

HOW TO COPY JSON DATA TO AN AMAZON REDSHIFT TABLE



Here we show how to load JSON data into Amazon Redshift. In this example, Redshift parses the JSON data into individual columns. (It is possible to store JSON in char or varchar columns, but that's another topic.)

First, review [this introduction on how to stage the JSON data in S3](#) and instructions on how to get the Amazon IAM role that you need to copy the JSON file to a Redshift table.

In this example, we load 20 years of temperature data for Paphos, Cyprus. We purchased that data for \$10 from [OpenWeather](#). Of course, you could use any data.

Create a Redshift Table

First we create a table. We only want the date and these three temperature columns. We will give Redshift a JSONParse parsing configuration file, telling it where to find these elements so it will discard the others.

```
create table paphos (  
  
dt_iso timestamp not null distkey sortkey,  
  
temp real,  
  
temp_min real,
```

```
temp_max real
```

```
);
```

The weather data looks like this:

```
{
```

```
"city_name": "Paphos Castle",
```

```
"lat": 34.753637,
```

```
"lon": 32.406951,
```

```
"main": {
```

```
"temp": 55.35,
```

```
"temp_min": 51.8,
```

```
"temp_max": 65.53,
```

```
"feels_like": 49.44,
```

```
"pressure": 1016,
```

```
"humidity": 73
```

```
},
```

```
"wind": {
```

```
"speed": 9.17,
```

```
"deg": 20
```

```
},
```

```
"clouds": {
```

```
"all": 1
```

```
},
```

```
"weather": ,
```

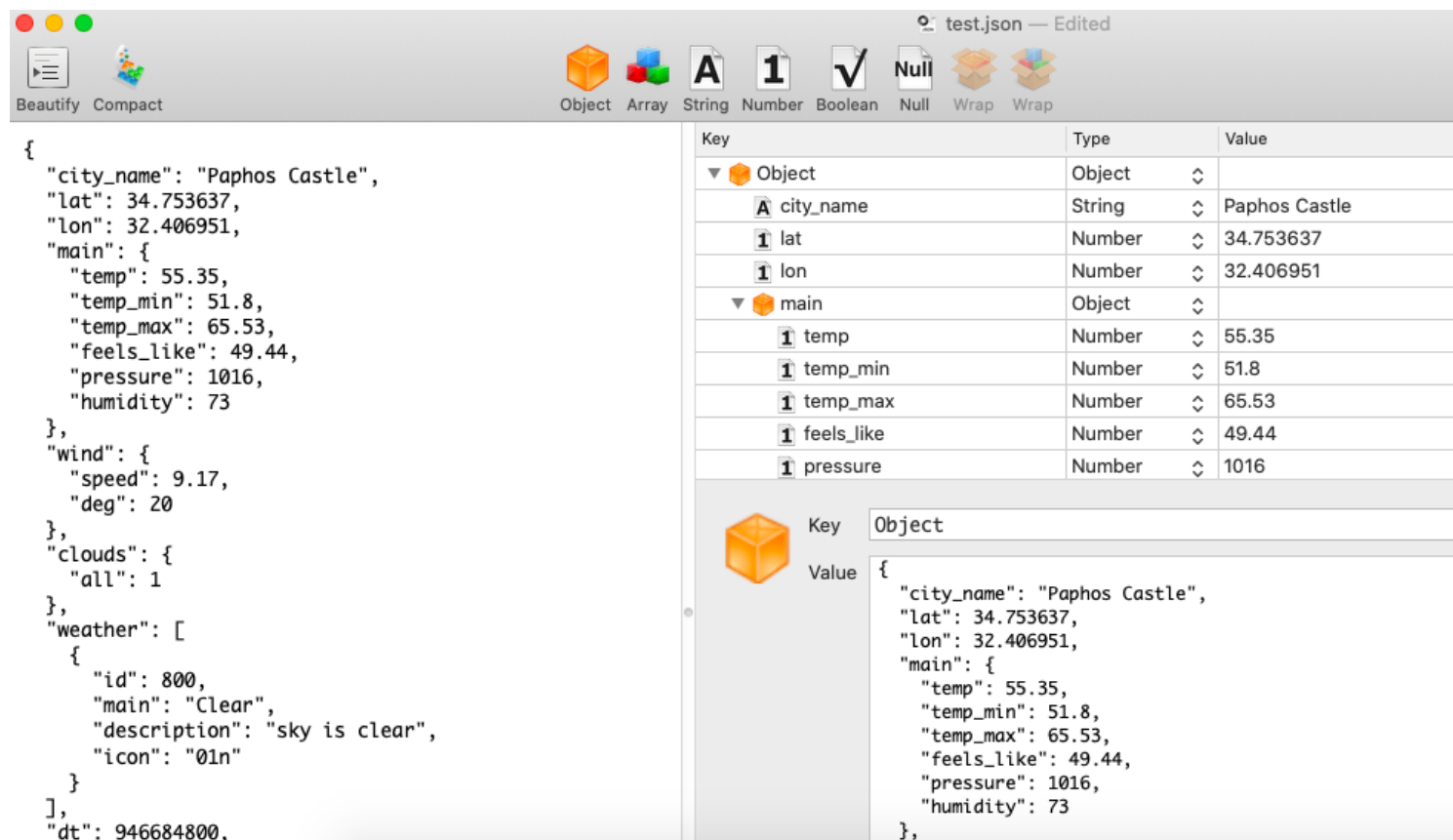
```
"dt": 946684800,
```

```
"timezone": 7200
```

```
}
```

Here is one record in the [JSON Power Editor for Mac](#).

