


# Querying Large(ish) Datasets fast and Efficiently via Salesforce’s REST API



Charlie Jonas

Nov 8, 2019 · 5 min read

Below we will dive deep into various ways we can restructure our query requests using the [Composite Resources](#) to achieve better performance with fewer API calls.

But first a pop-quiz...

***Given exactly two REST API calls (no BULK), whats the maximum number of records you can retrieve via a SOQL query?***

- A: 4000
- B: 52000
- C: 60000

*You might already be thinking, “why not just use the BULK API?”.*

*The BULK API is, without doubt, the fastest way to pull down very large record sets.*

*It is also more complex to implement, can consumes lots of API calls, and has limitations around the query and results format. Unless performance is your only concern, it only really starts to make sense as you approach 100k records.*

*We will be exploring how to optimize query requests using ONLY the REST API.*

*To be clear, I am not blindly recommending the following approaches over the BULK API. As always, the right tool for the job varies from scenario to scenario.*

## Standard Approach

Consider the following query:

```
SELECT Id, Name FROM Account LIMIT 52000
```

To make this query via REST, we would do something like this:

```
GET /services/data/v44.0/query?q=SELECT Id, Name FROM Account LIMIT 52000
```

\*<sub>q</sub> *intentionally left unencoded for readability*

However, that would only return the first 2000 records via response like this:

```
{
  done:false,
  totalSize:52000,
  nextRecordsUrl:"/services/data/v44.0/query/01g3A000071ZwLKQA0-2000",
  records: [...] //0-2000
}
```

To get all 52000 records, we’d continue making requests to `nextRecordsUrl` , until `done==true` :

```
GET /services/data/v44.0/query/01g3A000071ZwLKQA0-2000
GET /services/data/v44.0/query/01g3A000071ZwLKQA0-4000
GET /services/data/v44.0/query/01g3A000071ZwLKQA0-6000
GET /services/data/v44.0/query/01g3A000071ZwLKQA0-8000
GET /services/data/v44.0/query/01g3A000071ZwLKQA0-10000
... +20 more
```

By the time we are done with this, we would have made 26 API requests...

**Yikes!**

## Composite Batch with `nextRecordsUrl` interpolation

Luckily, there are more efficient way.

We'll start by making the initial query request same as before.

*However*, instead of calling `nextRecordsUrl` one by one, we're going to interpolate ALL the “next record URLs” upfront. In javascript, this could be as simple as:

```
let response = query('SELECT Id, Name FROM Account LIMIT 52000');
let baseNextUrl = response.nextRecordsUrl.split('-')[0];
let nextUrls = [];
for(let i = 1; i < response.totalSize / 2000; i++){
  nextUrls.push(`${baseNextUrl}-${i*2000}`);
}
```

Now we can leverage the Composite Batch API to pull the remaining 50k records in a single request!

```
{
  "batchRequests" : [
    {
      "method" : "GET",
      "url" : "/services/data/v44.0/query/01g3A000071ZwLKQA0-2000"
    },
    {
      "method" : "GET",
      "url" : "/services/data/v44.0/query/01g3A000071ZwLKQA0-4000"
    },
    {
      "method" : "GET",
      "url" : "/services/data/v44.0/query/01g3A000071ZwLKQA0-6000"
    }
    //... continue all the way to 50k
  ]
}
```

That's **24 LESS API** calls than the “Standard” approach!

The Composite batch request can only contain 25 sub-requests, but if you needed to query more than 50k records, just create a second or third batch request.

### A Caveat

You might have already recognized that the “interpolation” is a bit of a hack & relies on an assumptions that `nextRecordUrl` will **always** increment by 2000 records. Unfortunately, this is not always the case. While you can set the result size via a header, there is no guarantee Salesforce will honor it. That said, the only scenario I've come across that causes the result set to be less than 2000k is queries with children.

```
SELECT Id, (SELECT Id FROM Contacts) FROM Account
```

The `nextRecordUrl` is calculated by limiting the results to 2000 records per request, including child records. So if each Account had 9 child contacts, the `nextRecordUrl` would only increment by `200` .

*\*Note: In my testing only child relationships count toward the limit. Objects returned via Parent relationships don't.*

## Composite Chaining

If the above approach made you feel dirty, don't go running back to the “traditional method” just yet!

We can still do much better.

The Composite API only allows us to make up to 5 queries in a single request. However, unlike Composite Batch, you can pass the result from a previous request into a following one!

We can leverage this to make the **initial query + 4 additional “nextRecordsUrl” in a single call**:

