Using Cloud Data Warehousing to Analyze Structured and Semi-Structured data sets

Kevin Bair
Solution Architect
Kevin.Bair@snowflake.net
Topics this presentation will cover

1. Who are we?
2. Market Trends and Complexity
3. Cloud Data Warehousing
4. Demo
5. Semi-Structured Discussion
6. Demo - Processing JSON via SQL
**Introducing Snowflake: An experienced team of data experts with a vision to reinvent the data warehouse**

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bob Muglia</strong></td>
<td>CEO</td>
<td>Former President of Microsoft’s Server and Tools Business</td>
</tr>
<tr>
<td><strong>Benoit Dageville, PhD</strong></td>
<td>CTO &amp; Founder</td>
<td>Lead architect of Oracle parallel execution and a key manageability architect</td>
</tr>
<tr>
<td><strong>Marcin Zukowski, PhD</strong></td>
<td>Founder &amp; VP of Engineering</td>
<td>Inventor of vectorized query execution in databases</td>
</tr>
<tr>
<td><strong>Thierry Cruanes, PhD</strong></td>
<td>Founder Architect</td>
<td>Leading expert in query optimization and parallel execution at Oracle</td>
</tr>
</tbody>
</table>

---

**Team**

- [Oracle](https://oracle.com)
- [salesforce.com](https://salesforce.com)
- [aster data](https://asterdata.com)
- [Elastic](https://elastic.co)
- [vectorwise](https://vectorwise.com)
- [Twitter](https://twitter.com)

---

**Investors**

- [Redpoint](https://redpoint.com)
- [Sutter Hill Ventures](https://sutterhillventures.com)
- [Wing](https://wing.com)
Today’s data: big, complex, moving to cloud

- 25% Surge in cloud spending and supporting technology (IDC)
- $100B
- 2/3 Of workloads will be processed in cloud data centers (Cisco)
- 73% in 2016
- 66% in 2014
- Data in the cloud today is expected to grow in the next two years. (Gigaom)
Structured data and Semi-Structured data

- Transactional data
- Relational
- Fixed schema
- OLTP / OLAP

- Machine-generated
- Non-relational
- Varying schema
- Most common in cloud environments
Current architectures can’t keep up

Data Warehousing
- Complex: manage hardware, data distribution, indexes, ...
- Limited elasticity: forklift upgrades, data redistribution, downtime
- Costly: overprovisioning, significant care & feeding

Hadoop
- Complex: specialized skills, new tools
- Limited elasticity: data redistribution, resource contention
- Not a data warehouse: batch-oriented, limited optimization, incomplete security
Data Pipeline / Data Lake Architecture - “ETL”

- **Source**
  - Website Logs
  - Operational Systems
  - External Providers
  - Stream Data

- **Stage**
  - S3 • 10TB

- **Data Lake**
  - Hadoop • 30 TB

- **Stage**
  - S3 • 5 TB
  - Summary

- **EDW**
  - MPP • 10 TB Disk

10 TB = 45 TB
One System for all Business Data

Structured Storage

HDFS

Map-Reduce Jobs

Data Sink

Structured

Relational

Databases

Structured data

Apple
Pear
Orange

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Semi-structured data

Other Systems

✓ Multiple Systems
✓ Specialized Skillset
✓ Slower/More Costly Data Conversion

Other Systems

✓ Multiple Systems
✓ Specialized Skillset
✓ Slower/More Costly Data Conversion

Snowflake

✓ One System
✓ One Common Skillset
✓ Faster/Less Costly Data Conversion
✓ For both Structured and Semi-Structured Business Data

Structured data

{ "firstName": "John", "lastName": "Smith", "height_cm": 167.64, "address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": "10021-3100" } }

Semi-structured data

Apple 101.12 250 FIH-2316
Pear 56.22 202 IHO-6912
Orange 98.21 600 WHQ-6090
Data Pipeline / Snowflake Architecture - “ELT”

Source
- Website Logs
- Operational Systems
- External Providers
- Stream Data

Stage
- S3
  - 10TB

EDW
- Snowflake
  - 2 TB Disk

10 TB = 2+ TB
Cloud Architecture

- Software / SaaS
- Platform / PaaS
- Infrastructure (Virtualized) / IaaS
- Physical (Compute, Storage, Network)

Public Cloud
Moving to the Cloud

IaaS

1. Size your cluster
2. Networking
3. Security
4. Load your Software
5. Unload your data
6. Stage
7. Reload
Done! But wait.....