Note: I recommend this editor if you work with JSON a lot, as it makes editing JSON files a lot easier. You can work with objects in the right-hand screen which will create the text in the left-hand screen. That saves you the trouble of having to fix syntax error and line up curly brackets.



The screenshot shows the JSON Power Editor for Mac interface. The left pane displays a JSON object for Paphos Castle weather data. The right pane shows a visual tree view of the same object, with a table listing keys, types, and values. Below the table, a visual representation of the object is shown with a key-value pair.

Key	Type	Value
Object	Object	
city_name	String	Paphos Castle
lat	Number	34.753637
lon	Number	32.406951
main	Object	
temp	Number	55.35
temp_min	Number	51.8
temp_max	Number	65.53
feels_like	Number	49.44
pressure	Number	1016

Key: Object
Value: {
 "city_name": "Paphos Castle",
 "lat": 34.753637,
 "lon": 32.406951,
 "main": {
 "temp": 55.35,
 "temp_min": 51.8,
 "temp_max": 65.53,
 "feels_like": 49.44,
 "pressure": 1016,
 "humidity": 73
 },
}

Create JSONPath file

We create a JSONPath file, which tells Redshift which elements to get. We have to give it the path of the item all the way down to the item. In other words, we can't put just the top-level key **weather** and it will get **temp**, **temp_min**, and **temp_max**. We have to give it the full path of JSON keys **main->temp**.

We don't have any arrays in this example, but it supports that using notation.

```
{
```

```
"jsonpaths": "
```

```
"$"
```

```
"$"
```

```
"$"
```

```
]
}
```

Prepare and upload JSON file to S3

This text file is 64 MB of daily weather records for the past 20 years. Unfortunately, it's made all as one JSON array. So, we have to remove the first bracket and last bracket characters from the file.

Also, Redshift seems to require for the JSONP format that each record have a line feed at the end. Then means we need to insert a line feed for each. So, use these three sed statements to do that.

Note: JSONP file format means having one record right after another. So, taken together it's not a valid JSON object. Instead it's just a way to put many JSON objects into a file for bulk loading.

```
sed 's/^.//'
```

```
sed 's/.$//'
```

```
sed 's/,{"city/\n{"city/g'
```

Copy this file and the JSONPaths file to S3 using:

```
aws s3 cp (file) s3://(bucket)
```

Load the data into Redshift

We use this command to load the data into Redshift. **paphosWeather.json** is the data we uploaded. **paphosWeatherJsonPaths.json** is the JSONPath file.

```
copy paphos
```

```
from 's3://gluebmcwalkerrowe/paphosWeather.json'
```

```
credentials 'aws_iam_role=arn:aws:iam::xxxxx:role/Redshift'
```

```
region 'eu-west-3'
```

```
json 's3://gluebmcwalkerrowe/paphosWeatherJsonPaths.json';
```

Common errors

If you have formatted the text or JSONPaths table wrong or illogically you will get any of these errors.

The first says to look into **select * from stl_load_errors** for more details:

```
ERROR: Load into table 'paphos' failed. Check 'stl_load_errors' system table for details.
```

This error says we have five elements in the JSONPath file but have created a database table with 13

columns. Go back and fix those.

```
ERROR: Number of jsonpaths and the number of columns should match. JSONPath
size: 5, Number of columns in table or column list: 13 Detail: -----
----- error: Number of jsonpaths and the number
of columns should match. JSONPath size: 5, Number of columns in table or
column list: 13 code: 8001 context: query: 273 location: s3_utility.cpp:780
process: padbmaster -----
```

If you put all your JSON data into an array instead of the JSONP format it will be too large. Then you might get:

String length exceeds DDL length

Check the loaded data

Here we look at the first 10 records:

```
select * from paphos limit 10;
```

Rows returned (10)				Export
<input type="text" value="Search rows"/>				< 1 >
dt_iso	temp	temp_min	temp_max	
2000-01-01 02:00:00	55.669998	51.799999	65.260002	
2000-01-01 03:00:00	54.09	48.919998	65.110001	
2000-01-01 08:00:00	64.830002	64.400002	66.580002	
2000-01-01 09:00:00	65.970001	65.150002	66.739998	
2000-01-01 11:00:00	65.82	64.519997	66.199997	
2000-01-01 13:00:00	64.510002	62.32	66.199997	
2000-01-01 14:00:00	63	60.639999	64.400002	
2000-01-01 15:00:00	61.860001	60.439999	63.91	
2000-01-01 16:00:00	61.23	59	64.510002	

Here we count them. As you can see there are 181,456 weather records.

```
select count(*) from paphos;
```

Run **Clear**

Query history | **Query results** | Table details

Query **1958** [↗](#)

✔ Completed, started on September 18, 2020 at 14:09:56
ELAPSED TIME: 00 m 24 s

Rows returned (1)

count
181465

Additional resources

For more on this topic, explore these resources:

- [BMC Machine Learning & Big Data Blog](#)
- [AWS Guide](#), with 15+ articles and tutorials
- [Amazon Braket Quantum Computing: How To Get Started](#)