# SQL 2016 and Json

SQL2016-02 Foundation spends a lot of pages on JSON, but JSON is not an SQL data type, and so you cannot declare a variable of type JSON or a routine that returns JSON. You can use FORMAT and JSON constructors to create JSON values from SQL string values, and you can use path-expressions to extract SQL values from JSONValues. The resulting arms-length attitude results in some anomalies. I will be interested if someone can find I have missed something that could make things easier.

## JSON is defined in RFC 7159

Object = ‘{‘ [ Member { ‘,’ Member } ] ‘}’.

Value can be an object, array, number or string or one of the three literal values (lower-case only) false null true . Non-zero numbers cannot have leading zeroes. e and E can be used to introduce an exponent.

Array = ‘[‘ [ Value { ‘,’ Value } ] ‘]’ .

Member = String ‘:’ Value .

“An object whose names are all unique is interoperable in the sense that all software implementations receiving that object will agree on the name-value mappings. When the names within an object are not unique, the behavior of software that receives such an object is unpredictable. Many implementations report the last name/value pair only. Other implementations report an error or fail to parse the object, and some implementations report all of the name/value pairs, including duplicates.

JSON parsing libraries have been observed to differ as to whether or not they make the ordering of object members visible to calling software. Implementations whose behavior does not depend on member ordering will be interoperable in the sense that they will not be affected by these differences.”

Strings must be delimited by double-quotes.

## 10.12 <JSON value expression> and FORMAT

A JSONValue parses an SQL value to obtain a JSON object.

JSONValue = *Char\_*Value [FORMAT (JSON|..) ] .

For example

'true'

'236'

'"Fred"'

'{ "a": 23 , "b" : null , "c" : "Fred" }'

The FORMAT specification is not required if the context requires a JSON Value (e.g. within a JSONValueConstructor). The ellipsis in the optional FORMAT specification indicates that other JSON representation options can be implementation-defined. Note that to get the string JSONValue we needed two levels of quotes. I imagine we could allow some laxity here.

## 6.33 <JSON value constructor>

JSONValueConstructor allows a JSON object to be built from JSONValues

JSONValueConstructor = JSONObjectConstructor | JSONArrayConstructor .

JSONObjectConstructor = JSON\_VALUE ‘(‘

[ JSONNameAndValue { ‘,’ JSONNameAndValue }

[(NULL|ABSENT) ON NULL]

[(WITH|WITHOUT) UNIQUE [KEYS]]]

[RETURNING Type [FORMAT (JSON|..)]] ‘)’.

The FORMAT clause can be omitted here.

JSONNameAndValue = [KEY] JSONName VALUE JSONValue

| JSONName ‘:’ JSONValue .

Obviously the second form here is easier.

JSONName = *Char\_*Value .

This is ugly because in SQL all character literals are enclosed in single quotes: identifiers would be better.

For example:

JSON\_VALUE ( 'a' : 23 , 'b' **:** 'null' , 'c' : 'Fred' )

We could decide to allow { *content* } as an alternative to JSON\_VALUE( *content* ) , but then this would require single quotes as in

{ 'a' : 23 , 'b': 'null' , 'c' : 'Fred' }

and the differences between this version and the 10.12 version above will be hard to explain to students.

## 6.28 <value-expression>

Unchanged from SQL2011 except that JSONValueFunction is allowed as a ValueSpecification in sec 6.3. The JSONValueFunction extracts an SQL value of a predefined type from a JSON value:

JSONValueFunction = JSON\_VALUE ‘(‘*Context\_*JSONValue ‘,’ *Path\_*string

[AS id] [PASSING JSONValue AS id {‘,’ JSONValue AS id }]

[RETURNING Type] [JSONValueBehavior ON EMPTY ]

[JSONValueBehavior ON ERROR] ‘)’ .

JSONValueBehavior = ERROR | NULL | DEFAULT Value .

For example, if D is the JSONValue defined above, we can extract the integer value 23 with the Value expression

JSON\_VALUE(D,'a' RETURNING INT)

Note that this expression is a <value-expression> (an INT ) and not a <JSON value expression> or JSONValue, despite the similar-looking use of the JSON\_VALUE reserved word.

The interaction between context and path is defined in section 9.39 SQL.JSON Path Lenguage: syntax and semantics. As one might expect, there are many options.