PLURALSIGHT | HELP CENTER

Aggregating data

Tags: Flow

Aggregation is a powerful function that returns a single value based on a set of values. A common use case is generating results based on aggregating values as you would in SQL. The Flow API provides aggregate functions to spare you from ingesting large quantities of data to perform your own aggregation.

In this article

Data aggregation

Aggregation is a powerful function that returns a single value based on a set of values. You should be familiar with the GROUP BY clause in SQL to use this feature. Flow provides a robust Rest-based syntax that emulates SQL aggregates.

For example, you may want the average rework over the last year or the largest commit ever. Normally, you would have to find all the commits and perform the aggregate function locally in your client. Using Flow's built-in aggregate function, you can accomplish this with a single call.

This functionality includes:

- Multiple aggregate functions in a single request
- Customizable grouping
- Ordering that includes aggregate fields

These are the supported aggregate functions:

- **avg** – Mean average
- **count** – Count
- **max** – Maximum
- **min** – Minimum
- **std** – Standard deviation
- **sum** – Sum
- **var** – Variance

back to top

Basic API usage

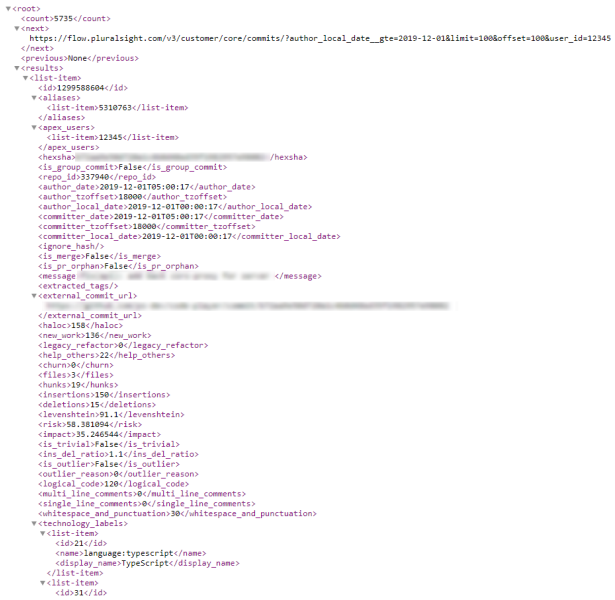
Let's look at an example of what a basic request might look like: we want to count the number of new lines of code in the last 30 days by User Bob Smith who has a `user_id` of 12345. In order to accomplish this without the benefit of aggregates, we would likely need to write a function that first collects all of Bob's commits by iterating through numerous sets of results using limit and offset in individual GET requests. In a second loop, we'd then have to iterate through each commit document and sum the value of the `new_work` field. That is a lot of work and overhead in order to simply add up a single set of values.

This would likely be the first request in the process, without using aggregation. Notice the length of the response:

Request

```
https://flow.pluralsight.com/v3/customer/core/commits/?user_id=[12345]&author_local_date__gte=[yyyy-mm-dd]
```

Response:



back to top

Advanced aggregate usage

To start, we'll modify the previous request and use the `sum` aggregation function to find the sum of `new_work` across all commits.

Request:

```
https://flow.pluralsight.com/v3/customer/core/commits.agg/?user_id=[12345]&author_local_date__gte=[yyyy-mm-dd]&aggregate[sum]=new_work
```

With this response:

```
{
  "count": 1

  "results": [
            {
               "new_work_sum": 8989
            }
  ],
  "records": 120
 }
```

You'll notice that a few things are different:

- The `.agg` suffix on the Commits URL tells the API that we are going to process this request as an aggregated request.
- The `aggregate` query parameter key defines which aggregate function to apply, while the value after the equal sign is the field to perform the aggregate function on.
- Our response document has changed. `count` still reflects the number of items in the result array. `records` has been added and reflects the number of rows that match our filter criteria. In our example, you can see that Bob Smith had 120 matching commits based on our 30-day filter criteria with a total `new_work` sum of 8,989 lines of code.
- The naming convention for fields in our aggregate `results` is `{field_name}_{aggregatefunction}`. In our example, we performed a sum against the field `new_work`, resulting in the field name `new_work_sum`.

Let's say we wanted to see the total number of commits and the average `new_work` lines of code per commit for each user on the team for a particular date range. We also want to group the data by the user's alias ID as well as the unique ID of the repo.

Request:

```
https://flow.pluralsight.com/v3/customer/core/commits.agg/?user_id__in=
```

[12345,67891]&author_local_date__gte=[yyyy-mm-dd]&aggregate[count]=id&aggregate[avg]=new_work&group_by[apex_user_id,repo_id]

Response:


```
{
      “count”: 3,

      “results”: [
            {
                  “apex_user_id”: 12345,
                  “repo_id: 1,
                  “new_work_avg”: 560,
                  “id_count”: 25
             }
             {
                  “apex_user_id”: 12345,
                  “repo_id”: 2,
                  “new_work_avg”: 39
                  “id_count”: 75,
            }
            {
                  “apex_user_id”: 67891,
                  “repo_id”: 1,
                  “new_work_avg”: 14
                  “id_count”: 50,
            }
  ]
  ;"records": 150
}
```


You can see from this example, there are three items in the results array. User 12345 has been working on both Repo 1 and Repo 2, while User 67891 is working only on Repo 1. There have been 150 total commits by the two users on this team. Also, note that you can see that the average quantity and distribution of their `new_work` differs significantly.

back to top

Advanced aggregates

These are relatively simple examples. You can use as many aggregate functions as you want and group by as many fields as you like based on the fields made available by each resource.

Additionally, ordering your aggregate results works in the exact same way as the conventional API calls, with the ordering query parameter and a comma-separated list of fields where fields prefixed with `-` will order descending. If you wish to order by one of your aggregate fields such as `new_work_avg`, you can do that by saying `&ordering=-new_work_avg`.

With this in mind, we could easily build more complex queries, such as:

Request:

https://flow.pluralsight.com/v3/customer/core/commits.agg/?author_local_date__gte=[yyyy-mm-dd]&aggregate[count]=id&aggregate[sum]=new_work,churn&aggregate[avg]=new_work,churn&aggregate[max]=author_date&group_by[
id_count

This request translates to:

Give me the total number of commits [for the given `author_local_date`], grouped by user. For each user I want to see the total lines of new work and rework, the average lines of new work and rework, and the most recent commit date of each user, ordered by total commits in descending order.

If you need help, please contact Pluralsight Support.