XML vs JSON - Battle Royale

@RileyMajor

SQL Community

- PASS
- Twitter
  - #SQLSatMadison
  - #tsql2sday
  - @RileyMajor
- Blogs
  - http://TSQLTuesday.com/
  - http://SQLServerCentral.com/
  - http://LessThanDot.com/
  - http://DBA.StackExchange.com/
  - http://blogs.SentryOne.com/
  - http://scribnasium.com/
- Minnesota
  - PASSMN – Twin Cities
    - @PASSMN
    - http://MNSSUG.org
- Wisconsin
  - FoxPASS - Appleton, WI
  - MADPASS - Madison, WI
  - Microsoft BI Professionals - Wisconsin: Greendale, WI
  - Western Wisconsin PASS - Eau Claire, WI
  - WausauPASS - Wausau, WI
  - WI SSUG - Waukesha, WI

Background

XML

- eXtensible Markup Language
- Introduced in 1998.
- Derived from SGML (parent of HTML) by W3C.
- Human & Machine Readable
- Elements and Attributes
- T-SQL Support in 2000

JSON

- JavaScript Object Notation
- Hints in 1996. More like 2002. RFC 4627 in 2006.
- Formalized by ECMA (makers of JavaScript) in 2013.
- Human & Machine Readable
- Name/Value Pairs.
- T-SQL Support in 2016

Basic Structure

XML

```
<?xml version="1.0"?>
<element
   attribute="value"
>
   Character Data
</element>
```

JSON

```
{
   "name":"value"
}
```

Arrays

XML

```
<array>
   <item>data</item>
   <item>data</item>
</array>
```

JSON

```
{
   "array":
   [
      "data",
      "data"
   ]
}
```

Nesting

XML

```
<Level1>
   <Level2>
      <Level3>
         Data
      </Level3>
   </Level2>
</Level1>
```

JSON

```
{
"Level1":
   {
   "Level2":
      {
      "Level3":
         "Data"
      }
   }
}
```

Data Types

XML

- Natively, none.
- With Schemas:
  - String
  - Boolean
  - Decimal
  - dateTime
  - anyURI
  - …more…

JSON

- Strings (quotes)
- Numeric (no quotes; scientific notation supported)
- Boolean (true, false)
- null

Special Characters

XML

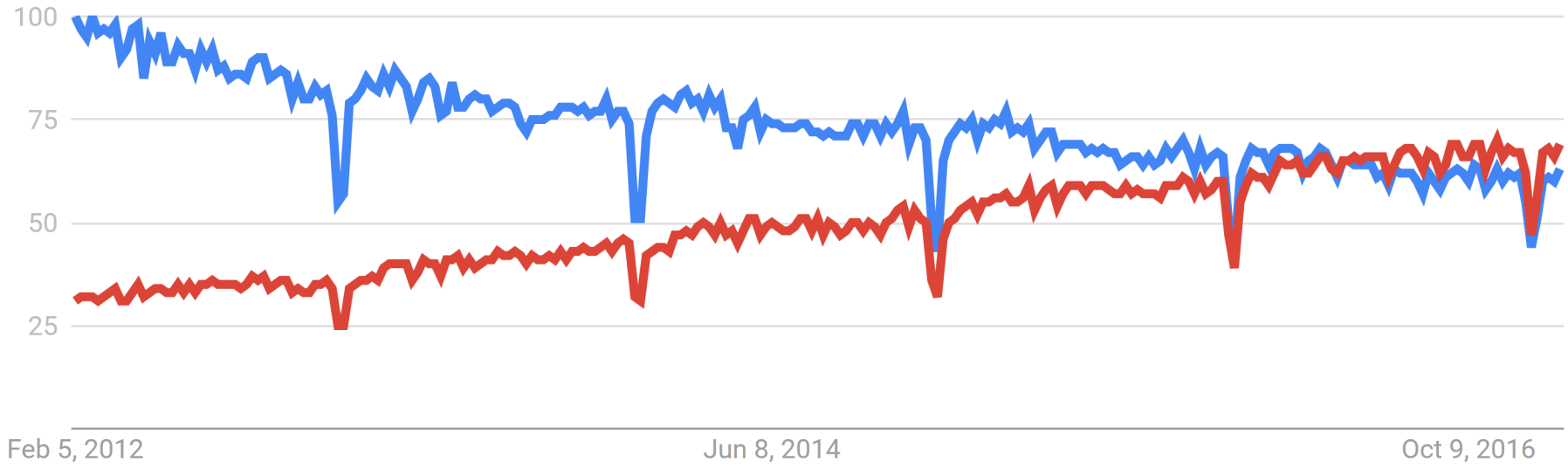
- Elements should be letters and numbers, with no spaces. Can use:
  - . - _ :
- In data and attributes, must encode:
  - < as <
  - & as &
- Encode chosen quotes in attributes.
- Control characters (except CR LF TAB) are not allowed.

