Brazilian Weather Querying in SQL and NoSQL Databases

Modern Database Systems
Studiengang Digital Science Master
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Finn Heydemann
Matrikel-Nr.:	111 25 390
Adresse:	Venloer Straße. 601
	50827 Köln
	finn.heydemann@th-koeln.de
eingereicht bei:	Prof. Dr. Johann Schaible

Köln, 28.06.2023

Abstract

This paper examines the significance of weather data in comprehending global climate warming, specifically addressing the querying of surface temperatures in Brazil. The suitability of SQL and NoSQL databases, particularly InfluxDB, for managing this weather dataset is explored, considering factors such as storage, performance, and query language ease.

1 Introduction

Weather data plays an important role in assessing climate warming on Earth. These data provide insights into the extent of climate change, the rate at which different regions are warming, and the impact of countermeasures. The present study focuses on querying a database for surface temperatures in a specific region of the Earth. An existing dataset containing weather records from 2000 to 2021 in Brazil is utilized for this purpose. In order to determine which database is better suited for this task, identical queries are performed on two different databases (SQL and NoSQL).

Accurate measurement and analysis of weather data are crucial for understanding and monitoring the effects of climate change. By examining changes in surface temperatures over time, scientists can gain insights into the magnitude and patterns of global warming. These insights help in identifying regions that are experiencing rapid warming, allowing for targeted mitigation strategies and adaptation measures to be implemented.

To effectively analyze the dataset and extract meaningful information, the study explores two different types of databases: SQL and NoSQL. Both databases have their own strengths and characteristics that can influence the performance and efficiency of data retrieval. By conducting identical queries on both databases, the research aims to determine which database type is better suited for handling and querying the weather data in this specific context.

By comparing the results obtained from the SQL and NoSQL databases, the study will assess their capabilities in terms of query execution speed, flexibility, and scalability. Additionally, it will evaluate the ease of use and compatibility of each database with the specific requirements of climate data analysis. The findings of this research will provide valuable insights into the most suitable database solution for storing and retrieving weather data, facilitating future studies in climate science and aiding in the development of effective climate change mitigation strategies.

2 Data Description

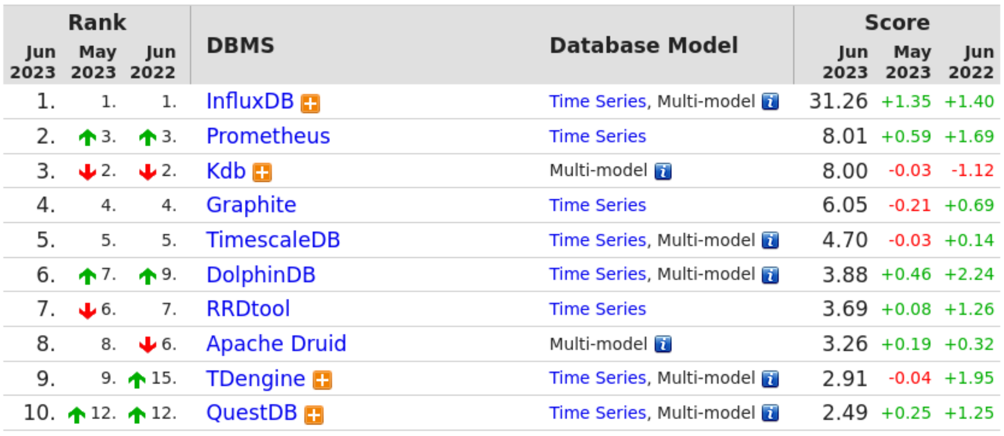
The dataset comprises a comprehensive collection of 623 geographically distributed weather stations spanning the vast geographical area of Brazil [1]. These stations collect weather data every hour. The recorded data includes vital meteorological parameters, such as atmospheric pressure, temperature, humidity, and wind direction, ensuring a comprehensive depiction of the prevailing weather conditions. Additionally, the dataset includes essential localization information, such as longitude and latitude coordinates, along with the corresponding province for each station. Moreover, naming attributes in the form of unique station codes and names are also diligently recorded, facilitating identification and reference.

It is noteworthy that the temporal dynamics of station operations influenced the data collection process. Consequently, only a limited number of stations initiated data recording as early as 2000, while the majority commenced their observations at later dates. This temporal variation underscores the importance of considering a broader time span to capture the full range of climate patterns and trends.