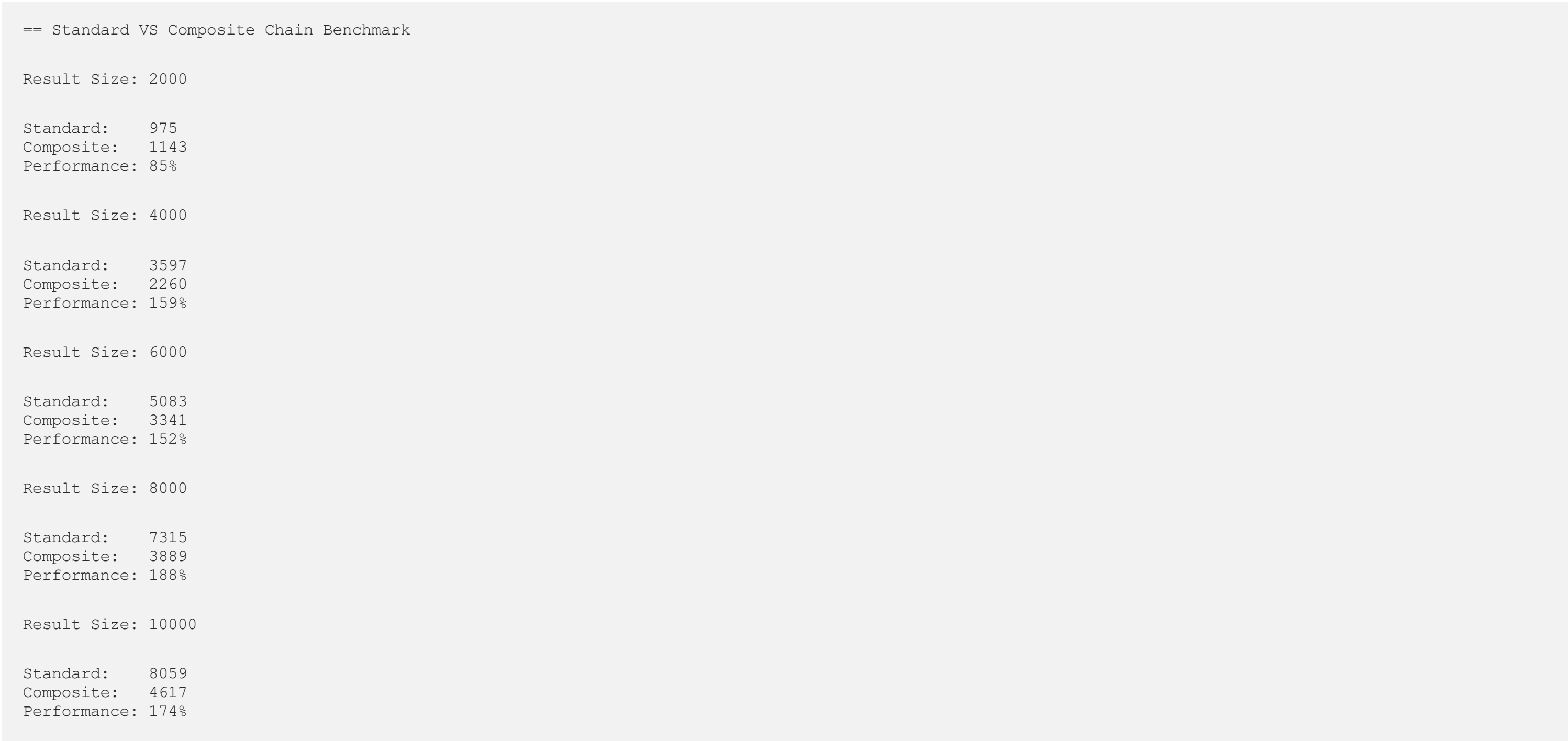
```
{
  "compositeRequest": [
    {
      "method": "GET",
      "url": "/services/data/v42.0/query?SELECT Id, (SELECT Id FROM Contacts) FROM Account",
      "referenceId": "req1",
    },
    {
      "method": "GET",
      "referenceId": "req2",
      "url": "@{req1.nextRecordsUrl}"
    },
    {
      "method": "GET",
      "referenceId": "req3",
      "url": "@{req2.nextRecordsUrl}"
    },
    {
      "method": "GET",
      "referenceId": "req4",
      "url": "@{req3.nextRecordsUrl}"
    },
    {
      "method": "GET",
      "url": "@{req4.nextRecordsUrl}"
    }
  ]
}
```

```
    }
  ]
}
```

Rinse and repeat until `done==true` .

In my opinion, you might as well just use this method every time you make a SOQL query via the REST API (unless you *know* your query will complete in a single request).

If the query is completed early, any remaining request will just error out. In my testing, these failed request have a very minimal impact on performance. In all but the worst case, the “Composite Chaining” should out-perform the “Standard” approach:



***Back to the original question...***

If we combine this approach with the “Composite Batch” method above, we could retrieve up to **60k records in just two API calls!**

## Multiple Queries

So far we’ve only been considering how to optimize a single query.

Imagine now that you need to perform a series **independent queries**, each of which potentially return a large data set (maybe during your apps initial load sequence):

```
SELECT Id, (SELECT Id FROM Tasks) FROM Account LIMIT 10000

SELECT Id, Name FROM Contact LIMIT 10000

SELECT Id, Name FROM Opportunity LIMIT 10000
...
```

Instead of executing these individually, we can use “Composite Batch” to combine them into a single request:

```
{
  "batchRequests" : [
    {
      "method" : "GET",
      "url" : "/services/data/v44.0/query/q?=SELECT Id, (SELECT Id FROM Tasks) FROM..."
    },
    {
      "method" : "GET",
      "url" : "SELECT Id, Name FROM Contact WHERE OwnerId = '001231'"
    },
    {
      "method" : "GET",
      "url" : "SELECT Id, Name FROM Opportunity WHERE OwnerId = 'asdasb'"
    }
    ... continue with the rest of the queries
  ]
}
```

We now check each sub-request result for `done==true` and make another “Composite Batch” request with the `nextRecordsUrl`. Repeat until all sub-requests are “done”. **Each subsequent request will give us the next 2000 records from EACH query.**

You could optimize this even further by combining this with the “nextRecordsUrl Interpolation” approach (where possible), but the nice thing about this pattern is that we aren’t guessing the `nextRecordsUrl`. It might not be as efficient, but it will **always** complete in `floor(largestRecordSize / 2000)+1` total API calls.

*NOTE: if you have less than  $\leq 5$  queries, it’s always more efficient to use the “Composite Chaining” method.*