JSON

- Keys and string data must be quote (") encapsulated.
- Quotes ("), "reverse solidus" aka backslash (\), and control characters (up through code 31, even tabs).
- Encode using backslash and unicode code point or shortcut (\r\n).

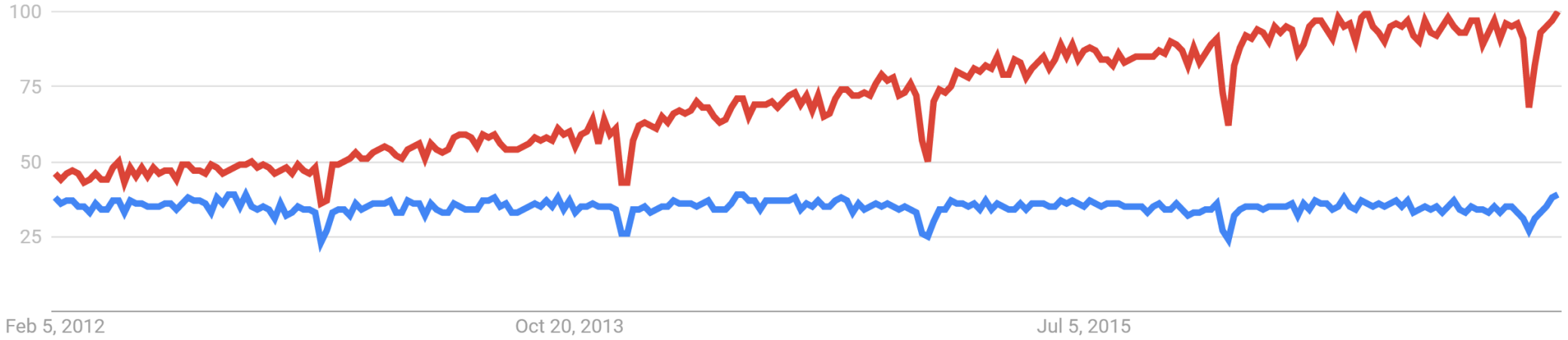
Search Trends

- XML (Blue) vs JSON (Red)



Web Ecosystem

- The world wide web loves JSON.
- SOAP (Blue): a complex XML-based API method.
- REST (Red): a simpler API method, usually using JSON.



Microsoft Ecosystem

XML

- SQL Server Query Plans
- SQL Server Extended Events
- BIML
- SSIS Packages & Configuration
- SSRS Configuration
- SSAS XMLA
- PowerBI Configuration
- Office File Formats
- SQLSaturday.com Data
- XAML

JSON

- TypeScript Configuration
- SSAS Tabular 2016 (TMSL)
- Visual Studio Team Services
- Various REST web services.

SQL Server Support

XML

- SQL Server 2000
  - FOR XML
  - OPENXML
- SQL Server 2005
  - XML Data Type
  - XML Indexing
  - XML Schema Processing
  - XML FLWOR Support
  - Functions: query, value, exist, nodes, modify

JSON

- SQL Server 2016
  - FOR JSON
  - OPENJSON
  - Functions: ISJON, JSON_VALUE, JSON_QUERY, JSON_MODIFY
- Differences
  - No “prepare document” step for OPENJSON
  - No “nodes” function.