To enhance the analysis and facilitate an organized representation of the stations, a clustering approach was employed. Each station was assigned to one of the five cardinal directions, resulting in the formation of distinct region clusters named "Central West," "South," "North," "Northeast," and "Southeast." This clustering methodology offers a structured framework for comparing and contrasting weather patterns and characteristics across different regions of Brazil, providing valuable insights into the spatial distribution of climatic variations within the country.

3 Database Description

In order to query the weather data from Brazil, two different database types are to be compared. On the one hand the classic SQL database, on the other hand a modern (NoSQL) database will be used for the query. Since the weather data is time-related data, each of which is time-stamped, a time series database is chosen. By far the most popular database in this segment is InfluxDB (compare figure 1).



Rank			DBMS	Database Model	Score		
Jun 2023	May 2023	Jun 2022			Jun 2023	May 2023	Jun 2022
1.	1.	1.	InfluxDB	Time Series, Multi-model	31.26	+1.35	+1.40
2.	↑ 3.	↑ 3.	Prometheus	Time Series	8.01	+0.59	+1.69
3.	↓ 2.	↓ 2.	Kdb	Multi-model	8.00	-0.03	-1.12
4.	4.	4.	Graphite	Time Series	6.05	-0.21	+0.69
5.	5.	5.	TimescaleDB	Time Series, Multi-model	4.70	-0.03	+0.14
6.	↑ 7.	↑ 9.	DolphinDB	Time Series, Multi-model	3.88	+0.46	+2.24
7.	↓ 6.	7.	RRDtool	Time Series	3.69	+0.08	+1.26
8.	8.	↓ 6.	Apache Druid	Multi-model	3.26	+0.19	+0.32
9.	9.	↑ 15.	TDengine	Time Series, Multi-model	2.91	-0.04	+1.95
10.	↑ 12.	↑ 12.	QuestDB	Time Series, Multi-model	2.49	+0.25	+1.25

Figure 1 Most Popular Time Series Database in June 2023 [2]

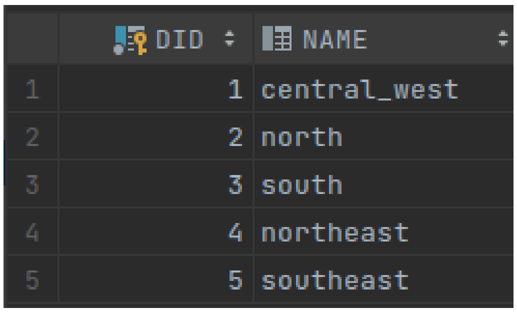
InfluxDB is a powerful and scalable time series database designed specifically for storing, querying and analyzing time-related data. It is used in a wide range of applications and scenarios where the efficient processing of large volumes of time series data is critical. InfluxDB's main use is in the collection and analysis of measurement data, sensor information, and other time-based metrics. It is widely used in areas such as Internet of Things (IoT), surveillance systems, DevOps, financial services and machine learning. With its particular focus on time series data, InfluxDB offers specific features and query capabilities that facilitate the analysis and visualization of changes over time. The database enables high write and query speeds, optimized data compression, and flexible data modeling. InfluxDB also supports a variety of integration options, including different programming languages, frameworks and tools. This allows it to be seamlessly integrated into existing data pipelines and analytics environments.

4 Data Modelling

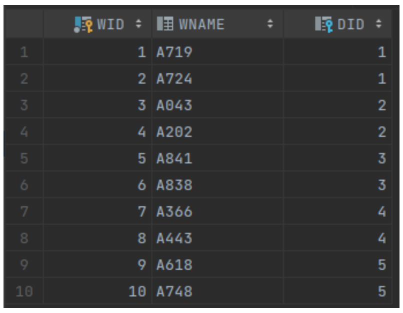
Due to the limited storage capacity of the SQL database, which allows for a maximum of 50 megabytes (MB) of storage space, it becomes necessary to significantly reduce the size of the extremely large data set, which exceeds 10 gigabytes (GB). To achieve this, a specific approach is adopted wherein two stations are randomly chosen from each of the five region clusters. From these selected stations, only the temperature column is extracted, resulting in a smaller data set that falls within the 50MB limit and can be effectively utilized for executing database queries within the SQL database.

