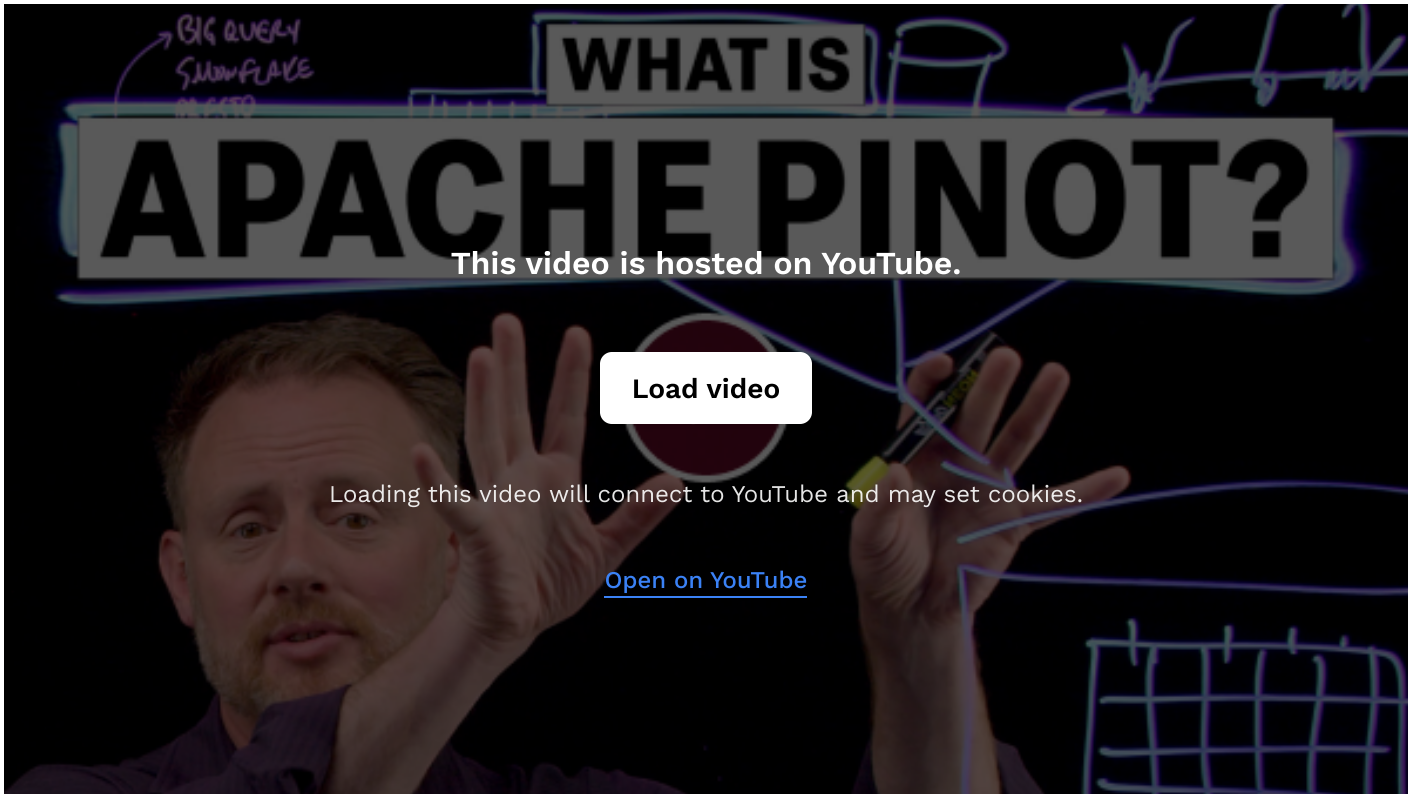
Apache Pinot™ 0.11 - Timestamp Indexes

By: Mark Needham

November 22nd, 2022 • 8 min read

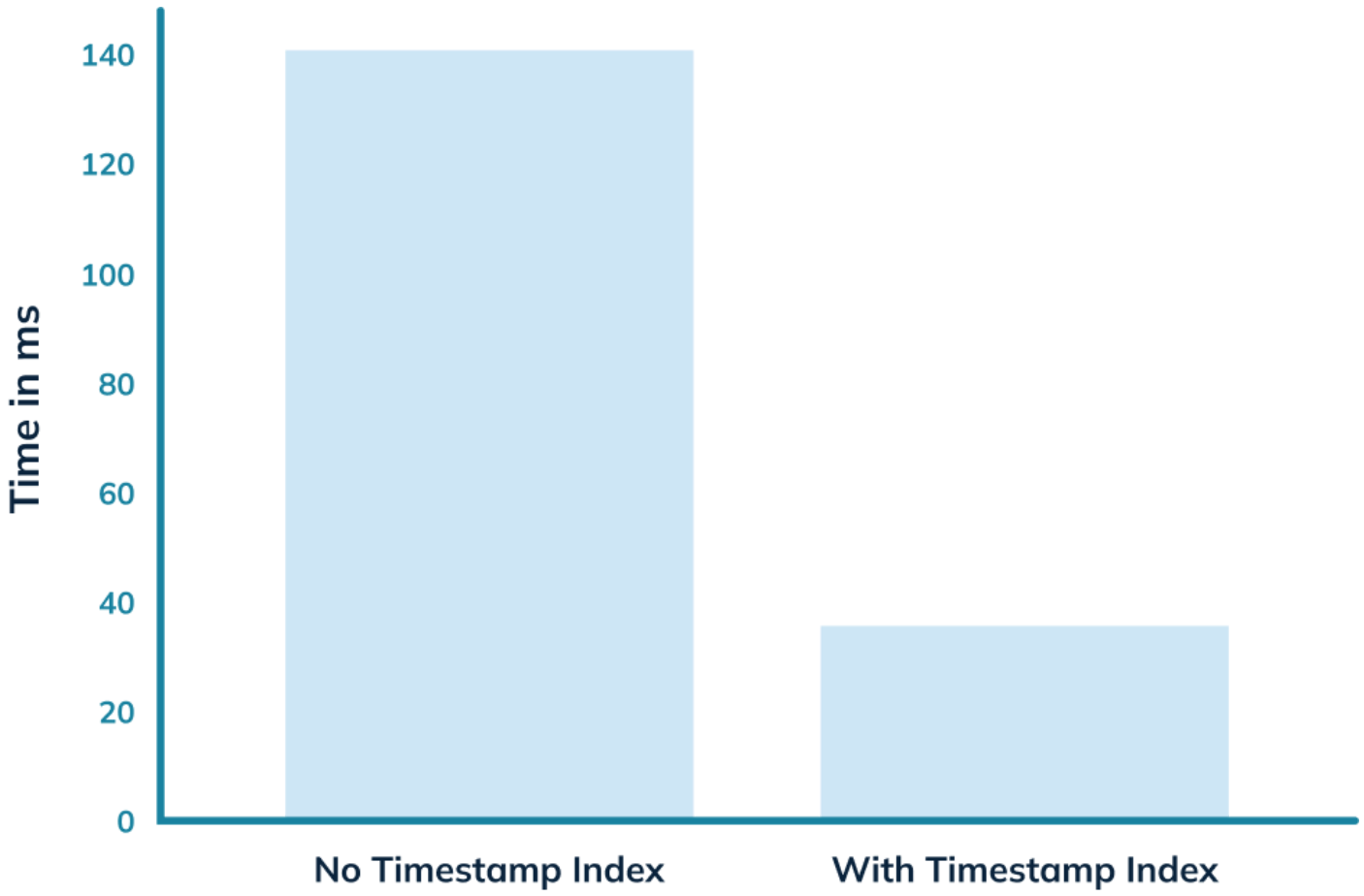


The recent Apache Pinot™ 0.11.0 release has lots of goodies for you to play with. This is the third in a series of blog posts showing off some of the new features in this release.

Pinot introduced the TIMESTAMP data type in the 0.8 release, which stores the time in millisecond epoch long format internally. The community feedback has been that the queries they're running against timestamp columns don't need this low-level granularity.