XML vs JSON - Sample Data

```
DECLARE @Orders TABLE
(
        OrderID bigint IDENTITY,
        OrderDate datetime
);
```

```
DECLARE @OrderDetails TABLE
(
        OrderDetailsID bigint IDENTITY,
        OrderID bigint,
        ProductID varchar(50),
        Qty int
);
```

OrderID	OrderDate	ProductID	Qty
1	2015-10-10	Bike	2
1	2015-10-10	Helmet	2
1	2015-10-10	Wheels	4
2	2015-10-09	Ball	10

XML vs JSON – Creation (Path)

XML

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         XML PATH;
```

JSON

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         JSON PATH;
```

XML vs JSON – Creation (Path)

XML

```
<row>
    <OrderID>1</OrderID>
    <OrderDate>2015-10-10T00:00:00
        </OrderDate>
    <ProductID>Bike</ProductID>
    <Qty>2</Qty>
</row>
<row>
    <OrderID>1</OrderID>
    <OrderDate>2015-10-10T00:00:00
        </OrderDate>
    <ProductID>Helmet</ProductID>
    <Qty>2</Qty>
</row>…
```

JSON

```
[{
    "OrderID":1,
    "OrderDate":"2015-10-10T00:00:00",
    "ProductID":"Bike",
    "Qty":2
},
{
    "OrderID":1,
    "OrderDate":"2015-10-10T00:00:00",
    "ProductID":"Helmet",
    "Qty":2
}…]
```

XML vs JSON – Creation (Auto)

XML

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         XML AUTO;
```

JSON

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         JSON AUTO;
```

XML vs JSON – Creation (Auto)

XML

```
<Orders
    OrderID="1"
    OrderDate="2015-10-10T00:00:00"
>
    <OrderDetails
        ProductID="Bike" Qty="2" />
    <OrderDetails
        ProductID="Helmet" Qty="2" />
    <OrderDetails
        ProductID="Wheels" Qty="4" />
</Orders>…
```

JSON

```
[{
"OrderID":1,
"OrderDate":"2015-10-10T00:00:00",
"OrderDetails":[
    {"ProductID":"Bike","Qty":2},
    {"ProductID":"Helmet","Qty":2},
    {"ProductID":"Wheels","Qty":4}]
},…
]
```

XML vs JSON - Get Values

XML

```
DECLARE
    @x xml = '<x>y</x>';

SELECT
    @x.value
    (
        '(/x/text())[1]',
        'varchar(50)'
    );
```

JSON

```
DECLARE
    @j varchar(50) = '{"x":"y"}';

SELECT
    JSON_VALUE
    (
        @j,
        '$.x'
    );
```

XML vs JSON - Getting Subsets

XML

```
DECLARE
    @x xml =
    '<x><y>z</y></x>';

SELECT
    @x.query ('(/x/y)');
```

Result:

```
<y>z</y>
```

JSON

```
DECLARE
    @j varchar(50) =
    '{"x":{"y":"z"}';

SELECT
    JSON_QUERY (@j,'$.x');
```

Result:

```
{"y":"z"}
```

XML vs JSON – Getting Rows

- XML has OPENXML and nodes function. Both support XQuery.
- OPENXML
  - Requires “prepare document” step.
  - Separate T-SQL Statement– can’t be used in views or inline functions.
  - Might be faster for repeat access.
  - You have to remove the document from memory manually.
- Nodes
  - Can be used as part of T-SQL statement.
- OPENJSON works like nodes, but without the XQuery.

OPENXML - Query

```
DECLARE @i int, @x xml =
'<x>
  <Element attribute="Attribute Value">
      Element Value
  </Element>
  <y><z>Hello</z></y>
</x>';
EXEC sp_xml_preparedocument @i OUTPUT, @x;
SELECT * FROM OPENXML (@i,'/');
```

OPENXML - Results

id	parentid	nodetype	localname	prefix	namespaceuri	datatype	prev	text
0	NULL	1	x	NULL	NULL	NULL	NULL	NULL
2	0	1	Element	NULL	NULL	NULL	NULL	NULL
3	2	2	attribute	NULL	NULL	NULL	NULL	NULL
7	3	3	#text	NULL	NULL	NULL	NULL	Attribute Value
4	2	3	#text	NULL	NULL	NULL	NULL	Element Value
5	0	1	y	NULL	NULL	NULL	2	NULL
6	5	1	z	NULL	NULL	NULL	NULL	NULL
8	6	3	#text	NULL	NULL	NULL	NULL	Hello