4.1 SQL Data Modelling

To effectively query the SQL database, three tables are required. The first table encompasses the names of the five region clusters, each assigned a unique number, serving as the primary key. The second table includes the station names, with a primary key assigned to each station, along with a key linking it to the corresponding region cluster. Consequently, a one-to-many relationship is established between the first and second tables (Compare figure 2).



	DID	NAME
1	1	central_west
2	2	north
3	3	south
4	4	northeast
5	5	southeast

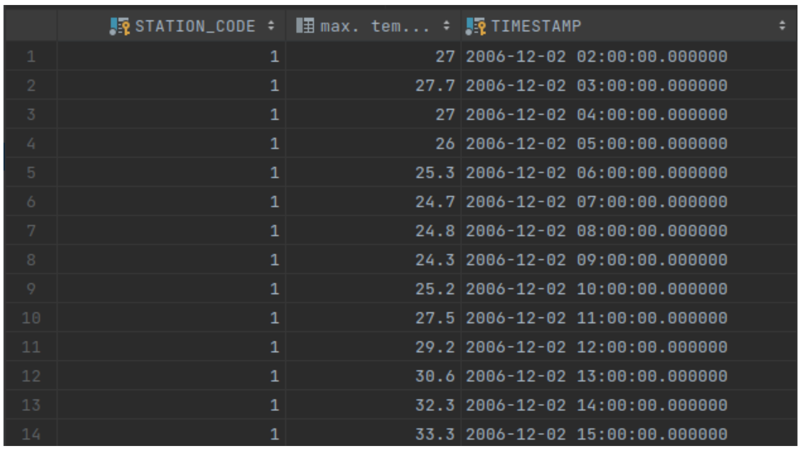


	WID	WNAME	DID
1	1	A719	1
2	2	A724	1
3	3	A043	2
4	4	A202	2
5	5	A841	3
6	6	A838	3
7	7	A366	4
8	8	A443	4
9	9	A618	5
10	10	A748	5

Figure 2 SQL-Tables: Left: Region with Direction-ID; right: Station Code with Direction-ID

The third table holds the actual data intended for querying. Its primary key is formed by combining the station number with the timestamp, while the temperature data is stored in a separate column (compare figure 3). Thus, a one-to-many relationship is formed between table two and three.

While the first two tables are comparatively small and are therefor simply programmed in SQL, a Python script (see [3]) is programmed for the generation of the last table. The Python script outputs a CSV file that contains the data in the required form (compare figure 3).



	STATION_CODE	max. tem...	TIMESTAMP
1	1	27	2006-12-02 02:00:00.000000
2	1	27.7	2006-12-02 03:00:00.000000
3	1	27	2006-12-02 04:00:00.000000
4	1	26	2006-12-02 05:00:00.000000
5	1	25.3	2006-12-02 06:00:00.000000
6	1	24.7	2006-12-02 07:00:00.000000
7	1	24.8	2006-12-02 08:00:00.000000
8	1	24.3	2006-12-02 09:00:00.000000
9	1	25.2	2006-12-02 10:00:00.000000
10	1	27.5	2006-12-02 11:00:00.000000
11	1	29.2	2006-12-02 12:00:00.000000
12	1	30.6	2006-12-02 13:00:00.000000
13	1	32.3	2006-12-02 14:00:00.000000
14	1	33.3	2006-12-02 15:00:00.000000

Figure 3 Small Section of the SQL Table holding the actual data

4.2 InfluxDB Data Modelling

Unlike the SQL database, the Influx database operates with a single table. Each row in this table must include specific columns: _measurement, _field, _value, and _time. The _measurement column denotes the name, _field represents the type of measurement value, and _value holds the actual measurement value. Additionally, a timestamp (_time) is required for each measured value, which can be specified in either Unix nanosecond format or RFC3339 format [4]. The RFC3339 format is more human-readable, consisting of a date in YYYY-MM-DD format, followed by a "T" or a space, and the time in HH:MM:SS format. The time zone is indicated in the format, where "Z" represents UTC+0, and a value like -4:00 indicates UTC-4.

In addition to the mandatory columns, it is possible to include tags in the rows. These tags can provide information about the location of the corresponding measurement.

