Technology
Arts Sciences
TH Köln

Semester Assignment: Modern Database Systems
Research Project

1. Student: Florian Veitz Matr.No: 11422671
2. Student: Matthias Zajcev Matr.No: 11142740

Module: **Modern Database Systems**
Course: Digital Sciences (Ma)

Lecturer: Dr. Matthäus Zloch

June 22, 2025

Contents

List of Figures

Abbreviations

Abbreviation	Meaning
IoT	Internet of Things
UI	User Interface
CSV	Comma-Separated Values
SQL	Structured Query Language
NoSQL	Not Only SQL (non-relational database model)
ms	Milliseconds
ID	Identifier

Abstract

This paper compares the performance of two database systems InfluxDB 2.7 and Oracle Database 23ai Free for storing and querying large volumes of IoT telemetry data. The same dataset, which includes temperature, humidity, gas, and motion readings from sensor devices, was imported into both systems. Several typical time-series queries were executed, such as calculating average humidity per device, detecting maximum temperature, and estimating air quality based on gas values.

We implemented quantitative methods to investigate time performance for data extraction for each database. The results show that Oracle was generally faster for most queries, while InfluxDB offered better support for native time-series operations. These findings highlight important trade-offs when selecting a database system for time-centric IoT applications. The code base of this project can be find in the following GitHub Repository.

Project Repository: `github.com/florivz/mds-influxdb`

1 Introduction

InfluxDB is a time-series database designed for high performance and scalability. It uses a schema-less model that lets users add new fields without modifying existing schemas. InfluxDB supports specialized query languages such as InfluxQL and Flux, and also standard SQL. It can ingest large volumes of metrics and IoT data at high speed and is commonly used for monitoring, alerting, and real-time analytics[1]. Therefore, we decided to take a closer look at this open source project to investigate the real-world performance compared to a state-of-the-art SQL Database from one of the leading companies in that field.

2 Data Overview

The experiments in this project are based on the Environmental Sensor Telemetry Data set by Gary A. Stafford, published on Kaggle[1]. The dataset file `iot_telemetry_data.csv` is approximately 61.9 MB in size and contains around 405,000 sensor readings. These readings were collected between **12 July 2017 and 19 July 2017** from three Raspberry Pi devices, each deployed in a different environment. The devices recorded various environmental variables and streamed the data to AWS IoT Core.

Field	Description	Unit / Type
`ts`	Timestamp	seconds
`device`	Device ID (`device_1`–`device_3`)	string
`temp`	Temperature	°F
`humidity`	Relative humidity	%
`co`	Carbon monoxide concentration	ppm
`lpg`	Liquefied petroleum gas concentration	ppm
`smoke`	Smoke concentration	ppm
`light`	Light sensor (binary)	boolean
`motion`	Motion sensor (binary)	boolean

Table 1: Fields and types of the environmental sensor dataset.

Data Preparation and Cleaning

Before loading the data into the databases, several transformations were performed in the respective initialization notebooks:

- **Missing values:** Rows with missing values were removed in both InfluxDB and Oracle workflows.

[1] `https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k`

- **Timestamp conversion:** The `ts` column was converted from String to a datetime format using `pandas.to_datetime`. In Oracle, it was further typed as `TIMESTAMP WITH TIME ZONE`.
- **Boolean values:** The fields `light` and `motion` were converted to `bool` for InfluxDB, but stored as `string` in Oracle to simplify schema compatibility.
- **Numeric values:** All gas and temperature fields were explicitly cast to `float` (InfluxDB) or `DOUBLE PRECISION` (Oracle).

Schema Implementation

The telemetry dataset was first cleaned and transformed before being loaded into both database systems.

In Oracle, the data was imported into a structured relational table with typed columns. Timestamps were stored as `TIMESTAMP`, numeric sensor values as `NUMBER`, and boolean states (`light`, `motion`) as strings due to the lack of native boolean support. Each row was assigned a unique ID. The data was inserted into the table `IOT_TELEMETRY_DATA` using a batch insert query.

In InfluxDB, the same data was stored as time-series points in the measurement `iot_telemetry`. Each point included a high-precision timestamp, a device tag, and multiple sensor fields such as humidity, temperature, and gas values. Fields were cast to floating-point or boolean types. A dedicated bucket named `iot_telemetry_data` was created to hold the measurement.

Schema Visualization

Figure 1 shows the InfluxDB structure. Figure 2 illustrates the Oracle table layout.