Data Center

Public Cloud
## Management responsibilities & TCO

<table>
<thead>
<tr>
<th>Area</th>
<th>Responsibility</th>
<th>On Premises/IaaS Data Warehouse</th>
<th>PaaS Data Warehouse</th>
<th>SaaS Data Warehouse (Snowflake)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware infrastructure (facilities, server, storage, network, etc.)</strong></td>
<td>Datacenter management</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Hardware selection &amp; configuration</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Hardware management</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Upgrades</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Scaling</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td><strong>Software infrastructure</strong></td>
<td>Operating system</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Database software (install, management, upgrades)</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td><strong>Data loading &amp; management</strong></td>
<td>Data distribution</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>File management</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Data protection (storage availability, backup, etc.)</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td><strong>Data warehouse management &amp; optimization</strong></td>
<td>Indexing</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Sort keys</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Metadata updates</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Query optimization</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Physical security</td>
<td>Customer</td>
<td>Vendor</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Deployment security</td>
<td>Customer</td>
<td>Customer</td>
<td>Snowflake</td>
</tr>
<tr>
<td></td>
<td>Database security</td>
<td>Customer</td>
<td>Customer</td>
<td>Customer</td>
</tr>
<tr>
<td><strong>Human Capital</strong></td>
<td>Percent of Systems Administrators time</td>
<td>100%</td>
<td>26%</td>
<td>0.03%</td>
</tr>
<tr>
<td></td>
<td>Percent of DBA time</td>
<td>100%</td>
<td>26%</td>
<td>0.03%</td>
</tr>
</tbody>
</table>
Characteristics of a Cloud Data Warehouse

All new code, built “in” and “for” the cloud, fully SQL-compatible relational database

- **Data warehouse as a service**
  - No infrastructure to manage, no knobs to tune

- **Multidimensional elasticity**
  - On-demand scalability of data, workloads, and users

- **All business data**
  - Native support for structured + semi-structured data

- **Best Price / Performance**
  - Twice the performance at the same cost, monetize your data
Data Warehousing Cloud Service

Database is separate from Virtual Warehouse

One Virtual Warehouse, multiple Databases

One Database, multiple Virtual Warehouses

Virtual Warehouse scales independently from Database

Data loading does not impact query performance
Data Warehousing Cloud Service

Supports structured and semi-structured data: JSON and Avro

The tools you know + Snowflake web UI
# Snowflake Architecture

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Cloud Services</th>
<th>Virtual Warehouse Processing</th>
<th>Database Storage</th>
<th>Cloud Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC Driver</td>
<td>Optimization</td>
<td>Customer Service</td>
<td>Snowflake Data</td>
<td>Amazon AWS</td>
</tr>
<tr>
<td>JDBC Driver</td>
<td>Query Mgmt</td>
<td>Financial Analysts</td>
<td>Sales</td>
<td></td>
</tr>
<tr>
<td>Web UI</td>
<td>Warehouse Mgmt</td>
<td>Quality Control</td>
<td>Marketing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td>Loading</td>
<td>Materials</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metadata</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Snowflake Architecture Components**

- **User Interface**: ODBC Driver, JDBC Driver, Web UI
- **Cloud Services**: Optimization, Query Mgmt, Warehouse Mgmt, Security, Metadata
- **Virtual Warehouse Processing**: Customer Service, Financial Analysts, Quality Control, Loading
- **Database Storage**: Snowflake Data, Sales, Marketing, Materials
- **Cloud Infrastructure**: Amazon AWS
Snowflake Architecture

