SQL Server 2016: JSON native support

Alessandro Alpi

@suxstellino

www.alessandroalpi.net

http://speakerscore.com/SQLJSON

Sponsors

Gold Sponsors

Silver Sponsors

Bronze Sponsors

Organizers

Alessandro Alpi | @suxstellino

- Microsoft MVP – SQL Server since 2008
  - Blog ITA: http://blogs.dotnethell.it/suxstellino
  - Blog ENG: http://suxstellino.wordpress.com/
  - Website: http://www.alessandroalpi.net
- CTO and Co-Founder of Engage IT Services S.r.l.
  - www.engageitservices.it
  - Team leader (Agile)
- Communities
  - Getlatestversion.it

Keep in mind

ONLY SQL Server 2016

- CTP 2 (2.4 now)
- CTP 3 (Future)
- Frequent changes
- V1 features

Agenda

JSON format - intro

- Concepts, why JSON?
- JSON vs XML

JSON from a SQL Server perspective

- Options
- Features on CTP2 and 3

Additional concepts

- How to index it
- Restrictions

JSON – concepts

What's JSON

- JavaScript Object Notation Acronym
- Based on JavaScript, independent at the same time
- Common data types: bool, int, real, string, array | null
- String, not markup
- Sample:

```
{
        "first": "Alessandro",
        "last": "Alpi"
        "age": "34",
        "courses": [
             {"code": "C001", "name": "SQL Server Querying"},
             {"code": "C002", "name": "Administering SQL Server"},
             {"code": "C003", "name": "Troubleshooting SQL Server"},
        ]
}
```

JSON – concepts

Why JSON

- Lightweight and simple format
- Data representation
- Readable/Portable/X-platform (it can replace XML)
- Optimal for AJAX/Javascript

Notations

- " for "name" and "values"
- : for "name":"value"
- Special char represented with escape (\r, \t, ecc.)
- { } objects (curly braces)
- [,] arrays (square brackets)
- [{ }, { }] arrays of objects

JSON – concepts

JSON vs XML – format

- Data exchange formats
- JSON is open, like XML, but no “real” rules (consortium)
- JSON doesn’t need DTD
- JSON doesn’t need extension, it’s not markup
- JSON represents data, XML is for documents

JSON – concepts

JSON vs XML - performances & productivity

- JSON is built-in (some languages syntax)
- Many serializers (like JSON.Net)
- Less resources intensive when (de)serializing
- JSON is “smaller”, quick to transfer
- JSON doesn’t “carry” media (XML CDATA)

DEMO

JSON & XML: objects

Native JSON format support

SQL Server 2016

- Supported from the CTP 2 (also In-Memmory OLTP)
- Before 2016, complex and CPU intensive t-sql

No storage changes

- It's not a data type like XML
- It's a string (nvarchar, varchar)
- Native parser right after the query execution
- For additional features, please use CONNECT ☺

JSON Conversions

Conversions

- nvarchar, varchar, nchar, char -> string
- int, bigint, float, decimal, numeric -> number
- bit -> Boolean (true, false)
- datetime, date, datetime2, time, datetimeoffset -> string
- Uniqueidentifier, money, binary -> string and BASE64 string
- CLR not supported (except some type, like **`hierarchyid`**)

Export features

"FOR JSON" Clause

- PATH
  - **FOR JSON PATH**
- AUTO
  - **FOR JSON AUTO**

Utilities

- ROOT
  - **FOR JSON AUTO, ROOT('info')**
- INCLUDE_NULL_VALUES
  - **FOR JSON AUTO, INCLUDE_NULL_VALUES**

Number	Date	Customer	Price	Quantity
SO43659	2011-05-31T00:00:00	AW29825	59.99	1
SO43661	2011-06-01T00:00:00	AW73565	24.99	3


```
[
   {
      "Number":"SO43659",
      "Date":"2011-05-31T00:00:00"
      "AccountNumber":"AW29825",
      "Price":59.99,
      "Quantity":1
    },
    {
      "Number":"SO43661",
       "Date":"2011-06-01T00:00:00“
        "AccountNumber":"AW73565“,
        "Price":24.99,
         "Quantity":3
    }
]
```

FOR JSON PATH

Results

- Without FROM: single JSON objects
- With FROM: array of JSONobjects
- Each column is a JSON "property"

Nesting

- Alias with "." as separator (for nesting depth)
- Subquery for sub-JSON

FOR JSON AUTO

Results

- Render based on column/tables order
  - ONLY with FROM clause
- STRICT Render (cannot be modified)
- Each column is a JSON “property”

Nesting

- With JOIN, the “left table” is the root, the “right” one is nested
- Subquery

JSON output in .net

Use the StringBuilder() .net object

```
var cmd = new SqlCommand(queryWithForJson, conn);
conn.Open();
var jsonResult = new StringBuilder();
var reader = cmd.ExecuteReader();
if (!reader.HasRows())
 jsonResult.Append("[]");
else
{
    while reader.Read()
    {
        jsonResult.Append(reader.GetValue(0).ToString());
    }
}
```

Empty JSON

Append rows

DEMO

Export features

Import features

Import and transform

- TVF OPENJSON()
  - Filter param
  - Syntax JS ($.Collection.Property)
  - … **FROM OPENJSON (@JSalestOrderDetails, '$.OrdersArray') WITH (definition);**

Why?

