# DuckDB as the New jq (https://www.pgrs.net/2024/03/21/duckdb-as-the-new-jq/)

March 21, 2024 • 3 minute read

Recently, I've been interested in the DuckDB (https://duckdb.org/) project (like a SQLite (https://www.sqlite.org/) geared towards data applications). And one of the amazing features is that it has many data importers included without requiring extra dependencies. This means it can natively read and parse JSON as a database table, among many other formats.

I work extensively with JSON day to day, and I often reach for jq (https://jqlang.github.io/jq/) when exploring documents. I love `jq`, but I find it hard to use. The syntax is super powerful, but I have to study the docs anytime I want to do anything beyond just selecting fields.

Once I learned DuckDB could read JSON files directly into memory, I realized that I could use it for many of the things where I'm currently using `jq`. In contrast to the complicated and custom `jq` syntax, I'm very familiar with SQL and use it almost daily.

Here's an example:

First, we fetch some sample JSON to play around with. I used the GitHub API to grab the repository information from the golang org:

```
% curl 'https://api.github.com/orgs/golang/repos' > repos.json
```

Now, as a sample question to answer, let's get some stats on the types of open source licenses used.

The JSON structure looks like this:

```
[
  {
    "id": 1914329,
    "name": "gddo",
    "license": {
      "key": "bsd-3-clause",
      "name": "BSD 3-Clause \"New\" or \"Revised\" License",
      ...
    },
    ...
  },
  {
    "id": 11440704,
    "name": "glog",
    "license": {
      "key": "apache-2.0",
      "name": "Apache License 2.0",
      ...
    },
    ...
  },
  ...
]
```

This might not be the best way, but here is what I cobbled together after searching and reading some docs for how to do this in `jq`:

```
 % cat repos.json | jq \
  'group_by(.license.key)
  | map({license: .[0].license.key, count: length})
  | sort_by(.count)
  | reverse'
[
  {
    "license": "bsd-3-clause",
    "count": 23
  },
  {
    "license": "apache-2.0",
    "count": 5
  },
  {
    "license": null,
    "count": 2
  }
]
```

And here is what it looks like in DuckDB using SQL:

```
% duckdb -c \
  "select license->>'key' as license, count(*) as count \
  from 'repos.json' \
  group by 1 \
  order by count desc"
┌──────────────┬───────┐
│   license    │ count │
│   varchar    │ int64 │
├──────────────┼───────┤
│ bsd-3-clause │    23 │
│ apache-2.0   │     5 │
│              │     2 │
└──────────────┴───────┘
```

For me, this SQL is much simpler and I was able to write it without looking at any docs. The only tricky part is querying nested JSON with the `->>` operator. The syntax is the same as the PostgreSQL JSON Functions (https://www.postgresql.org/docs/current/functions-json.html), however, so I was familiar with it.

And if we do need the output in JSON, there's a DuckDB flag for that:

```
% duckdb -json -c \
  "select license->>'key' as license, count(*) as count \
  from 'repos.json' \
  group by 1 \
  order by count desc"
[{"license":"bsd-3-clause","count":23},
{"license":"apache-2.0","count":5},
{"license":null,"count":2}]
```

We can still even pretty print with `jq` at the end, after using DuckDB to do the heavy lifting:

```
% duckdb -json -c \
  "select license->>'key' as license, count(*) as count \
  from 'repos.json' \
  group by 1 \
  order by count desc" \
  | jq
[
  {
    "license": "bsd-3-clause",
    "count": 23
  },
  {
    "license": "apache-2.0",
    "count": 5
  },
  {
    "license": null,
    "count": 2
  }
]
```

JSON is just one of the many ways of importing data into DuckDB. This same approach would work for CSVs, parquet, Excel files, etc.

And I could choose to create tables and persist locally, but often I'm just interrogating data and don't need the persistence.

Read more about DuckDB's great JSON support in this blog post: Shredding Deeply Nested JSON, One Vector at a Time (https://duckdb.org/2023/03/03/json.html)

**Update:**

I also learned that DuckDB can read the JSON directly from a URL, not just a local file:

```
% duckdb -c \
  "select license->>'key' as license, count(*) as count \
  from read_json('https://api.github.com/orgs/golang/repos') \
  group by 1 \
  order by count desc"
```

**Updated:** March 21, 2024