Compute
EC2

Financial Analysts

User Interface
ODBC Driver  JDBC Driver  Web UI

Cloud Services
Optimization  Query Mgmt  Warehouse Mgmt  Security

Cluster

Storage
S3

Compute
EC2

Financial Analysts

User Interface
ODBC Driver  JDBC Driver  Web UI

Cloud Services
Optimization  Query Mgmt  Warehouse Mgmt  Security

Cluster

Storage
S3
Snowflake High Availability Architecture

- Load Balancer
- Cloud Services
- Fully Replicated Metadata
- Virtual Warehouses
- Fully Replicated Database Storage

SQL
REST

Availability zone 1
Availability zone 2
Availability zone 3
Enterprise-class data warehouse: Security

**Authentication**
- Embedded multi-factor authentication server
- Federated authentication via SAML 2.0 (*in development*)

**Access control**
- Role-based access control model
- Granular privileges on objects & actions

**Data encryption**
- Encryption at rest for database data
- Encryption of Snowflake metadata
- Snowflake-managed keys

*Controls & processes validated through SOC certification & audit*
Demo Time!
Scenario

- Peter – DBA
  - Create Warehouse
  - Load Data
  - Clone

- Marie – Analyst
  - Campaign Analysis combining structure and unstructured
  - BI tools view
Break!
What does Semi Structured mean?

- Data that may be of any type
- Data that is incomplete
- Structure that can rapidly and unpredictably change
- Usually Self Describing

Examples
- XML
- AVRO
- JSON
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
    <calories>900</calories>
  </food>
</breakfast_menu>
JSON Example

```json
{
    "custkey": "450002",
    "useragent": {
        "devicetype": "pc",
        "experience": "browser",
        "platform": "windows"
    },
    "pagetype": "home",
    "productline": "none",
    "customerprofile": {
        "age": 20,
        "gender": "male",
        "customerinterests": [
            "movies",
            "fashion",
            "music"
        ]
    }
}
```
Avro Example
Why is this so hard for a traditional Relational DBMS?

• Pre-defined Schema

• Store in Character Large Object (CLOB) data type

• Inefficient to Query
Relational Processing of Semi-Structured Data

1. Variant data type compresses storage of semi-structured data
2. Data is analyzed during load to discern repetitive attributes within the hierarchy
3. Repetitive attributes are columnar compressed and statistics are collected for relational query optimization
4. SQL extensions enable relational queries against both semi-structured and structured data
Why Support Semi-structured data via SQL?

• Integrate with existing data
• Reduced administrative costs
• Improved security
• Transaction management
• Better performance
• Better resource allocation
• Increased developer productivity
• SQL is proven model for performing queries, especially joins.
What makes Snowflake unique for handling Semi Structured Data?

- Compression
- Encryption / Role Based Authentication
- Shredding
- History/Results
- Clone
- Time Travel
- Flatten
- Regexp
- No Contention
- No Tuning
Functions

- REGEXP
- REGEXP_COUNT
- REGEXP_INSTR
- REGEXP_LIKE
- REGEXP_REPLACE
- REGEXP_SUBSTR
- RLIKE
  - CHECK_JSON
  - PARSE_JSON
  - OBJECT_CONSTRUCT
  - OBJECT_INSERT
  - GET
  - GET_PATH
  - AS_type
  - IS_type
  - IS_NULL_VALUE
  - TO_JSON
  - TYPEOF

- ARRAYAGG, ARRAY_AGG
- ARRAY_APPEND
- ARRAY_CAT
- ARRAY_COMPACT
- ARRAY_SIZE
- ARRAY_CONSTRUCT
- ARRAY_CONSTRUCT_COMPACT
- ARRAY_INSERT
- ARRAY_PREPEND
- ARRAY_SLICE
- ARRAY_TO_STRING

snowflake
Scenario

• User – Me
  • Receiving Clickstream data from a Website in JSON format
  • “ELT”
    • Load Semi-Structured, JSON data
    • Load/Transform into Relational Tables
  • Receive new data
    • Clone existing data set, update/insert new data
    • Merge with existing
JSON Example

```json
{
    "custkey": "450002",
    "useragent": {
        "devicetype": "pc",
        "experience": "browser",
        "platform": "windows"
    },
    "pagetype": "home",
    "productline": "none",
    "customerprofile": {
        "age": 20,
        "gender": "male",
        "customerinterests": [
            "movies",
            "fashion",
            "music"
        ]
    }
}
```
Parsing JSON using Snowflake SQL

