databricks

Cloud Storage with LakeFlow Connect
Standard Connectors

LECTURE

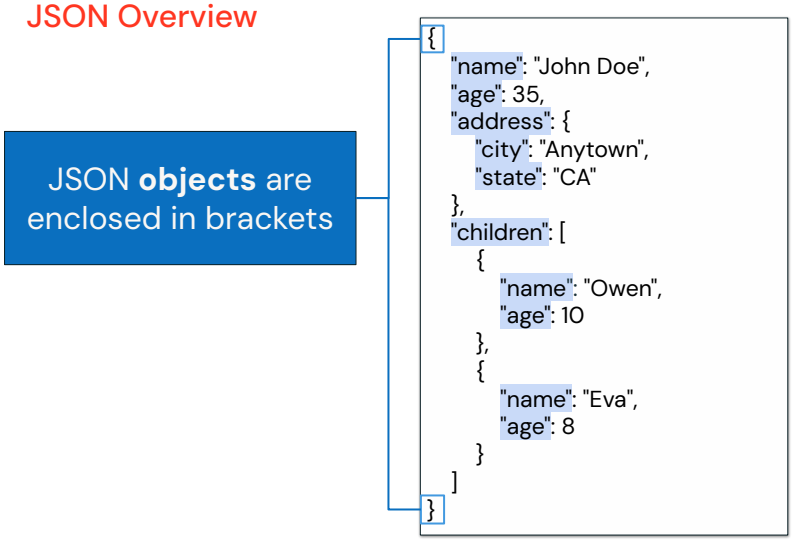
Ingesting Semi-Structured Data: JSON

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Ingesting semi-structured data like JSON enables efficient parsing and transformation of complex, nested input into structured Delta tables for advanced analytics in the Lakehouse.

Ingesting Semi-Structured Data: JSON

JSON Overview



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

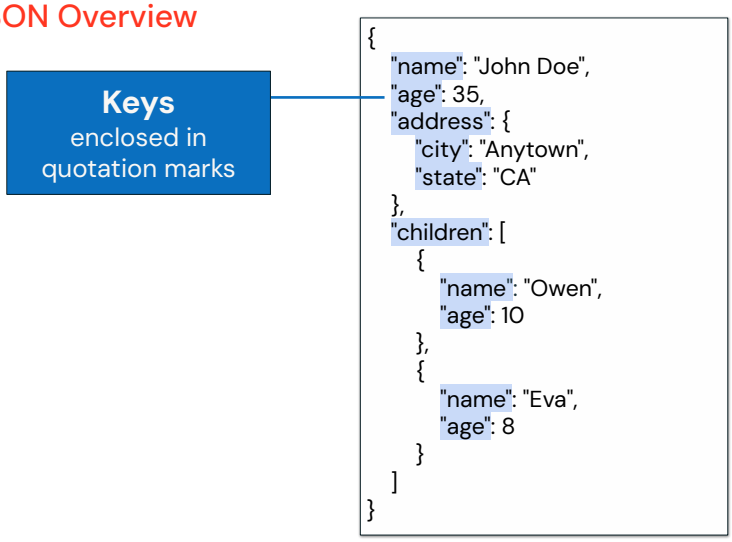
Ingesting semi-structured data, such as JSON files, is a common task for data engineers, especially when dealing with event data, logs, or data from APIs.

Before we get into how to work with JSON in Databricks, let's first review the basic structure of a JSON file.

JSON data is made up of JSON objects, which are typically enclosed in curly brackets {}.

Ingesting Semi-Structured Data: JSON

JSON Overview



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Within the curly brackets, JSON objects contain key-value pairs.

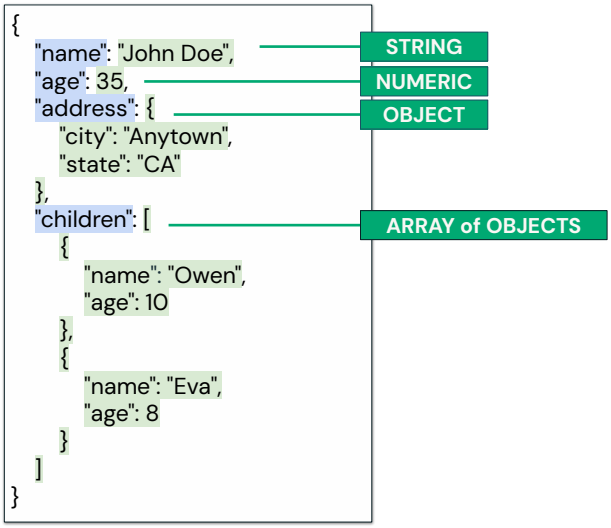
Each key is always a string enclosed in quotation marks. Each key contains a value.