For the specific use case addressed in this project, the table's columns are populated as illustrated in figure 4. The temperature value is written to the _value column, the timestamp is converted to RFC3339 format as required and written to the _time column, and the region and station code are included as tags in the row. In order to compare whether the SQL and Influx databases provide the same results, the timestamp is not appended with the actual correct time zone but simply with a "Z".

As the resulting CSV file is too large to be directly written into the Influx database,

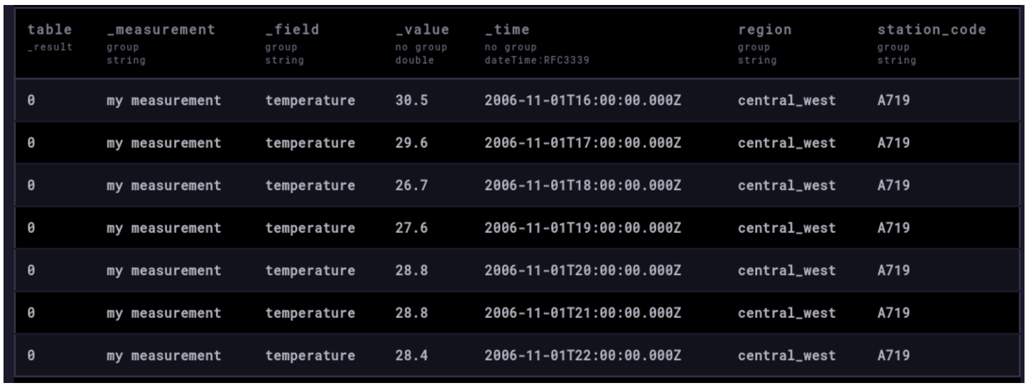


table _result	_measurement group string	_field group string	_value no group double	_time no group dateTime:RFC3339	region group string	station_code group string
0	my measurement	temperature	30.5	2006-11-01T16:00:00.000Z	central_west	A719
0	my measurement	temperature	29.6	2006-11-01T17:00:00.000Z	central_west	A719
0	my measurement	temperature	26.7	2006-11-01T18:00:00.000Z	central_west	A719
0	my measurement	temperature	27.6	2006-11-01T19:00:00.000Z	central_west	A719
0	my measurement	temperature	28.8	2006-11-01T20:00:00.000Z	central_west	A719
0	my measurement	temperature	28.8	2006-11-01T21:00:00.000Z	central_west	A719
0	my measurement	temperature	28.4	2006-11-01T22:00:00.000Z	central_west	A719

Figure 4 Small section of the Influx Database

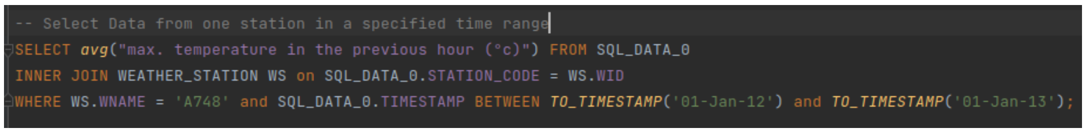
a Python script is developed to individually write each measured value into the database (see [3]).

5 Querying

In both the SQL and Influx databases, it is desired to perform the same set of queries. The objective is to retrieve the average or median temperature within a specified time range, either for a particular region or an individual station. Furthermore, there is a requirement to specify whether the average or median should be calculated across all measurements or only for each year or month separately.

5.1 SQL Queries

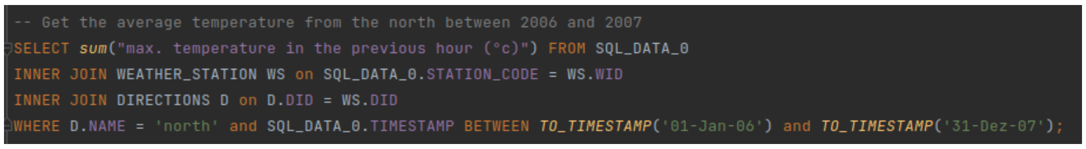
The number of "joins" to be performed varies depending on whether the query pertains to an individual weather station or a region. When querying for a single station, only one "join" is required between the data table and the table containing the stations. The "where" statement should then filter based on the corresponding station and the specified time range. For example, to determine the average temperature between January 1, 2012, and January 1, 2013, at station "A748," the query in figure 5 should be executed.



```sql
-- Select Data from one station in a specified time range
SELECT avg("max. temperature in the previous hour (°c)") FROM SQL_DATA_0
INNER JOIN WEATHER_STATION WS on SQL_DATA_0.STATION_CODE = WS.WID
WHERE WS.WNAME = 'A748' and SQL_DATA_0.TIMESTAMP BETWEEN TO_TIMESTAMP('01-Jan-12') and TO_TIMESTAMP('01-Jan-13');
```