(After loading JSON file into Snowflake table)

```sql
SELECT 'The First Person is '|| fullrow:fullName || ' He is '|| fullrow:age || ' years of age. '|| ' His children are: '|| fullrow:children[0].name || ' Who is a '|| fullrow:children[0].gender || ' and is '|| fullrow:children[0].age || ' year(s) old '
   || fullrow:children[1].name || ' Who is a '|| fullrow:children[1].gender || ' and is '|| fullrow:children[1].age || ' year(s) old ' Result
FROM json_data_table
WHERE fullrow:fullName = 'John Doe';
```
FLATTEN() Function and its Pseudo-columns

FLATTEN() Converts a repeated field into a set of rows. FLATTEN() Returns Pseudo-columns in addition to the data result.

SELECT S.fullrow:fullName, t.value:name, t.value:age, t.SEQ, t.KEY, t.PATH, t.INDEX, t.VALUE
FROM json_data_table AS S, TABLE(FLATTEN(S.fullrow, 'children')) t;

<table>
<thead>
<tr>
<th>row#</th>
<th>S.fullrow:fullName</th>
<th>t.value:name</th>
<th>t.value:age</th>
<th>SEQ</th>
<th>KEY</th>
<th>PATH</th>
<th>INDEX</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Doe</td>
<td>Jane</td>
<td>6</td>
<td>1</td>
<td>NULL</td>
<td>children[0]</td>
<td>0</td>
<td>{ &quot;age&quot;: &quot;6&quot;, &quot;gender&quot;: &quot;Female&quot;, &quot;name&quot;: &quot;Jane&quot; }</td>
</tr>
<tr>
<td>2</td>
<td>John Doe</td>
<td>John</td>
<td>15</td>
<td>1</td>
<td>NULL</td>
<td>children[1]</td>
<td>1</td>
<td>{ &quot;age&quot;: &quot;15&quot;, &quot;gender&quot;: &quot;Male&quot;, &quot;name&quot;: &quot;John&quot; }</td>
</tr>
<tr>
<td>3</td>
<td>Mike Jones</td>
<td>Earl</td>
<td>10</td>
<td>2</td>
<td>NULL</td>
<td>children[0]</td>
<td>0</td>
<td>{ &quot;age&quot;: &quot;10&quot;, &quot;gender&quot;: &quot;Male&quot;, &quot;name&quot;: &quot;Earl&quot; }</td>
</tr>
<tr>
<td>4</td>
<td>Mike Jones</td>
<td>Sam</td>
<td>6</td>
<td>2</td>
<td>NULL</td>
<td>children[1]</td>
<td>1</td>
<td>{ &quot;age&quot;: &quot;6&quot;, &quot;gender&quot;: &quot;Male&quot;, &quot;name&quot;: &quot;Sam&quot; }</td>
</tr>
<tr>
<td>5</td>
<td>Mike Jones</td>
<td>Kit</td>
<td>8</td>
<td>2</td>
<td>NULL</td>
<td>children[2]</td>
<td>2</td>
<td>{ &quot;age&quot;: &quot;8&quot;, &quot;gender&quot;: &quot;Male&quot;, &quot;name&quot;: &quot;Kit&quot; }</td>
</tr>
</tbody>
</table>
FLATTEN() in Snowflake SQL
(Removing one level of nesting)

SELECT S.fullrow:fullName, t.value:name, t.value:age
FROM json_data_table as S,
    TABLE(FLATTEN(S.fullrow,'children')) t
WHERE s.fullrow:fullName = 'Mike Jones'
AND t.value:age::integer > 6 ;
FLATTEN(): Two levels of un-nesting

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>gender</th>
<th>citiesLived.place</th>
<th>citiesLived.yearsLived</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1989</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1993</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>2002</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>1990</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>1993</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>2008</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>1993</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>2003</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>2005</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Austin</td>
<td>1973</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Austin</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Austin</td>
<td>2001</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Austin</td>
<td>2005</td>
</tr>
</tbody>
</table>
FLATTEN() in Snowflake SQL
(Removing two levels of nesting)