Instead, users write queries that use the datetrunc function to filter at a coarser grain of functionality. Unfortunately, this approach results in scanning data and time value conversion work that takes a long time at large data volumes.

The timestamp index solves that problem! In this blog post, we'll use it to get an almost 5x query speed improvement on a relatively small dataset of only 7m rows.



Spinning up Pinot

We're going to be using the Pinot Docker container, but first, we're going to create a network, as we'll need that later on:

docker network create timestamp_blog

We're going to spin up the empty QuickStart in a container named pinot-timestamp-blog:

```
docker run \
  -p 8000:8000 \
  -p 9000:9000 \
 --name pinot-timestamp-blog \
  --network timestamp_blog \
  apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0 \
  QuickStart -type EMPTY
```

Or if you're on a Mac M1, change the name of the image to have the arm-64 suffix, like this:

```
docker run \
  -p 8000:8000 \
  -p 9000:9000 \
    --network timestamp_blog \
 --name pinot-timestamp-blog \
  apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0-arm64 \
  QuickStart -type EMPTY
```

Once that's up and running, we'll be able to access the Pinot Data Explorer at http://localhost:9000, but at the moment, we don't have any data to play with.

Importing Chicago Crime Dataset

The Chicago Crime dataset is a small to medium-sized dataset with 7 million records representing reported crimes in the City of Chicago from 2001 until today.

It contains details of the type of crime, where it was committed, whether an arrest was recorded, which beat it occurred on, and more.

Each of the crimes has an associated timestamp, which makes it a good dataset to demonstrate timestamp indexes.

You can find the code used in this blog post in the Analyzing Chicago Crimes recipe section of Pinot Recipes GitHub repository. From here on, I'm assuming that you've downloaded this repository and are in the recipes/analyzing-chicago-crimes directory.

We're going to create a schema and table named crimes by running the following command:

```
docker run \
    --network timestamp_blog \
    -v $PWD/config:/config \
    apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0-arm64 AddTable \
      -schemaFile /config/schema.json \
      -tableConfigFile /config/table.json \
      -controllerHost pinot-timestamp-blog \
     -exec
```

We should see the following output:

```
   2022/11/03 13:07:57.169 INFO \[AddTableCommand\] \[main\] {"unrecognizedPropert
```

A screenshot of the schema is shown below:

TABLE SCHEMA JSON Format

Search...

Column	Type	Field Type
ID	INT	Dimension
CaseNumber	STRING	Dimension
Block	STRING	Dimension
IUCR	STRING	Dimension
PrimaryType	STRING	Dimension
Arrest	BOOLEAN	Dimension
Domestic	BOOLEAN	Dimension
Beat	STRING	Dimension
District	STRING	Dimension
Ward	STRING	Dimension
CommunityArea	STRING	Dimension
FBICode	STRING	Dimension
Latitude	DOUBLE	Dimension
Longitude	DOUBLE	Dimension
DateEpoch	TIMESTAMP	Date-Time

We won't go through the table config and schema files in this blog post because we did that in the last post, but you can find them in the config directory on GitHub.

Now, let's import the dataset.

```
docker run \
    --network timestamp_blog \
    -v $PWD/config:/config \
    -v $PWD/data:/data \
    apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0-arm64 LaunchDataIngestion
```

```
    -jobSpecFile /config/job-spec.yml \
     -values controllerHost=pinot-timestamp-blog
```

It will take a few minutes to load, but once that command has finished, we're ready to query the crimes table.

Querying crimes by date

The following query finds the number of crimes that happened after 16th January 2017, grouped by week of the year, with the most crime-filled weeks shown first:

```
select datetrunc('WEEK', DateEpoch) as tsWeek, count(*)
from crimes
WHERE tsWeek > fromDateTime('2017-01-16', 'yyyy-MM-dd')
group by tsWeek
order by count(*) DESC
limit 10
```

If we run that query, we'll see the following results:

QUERY RESULT

Search...

tsWeek	count(*)
1527465600000	6128
1501459200000	6095
1532908800000	6068
1529884800000	5933
1530489600000	5904
1535328000000	5848
1564358400000	5824
1534118400000	5804

And, if we look above the query result, there's metadata about the query, including the time that it took to run.

timeUsedMs	numDocsScanned	totalDocs
141	1394147	7663021

The query took 141 ms to execute, so that's our baseline.