Ingesting Semi-Structured Data: JSON

JSON Overview

Values

- String
- Number
- Boolean
- Array
- Object



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

The value of a key can be a string, number, boolean, array, or another JSON object, or even null.

These objects can be flat, meaning all key-value pairs are at one level, or they can be nested, where values themselves are JSON objects. The complexity depends on how the data is structured in the source.

Understanding this format is important because it affects how we parse and transform the data during ingestion.

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted STRING Column

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

Columns in tables can hold **JSON formatted strings** as values

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Now, when working with JSON data, it's common that after ingestion one or more columns in your table might contain JSON-formatted strings as values.

So the question becomes, how do you work with columns that store JSON formatted strings?

This is a common scenario when JSON isn't fully parsed during ingestion, or when JSON data is embedded within another field, like a log message or a nested structure. We'll explore techniques to parse, extract, and manipulate those JSON strings using SQL or DataFrame operations, so you can flatten and or access the nested fields just like regular columns.

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted Column as a STRING

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

*Working with JSON data can be done using different **column data types***

1. STRING Data Type

- JSON can be stored as a simple STRING
- Can hold **any JSON STRING** without constraints
- Less **performant**

Use **:** (colon) syntax to access subfields in JSON formatting strings

```
SELECT json_column:name
```
→ *John Doe*

```
SELECT json_column:address:city
```
→ *Anytown*

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

One technique for working with a JSON-formatted string column is to access values directly from the STRING data type column.

A few key points to remember:

- A column can simply store JSON data as a plain STRING.
- Since it's stored as a string, the column can hold any JSON string without constraints—it's just raw text from the system's perspective.
- However, this approach is less performant compared to other methods we will cover next.

To access subfields within JSON-formatted string columns, you can use the colon (:) syntax.

For example, if your column is named json_column, and you want to access the subfield "name", you would specify it as: *json_column:name*

This syntax allows you to extract specific fields directly from the JSON string stored in the column.

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted Column as a STRING

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

*Working with JSON data can be done using different **column data types***

1. STRING Data Type

- JSON can be stored as a as a simple STRING
- Can hold **any JSON STRING** without constraints
- Less **performant**

2. STRUCT Data Type

- You can parse JSON data into a STRUCT type, with a **defined schema**
- STRUCT **enforces** the JSON schema
- Is **more efficient** for querying than a JSON formatted STRING

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Another method to work with a JSON-formatted string column is to convert the column to a STRUCT data type.

Here are a few key points to remember:

- You can parse JSON data into a STRUCT type by defining a schema.
- The STRUCT enforces the JSON schema, ensuring data types and structure are consistent.
- Querying a STRUCT is more efficient than working with a raw JSON-formatted STRING.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

JSON String Types	Databricks SQL Data Type
String	**STRING**
Number	**INT/FLOAT/DOUBLE**
Boolean	**BOOLEAN**
Object	**STRUCT <>**
Array	**ARRAY <>**

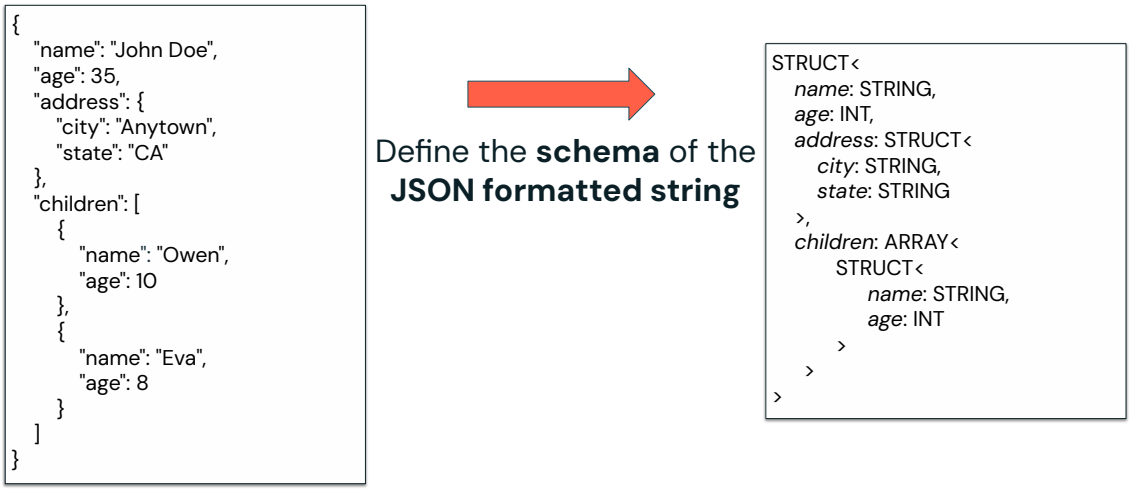
© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