Getting cities Mike Jones lived and when

TABLE (Snowflake syntax)

```
SELECT
  p.fullrow:fullName::varchar as name,
  p.fullrow:age::int as age,
  p.fullrow:gender::varchar as gender,
  cl.value:place::varchar as city,
  yl.value::int as year
FROM json_data_table p,
TABLE (FLATTEN(p.fullrow,'citiesLived')) cl,
TABLE (FLATTEN(cl.value:yearsLived,'')) yl
WHERE name = 'Mike Jones';
```

LATERAL (ANSI syntax, also supported)

```
SELECT
  p.fullrow:fullName::varchar as name,
  p.fullrow:age::int as age,
  p.fullrow:gender::varchar as gender,
  cl.value:place::varchar as city,
  yl.value::int as year
FROM json_data_table p,
LATERAL FLATTEN(p.fullrow,'citiesLived') cl,
LATERAL FLATTEN(cl.value:yearsLived,'') yl
WHERE name = 'Mike Jones';
```

Output

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>gender</th>
<th>citiesLived.place</th>
<th>citiesLived.yearsLived</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1989</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1993</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Los Angeles</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>2002</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>1990</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Washington DC</td>
<td>1993</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>2003</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>1998</td>
</tr>
<tr>
<td>Mike Jones</td>
<td>35</td>
<td>Male</td>
<td>Portland</td>
<td>2005</td>
</tr>
</tbody>
</table>
Parsing JSON using Snowflake SQL (Without loading the JSON file into a Snowflake table)

```sql
SELECT 'The First Person is ' ||
S.$1:fullName || ' He is ' || S.$1:age || ' years of age.' || ' His children are: ' || S.$1:children[0].name || ' Who is a ' || S.$1:children[0].gender || ' and is ' || S.$1:children[0].age || ' year(s) old ' FROM @~/json/json_sample_data (FILE_FORMAT => 'json') as S
WHERE S.$1:fullName = 'John Doe';
```
Parsing JSON Records: PARSE_JSON

Interprets an input string as a JSON document, producing a VARIANT value

```
SELECT  s.fullrow:fullName Parent, c.value Children_Object,
        c.value:name Child_Name, c.value:age Child_Age
FROM     json_data_table AS S,
         TABLE(FLATTEN(S.fullrow,'children')) c
WHERE    PARSE_JSON(c.value:age) > 8;
```
Valid JSON will produce NULL

```sql
SELECT CHECK_JSON('{
     "age": "15",
     "gender": "Male",
     "name": "John"}');
```

<table>
<thead>
<tr>
<th>row#</th>
<th>CHECK_JSON({&quot;AGE&quot;: &quot;15&quot;, &quot;GENDER&quot;: &quot;MALE&quot;, &quot;NAME&quot;: &quot;JOHN&quot;})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Valid JSON

Invalid JSON will produce error message

```sql
SELECT CHECK_JSON('{
     "age": "15",
     "gender": "Male",
     "name": "John"
}');
```

Missing : missing colon, line 3, pos 28

Invalid JSON will produce error message

```sql
SELECT CHECK_JSON('{
     "age": "15",
     "gender": "Male",
     "name": "John"
}');
```

Missing } incomplete object value, line 3, pos 35
Parsing JSON Records: CHECK_JSON

Validate JSON records in the S3 file before loading it. Use SELECT with CSV file format

```sql
SELECT S.$1, CHECK_JSON(S.$1)
FROM @~/json/json_sample_data (FILE_FORMAT => 'CSV') AS S;
```

Validate JSON records in the S3 file before loading it. Use COPY with JSON file format

```
COPY INTO json_data_table
FROM @~/json/json_sample_data.gz
FILE_FORMAT = 'JSON' VALIDATION_MODE = 'RETURN_ERRORS';
```
Demo Time!
Where to Get More Info

• Visit us: http://www.snowflake.net/

• Email us:
  • Sales: sales@snowflake.net
  • General: info@snowflake.net

• Q&A