The Power of Leveraging Pre-aggregated Data

1 month ago • 4 replies

yul pertierra

Overview

Creating pre-aggregated data with Devo's Aggregated Tasks feature can be advantageous for subsets of datasets that are frequently searched.

With Aggregated Tasks, we can create a pre-aggregated cut of the original data source-- so that the data we most frequently request is more highly available to us, as it's:

- **A smaller subset of data we are querying.**
- **More responsive for our searches**.

> - The **Aggregated Task's query in of itself must be efficiently written**, as it's always working over the source data to produce the pre-aggregated dataset.
> - Please note that using aggregation tasks **may entail an extra charge** to your usual Devo subscription fee.

Pre-aggregated data should be generated with consideration and a proper plan in mind.

Examples:

- To assist frequently used activeboards queries.
- To assist alerts that have matching query contexts.

Check the example below, in which we'll be querying a 1 TB source to produce a 50 GB source.

Let's imagine that we have a tag1.tag2.tag3.tag4 table with 1 terabyte (TB) of data every day (24 hours), which we query based on 1-day intervals to retrieve the same 50 gigabytes (GB) of data every time.

In order to implement pre-aggregated data, we will use the following query:

Query example:

```
from tag1.tag2.tag3.tag4 where field1 = "x" where field2 -> "substring" group every 5m by field3, field4 select sum(field5) as summedValue
```

This query will result in the end-user's query only parsing 50 GB of data instead of the original 1 TB.

Examples

Type of queries that benefit from pre-aggregated data

> This feature becomes ***very* advantageous as the source table grows** in size and extends more columns of data.

First, let’s explain some terms to help drive the example:

- `filters`: this is the `where <clause>` portion of the Aggregated Tasks.
- `grouping keys`: these are the fields that are aggregated/grouped by; `group by <field>`.
- `benefits`: these are the statistical datasets that benefit, as they’re already calculated from source in the pre-aggregated subset; `select sum(<field>) as sumStat`.

Obviously, the *same* query would leverage the pre-aggregated data available.

For different queries, Aggregated Tasks will create a subset of data, respective to the results of the query defined in the task.

Example

To leverage the pre-aggregated data, our query must **use the grouping keys**. This means we must include in the query the field or fields we grouped by.

If the grouping keys are in use any filters applied to them **will be *free* because these fields have been already grouped**. When you group by a field, you aggregate by every value in that field so filtering by a field that was used as a `grouping key` in Aggregate Tasks becomes *free* (allowing the subset of data to be easily accessed).

The aggregated operations used after the group become *free* as well, as the statistic column becomes available from the Aggregated Tasks results, implying the named `benefits`.

> Comments inside LINQ are **heading by //.**

```
// table from tag1.tag2.tag3.tag4  // filters where field1 = "x" where field2 -> "substring" // grouping keys group every 5m by field3, field4 // benefits select sum(field5) as summedValue
```

Step-by-step example

Let’s make up a table as well as some ‘fake’ events to help visualize this concept.

Source data as it’s queried from Devo:

> For this example, **we’ll pretend this is source data**. Devo actually stores the data *raw* and normalizes it at query execution time.

```
eventTime,user,dept,state,country,ip,role,event 2022-01-11 16:01:00, devo, cs, TX, USA, 20.478.93.2, attempted login 2022-01-11 16:01:45, devo, cs, TX, USA,
20.478.93.2, login failed! 2022-01-11 16:02:38, devo, cs, TX, USA, 20.478.93.2,
attempted login 2022-01-11 16:03:10, devo, cs, IL, USA, 48.78.312.9, attempted
login 2022-01-11 16:03:11, devo, cs, TX, USA, 20.478.93.2, login failed! 2022-01-
11 16:03:14, devo, cs, IL, USA, 48.78.312.9, login successful!
```

In this use case, we find ourselves querying this dataset similarly every day.

If it is a very valuable dataset from the point of view of our business, it is likely that we will end up using this data to create:

- An activeboard for accessing the data through visualizations and analytics tools.
- Alerts for monitoring tasks.

Usually, these activeboards and alerts will be built using the following query format:

```
from table where event = "login failed!" or event = "login successful!" group
every 5m by user select round(hllppcount(ip)) as uniqueIps
```

And the results would look as such:

```
2022-01-11 16:00:00, someUser, 1 2022-01-11 16:00:00, anotherUser, 1 2022-01-11
16:05:00, devo, 2
```

If we create an Aggregated Task to generate a pre-aggregated subset of this data, we could query this `table` filtering exclusively by `user` for *free* for events that match the condition `where event = "login failed!" or event = "login successful!"`.

```
from table where event = "login failed!" or event = "login successful!" where
user = "suspect1" or user = "suspect2" or user = "suspect3" group by user select
round(hllppcount(ip)) as ips
```

As a bonus, we can access the `round(hllppcount(ip))`-generated stat column `uniqueIps` for *free*; we only need to use the same operation at the end of our grouping.

```
from table where event = "login failed!" or event = "login successful!" group by
user select round(hllppcount(ip)) as ips where ips > 1
```

Walk away tips

- Devo's query engine will ***always*** try to leverage pre-aggregated data in its queries when possible.
- Devo's Activeboard **widgets can display whether a query leverages pre-aggregated data or not**. See this article for more details.

Thanks to @twood (pre-aggregated data rockstar) for providing valuable technical insights.

aggregation

4 people like this