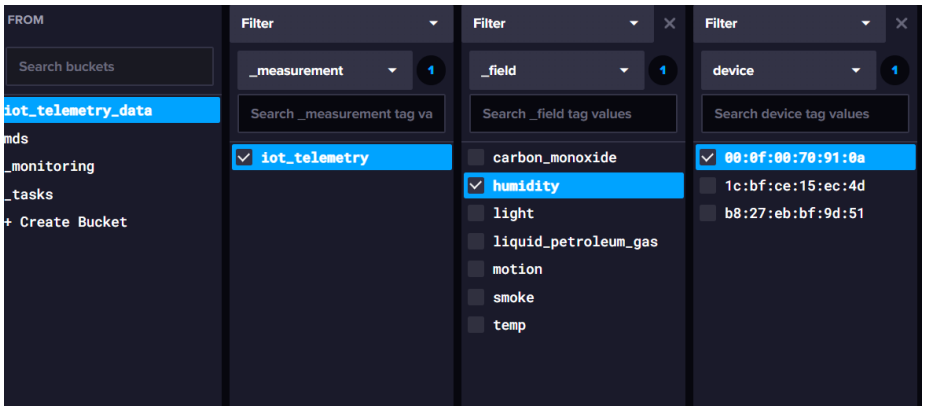
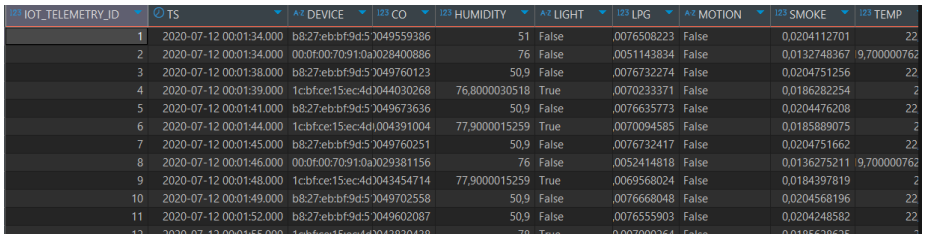


Figure 1: InfluxDB measurement schema with tags and fields.



IOT_TELEMETRY_ID	TS	DEVICE	CO	HUMIDITY	LIGHT	LPG	MOTION	SMOKE	TEMP
1	2020-07-12 00:01:34.000	b8:27:eb:bf:9d:5	0049559386	51	False	,0076508223	False	0,0204112701	22
2	2020-07-12 00:01:34.000	00:0f:00:70:91:0a	0028400886	76	False	,0051143834	False	0,0132748367	9,700000762
3	2020-07-12 00:01:38.000	b8:27:eb:bf:9d:5	0049760123	50,9	False	,0076732274	False	0,0204751256	22
4	2020-07-12 00:01:39.000	1c:bf:ce:15:ec:4d	0044030268	76,8000030518	True	,0070233371	False	0,0186282254	2
5	2020-07-12 00:01:41.000	b8:27:eb:bf:9d:5	0049673636	50,9	False	,0076635773	False	0,0204476208	22
6	2020-07-12 00:01:44.000	1c:bf:ce:15:ec:4d	004391004	77,9000015259	True	,0070094585	False	0,0185889075	2
7	2020-07-12 00:01:45.000	b8:27:eb:bf:9d:5	0049760251	50,9	False	,0076732417	False	0,0204751662	22
8	2020-07-12 00:01:46.000	00:0f:00:70:91:0a	0029381156	76	False	,0052414818	False	0,0136275211	9,700000762
9	2020-07-12 00:01:48.000	1c:bf:ce:15:ec:4d	0043454714	77,9000015259	True	,0069568024	False	0,0184397819	2
10	2020-07-12 00:01:49.000	b8:27:eb:bf:9d:5	0049702558	50,9	False	,0076668048	False	0,0204568196	22
11	2020-07-12 00:01:52.000	b8:27:eb:bf:9d:5	0049602087	50,9	False	,0076555903	False	0,0204248582	22

Figure 2: Oracle relational table for telemetry data.

Summary

The dataset combines continuous variables, binary states, and a small amount of missing values. These were cleaned and typed appropriately before loading. The resulting data structure is well-suited for evaluating different database models in time-series analytics.

3 Architecture Comparison

3.1 InfluxDB: Time-Series Architecture

InfluxDB uses column-oriented storage for high write throughput and fast time-based queries. Data is organized into measurements with indexed tags and unindexed fields, each tied to a timestamp [2]. On top Influx in the major release version 2 comes with it‘s own query language called Flux and a standard SQL Query language. Flux is a query language designed for working with time stamps and allows read operations with relative time windows [3].