OPENJSON

```
DECLARE @j varchar(max) =
'{
        "NULL": null,
        "String": "Hello",
        "Number": 123.4E05,
        "Boolean": true,
        "Array":[1,2,3],
        "JSON": {"a":"b"}
}';
SELECT          *
FROM        OPENJSON(@j);
```

key	value	type
NULL	NULL	0
String	Hello	1
Number	1.2E+07	2
Boolean	TRUE	3
Array	[1,2,3]	4
JSON	{"a":"b"}	5

XML vs JSON – Consuming (OPEN*)

OPENXML

```
DECLARE
    @i int, @x xml =
    '<x><a>1</a><a>2</a></x>';

EXEC sp_xml_preparedocument
@i OUTPUT, @x;

SELECT * FROM
        OPENXML (@i, '/x/a', 2)
WITH (a int '.');
```

OPENJSON

```
DECLARE
  @j varchar(max) =
  '{"x":[{"a":1},{"a":2}]}';

SELECT a.value FROM
OPENJSON (@j) AS x
CROSS APPLY OPENJSON (x.
[value]) AS a_array
CROSS APPLY OPENJSON
(a_array.[value]) AS a;
```

XML vs JSON – Consuming (Nodes)

XML Nodes()

```
DECLARE
  @x xml =
  '<x><a>1</a><a>2</a></x>';

SELECT
  a.value('.','int')
FROM @x.nodes('/x/a') AS x(a);
```

OPENJSON

```
DECLARE
  @j varchar(max) =
  '{"x":[{"a":1},{"a":2}]}';

SELECT a.value FROM
OPENJSON (@j) AS x
CROSS APPLY OPENJSON (x.
[value]) AS a_array
CROSS APPLY OPENJSON
(a_array.[value]) AS a;
```

XML vs JSON – Consuming (JSON v JSON)

OPENJSON

```
SELECT a.value
FROM
    OPENJSON (@j) AS x
CROSS APPLY
    OPENJSON
    (x.[value]) AS a_array
CROSS APPLY
    OPENJSON
    (a_array.[value]) AS a;
```

Combo

```
SELECT JSON_VALUE
(a_array.value,'$.a') FROM
(
        SELECT
        JSON_QUERY(@j,'$.x')
AS x
) xtable
CROSS APPLY OPENJSON
(xtable.x) AS a_array;
```

XML vs JSON – Data Type

- XML has a native type, but can be stored as nvarchar or varchar.
- JSON does *not* have a native type. Use nvarchar or varchar.
- Why not?
  - Already being stored as text.
    - But so was XML.
    - And so what? Convert over time. Convert on the fly.
  - Don’t have to update other SQL Server tools.
    - Boo hoo. Ok for now, but convert over time.
  - Client apps can handle native XML but not JSON.
    - Wait, what?
    - And so what if it’s text to the outside world; what about in-database performance?

XML vs JSON – Data Type – Validation

- Without JSON type, can’t use TRY_CONVERT() to validate.
- Use ISJSON() instead.
- Can use in CHECK constraint to ensure text field has valid JSON.
- Can then safely create calculated field based off JSON contents.

XML vs JSON – Data Type – Nesting Issue

XML

```
SELECT
CONVERT(xml,
'<TextXML>I typed this.</TextXML>'
) AS 'OuterTag'
FOR XML PATH('');
```

Results:

```
<OuterTag>
<TextXML>I typed this.</TextXML>
</OuterTag>
```

JSON

```
SELECT
'{"TextJSON":"I typed this."}' AS
'OuterTag'
FOR JSON PATH;
```

Results:

```
{"OuterTag":"{\"TextJSON\":\"I typed
this.\"}"}
```

XML vs JSON – Data Type – Nesting Fix

```
SELECT
	(
		SELECT
			'I typed this.' AS TextJSON
		FOR JSON PATH
	) AS 'OuterTag'
FOR JSON PATH;
```

Results:

```
{"OuterTag":{"TextJSON":"I typed this."}}
```

Additional Features (in SQL Server)

XML

- XPath
- DTDs
- Entities
- Schema
- Namespaces
- FLWOR
- XHTML (Sort of)
- SQLXML (Deprecated)

JSON

XML Feature: XQuery

```
DECLARE
  @x xml = '<r><x a="1">y</x><x a="2">z</x></r>';

SELECT
  @x.query('//x[@a>1]'),
  @x.query('//x[text()="z"]');
```

Result:

```
  <x a="2">z</x>
```

XML Feature: XQuery - More Complex

```
DECLARE @x xml =
  '<r>
    <x a="1" b="2">
      <y b="2">PickMe!</y>
      <y b="3">No</y>
    </x>
    <x a="1" b="3">
      <y b="2">No</y>
    </x>
    <x a="2" b="2">
      <y b="2">No</y>
    </x>
  </r>';
```

```
SELECT
  @x.value('
  '(/r/x[@a=1 and @b=2]/y)[1]',
  'varchar(50)');
```

Result:

PickMe!