Figure 5 SQL-Query for querying the average temperature of station "A748" between January 2012 and January 2013

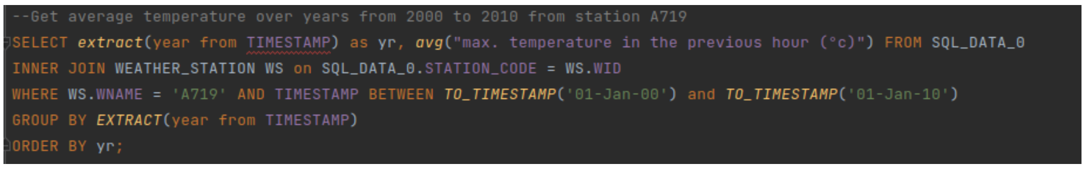
If the query pertains to a region, an additional "join" operation is needed. The region table should be joined with the station table, and the "where" statement should filter accordingly based on the region (see figure 6).



```sql
-- Get the average temperature from the north between 2006 and 2007
SELECT sum("max. temperature in the previous hour (°c)") FROM SQL_DATA_0
INNER JOIN WEATHER_STATION WS on SQL_DATA_0.STATION_CODE = WS.WID
INNER JOIN DIRECTIONS D on D.DID = WS.DID
WHERE D.NAME = 'north' and SQL_DATA_0.TIMESTAMP BETWEEN TO_TIMESTAMP('01-Jan-06') and TO_TIMESTAMP('31-Dez-07');
```

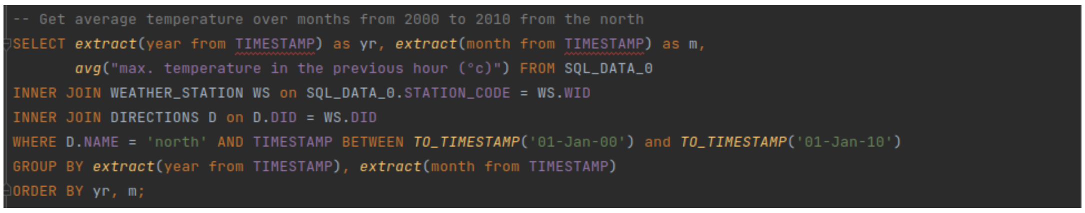
Figure 6 SQL-Query for querying the average temperature in the north between January 2006 and January 2007

Queries that involve time breakdowns are more complex. In such cases, besides calculating the average or median temperature, the year and possibly the month need to be extracted from the timestamp. Finally, grouping and sorting based on time should be performed. For instance, to retrieve the average temperature between 2000 and 2010, broken down by years, the query in figure 7 is executed. However, if the same query needs to be broken down by months and years, encompassing all stations in the northern region, the query in figure 8 is executed.



```sql
--Get average temperature over years from 2000 to 2010 from station A719
SELECT extract(year from TIMESTAMP) as yr, avg("max. temperature in the previous hour (°c)") FROM SQL_DATA_0
INNER JOIN WEATHER_STATION WS on SQL_DATA_0.STATION_CODE = WS.WID
WHERE WS.WNAME = 'A719' AND TIMESTAMP BETWEEN TO_TIMESTAMP('01-Jan-00') and TO_TIMESTAMP('01-Jan-10')
GROUP BY EXTRACT(year from TIMESTAMP)
ORDER BY yr;
```

Figure 7 SQL-Query for average temperature between 2000 and 2010 by years for station "A719"