- **Measurement**: A named collection of related time-series data, similar to a table in SQL. It groups points of the same type (e.g., “temperature”).
- **Tags**: Key–value pairs used as indexed metadata. Tags provide fast filtering and grouping (e.g., sensor=“UUID”).
- **Fields**: The actual data values for a point (e.g., temperature=22.5). Fields are not indexed and can vary in type.
- **Point**: A single record composed of a timestamp, one measurement name, zero or more tags, and one or more fields. It represents one data observation in time.

3.2 Oracle Database: Relational Architecture

Oracle 23ai Free Database is a row-based relational system. It supports complex SQL, transactions, and strong consistency by using the System Global Area (SGA) and background processes to manage memory and I/O [4]. The SGA is a block of shared memory where Oracle keeps data and SQL code so queries run faster and the disk is accessed less often.

3.3 Architectural Differences

Purpose and Specialization Oracle and similar SQL databases serve general applications transactions like standard SQL and strong consistency. InfluxDB is built for high volume, append-only time-series data with fast ingestion and time-based aggregations [3].

Time as Primary Key In InfluxDB, each point's timestamp is mandatory and acts as the primary key, optimizing storage and queries by time. In SQL databases, time is just another column and needs separate indexing for efficient time-based queries [3].

Schema and Flexibility SQL databases require a fixed schema before inserting data, and changes need migrations. InfluxDB infers schema on write, allowing new fields without migration; this speeds up development but can lead to inconsistent data [3].

Data Model

- **InfluxDB:** Measurements (like tables), tags (indexed), fields (values), points (timestamped records).
- **Oracle:** Tables with fixed columns, manual indexes, and defined data types.

Query Capabilities SQL supports JOINs, subqueries, window functions, and transactions. InfluxQL offers basic SELECT queries without JOINs; Flux adds joins and transformations but remains optimized for time-series analytics [3].

Data Operations SQL fully supports Create, Read, Update, Delete with constraints. InfluxDB is append-only: updates overwrite points with identical timestamps/tags, and deletes apply only to time ranges or entire series [3].

Aspect	**InfluxDB (Time-Series)**	**SQL Databases (General)**
Design Purpose	Optimized for time-series data, high write speed, time-based queries	Designed for general-purpose data, transactions, structured queries
Primary Key	Timestamp (with tags) as fixed and central key	Any column can be used as primary key; timestamp optional
Schema Flexibility	Dynamic schema, fields defined at write time	Static schema, must be predefined and consistent
Data Model	Measurements, tags, fields, points	Tables, columns, rows, primary/foreign keys
Indexing	Tags are indexed automatically; fields are not	Indexes must be defined manually on any column
Query Language	InfluxQL (limited), Flux (functional, time-focused)	Full SQL (rich query language with JOINs, subqueries, etc.)
Joins	Not supported in InfluxQL; limited in Flux	Fully supported across tables
CRUD Support	Focus on Create and Read; limited Update/Delete	Full Create, Read, Update, Delete support
Tag and Field Operations	Tags cannot be renamed/deleted; fields can be overwritten only	Columns can be changed, renamed, or dropped
Performance	High insert and query speed for time-series	Slower for time-series without optimization; better for complex queries
Best Use Cases	IoT, monitoring, metrics, logging, real-time analytics	Business systems, transactions, reporting, relational data

Table 2: Comparison between InfluxDB and SQL Databases

Summary InfluxDB is used for write-heavy, time-based workloads with flexible schema and independent scaling but lacks full relational features. SQL databases are versatile and support complex queries and transactions but are less efficient for large-scale data.

4 Classification of InfluxDB 2.7 in the NoSQL Context

4.1 CAP Theorem

With InfluxDB 2.7, Influx presents a hybrid clustering design, which is neither strictly CP (Consistency and Partition Tolerance) nor purely AP (Availability and Partition Tolerance) according to the CAP theorem. This approach com-

bines two separate systems: a CP subsystem using Raft consensus for managing cluster metadata, and an AP subsystem that handles data writes and reads with eventual consistency. This design specifically addresses the characteristics of time series data, which mostly involve immutable inserts, rarely include deletes or updates, and require high scalability and availability. The main benefits of this approach include high throughput, scalability, and resilience against node failures, making it suitable for large-scale time series applications [5].

4.2 Consistency

By default, InfluxDB operates with *eventual consistency*, meaning data changes are confirmed immediately and distributed subsequently across the system [6]. However, stricter consistency can be configured by setting quorum or complete replication, where multiple nodes must confirm before data is accepted.