XML Feature: DTDs / Entities

- SQL Server has “limited” DTD support.
- Provides Entity substitution.
- Provides default attribute values.
- Consumed by XML conversion. (One way trip.)
- Validation not supported by SQL Server.

XML Feature: DTDs / Entities

T-SQL

```
SELECT CONVERT(xml, N'
<!DOCTYPE Test [
<!ENTITY ReplaceMe
"Replacement">
<!ATTLIST Test Attr CDATA
"Default">]>
<Test>
   &ReplaceMe;
   &ReplaceMe;
</Test>
',2);
```

Result

```
<Test Attr="Default">
   Replacement
   Replacement
</Test>
```

XML Feature: Schema

- Provides data validation.
- Provides structure validation.
- Creates “typed” XML.
  - More efficient storage.
  - Allows XML indexes.
- Does not allow entity creation / substitution.
- Schema collection must be created in advance of use.

XML Feature: Schema

T-SQL

```
CREATE XML SCHEMA COLLECTION
TestSchema AS

N'<schema xmlns="http://
www.w3.org/2001/XMLSchema">

<element name="Test"
type="integer" />

</schema>';

GO

SELECT CONVERT(xml
(TestSchema), N'<Test>a</
Test>');

GO

DROP XML SCHEMA COLLECTION
TestSchema;
```

Result

```
Msg 6926, Level 16,
State 1, Line 6

XML Validation: Invalid
simple type value: 'a'.
Location: /*:Test[1]
```

Counterpoint - JSON “Validation”

T-SQL

```
SELECT * FROM
OPENJSON('{"a":test}');
```

Result

```
Msg 13609, Level 16,
State 4, Line 1

JSON text is not
properly formatted.
Unexpected character
't' is found at
position 5.
```

XML Feature - Namespaces

- Allows disambiguation of element names.
- Makes for very ugly XML.
- Namespace requires “prefix” and “namespace identifier”.
  - “Prefix” is shorthand way to reference in XML elements.
  - “Namespace identifier” must be a URL or URN.
    - URLs were chosen with the idea that you would buy the domain to guarantee you owned that “space”.
    - But these don’t have to be actual, Internet accessible locations.
    - SQL Server does not navigate to the URLs.
- Requires special handling and syntax in T-SQL.

XML Feature – Namespaces

T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
Test
</a:x>';
SELECT
   @x.value('(/a:x)
[1]','varchar(50)');
```

Results

```
Msg 2229, Level 16,
State 1, Line 3
XQuery [value()]: The
name "a" does not
denote a namespace.
```

XML Feature – Namespaces

T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
   Test
</a:x>';
SELECT
   @x.value('declare
namespace
a="example.com"; (/a:x)
[1]','varchar(50)');
```

Alternative T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
   Test
</a:x>';
SELECT
   @x.value('(/*:x)
[1]','varchar(50)');
```

XML Feature: FLWOR

- FOR, LET, WHERE, ORDER BY, RETURN
- There's a whole programming language inside of XML.
- You can loop, do calculations, and construct XML.
- There are special cases where this makes sense, but there are often better ways.

XML Feature: FLWOR

T-SQL

```
DECLARE @x xml = N'
   <x>   <a>1</a>
         <b>2</b>
         <c>3</c>   </x>';
SELECT  @x.query('
for $n in x/*
order by ($n/text())[1]*-1
return
   <num>
      {(($n/text())[1])+1}
   </num>     ');
```

Result

```
<num>4</num>
<num>3</num>
<num>2</num>
```

XML Feature: XHTML

- XHTML is not a SQL Server feature per se.
- You can construct XHTML code using T-SQL XML features.
- You could use this to construct entire web pages (laboriously).
- This will perform much more poorly than string concatenation.
- But you don’t have to worry about syntactical mistakes.
- You could use this to construct pretty HTML-style emails to be sent using SQL Server, without any outside toolset.

XML Feature: XHMTL

T-SQL