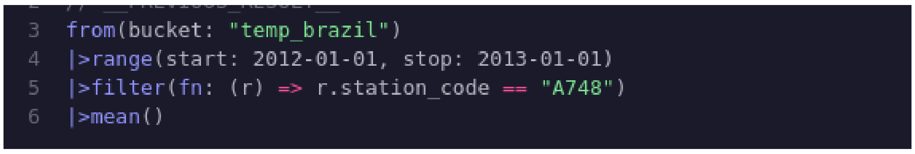


```sql
-- Get average temperature over months from 2000 to 2010 from the north
SELECT extract(year from TIMESTAMP) as yr, extract(month from TIMESTAMP) as m,
       avg("max. temperature in the previous hour (°c)") FROM SQL_DATA_0
INNER JOIN WEATHER_STATION WS on SQL_DATA_0.STATION_CODE = WS.WID
INNER JOIN DIRECTIONS D on D.DID = WS.DID
WHERE D.NAME = 'north' AND TIMESTAMP BETWEEN TO_TIMESTAMP('01-Jan-00') and TO_TIMESTAMP('01-Jan-10')
GROUP BY extract(year from TIMESTAMP), extract(month from TIMESTAMP)
ORDER BY yr, m;
```

Figure 8 SQL-Query for the average temperature between 2000 and 2010 by months and years for the region north

5.2 InfluxDB Queries

Queries in the Influx database, being a time-series database, must always include a time specification indicated by the keyword "range." Within the "range," a start time (mandatory) and an end time (optional, default=now) are defined to query the data within the specified interval. The data can be filtered using the "filter" keyword. Finally, the desired operation can be specified to be applied to the output data (e.g., "mean"). In figure 9, the Influx database is queried to retrieve the average temperature between 2012 and 2013 at weather station "A748" (similar to the SQL query in figure 5).

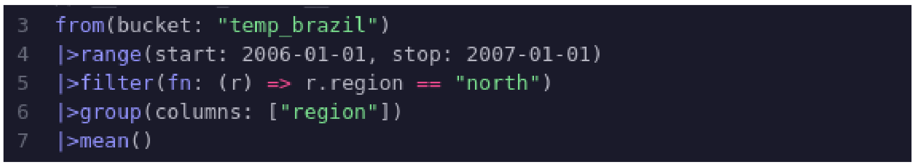


```
from(bucket: "temp_brazil")
|>range(start: 2012-01-01, stop: 2013-01-01)
|>filter(fn: (r) => r.station_code == "A748")
|>mean()
```

Figure 9 InfluxDB Query for the average temperature in 2012 from station "A748"

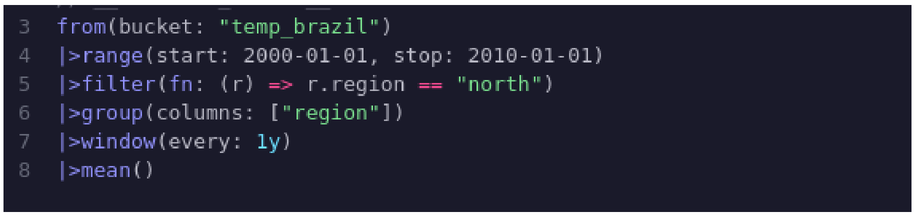
If the query is restricted to one of the five regions, an additional line with the keyword "group" is required. Here, it is specified to group by region instead of stations, enabling calculations such as the average temperature across all northern stations (see figure 10 and compare figure 6).

Time interval breakdowns in the Influx database are achieved by introducing another line. This line, starting with the keyword "window," specifies the time intervals for calculations to be performed (in figure 11, for example, calculations are done for each year). This syntax also allows for easy introduction of quarterly queries, as the line can be replaced with *|>window(every: 3mo)*.



```
from(bucket: "temp_brazil")
|>range(start: 2006-01-01, stop: 2007-01-01)
|>filter(fn: (r) => r.region == "north")
|>group(columns: ["region"])
|>mean()
```

