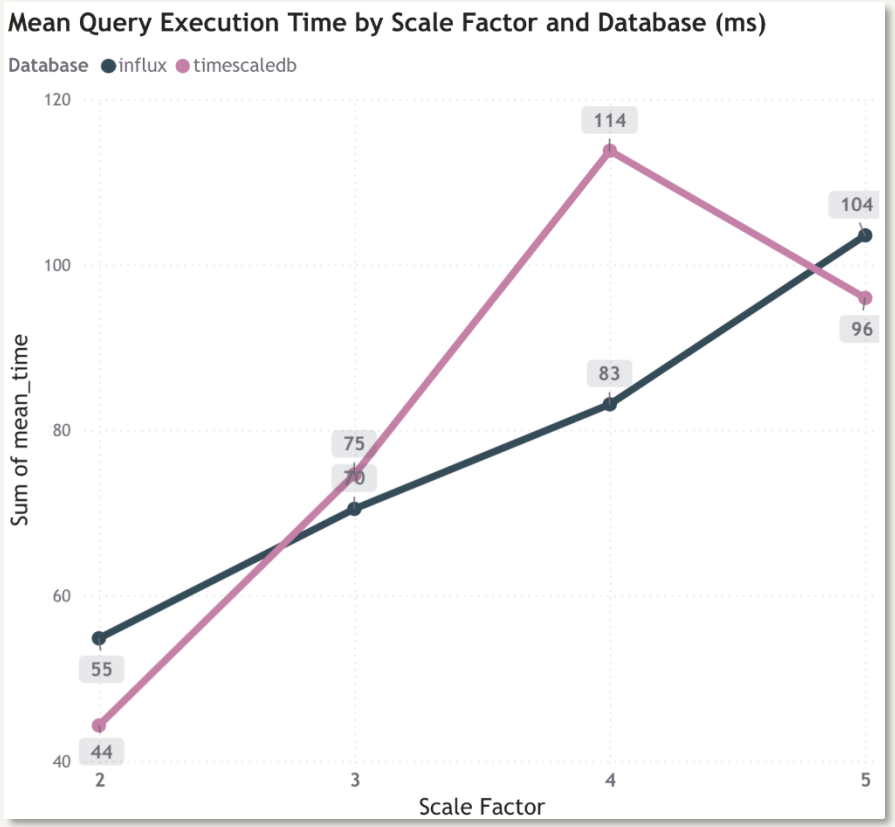
```
SELECT participant_id, MAX(temp) - MIN(temp)
AS temp_diff
FROM temperature_data
GROUP BY participant_id;
```

InfluxDB - Query-5

```
SELECT SPREAD("temp") AS temp_diff
FROM "temperature_data"
GROUP BY "participant_id"
```

Custom Benchmark - Results

Query-6



TimescaleDB - Query-6

```
SELECT participant_id, COUNT(ibi)
AS ibi_count
FROM ibi_data
WHERE ibi > 1
GROUP BY participant_id;
```

InfluxDB - Query-6

```
SELECT COUNT("ibi") AS ibi_count
FROM "ibi_data"
WHERE "ibi" > 1
GROUP BY "participant_id"
```

Custom Benchmark – Future Enhancements

- **Newer Versions of Tools**: Leverage updated versions for enhanced functionality and extended operation support.
- **Support for Complex Queries**: Explore capabilities for executing advanced analyses, such as glucose spike detection and correlation trends.

01 **Time Series Databases**

Introduction | Section2 | Section3

02 **Database Management Systems**

TimescaleDB | InfluxDB | Differences | Grafana

03 **Tool Comparison**

TSBS | Custom Benchmark

04 **Conclusion & Insights**

Tool Suitability *| Key Takeaways | Exercises*

INFLUXDB

Metric	Performance
Load time	InfluxDB outperforms across **7 out of 8** scale factors
Disk space	In TSBS benchmark, InfluxDB outperforms TimescaleDB, requiring at least **1.4** times less storage (solely time-series data)
Query execution	InfluxDB outperforms timescale requiring **one-third** of the execution time

Perfect for high-performance time-series data ingestion and storage and simple time-series querying

TIMESCALEDB

Metric	Performance
Load time	Timescale underperforms in most data loading cases
Disk space	In the custom benchmark, Timescale outperforms Influx in storing time-series relational tables.
Query execution	Timescale with PostgreSQL allow for advanced analytical queries, joins, and complex operations while underperforming in basic group by operations

Excellent for high-performance time-series data processing combined with relational database capabilities and advanced analytics

KEY TAKEAWAYS

TimescaleDB is the more suitable choice for this medical sensor application

Superior Disk Usage Efficiency: Optimized for storing both time-series and relational data at scale, ensuring cost-effectiveness

Advanced Query Capabilities: Supports sophisticated analysis to uncover critical insights (e.g., glucose spikes, stress detection, exercise patterns).

Scalability: Handles large-scale datasets efficiently, essential for growing medical applications.

Tailored for Medical Applications: Perfectly suited for sensor-driven healthcare solutions, where vast amounts of patient data (e.g., medical history, demographic, and lifestyle information) are stored in relational structures

Future-Ready: Positioned to support advanced predictive and diagnostic analytics for high-risk conditions (e.g., diabetes, cardiovascular diseases) in geriatric care and beyond

01 **Time Series Databases**

Introduction | Section2 | Section3

02 **Database Management Systems**

TimescaleDB | InfluxDB | Differences | Grafana

03 **Tool Comparison**

TSBS | Application | Benchmarking Setup | Benchmarking Results

04 **Conclusion & Insights**

*Tool Suitability | Key Takeaways | **Exercises***

EXERCISE SHEET

ETL is already done for the glycemic data.

Many query experiments for the custom benchmark

Some can be **transformed to exercises**

EXAMPLES

Find the first and last temperature measurement for each participant in the span of the 1st February 2020 and 1st March 2020.

```
SELECT participant_id,
FIRST(temp, event_time) AS
first_temperature_measurement,
LAST(temp, event_time) AS last_temperature_measurement
FROM temperature_data
WHERE event_time BETWEEN '2020-02-01 00:00:00' AND
'2020-03-01 23:59:59'
GROUP BY participant_id
ORDER BY participant_id;
```

Create a continuous aggregate on the temperature attribute that calculates the first, last, min, max and average of temperature measurement for each day and for each participant

```
CREATE MATERIALIZED VIEW temperature_stats_daily
WITH (timescaledb.continuous) AS
SELECT participant_id,
    time_bucket('1 day', event_time) AS day,
    FIRST(temp, event_time),
    LAST(temp, event_time),
    MIN(temp),
    MAX(temp),
    AVG(temp)
FROM temperature_data
GROUP BY participant_id, day;
```

Time series databases with TimeScaleDB and InfluxDB

check out our GitHub repository

TEAM MEMBERS

Kristóf Balázs: 000612294
Stefanos Kypritidis : 000606810
Nishant Sushmakar : 000606016
Olha Baliasina : 000603977