When converting a JSON-formatted STRING column to a STRUCT column, you need to understand how JSON types map to Databricks SQL data types:

- JSON String maps to STRING
- JSON Number maps to INT, FLOAT, or DOUBLE
- JSON Boolean maps to BOOLEAN
- JSON Object maps to STRUCT<>
- JSON Array maps to ARRAY<>

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS



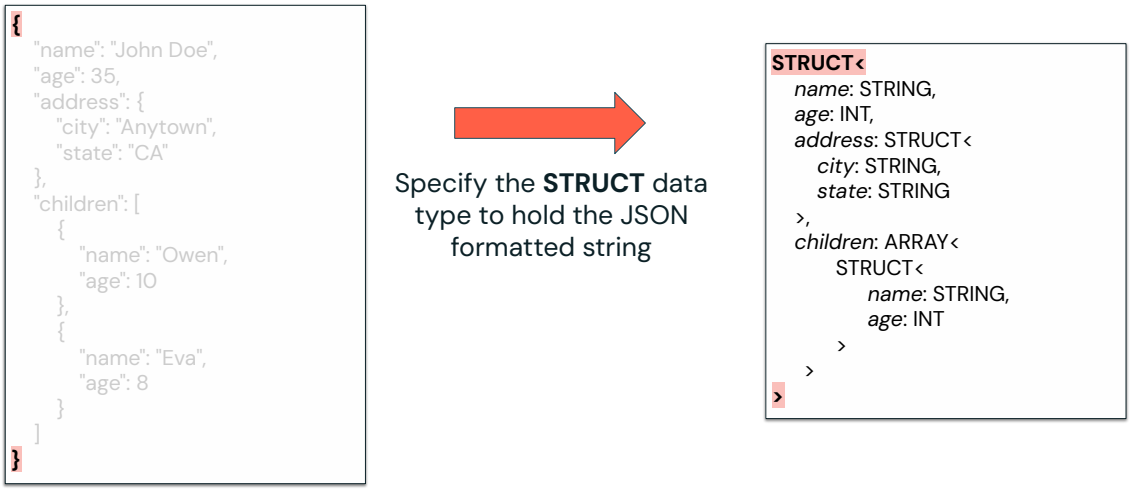
Now, let's go through the process of mapping a JSON-formatted STRING into a STRUCT column.

The first step is to define the schema (the structure) of the JSON-formatted string.

Defining the schema allows you to tell Databricks how to interpret each part of the JSON string and convert it into the appropriate data types within a STRUCT.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