Figure 10 InfluxDB Query for the average temperature in 2006 in the region north

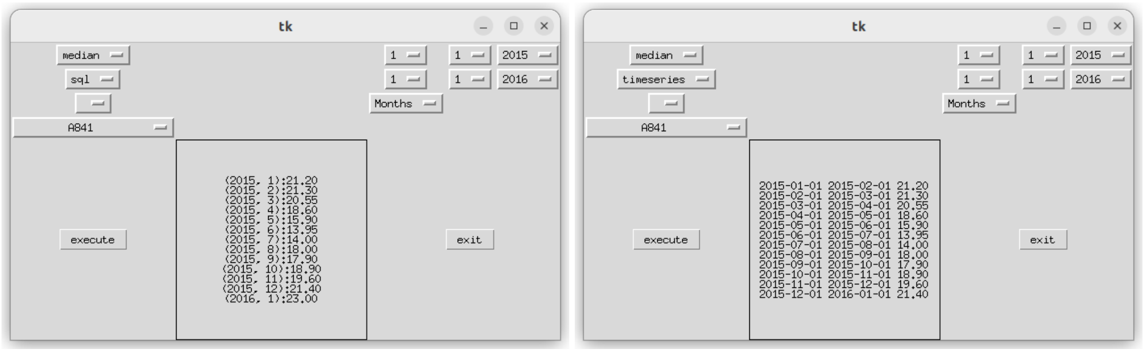


```
from(bucket: "temp_brazil")
|>range(start: 2000-01-01, stop: 2010-01-01)
|>filter(fn: (r) => r.region == "north")
|>group(columns: ["region"])
|>window(every: 1y)
|>mean()
```

Figure 11 Influx Query for the average temperature between 2000 and 2010 by years for region north

5.3 Querying Results

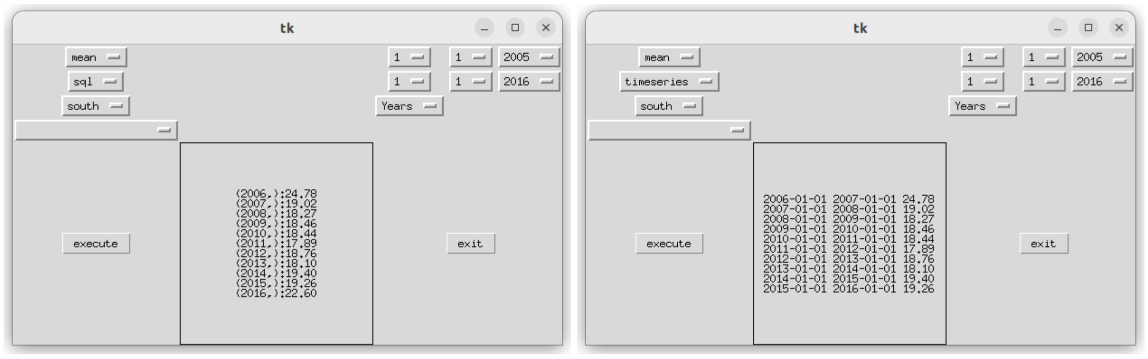
Using a rudimentary GUI, input parameters for the database query are captured [3]. Users can choose whether to apply the median or average as the function. Additionally, they can select the underlying database (SQL or time series), a region or individual station, the start and end dates of the query, and the time interval on which the function should be applied. By clicking the "execute" button, the query is created and sent to the respective database. The retrieved data is displayed in boxes with black borders. The two figures (figures 12 and 13) depict the GUI after executing an SQL and time series database query, respectively. Both databases yield the same result.



```
(2015, 1):21.20
(2015, 2):21.30
(2015, 3):20.55
(2015, 4):18.60
(2015, 5):15.90
(2015, 6):13.95
(2015, 7):14.00
(2015, 8):18.00
(2015, 9):17.90
(2015, 10):18.90
(2015, 11):19.60
(2015, 12):21.40
(2016, 1):23.00
```

```
2015-01-01 2015-02-01 21.20
2015-02-01 2015-03-01 21.30
2015-03-01 2015-04-01 20.55
2015-04-01 2015-05-01 18.60
2015-05-01 2015-06-01 15.90
2015-06-01 2015-07-01 13.95
2015-07-01 2015-08-01 14.00
2015-08-01 2015-09-01 18.00
2015-09-01 2015-10-01 17.90
2015-10-01 2015-11-01 18.90
2015-11-01 2015-12-01 19.60
2015-12-01 2016-01-01 21.40
```