4.3 Replication and Sharding

InfluxDB organizes data into *Buckets*, which are further divided into *Shard Groups*. These shard groups are time-based, enabling efficient storage and querying within defined time windows. Data replication is achieved via configurable streams, allowing data synchronization and distribution across multiple InfluxDB instances [7].

4.4 Storage and Query Language

Storage: The InfluxDB storage engine ensures that data is stored safely, remains consistent, and can be queried efficiently. It uses a combination of Write Ahead Log (WAL), in-memory cache, and Time-Structured Merge Tree (TSM) files to manage data durability and performance. When data is written, it is first added to the WAL and the in-memory cache. This ensures that data is not lost during unexpected failures. Periodically, the cache is flushed to disk as compressed TSM files, which are optimized for fast reads and storage efficiency. Additionally, the Time Series Index (TSI) helps keep query performance high, even when the number of series grows very large. TSI enables the engine to quickly identify existing measurements, tag keys, and fields, making it suitable for high volatile time series datasets [8].

Query Language: For data querying, InfluxDB 2.7 primarily utilizes the functional and declarative query language *Flux*, which is specifically optimized for analysis, transformation, and aggregation of time series data and standard SQL. Due to few usage of Flux InfluxDB has announced to discontinue maintain the query language according to the official documentation.

4.5 Conclusion

InfluxDB 2.7 stands out due to its high flexibility and efficiency, particularly suited for time-based applications. Its strengths include efficient management of large datasets, easy scalability, and a user-friendly design. Comparative analysis with relational databases such as Oracle has shown that while InfluxDB may offer additional functionalities tailored for time series data, traditional databases can sometimes provide faster query responses in specific scenarios.

5 Experiment and Observations

5.1 Experiment Setup

To ensure a fair and reproducible comparison, both database systems Oracle Database 23ai Free and InfluxDB 2.7 were installed using Docker containers. This allowed us to control the execution environment and avoid side effects from background processes or prior system states.

Each performance test was executed inside a fresh container instance. Before running each query group, the container was restarted and caches were cleared. This fresh cache scenario helps simulate first time access and reduces any advantage from internal memory optimizations. All analyses were performed using Python notebooks that were executed under identical conditions for both systems. On top the experiments were run on two different systems by the authors to ensure independence.

5.2 Experiment

For the performance study, we executed five analytical queries on both databases. The queries represent common patterns in time-series workloads: aggregation, extreme value detection, and compound metric computation. The curated queries also differ in complexity to further identify strengths and weaknesses of the databases.

Q1 – Average humidity per device *Task.* Compute the mean of `humidity` for each device (`GROUP BY device`). *Motivation.* Tests per-device roll-ups over the full time span.

Q2 – Maximum temperature (global) *Task.* Return the highest value of `temp` across all devices and timestamps. *Motivation.* Exercises a global extreme value search over time.

Q3 – Worst air-quality score per device *Task.* Derive an air-quality index AQI = `co` + `lpg` + `smoke` and report the maximum AQI per device. *Motivation.* Combines multiple columns and stresses aggregation.

Q4 – Temperature-Humidity Ratio (daily) *Task.* Calculate the average of `temp/humidity` for each day. *Motivation.* Involves arithmetic over two fields and grouping by time.

Q5 – Temperature-Humidity Ratio (per minute) *Task.* Calculate the average of `temp/humidity` per minute. *Motivation.* Evaluates fine-grained time grouping with calculation.

Each query (Q1–Q5) was executed 5 times on both systems after container startup. No other workloads were active. Query runtimes were measured with Python timing functions and averaged to reduce fluctuation.

5.3 Execution Results

Table 3 shows the average runtime for each query on both systems. Values are in seconds and reflect cold start performance under identical conditions.

Query	**Oracle DB (sec)**	**InfluxDB (sec)**
Q1 – Average humidity	0.103	0.251
Q2 – Maximum temperature	0.033	0.033
Q3 – Worst air-quality score	0.084	1.729
Q4 – Temp/Humidity Ratio (daily)	0.129	2.440
Q5 – Temp/Humidity Ratio (min)	0.443	3.038

Table 3: Average query execution time (in seconds).

5.4 Preliminary Observation

The results show that Oracle DB outperforms InfluxDB in most cases. While both systems show equal performance for the global maximum temperature query (Q2), Oracle shows faster execution in complex analytical queries (Q3–Q5). InfluxDB performs slower, especially for queries with fine time granularity and arithmetic operations.

5.5 Interpretation of Results

The experiment shows that Oracle Database was faster than InfluxDB in almost all queries. This result is surprising and goes against our initial assumption that a time-series database like InfluxDB would perform better in this context.