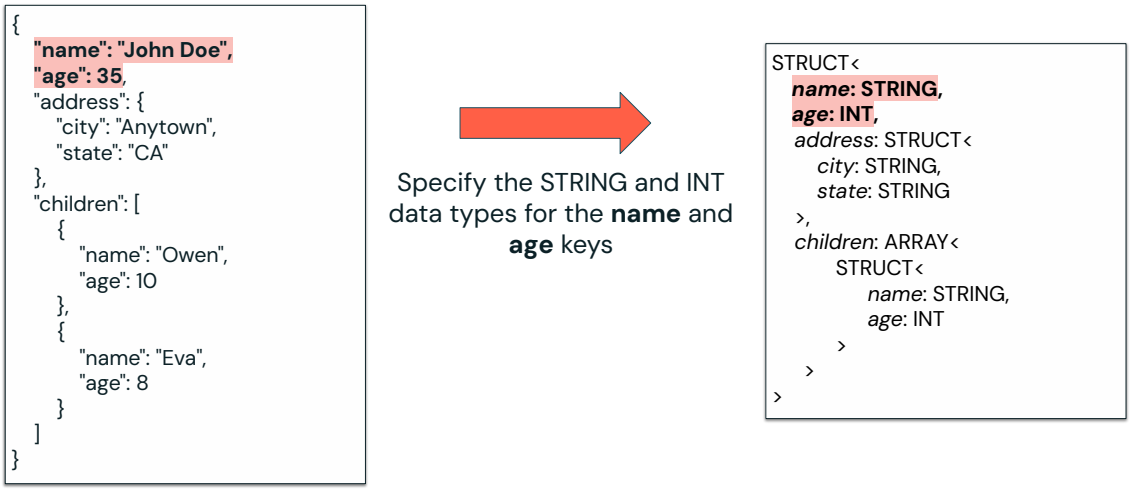


First, the STRUCT<> data type holds the entire structure of the JSON-formatted string.

It acts as a container for all the fields defined in the JSON, preserving their data types and hierarchy within a single column.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

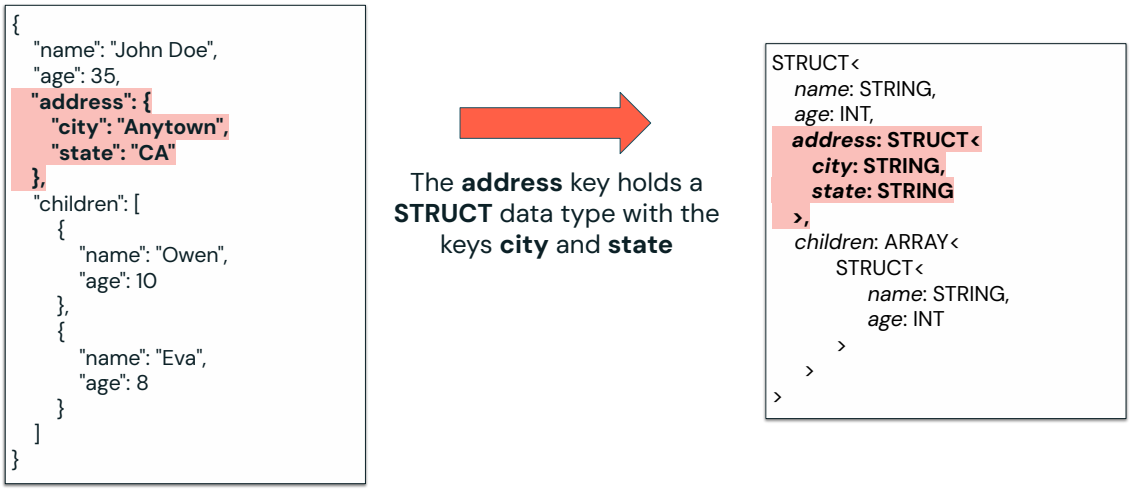
Next, go through the JSON key-value pairs one by one and define the structure for each.

First, let's look at the name and age keys.

- The name key holds a string.
- The age key holds an integer.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS



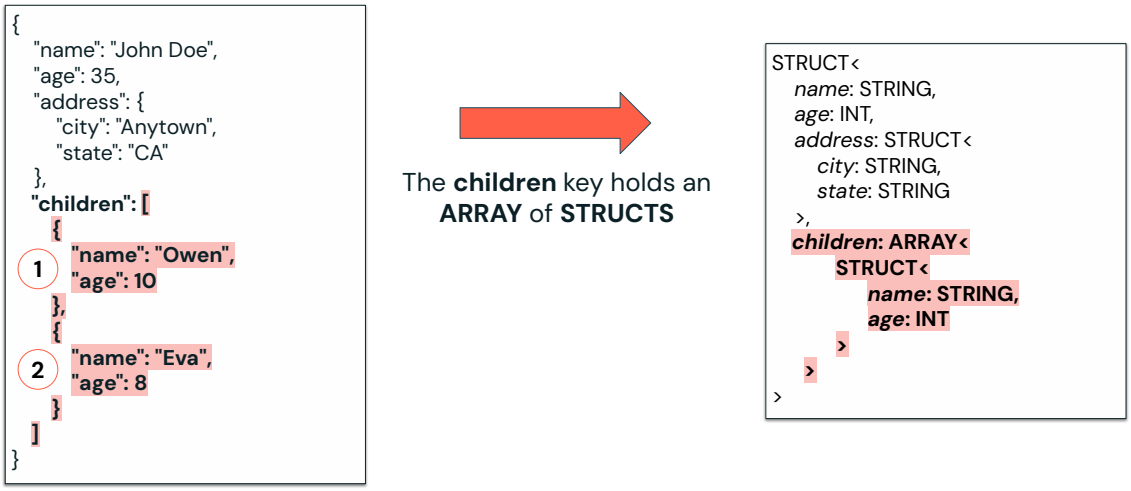
The address key contains another object, which we represent as a nested STRUCT.

