Getting Started with Time Series Data Modeling

White Paper

Cassandra is awesome at time series

Cassandra's data model works well with data in a sequence. That data can be variable in size, and Cassandra handles large amounts of data excellently. When writing data to Cassandra, data is sorted and written sequentially to disk. When retrieving data by row key and then by range, you get a fast and efficient access pattern, due to minimal disk seeks. Time series data is an excellent fit for this type of pattern. For these examples, we'll use a weather station that is creating temperature data every minute. You will see how using the row key and sequence can be a powerful data modeling tool.

Single device per row - Time Series Pattern 1

The simplest model for storing time series data is creating a wide row of data for each source. In this first example, we will use the weather station ID as the row key. The timestamp of the reading will be the column name and the temperature the column value (figure 1). Since each column is dynamic, our row will grow as needed to accommodate the data. We will also get the built-in sorting of Cassandra to keep everything in order.


```
CREATE TABLE temperature (
   weatherstation_id text,
   event_time timestamp,
   temperature text,
   PRIMARY KEY (weatherstation_id,event_time)
);
```

Now we can insert a few data points for our weather station.

```
INSERT INTO temperature(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:01:00','72F');

INSERT INTO temperature(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:02:00','73F');

INSERT INTO temperature(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:03:00','73F');

INSERT INTO temperature(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:04:00','74F');
```

A simple query looking for all data on a single weather station.

```
SELECT event_time,temperature
FROM temperature
WHERE weatherstation_id='1234ABCD';
```

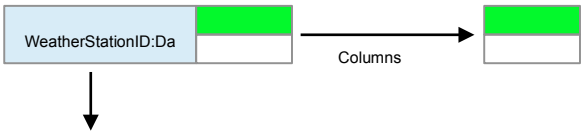
A range query looking for data between two dates. This is also known as a slice since it will read a sequence of data from disk.

```
SELECT temperature
FROM temperature
WHERE weatherstation_id='1234ABCD'
AND event_time > '2013-04-03 07:01:00'
```

Partitioning to limit row size - Time Series Pattern 2

In some cases, the amount of data gathered for a single device isn't practical to fit onto a single row. Cassandra can store up to 2 billion columns per row, but if were storing data every second

you wouldn't even get a month's worth of data. The solution is to use a pattern called row partitioning by adding data to the row key to limit the amount of columns you get per device. Using data already available in the event, we can use the date portion of the timestamp and add that to the weather station id. This will give us a row per day, per weather station, and an easy way to find the data. (figure 2)




```
CREATE TABLE temperature_by_day (
   weatherstation_id text,
   date text,
   event_time timestamp,
   temperature text,
   PRIMARY KEY ((weatherstation_id,date),event_time)
);
```

Note the (*weatherstation_id,date*) portion. When we do that in the PRMARY KEY definition, the key will be compounded with the two elements. Now when we insert data, the key will group all weather data for a single day on a single row.

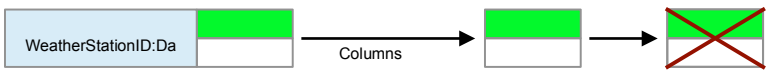
```
INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
VALUES ('1234ABCD','2013-04-03','2013-04-03 07:01:00','72F');

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
VALUES ('1234ABCD','2013-04-03','2013-04-03 07:02:00','73F');

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
VALUES ('1234ABCD','2013-04-04','2013-04-04 07:01:00','73F');

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
VALUES ('1234ABCD','2013-04-04','2013-04-04 07:02:00','74F');
```

To get all the weather data for a single day, we can query using both elements of the key.

```
SELECT *
FROM temperature_by_day
WHERE weatherstation_id='1234ABCD'
AND date='2013-04-03';
```

Reverse order timeseries with expiring columns - Time Series Pattern 3

Another common pattern with time series data is rolling storage. Imagine we are using this data for a dashboard application and we only want to show the last 10 temperature readings. Older data is no longer useful, so can be purged eventually. With many other databases, you would have to setup a background job to clean out older data. With Cassandra, we can take advantage of a feature called expiring columns to have our data quietly disappear after a set amount of seconds. (figure 3)



```
CREATE TABLE latest_temperatures (
   weatherstation_id text,
   event_time timestamp,
   temperature text,
   PRIMARY KEY (weatherstation_id,event_time),
) WITH CLUSTERING ORDER BY (event_time DESC);
```

Now when we insert data. Note the TTL of 20 which means the data will expire in 20 seconds.

```
INSERT INTO
latest_temperatures(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:03:00','72F') USING TTL 20;

INSERT INTO
latest_temperatures(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:02:00','73F') USING TTL 20;

INSERT INTO
latest_temperatures(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:01:00','73F') USING TTL 20;

INSERT INTO
latest_temperatures(weatherstation_id,event_time,temperature)
VALUES ('1234ABCD','2013-04-03 07:04:00','74F') USING TTL 20;
```

As soon as you insert the data, start selecting all rows over and over. Eventually you will see all the data disappear. This is an example of the TTL period expiring. Imagine what kind of interesting things you could do with your application data model using these.

Conclusion

Time series is one of the most compelling data models for Cassandra. It's a natural fit for the big table data model and scales well under a variety of variations. Many production use cases are similar to the examples above. For those users the problem of storing data at machine generating speeds and still keeping it organized in a useful manner is no longer a challenge. Hopefully this getting started guide will get your creative juices flowing.