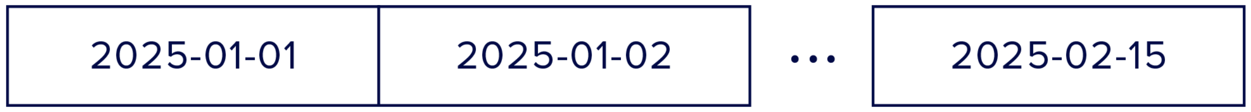
Partitioning data for Query Performance in InfluxDB 3.0

Query performance is critical in any database. Data partitioning is a mechanism that helps prune unnecessary data, allowing queries to run faster. However, there are always trade-offs between large and small numbers of partitions. For instance, fine-grained partitioning on high cardinality columns can reduce performance. This post describes different partitioning schemes supported by InfluxDB 3.0 and explains their trade-offs.

Only InfluxData's Cloud Dedicated and Clustered products support user-defined partitioning. Note that InfluxDB Cloud Serverless always partitions data by day and there is no option to modify that.

Default partitioning

Because InfluxDB is a time series database, it filters most queries by a time range. If you load data without specifying how your table is partitioned, by default, InfluxDB partitions your data by day, as shown in Figure 1. It stores all data with `` `time` `` on the same day in the same partition. In practice, this is a good partitioning scheme for most **moderate-volume** use cases and helps balance ingest efficiency and query performance.



```
<br>
```

Figure 1: data of my_table partitioned by day

Queries that select a specific time range only need to read data from the relevant partitions. For example, the following SQL query only needs to read data from two partitions, *2025-01-01* and *2025-01-02*. InfluxDB 3.0 uses information from its catalog to avoid reading any other partitions.

```
<pre><code class="language-sql">SELECT ...
FROM   my_table
WHERE  time >= '2025-01-01 18:00:00' AND time <= '2025-01-02 03:00:00';
```

</code></pre>

Query 1: filtering data on a time range

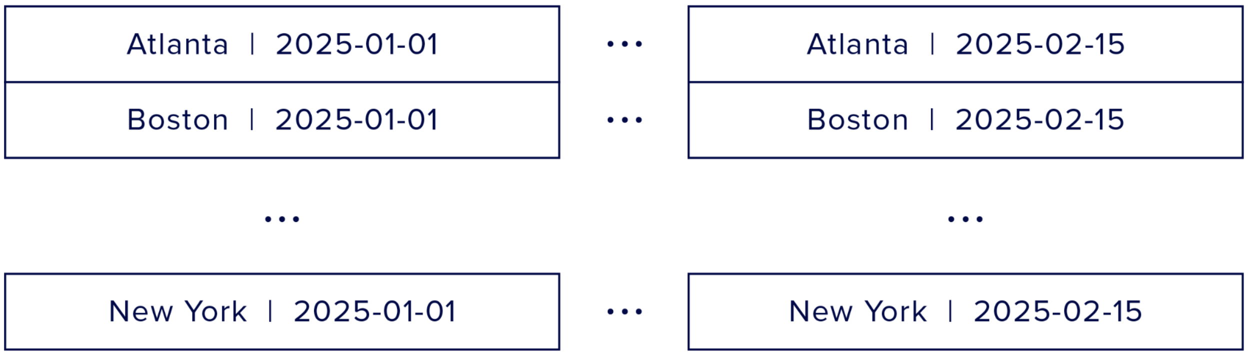
User-defined / custom partitioning

For cases when a single day contains too much data (e.g., GBs of Parquet files), and the query includes filters on additional tag columns, InfluxDB allows you to partition your data on your tag(s) and time. For instance, if your common queries request data for a specific city in a specific time range, as in the following query, you can use custom partitioning to partition the data on `` `city_name` `` and `` `time.` ``

```
<pre><code class="language-sql">SELECT ...
FROM   my_table
WHERE  time >= '2025-01-01 18:00:00' AND time <= '2025-01-02 03:00:00'
   AND city_name = `Boston`;
</code></pre>
```

Query 2: filtering data for a city and a time range

If you partition data on `` `city_name` `` and `` `day,` `` you will have **more partitions** and they will look like this:



<br>

Figure 2: data of my_table partitioned by city_name and day

Query 2 only needs to read two partitions, `Boston | 2025-01-01` and `Boston | 2025-01-02.` If your data contains many cities, each partition will be smaller than the day-default partition, and you will need less data for your query.

Note that your custom partitioning tag columns must always have values for InfluxDB to store them in the correct partitions. Without a value, InfluxDB won't have enough information to apply filters, and your query will end up reading all partitions.