This nested STRUCT has two keys: city and state.

Both city and state are of the string data type.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

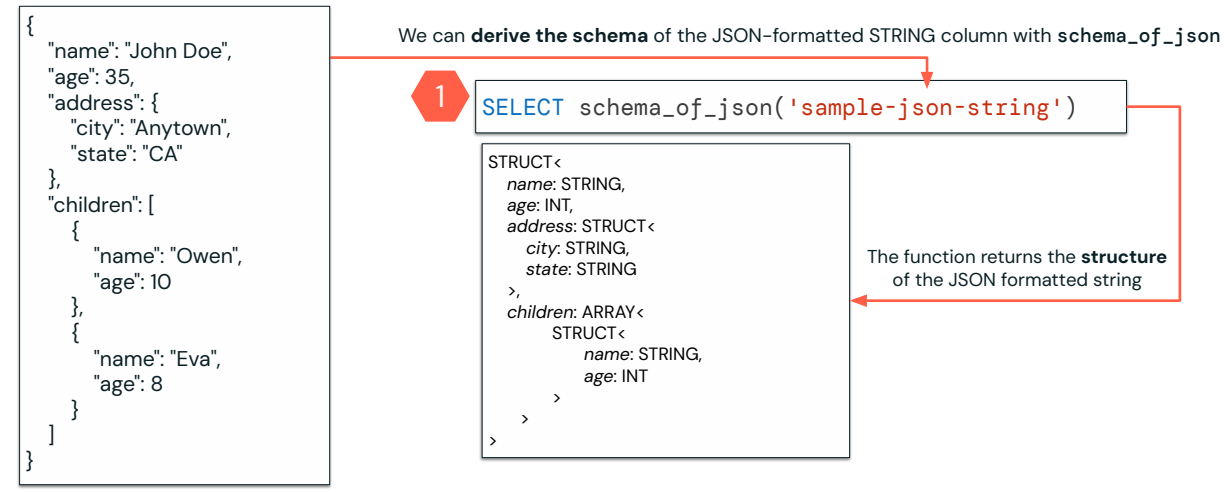


The children key holds an ARRAY of STRUCTS. Each STRUCT in the array contains two keys: name and age.

- The name key contains a string.
- The age key contains an integer.

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

After reviewing how to map a JSON-formatted STRING to a STRUCT column, let's learn how to easily determine the structure of the JSON string.

This can be done in two steps.

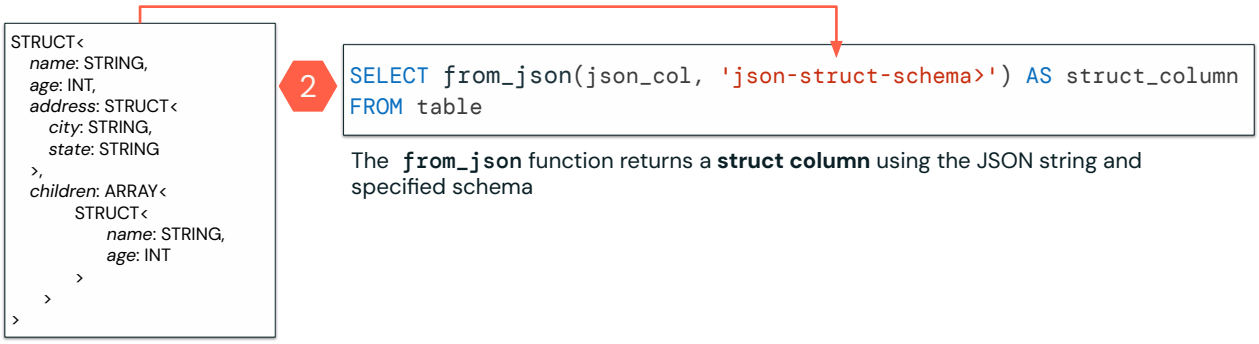
The first step is to get the schema of the JSON-formatted string.