Unexpected outcome: Oracle is faster. We expected InfluxDB to have better performance because it is designed for time-series workloads. However, Oracle SQL was not only faster in simple queries but also in more complex ones (e.g., Q3–Q5).

5.6 Investigation of Performance Differences

At this stage, we investigated why Oracle Database outperformed InfluxDB in our experiment. Because Oracle is closed source, we cannot examine its internal implementation in detail. However, several factors may explain its superior speed:

- **Maturity and Optimization:** Oracle has been under active development for decades. Over this time, engineers have refined indexing schemes and implemented advanced caching strategies to reduce query latency.
- **High-Performance Implementation:** The Oracle backend is written in optimized languages such as C++ and assembly. These languages allow tight control over memory and processor usage, which can yield faster execution.
- **InfluxDB's Current Stack:** In contrast, InfluxDB 2.x relies heavily on Node.js, a higher-level runtime that, while flexible, generally runs slower than lower-level languages.
- **Future Directions:** InfluxData has announced plans to rewrite the entire code base in Rust. This change could improve performance in future releases and potentially narrow the gap with Oracle.

5.7 Benefits of Using InfluxDB

Despite the slower performance, InfluxDB offers several advantages for time-series use cases:

- It has a built-in web UI that makes it fast and easy to manage the database.
- Users can filter and aggregate and visualize data based on time in the UI which helps with understanding the data.
- Tasks: InfluxDB is able to create automatic tasks. They can be used for recurrent tasks like filtering out only the most important by statistical methods and more.
- Retention Time: Buckets can have fixed retention therefore automatically delete all the data after a specific time window.
- Nano seconds: Time Stamps can be saved up to Nano seconds which allow for detailed investigation especially for highly sensible IoT data.
- Flexibility and customization: InfluxDB allows for creating custom data schemas and types allowing to tailor the database to the project requirements [9].

- Open Source: Due to it´s open source nature, InfluxDB is especially interesting for smaller comapanies with less budget.

Analyzing time-series data is not always just about time performance. There are many benefits that InfluxDB comes with by default which makes working and understanding Big Data easy and fast.

6 Conclusion

This study compared InfluxDB 2.7 and Oracle Database 23ai Free for storing and querying IoT telemetry data. Both systems were evaluated under controlled conditions using real sensor data and three representative analytical queries.

The experimental results indicate that Oracle Database outperforms InfluxDB in overall query execution time. Its mature SQL engine and optimized relational model make it well suited for complex analytical tasks on large datasets. In contrast, InfluxDB offers a complete and useful ecosystem that simplifies the understanding, management, and automation of time-series data for end users. Although execution speed is important, the functionality and ecosystem support provided by a database are equally critical. In this respect, InfluxDB shows clear advantages for business-oriented applications.

The upcoming major release of InfluxDB (version 3) seems promising because it is built on a new technology stack written in Rust. In future work, it would be interesting to evaluate InfluxDB 3 experimentally in order to quantify any latency improvements resulting from this rewrite.

References

[1] Team Timescale, "The Best Time-Series Databases Compared," May 2024, accessed: 17 June 2025. [Online]. Available: https://www.tigerdata.com/learn/the-best-time-series-databases-compared

[2] Influx. (2025) Influxdb key concepts. Accessed: 2025-06-06. [Online]. Available: https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/

[3] Influx, "Compare influxdb to sql databases, year = , howpublished = https://docs.influxdata.com/influxdb/v1/concepts/crosswalk/," accessed: 2025-06-06.

[4] SOAIS, "Oracle database architecture: An in-depth overview," https://soais.com/oracle-database-architecture/, 2023, accessed: 2025-06-06.

[5] P. Dix, "Influxdb clustering design - neither strictly cp or ap," https://www.influxdata.com/blog/influxdb-clustering-design-neither-strictly-cp-or-ap/, Jun 2015, accessed: 2024-06-18.

[6] K. Farmer, "Eventual consistency: The hinted handoff queue," https://www.influxdata.com/blog/eventual-consistency-the-hinted-handoff-queue/, May 2018, accessed: 2024-06-18.

[7] InfluxData, "Influxdb shards and shard groups," https://docs.influxdata.com/influxdb/v2/reference/internals/shards/, 2024, accessed: 2024-06-18.

[8] ——, "Influxdb storage engine," https://docs.influxdata.com/influxdb/v2/reference/internals/storage-engine/, 2024, accessed: 2024-06-18.

[9] devneagu, "InfluxDB – What, When, Why," https://dev.to/devneagu/influxdb-what-when-why-4lmf, Dec. 2022, dEV Community.