Adding the timestamp index

We could add a timestamp index directly to this table and then compare query performance, but to make it easier to do comparisons, we're going to create an identical table with the timestamp index applied.

The full table config is available in the config/table-index.json file, and the main change is that we've added the following section to add a timestamp index on the DateEpoch column:

```
"fieldConfigList": [
  {
    "name": "DateEpoch",
    "encodingType": "DICTIONARY",
    "indexTypes": ["TIMESTAMP"],
    "timestampConfig": {
      "granularities": [
        "DAY",
        "WEEK",
        "MONTH"
      ]
    }
  }
],
```

encodingType will always be 'DICTIONARY' and *indexTypes* must contain 'TIMESTAMP'. We should specify granularities based on our query patterns.

As a rule of thumb, work out which values you most commonly pass as the first argument to the datetrunc function in your queries and include those values.

The full list of valid granularities is: *millisecond*, *second*, *minute*, *hour*, *day*, *week*, *month*, *quarter*, and *year*.

Our new table is called crimes_indexed, and we're also going to create a new schema with all the same columns called crimes_indexed, as Pinot requires the table and schema names to match.

We can create the schema and table by running the following command:

```
docker run \
   --network timestamp_blog \
   -v $PWD/config:/config \
```

```
    apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0-arm64 AddTable \
      -schemaFile /config/schema-index.json \
      -tableConfigFile /config/table-index.json \
      -controllerHost pinot-timestamp-blog \
     -exec
```

We'll populate that table by copying the segment that we created earlier for the crimes table.

```
docker run \
    --network timestamp_blog \
    -v $PWD/config:/config \
    -v $PWD/data:/data \
    apachepinot.docker.scarf.sh/apachepinot/pinot:0.11.0-arm64 LaunchDataIngestion
     -jobSpecFile /config/job-spec-download.yml \
      -values controllerHost=pinot-timestamp-blog
```

If you're curious how that job spec works, I wrote a blog post explaining it in a bit more detail.

Once the Pinot Server has downloaded this segment, it will apply the timestamp index to the DateEpoch column.

For the curious, we can see this happening in the log files by connecting to the Pinot container and running the following grep command:

```
docker exec -iti pinot-timestamp-blog \
  grep -rni -A10 "Successfully downloaded segment:.*crimes_indexed_OFFLINE.*" \
  logs/pinot-all.log
```

We'll see something like the following (tidied up for brevity):

```
[BaseTableDataManager]  Successfully downloaded segment: crimes_OFFLINE_0 of tabl
[V3DefaultColumnHandler]  Starting default column action: ADD_DATE_TIME on column
[SegmentDictionaryCreator]  Created dictionary for LONG column: $DateEpoch$DAY wi
[V3DefaultColumnHandler]  Starting default column action: ADD_DATE_TIME on column
```

```
[SegmentDictionaryCreator]  Created dictionary for LONG column: $DateEpoch$WEEK w
[V3DefaultColumnHandler]  Starting default column action: ADD_DATE_TIME on column
[SegmentDictionaryCreator]  Created dictionary for LONG column: $DateEpoch$MONTH
[RangeIndexHandler]  Creating new range index for segment: crimes_OFFLINE_0, colu
[RangeIndexHandler]  Created range index for segment: crimes_OFFLINE_0, column: $
[RangeIndexHandler]  Creating new range index for segment: crimes_OFFLINE_0, colu
[RangeIndexHandler]  Created range index for segment: crimes_OFFLINE_0, column: $
```

