**How Postgres JSON Query Handles Missing Key**

When we transform JSON to a structured format with a programming language in JSON data ingestion, we have to handle missing key. This is because JSON is schema-less and it doesn’t always have the same keys in all records as opposed to relational database tables.

In Python, the missing key handling looks like this.

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

try:

tmp.append(record['info']['postcode'])

except KeyError:

tmp.append('NA')

[/cc]

In the previous post, we examined Postgres JSON support and discussed how we could use it as a new JSON ingestion strategy ([**New JSON Data Ingestion Strategy By Using the Power of Postgres**](https://www.mydatahack.com/new-json-data-ingestion-strategy-by-using-the-power-of-postgres/)).

The first question comes to mind is how Postgres handles missing key when you are used to transforming JSON in a programming language.

I start with the conclusion. Postgres puts null value when the key is missing. It also puts null value when the key value is null. This is awesome because the database can handle the missing key without being specific about key error handling.

Let’s have take a look.

**Record Example**

We have 6 JSON records.

1. Complete record
2. Note is null
3. Note key is missing
4. Payment is null
5. Payment Type key is missing
6. Payment is missing

**JSON File**

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

[/cc]

**Loading to Postgres with Python**

Let’s load this to Postgres with a target table as usermanaged.transaction.

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

[/cc]

**Postgres Query**

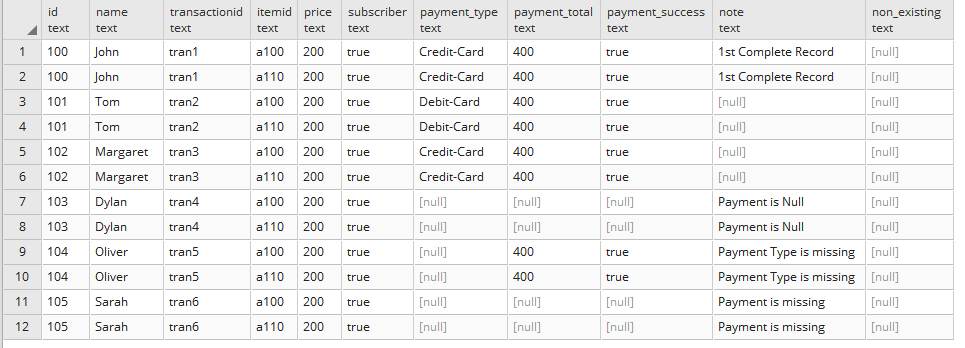
Let’s query the table to see how Postgres handles missing keys.

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

[/cc]

**Results**

As you can see, the missing key is set to null. If you have a key that does not exist in any record, it creates a column with null values. When the key value is null, it also sets the value to null.



Good times!