Figure 12 Median monthly temperature in 2015 for station "A841", left: SQL-query, right: influxDB-query



```
(2006,):24.78
(2007,):19.02
(2008,):18.27
(2009,):18.46
(2010,):18.44
(2011,):17.89
(2012,):18.76
(2013,):18.10
(2014,):19.40
(2015,):19.26
(2016,):22.60
```

```
2006-01-01 2007-01-01 24.78
2007-01-01 2008-01-01 19.02
2008-01-01 2009-01-01 18.27
2009-01-01 2010-01-01 18.46
2010-01-01 2011-01-01 18.44
2011-01-01 2012-01-01 17.89
2012-01-01 2013-01-01 18.76
2013-01-01 2014-01-01 18.10
2014-01-01 2015-01-01 19.40
2015-01-01 2016-01-01 19.26
```

Figure 13 Average yearly temperature between 2005 and 2016 for region south, left: SQL-query, right: influxDB-query

6 Performance Results

To compare both databases, certain benchmark values need to be established. Since InfluxDB, unlike the SQL database programmed in Jetbrains Datagrip, does not have the capability to calculate total costs, the query execution times need to be compared instead. However, comparing these times poses a challenge as the databases become faster with repeated identical queries. Therefore, both databases are restarted, and the first query execution times for the four queries described earlier (figures 5 to 8 or figures 9 to 11 respectively) are recorded. Table 1 presents the result of this test. It becomes evident that the time series database outperforms the SQL database in terms of query speed. Particularly, for simple queries involving only a time range and a filter, the time series database demonstrates significant advantages.

Table 1 Comparison of SQL and InfluxDB query times

Query No. / DB Type	1	2	3	4
SQL	264ms	228ms	365ms	314ms
Influx	7ms	8ms	31ms	234ms

7 Implementation of NoSQL concepts

The Influx database focuses on availability and partition tolerance, prioritizing them over strict consistency. However, the lack of immediate consistency becomes apparent, especially with high-frequency measurement data (ms intervals), which may not be immediately accessible to all readers. Sharding across different partitions should be done based on the timestamp using a range-based approach. This ensures efficient data retrieval through global secondary indexing.

Data replication occurs asynchronously, and full consistency cannot be guaranteed. In the worst-case scenario, data may be lost. The type of replication can be configured in the Influx database. There are different options to choose from, including "any," where data is replicated to another partition or written to a queue for later replication (Hinted Handoff), "One," where successful replication to another server is required, "Quorum," which replicates to $n/2+1$ partitions, and "all," where replication occurs to all partitions.

When deleting data, the "all" option must be chosen to prevent unintended replication. However, since the data is recorded and written at a very low frequency (1 per hour) and never needs to be deleted, the specific type of replication becomes less significant. Therefore, the decision is made in favor of the "any" replication method to achieve a higher query speed (lower latency).

As time series data does not undergo modification or updates, issues such as conflict resolution, which are typically resolved using Vector Clocks, do not arise. [5]

8 Conclusion

In this paper, the importance of weather data in understanding global climate warming, with a specific focus on querying surface temperatures in Brazil, is explored. We compare the suitability of SQL and NoSQL databases, particularly InfluxDB, for handling this weather dataset.

Furthermore, it is highlighted that InfluxDB excels in terms of query performance, with significantly faster execution times compared to SQL. Its specialized nature as a time series database contributes to this advantage, allowing for quicker and more efficient querying of time-based data. Additionally, InfluxDB offers an intuitive querying language, further enhancing its ease of use for extracting and manipulating relevant weather information.

InfluxDB's ability to deliver superior query speed and its user-friendly querying language make it a preferred choice in various industries where time series data plays a pivotal role. Whether it is for scientific research, environmental monitoring, or business analytics, InfluxDB proves to be an invaluable tool for storing, analyzing, and visualizing time-based data effectively.

References

[1] Kaggle: Climate Weather Surface of Brazil - Hourly `https://www.kaggle.com/datasets/PROPPG-PPG/hourly-weather-surface-brazil-southeast-region`

[2] DB-Engines: DB-Engines Ranking of Time Series DBMS `https://db-engines.com/en/ranking/time+series+dbms`

[3] Finn Heydemann @ github: Modern-Database-Systems `https://github.com/gjmm07/Modern-Database-Systems`

[4] InfluxDB: Operate on Timestamp with FLUX `https://docs.influxdata.com/influxdb/cloud/query-data/flux/operate-on-timestamps/`

[5] influxdata: InfluxDB Clustering Design - neither strictly CP or AP `https://www.influxdata.com/blog/influxdb-clustering-design-neither-strictly-cp-or-ap/`