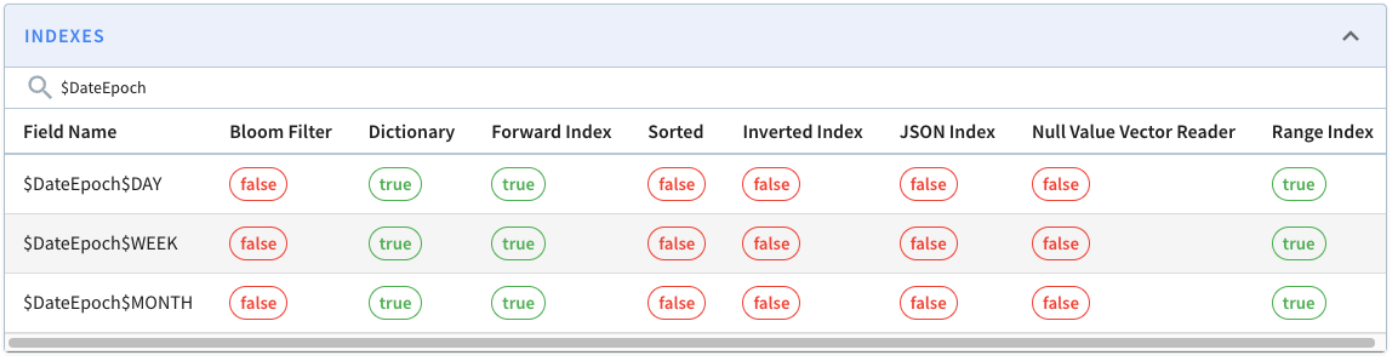
What does a timestamp index do?

So, the timestamp index has now been created, but what does it actually do?

When we add a timestamp index on a column, Pinot creates a derived column for each granularity and adds a range index for each new column.

In our case, that means we'll have these extra columns: $DateEpoch$DAY, $DateEpoch$WEEK, and $DateEpoch$MONTH.

We can check if the extra columns and indexes have been added by navigating to the segment_page and typing $Date$Epoch in the search box. You should see the following:



Field Name	Bloom Filter	Dictionary	Forward Index	Sorted	Inverted Index	JSON Index	Null Value Vector Reader	Range Index
$DateEpoch$DAY	false	true	true	false	false	false	false	true
$DateEpoch$WEEK	false	true	true	false	false	false	false	true
$DateEpoch$MONTH	false	true	true	false	false	false	false	true

These columns will be assigned the following values:

- $DateEpoch$DAY = dateTrunc('DAY', DateEpoch)
- $DateEpoch$WEEK = dateTrunc('WEEK', DateEpoch)
- $DateEpoch$MONTH = dateTrunc('MONTH', DateEpoch)

Pinot will also rewrite any queries that use the dateTrunc function with DAY, WEEK, or MONTH and the DateEpoch field to use those new columns.

This means that this query:

```
select datetrunc('WEEK', DateEpoch) as tsWeek, count(*)
from crimes_indexed
GROUP BY tsWeek
limit 10
```

Would be rewritten as:

```
select  $DateEpoch$WEEK as tsWeek, count(*)
from crimes_indexed
GROUP BY tsWeek
limit 10
```

And our query:

```
select datetrunc('WEEK', DateEpoch) as tsWeek, count(*)
from crimes
WHERE tsWeek > fromDateTime('2017-01-16', 'yyyy-MM-dd')
group by tsWeek
order by count(*) DESC
limit 10
```

Would be rewritten as:

```
select $DateEpoch$WEEK as tsWeek, count(*)
from crimes
WHERE tsWeek > fromDateTime('2017-01-16', 'yyyy-MM-dd')
group by tsWeek
order by count(*) DESC
limit 10
```

Re-running the query

Let's now run our initial query against the *crimes_indexed* table. We'll get exactly the same results as before, but let's take a look at the query stats:

timeUsedMs	numDocsScanned	totalDocs
36	1394147	7663021

This time the query takes 36 milliseconds rather than 140 milliseconds. That's an almost 5x improvement, thanks to the timestamp index.

Summary

Hopefully, you'll agree that timestamp indexes are pretty cool, and achieving a 5x query improvement without much work is always welcome!

If you're using timestamps in your Pinot tables, be sure to try out this index and let us know how it goes on the StarTree Community Slack . We're always happy to help out with any questions or problems you encounter.

Join our newsletter

Your email address