**New JSON Data Ingestion Strategy By Using the Power of Postgres**

Postgres always had a JSON support with somehow limited capability before the 9.2 version added the native JSON support. The release of version 9.3 has really taken the JSON feature to the next level with additional constructor and extractor methods. The capability of querying and transforming the JSON data type with Postgres gives you the new strategy to ingest JSON data from APIs or NoSQL databases.

To ingest JSON into a data warehouse, we traditionally use a programming language or ETL tool to transform JSON into a structured table format and upload it into a database table. When you think about it, this is not efficient. JSON is schema-less while the database table requires strictly defined schema. This makes transformation cumbersome especially when the schema often changes per record. Any change in the source JSON structure also requires additional development effort.

The ability to query and transform native JSON data type with Postgres enables us to load the JSON file as it is and use SQL to query or transform later. In this way, you can just adjust the query as new business requirement comes without further development effort.

It sounds amazing, doesn’t it? Yes, this is possible. Let’s have a look.

**Example**

Let’s use the same dataset from the previous example where we ingested JSON data from MongoDB [**How to get data from MongoDB with Python**](https://www.mydatahack.com/how-to-get-data-from-mongodb-with-python/).

In that example, we first pulled the data and transformed into a structured format with Python. Instead of transforming it, we can load each record directly into Postgres as it is. Then, we can use SQL to create the structured data.

In some sense, the way we ingested data in  [How to get data from MongoDB with Python](https://www.mydatahack.com/how-to-get-data-from-mongodb-with-python/) is a traditional way of creating structured tables from Json. Using the same will make it a good comparison between two methods.

In the previous post, we ingested the restaurant collection like below to create structured tables. Here, we will basically replicate it except that the final output is views of those two tables.

**Record Example**

**Data Model**

**Table Creation**

Each row corresponds to a JSON record. We only need one column with the data type of json.

[cc lang="sql" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

Create Table mongodb.restaurants\_json (

    data json

);

[/cc]

**Data Load**

We will use Python to insert JSON record into the table. I can present two different approaches. The first approach is to query the data from MongoDB and insert the record one by one. The second approach is to write the ingested data into a flat file and call copy command. When you do a bulk upload, this approach is faster.

If you need to have a recap on how to ingest MongoDB data, you can refer to  [How to get data from MongoDB with Python](https://www.mydatahack.com/how-to-get-data-from-mongodb-with-python/).

Whatever approach you take, you need to have a table with all the JSON records loaded in one column as below.

**Approach 1: Insert**

This is pretty straight forward. Note that if you have ‘ in the dataset, you will get an error. I replaced it with <>. As the record is inserted one by one, this takes longer than the bulk upload from a flat file.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

**Approach 2: Bulk Upload**

Each record is a row written as a long string. Once the file is generated, use copy\_expert() to upload the file. It seems WITH CSV Quote e’\x01’ Delimiter e’\02’ works the best for the copy command. I was getting a lot of errors without these conditions.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

**Create Views With Postgres JSON Functions**

Once the table is loaded, real fun begins! We can simply write SQL to query and transform the JSON records. The way we query JSON with Postgres is to map the key to the value with -> or ->> operators.

There are only a few things you need to know to get it started. Make sure that you understand how to map key to the value and the difference between -> and ->> (--> returns JSON object and ->> returns text) because the data type is important to use other JSON functions.

Json\_array\_elements() is the God-sent function which expands array into multiple rows. If we are to do this transformation in a programming language, we have to use loop. This function makes querying nested JSON so much easier. Note that the function takes only JSON object, not string. Therefore, use -> to get the array from the key.

It is probably a good idea to read [this tutorial](http://www.postgresqltutorial.com/postgresql-json/) first to get the idea. For further reference, I always use Postgres documentation [here](https://www.postgresql.org/docs/9.5/static/functions-json.html). I am using a few string functions. [Here](https://www.postgresql.org/docs/9.1/static/functions-string.html) is the reference.

**Main Table (Restaurant\_VW)**

[cc lang="sql" tab\_size="4" lines="-1" theme="mac-classic" ]

[/cc]

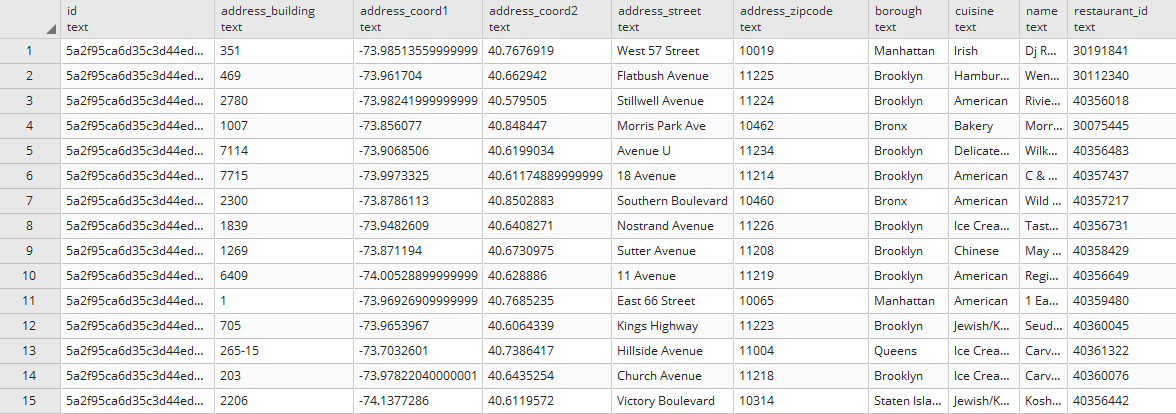
**Child Table (Grades\_VW)**

[cc lang="sql" tab\_size="4" lines="-1" theme="mac-classic" ]

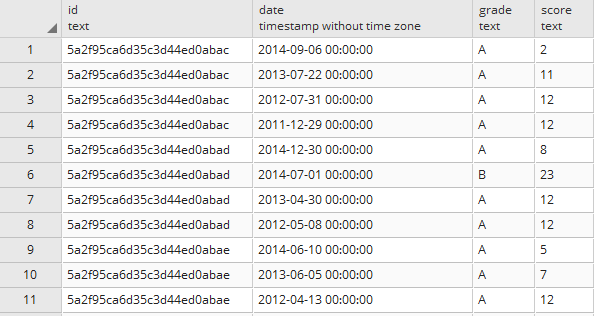
[/cc]

Now check out the view! We have the same output with the traditional method described here. But, this is more flexible and easy when you have JSON file changing dynamically and business is moving fast. If you are using Postgres, I would definitely recommend you to try it.

***Select \* From mongodb.restaurants\_vw;***



***Select \* From mongodb.grades\_vw;***



Cool!