```
SELECT
    'Hello, world!' AS 'div'
FOR
    XML
    PATH ('body'),
    ROOT('html'),
    TYPE;
```

Result

```
<html>
    <body>
        <div>Hello, world!</div>
    </body>
</html>
```

Gotchas

XML

- Must have root element (but SQL more forgiving).
- No repeated attribute names.
- Funky whitespace handling.
- No colons in element names.
- No low level ASCII (except CR LF TAB).
- Character restrictions for element names.
- Exact text not preserved in SQL Server XML data type.

JSON

- No comments.
- Repeated key names are variably supported. (Use array instead.)
- “Root” can be array or object.

Conciseness

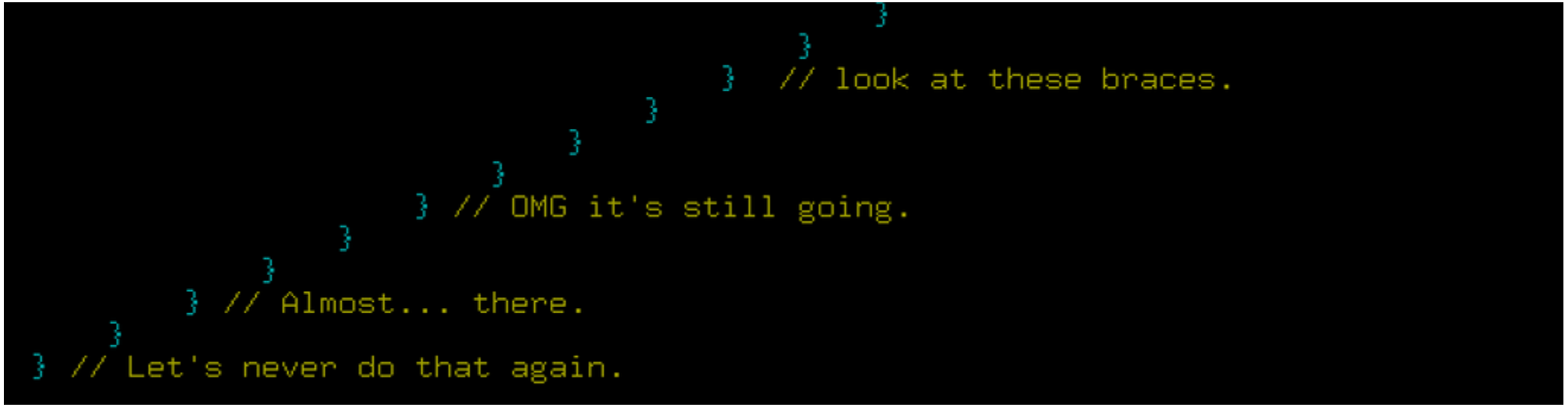
- Shorter is not necessarily better.
- Raw binary data is most efficient, but it’s not human readable.
- Even human readable code can be impractically terse.

This is a valid program written in the language 05AB1E. It is a “quine”, a program which prints itself without reading its source code.

0"D34çý"D34çý

Conciseness

- Sometimes, more characters are better.
- XML's extra characters come from labeling the end of a section.
- That can help with navigation in a complex document.



Conciseness

- XML stored as optimized binary (MS-BINXML).
- Compression is now in SQL Server Standard edition (SP1).
- HTTPs/HTTP2 makes automatic compression widespread.

Speed

- JSON parsing is significantly faster in SQL Server and elsewhere.
- XML, especially with multiple XQuery expressions, will create very complex query plans. Even if not slower to execute, slower to compile.

XML vs JSON - Winner?

XML

- Microsoft Ecosystem
- XQuery
- Features
- Close Tags

JSON

- Web Ecosystem
- Simpler
- Faster

Riley Major

- @RileyMajor | PASSMN@RileyMajor.com
- Enterprise Architect
- Manna Freight Systems, Inc.
- Worked with SQL Server since May of 2000
- PASSMN Board – Director of SQL Saturday
- Conference speaker
- Father of three girls

Image Credits

- Page 1
  - 4381948277_66eb46cc2e_o.jpg
  - GRU - Melee 333: Dragoon Jumping and Double Spears Technique
  - Kevin Thai
  - https://www.flickr.com/photos/kthai/4381948277