The cost of having too many partitions

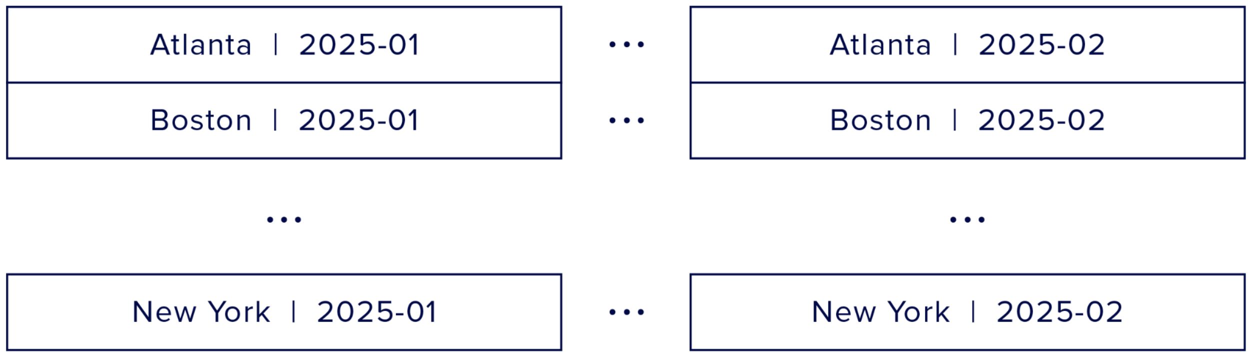
Partition designs always come with trade-offs. While smaller partitions help reduce the amount of data your query reads, it does not always mean your query will run faster. There are also side effects on ingester and compactor workloads. Having smaller partitions usually means having more, smaller Parquet files and will lead to:

1. Less storage efficiency—more files require more space to store the same data.
2. A higher ingester workload is required to group data into smaller partitions and files.
3. A higher compactor workload is required to compact more partitions and smaller files.
4. A higher metadata catalog volume—more partitions and a higher number of files require more pruning processing at query time.
5. Queries that do not have predicates and cover your entire partition design may end up reading many partitions and smaller files, degrading performance.

Below are the schemes for you to control the number of your partitions.

User-controlled number of partitions

If you want to control the number of partitions for a data set containing many cities but there is not much data for each city, you may consider partitioning your data over a greater time range. Figure 3 illustrates data partitioned by `city_name` and `month.` In this case, Query 2 will read data from one partition, `Boston | 2025-01,` which covers the entire month of data for Boston.



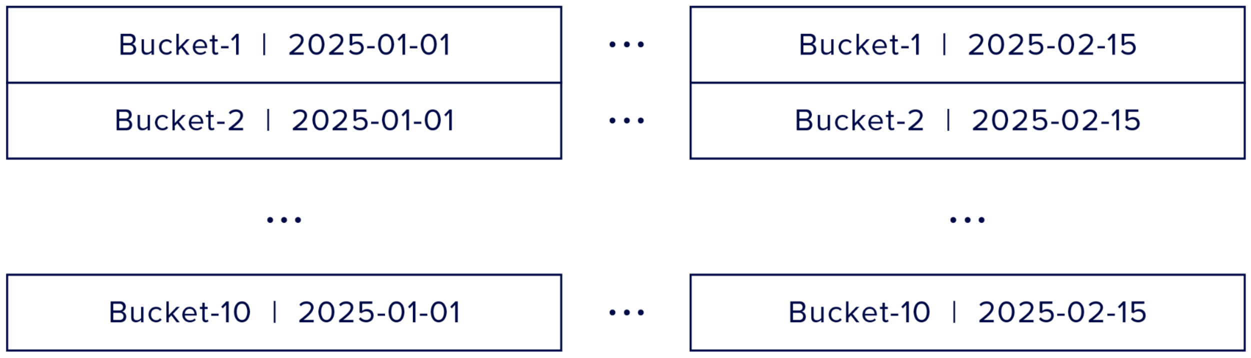
<br>

`Figure 3:  data of my_table partitioned by city_name and month`

Note: Defining a retention policy for a specific time allows you to control the number of partitions in the database.

User-defined number of partitions

At this time (Jan 2024), InfluxData is working on a feature named ***Server-Side Bucketing***, which will provide users with a simpler method for setting their desired number of partitions. For example, if you don't know how many cities will be in your data set but you know there will be many, you can cap the number of partitions by hashing many of them into the same partition. Figure 4 shows data partitioned by `*hash(city_name) % 10*` and `*day.*` In this case, there will be a max of 10 * 365 = 3,650 partitions for a year of data.



<br>

`Figure 4:  data of my_table partitioned by hash(city_name) % 10 and day.`

Final thoughts

Finding the right partitioning plan for your data helps increase your query performance, especially for high-volume ingest cases. However, you need to understand the nature and scale of your data to implement the right design and to ensure you don't have too many partitions in your system. Remember to avoid partitioning your data on **optional** tag values. If your data lacks a value, InfluxDB won't have enough information to apply filters, and your query will end up reading all partitions.

Refer to Partitioning for Performance in a Sharding Database System for other roles of data partitioning.