Now, instead of manually defining the schema, you can use the built-in schema_of_json function to automatically derive the schema from an example JSON string.

Simply pass an example JSON-formatted string as the argument to this function, and it will return the inferred schema (structure).

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

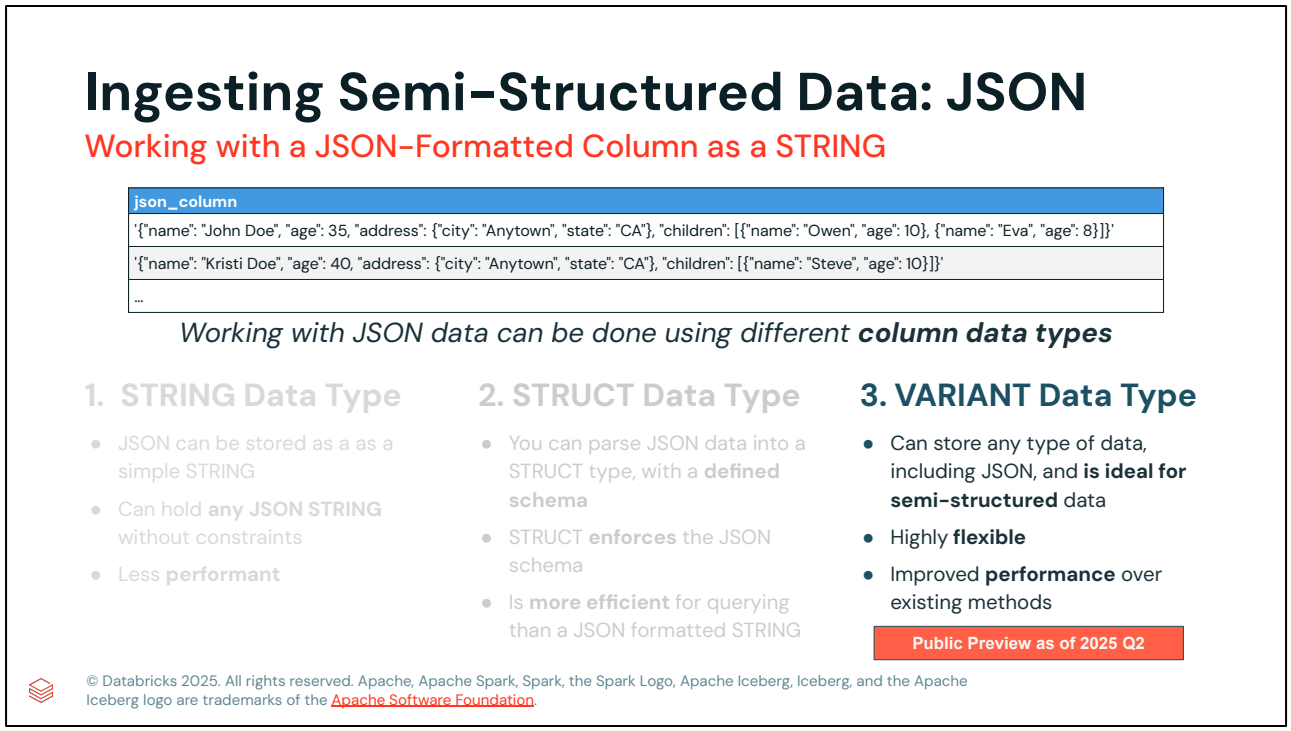


© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Once you have the structure of the JSON-formatted string, you can use the Spark from_json function.

This function takes the JSON string and the specified schema you obtained in the previous step, and returns a STRUCT column.

Using from_json will create a new column with the STRUCT data type, containing the parsed JSON data according to the defined schema.



Lastly, you can use the new VARIANT column data type.

As of 2025 Q2, VARIANT is in public preview, so keep an eye out for its General Availability (GA) release!

Some major benefits of the VARIANT data type include:

- It can store any type of data, including JSON, making it ideal for semi-structured data.
- It is highly flexible, adapting to different data shapes without rigid schemas.
- It offers improved performance compared to existing methods for handling semi-structured data.

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the Apache Software Foundation.

Thank you for completing this lesson and continuing your journey to develop your skills with us.