- Load in temp table
- Analyze columns

```
[
   {
      "Number":"SO43659",
      "Date":"2011-05-31T00:00:00"
      "AccountNumber":"AW29825",
      "Price":59.99,
      "Quantity":1
    },…]
```

SELECT * FROM **OPENJSON**(@json)

Number	Date	Customer	Price	Quantity
SO43659	2011-05-31T00:00:00	AW29825	59.99	1
SO43661	2011-06-01T00:00:00	AW73565	24.99	3

Import features

Additional built-in functions

- Validation: ISJSON(json_text)
  - Important for CHECK CONSTRAINT
  - … `WHERE ISJSON(tab.JCol) > 0`
- Querying: JSON_VALUE(json_text, path)
  - Gets a scalar value from the JSON (can be filtered by path)
  - … `AND JSON_VALUE(tab.JCol,'$.Order.Type') = 'C'`

Path param syntax

Based on JavaScript

- $ (the whole JSON text)
- $.prop1 (JSON property)
- $[n] (n-th element on the JSON array)
- $.prop1.prop2.array1[n].prop3.array[2].prop4 (complex traversing/navigation)

DEMO

Import features

Where?

Record extensions

- Common column table with JSON "extension"
  - Example: "log" info

Serialization

- Output format for services
- X-platform support (lightweight and simple)
- Possible SSIS integration

Worst practices

JSON everywhere

- Too much key-value-store based table (key + JSON)
  - SQL Server is RDBMS, not Key Value Store (like Redis)
- Store too JSON data
  - Can be heavy for the CPU (the parser is well optimized, however.. ☺)
- All queries with FOR JSON clause
  - JSON is a useful utility, not a rule
  - SQL Server is not an application server

Indexing

No storage design

- It's just a string
- Indexing in string -> index on varchar, nvarchar
- Full-text (obviously) supported

Indexing strategies

- JSON_VALUE can be used for a computed column (+ index)
- Computed columns with JSON_VALUE can be:
  - In the index key
  - In the INCLUDE of the index (leaf)

Indexing – sample

JSON_VALUE + Computed columns

```
CREATE TABLE SalesOrderRecord (
    Id int PRIMARY KEY IDENTITY,
    OrderNumber NVARCHAR(25) NOT NULL,
    OrderDate DATETIME NOT NULL,
    JOrderDetails NVARCHAR(4000),
    Quantity AS CAST(JSON_VALUE(JOrderDetails, '$.Order.Qty') AS
int),
    Price AS CAST(JSON_VALUE(JOrderDetails, '$.Order.Price') AS
decimal(18, 2)
)

CREATE INDEX idxJson ON SalesOrderRecord(Quantity) INCLUDE (Price);
```

Restrictions FOR JSON clause

Data type support

- No CLR (except some, like HIERARCHYID)

Statement

- No SELECT INTO

Rules

- Column aliases always for non-named values
- A table must exist for parsing (FOR JSON AUTO)

DEMO

Unsupported queries

Conclusions

Reducing the gap with competitors (PostgreSQL)

Optimized integrated parser (avoid CLR custom)

Poor additional t-sql (parameters and built-in functions)

V1 features

Future storage changes? Index structures?

Biz logic -> application, not everything on SQL Server

Resources

FOR JSON PATH: https://msdn.microsoft.com/en-us/library/dn921877.aspx

FOR JSON AUTO: https://msdn.microsoft.com/en-us/library/dn921883.aspx

INCLUDE_NULL_VALUES: https://msdn.microsoft.com/en-us/library/dn921878.aspx

ROOT: https://msdn.microsoft.com/en-us/library/dn921894.aspx

POST Jovan Popovic: http://blogs.msdn.com/b/jocapc/archive/2015/05/16/json-support-in-sql-server-2016.aspx

POST Aaron Bertrand: http://blogs.sqlsentry.com/aaronbertrand/sql-server-2016-json-support/

JSON Online Viewer: http://jsonviewer.stack.hu/

JSON Online Query Tool: http://www.jsonquerytool.com/

JSON Online Tool: https://www.jsonselect.com/

Q&A

Questions?

Evaluations

- Don't forget to compile evaluations form here
  - http://speakerscore.com/sqlsat454
  - This session: http://speakerscore.com/SQLJSON

#sqlsat454

THANKS!

HTTP://SPEAKERSCORE.COM/SQLJSON